

Inhaltsverzeichnis

Vorwort

[An wen sich dieses Buch richtet und wie man es verwendet](#)

[Was dieses Buch enthält](#)

[Einige wichtige Hinweise vor dem Beginn](#)

[Was Sie vor einem Start haben sollten](#)

Woche 1 im Überblick

Tag 1 Einführung & Grundlagen

[Grundlagen des Internets und des WWW](#)

[HTML](#)

[Style Sheets](#)

[Verschiedene Browser](#)

[Tools zur Webseitenerstellung und JavaScript-Programmierung](#)

[Weitere sinnvolle Programme](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 2 Das Wesen von JavaScript

[Was ist JavaScript?](#)

[Erste JavaScript-Beispiele](#)

[Client-seitige versus Server-seitige Scripte](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 3 Wichtige HTML-Grundlagen

[HTML-Grundlagen](#)

[HTML-Grundstrukturen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 4 Einbindung von Scripten und Style Sheets in Webseiten

[Die Einbindung von Scripten in eine Webseite](#)

[Die Einbindung von Style Sheets in eine Webseite](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 5 Wichtige HTML-Anweisungen

[Zeilen- und Absatzschaltungen](#)

[Ausrichtungsanweisungen](#)

[Allgemeine Gruppierungsanweisung](#)

[Trennlinien](#)

[Überschriften](#)

[Grafiken](#)

[Textformatierungen](#)

[Sonderzeichen](#)

[Schrift-Anweisung](#)

[Hyperlinks](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 6 JavaScript-Praxis

[Dynamischer Aufbau von Webseiten](#)

[Kontrollstrukturen](#)

[Zugriff auf den Browser](#)

[Zugriff auf die Besucherplattform](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 7 Fehlerbeseitigung und Optimierung in JavaScript

[Fehler](#)

[Fehler im Vorfeld vermeiden](#)

[Sicherungsmaßnahmen](#)

[Konkrete Fehlerbehandlung](#)

[Optimierung von Webseiten und Scripten](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Woche 1 im Rückblick

Woche 2 im Überblick

Tag 8 Variablen, Datentypen, Funktionen und Methoden

[Funktionen, Prozeduren und Methoden](#)

[Variablen, Datentypen und Literale](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 9 Schlüsselworte, Operatoren und Wertzuweisungen

[JavaScript-Schlüsselworte](#)

[Operatoren und Operanden](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 10 Anweisungen und Kontrollfluss-Steuerung

[Allgemeine Anweisungen](#)

[Auswahl- und Iterationsanweisungen](#)

[Kombinierte Anwendungen von Kontrollfluss-Anweisungen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 11 Der Aufruf von JavaScripten

[Was ist ein Ereignis?](#)

[Aufruf beim Start/Verlassen der HTML-Datei](#)

[Klick und Doppelklick auf eine Referenz](#)

[Überstreichen mit dem Mauszeiger](#)

[Mausbewegungen](#)

[Tastaturereignisse](#)

[Aktivierung und Verlassen eines Elements](#)

[Reaktion auf Änderung](#)

[Reaktion auf Zurücksetzen oder Absenden eines Formulars](#)

[Reaktion bei der Selektion von Text](#)

[Unterbrechung eines Bildladevorgangs](#)

[Der explizit Aufruf eines Event-Handlers](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 12 Erweiterte Gestaltungsmöglichkeiten einer Webseite

[Listen und Aufzählungen](#)

[Tabellen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 13 Vorgefertigte JavaScript-Funktionen und Top-Level-Eigenschaften

[Top-Level-Eigenschaften](#)

[Konvertierung eines Wertes in eine Zahl](#)

[Textausgabe von Werten](#)

[Kodieren und Dekodieren von Zeichenketten](#)

[Test auf Unendlichkeit](#)

[Schutzmaßnahmen](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 14 JavaScript und Java, allgemeine Multimedia-Techniken

[Allgemeines zur Einbindung von Java-Applets und multimedialen Objekten](#)

[PlugIns](#)

[Der <OBJECT>-Tag](#)

[Die Applet-Referenz mit <APPLET>](#)

[Zugriff auf Java-Funktionalitäten aus Scripten](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Woche 2 im Rückblick

Woche 3 im Überblick

Tag 15 Objekte in JavaScript - Grundlagen

[Was sind Objekte?](#)

[Die Standardobjekte von JavaScript](#)

[Objektfelder und Arrays allgemein](#)

[Die Objekthierarchie](#)

[Zugriff auf Objekt-Eigenschaften und -Methoden](#)

[Die with\(\)-Anweisung](#)

[Das Entstehen von Objekten](#)

[Eigene Objekte und das Schlüsselwort this](#)

[Prototype](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 16 Frames und das window-Objekt

[Das window-Objekt](#)

[Frames](#)

[JavaScript auf Frames anwenden](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 17 Globale Ereignisbehandlung

[Das event-Objekt in JavaScript](#)

[Das Netscape-Ereignismodell](#)

[Die Microsoft-Variante](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 18 Top-Level-Objekte und das Objekt navigator

[Array](#)

[Boolean](#)

[Date](#)

[Function](#)

[Math](#)

[Number](#)

[RegExp](#)

[screen](#)

[String](#)

[Das navigator-Objekt](#)

[Das history-Objekt](#)

[Das location-Objekt](#)

[Trennung von Browser- und Plattformwelten](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 19 Die Webseite als Objekt

[Dynamic HTML und DOM](#)

[Das document-Objekt](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 20 Formulare

[HTML-Grundlagen für Formulare](#)

[Der JavaScript-Zugriff auf Formulare](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Tag 21 Style Sheets und das Style-Objekt

[Das Style-Objekt](#)

[Style Sheets](#)

[JavaScript-Zugriff auf CSS](#)

[Zusammenfassung](#)

[Fragen und Antworten](#)

[Workshop](#)

[Kontrollfragen](#)

Woche 3 im Rückblick

Anhang A Wichtige Tabellen zu JavaScript, HTML, Maskierungen, Quellen usw.

[Die JavaScript-Token in Bezug auf die ECMA-Norm](#)

[Die unterstützten Varianten von JavaScript bzw. JScript in den wichtigsten Browserversionen](#)

[Die unterstützten HTML-Versionen in den wichtigsten Browserversionen](#)

[Die Auflistung aller HTML-Elemente und -Attribute](#)

[Tabelle mit Maskierungen](#)

[Umrechnungstabelle zwischen hexadezimalen und dezimalen Zahlen](#)

[MIME-Typen](#)

[Weiterführende Quellen zu JavaScript, HTML und Style Sheets](#)

[Anhang B Antworten auf die Kontrollfragen](#)

[Anhang C Die CD zum Buch](#)

[Stichwortverzeichnis](#)

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Vorwort

Für viele Autoren ist das Vorwort das schwierigste Kapitel bei einem neuen Buch. Für mich normalerweise nicht. So etwas geht mir recht locker von der Hand. Dieses Buch zu JavaScript ist jedoch beileibe nicht mein erstes Buch. Auch nicht mein erstes Buch zu JavaScript. So langsam ertappe ich mich bei dem Gedanken an den Prolog zu »JavaScript in 21 Tagen« dabei, bereits in anderen Büchern verwendete Elemente wiederholen zu wollen. Nicht schön. Auch dem Vorwort gehört Sorgfalt gewidmet. Immerhin soll es ja Sie - den/die potentielle(n) Leser/in - überzeugen (was natürlich im Interesse des Autors ist - es gilt eine Familie zu ernähren): im Wesentlichen von dem Thema, der Art der Behandlung und nicht zuletzt dem Stil des Autors. Die exakten fachlichen Details kann man eher dem Cover oder dem Inhaltsverzeichnis entnehmen. Also frisch ans Werk und eine individuelle Präambel geschaffen.

Frisch ist zugleich die Überleitung zu der Situation, wann und warum das Vorwort so entsteht, wie es entsteht. Ich sitze samt Andrea und meinen Zwillingen auf einer dänischen Insel, habe mir eigentlich Urlaub genommen, aber das Wetter ist einfach gerade zu frisch, um im Meer zu baden. Da das Notebook vorsorglich im Wohnmobil einen Platz gefunden hatte, kann die Zeit aber gut genutzt werden. Sand, Sonne, Wind und halt ein Vorwort zu einem JavaScript-Buch. Warum denn nicht? Andere Autoren erstellen ein Vorwort unter Zeitdruck kurz vor Abgabetermin.

Frisch ist obendrein eine recht interessante Charakterisierung von JavaScript. Frisch, spannend, leistungsfähig. Diese geniale Scriptsprache hat ziemlich frischen Wind in das Internet gebracht, frischt eingeschlafene Webseiten auf und erlaubt erfrischende Aktionen rund um Browser, WWW und die übrigen am gesamten Prozess beteiligten Rechner und Programme.

Zeitgleich zu der Erstellung dieses Buchs lief eine mehrmonatige umfangreiche Schulungsmaßnahme, in deren Rahmen die Teilnehmer zu Internet Publishern ausgebildet wurden. Mein Part als Dozent dabei bestand aus den Themen Internet-Grundlagen, HTML, Style Sheets und JavaScript - verteilt in mehrere Blöcke in dieser Reihenfolge. Insbesondere bei HTML und Style Sheets häuften sich die Fragen von Teilnehmern, ob man dieses oder jenes im Rahmen einer Webseite machen könne. Manche der Fragen ließen sich mit der gerade aktuellen oder einer davor besprochenen Technik behandeln, aber zunehmend lautete die Antwort: Dazu kommen wir bei JavaScript. Noch bevor das Thema JavaScript überhaupt erreicht wurde, verfestigte sich bei den Teilnehmern der Eindruck, dass JavaScript die Lösung für fast jedes tiefergehende Problem in einer Webseite beinhaltet. Ob Abfrage des Browser-Typs oder der Browser-Version, ob Animationen mit Style Sheets, ob Kontrolle von Anwendereingaben in Formularen oder der individuelle Aufbau von Webseiten. Ich musste auf eine neue Frage nur noch eine längere Pause folgen lassen - die Reaktion war Gelächter und die kollektive Antwort des Auditoriums: »Dazu kommen wir bei JavaScript«.

JavaScript ist sicher nicht die Lösung für jedes Problem im Web, aber es bietet so viele interessante Möglichkeiten, dass eine Beschäftigung damit äußert nützlich und vor allem spannend und interessant ist.

An wen sich dieses Buch richtet und wie man es verwendet

Zielgruppe für das Buch sind all diejenigen, die sich in den Kopf gesetzt haben, JavaScript von Grund auf zu lernen. Was für Voraussetzungen sollte ein(e) Leser(in) aus dieser Zielgruppe mitbringen? Sie oder er sollte

- einen Rechner bedienen können.
- mit einem grafischen Betriebssystem und einer Maus umgehen können.
- wissen, was Dateien sind, und Dateien von Verzeichnissen unterscheiden können.
- halbwegs das Tastaturfeld eines Rechners kennen und einigermaßen effektiv Texte abschreiben können.

Das Buch ist jedoch so aufgebaut, dass keinerlei Programmierkenntnisse vorausgesetzt werden, nicht einmal HTML. Von den Grundlagen der Programmierung sollen Sie in 21 Tageslektionen bis zu anspruchsvollen JavaScript-Themen geführt werden. Die Themen bauen so aufeinander auf, dass sich das Durcharbeiten der einzelnen Lektionen in der angegebenen Reihenfolge empfiehlt. Immer am Beginn eines Sieben-Tage- Abschnitts erfolgt eine knappe Vorschau auf das vor Ihnen liegende Material. Am Ende der Woche werden die erreichten Ziele zusammengefasst.

Wenn Sie sich sodann die Frage stellen, ob Sie in 21 Tagen wirklich programmieren lernen können, lautet die Antwort ja. Da verspricht der Titel wirklich nicht zu viel¹.

Können Sie auch in 21 Tagen mit JavaScript programmieren lernen? Antwort - ja. Wahrscheinlich viel besser und leichter als mit den meisten anderen Programmiersprachen, denn JavaScript ist von weit einfacherer Struktur als Sprachen wie C/C++, Java oder Delphi. JavaScript bietet zudem durch die einfache Integration in Webseiten bereits schnelle Anfangserfolge. Sie brauchen auch keine besonderen Programme zur Erstellung von JavaScripten zu lernen. Sie sollten nur einen einfachen Texteditor beherrschen und HTML-Dateien in einen Browser laden können (und auch das wird im Rahmen dieses Buchs behandelt).

Stellen wir noch eine dritte Frage: Werden Sie in 21 Tagen mit Hilfe dieses Buchs zum/r professionellen Programmierer/in? Antwort - vielleicht, aber wahrscheinlich nicht. In diesem Punkt muss man realistisch bleiben. Sie werden danach in JavaScript programmieren können, sogar eindrucksvolle Dinge wie Animationen oder Zugriffe auf Java-Elemente. Um jedoch wirklich professionell arbeiten zu können, benötigt man Erfahrung und ein gewisses Talent. Talent insofern, als manche Menschen einfach nicht für die Programmierung geschaffen sind. Meist haben diese aber auch kein Interesse daran (und werden dieses Vorwort nie lesen). Erfahrung hingegen kann man mit der Zeit erwerben. Und die müssen Sie sich auch nach der Durcharbeit des Buchs noch geben. Viel Programmieren, aber auch Fehler machen, Fehler finden und beseitigen und so die Erfahrung aufbauen, ein guter Programmierer oder eine gute Programmiererin zu werden. Das Buch verschafft Ihnen das Rüstzeug, die Werkzeuge dazu. Vielleicht beschreibt ein Vergleich die Situation ganz gut: Auch Goethe hatte nur das Rüstzeug wie alle anderen Deutschen seiner Zeit - das Alphabet und die deutsche Sprache. Gelernt hat er dies wohl wie seine Mitmenschen in der Schule (das soll in dem Beispiel der Aufgabe dieses Buchs entsprechen). Dennoch hat er sein so vermitteltes Rüstzeug professioneller eingesetzt als die meisten seiner Zeitgenossen. Goethe hatte das Talent zum Schreiben und mit der Zeit die Erfahrung, Worte und Buchstaben so einzusetzen, dass etwas Positives aus deren Aneinanderreihung entstand. Programmierung hat dazu einige Ähnlichkeiten (obgleich ich nicht so

vermessen sein möchte, JavaScript-Programmierung mit den Werken von Goethe gleichzusetzen).

Was dieses Buch enthält

Was steht Ihnen in den nächsten 21 Tagen bevor (oder der entsprechenden Zeit in Ihrem persönlichen Rhythmus), wenn Sie sich auf den Kurs einlassen? Grundsätzlich werden wie gesagt erst einmal keinerlei Programmierkenntnisse vorausgesetzt. Das Buch soll Sie über eine fundamentale Erklärung von Programmiertechniken und -begriffen an Programmierung heranzuführen. Wir werden uns dabei selbstredend besonders intensiv um die Umgebung von JavaScript kümmern. Also das Internet und das WWW sowie wie Scripte in Webseiten verankert werden, Browser, Protokolle und den Client-Server-Mechanismus. Natürlich wird geklärt, was JavaScript ist und wie es im Vergleich zu anderen Techniken wie HTML, VBScript oder Java zu sehen ist. Diese werden teilweise sogar genauer behandelt, wenn es für die Arbeit mit JavaScript sinnvoll ist. Auch das Schlagwort *Objektorientierte Programmierung* mit seinen Ausprägungen wird wesentlich behandelt. Ein weiterer wichtiger Schwerpunkt ist das so genannte Eventhandling, also die Reaktion auf Anwenderereignisse im Rahmen einer Webseite.

Einige wichtige Hinweise vor dem Beginn

Folgende Punkte sollten Sie generell im Buch beachten:

- Wenn Grafiken, Videos oder Tondateien innerhalb von Beispielen verwendet werden, finden Sie diese auf der Buch-CD. Die Dateien samt den Listings der Beispiele sind nach Tagen gegliedert dort vorhanden.
- In den Listings der Beispiele ist es oft auf Grund satztechnischer Gründe notwendig, dass Zeichenketten (in Hochkommata eingeschlossener Text) auf mehrere Zeilen verteilt werden. Bei der Programmierung dürfen Sie aber solche Zeichenketten nicht auf verschiedene Zeilen verteilen. Ausdrücke in eckigen Klammern sind Platzhalter in den Syntaxbeispielen.
- Grundsätzlich sollten Sie beim Abtippen der Listings auf Groß- und Kleinschreibung und auch sonst auf identische Schreibweisen achten.
- Die Schreibweise von einigen Begriffen ist gerade bei computerbezogenen Themen sehr umstritten. Wie soll nun ein aus dem Englischen kommender Begriff wie »PlugIn« geschrieben werden? So oder »Plugin« oder gar »Plug-In«? Oder was ist mit der Mehrzahl von »Script«? Ist nun »Scripts« oder »Scripte« korrekt? Oder was ist für den Fall einer elektronischen Post zu verwenden? »Email«, »EMail« oder »E-Mail«? Die Liste geht weiter: »Client-seitig« oder »clientseitig«, »Server-seitig« oder serverseitig«, »Eventhandling« oder »Event- Handling«? Es lassen sich viele Begriffe finden, die es in diversen Schreibweisen gibt. Noch schlimmer wird die Situation, wenn es um vorangestellte Artikel geht. »Der Tag« oder »das Tag«, »der Frame« oder »das Frame« usw. Wenn man sich schon in der deutschen Rechtschreibung nicht einigen kann² - wie soll sich dann eine verbindliche und vor allem sinnvolle Schreibweise für diese eingedeutschten Spezialworte etablieren? Und wer soll sie festlegen? Etwa Experten der deutschen Rechtschreibung, die weder von dem Fachthema eine Ahnung haben noch in der Szene verkehren und damit die dort übliche Sprache nicht kennen? Das ist sicher keine Lösung, denn damit wird diese Notation nie akzeptiert. Im Rahmen verschiedener Seminare und Vorträge habe ich meinen Zuhörern einige, mir mehrfach aufgefallene, Bezeichnungen vorgestellt und gebeten, sich jeweils für bestimmte Ausdrucksweisen zu entscheiden. Resultate waren Schreibweisen wie »der Tag« statt »das Tag« oder »Scripte« statt »Scripts« usw. Das

kann aber nur ein paar persönliche Momentaufnahmen darstellen. Und obwohl das ganze Dilemma ein schwebender Prozess ist, muss ein Autor sich bei einem Buch zu einem solchen Thema bei jedem Begriff für eine Schreibweise entscheiden. Sollten Sie jedoch eine andere Schreibweise favorisieren - ich bitte um gnädige Rücksicht ;-).

Was Sie vor einem Start haben sollten

JavaScript ohne einen Computer zu lernen gleicht dem Versuch, sich Schwimmen an Hand eines Buchs beizubringen und auf die konkreten Schwimmübungen im Wasser zu verzichten. Sie sollten auf jeden Fall einen Computer zum Üben parat haben. Welchen Typs ist nicht so wichtig, denn JavaScript ist nicht an einen bestimmten Computertyp gebunden. Ob Sie nun einen PC verwenden, einen Mac oder gar einen alten Großrechner, den Sie zufällig noch auf dem Speicher haben³ - im Prinzip ist es egal. Auch die Leistungsfähigkeit des Rechners und der restlichen Hardware spielt - weder bei JavaScript, noch im Internet allgemein (entgegen vielen Behauptungen in der Werbung) - keine primäre Rolle⁴. Sie sollten nur über ein passendes Betriebssystem verfügen und die nachfolgend noch angeführten Programme verwenden können. Ideale Kandidaten an Betriebssystemen sind natürlich Linux und alle Unix-Derivate, Mac-OS oder BeOS. Haben Sie nicht? Egal, auch Windows-Anwender (und das werden die meisten Leser sein) sind mit von der Partie. Nicht nur das - Windows wird - wegen seiner Verbreitung und umfangreichen Ergänzung mit darunter laufenden Internet-Programmen⁵ - unser Referenz-Betriebssystem. Windows 95/98 und NT/2000 sind uneingeschränkt sinnvoll, aber sogar Windows 3.x kann mit einigen Einschränkungen noch verwendet werden (obwohl neuere Dinge damit nicht mehr funktionieren). Viel entscheidender als der verwendete Rechner oder das zugrunde liegende Betriebssystem sind die konkret das Internet betreffenden Voraussetzungen.

Da JavaScript ein Internet-Thema ist, sollten Sie einen Internet-Zugang haben und auch schon etwas Übung damit. Sie sollten also in der Lage sein, das Internet (zumindest das World Wide Web) zu verwenden. Insbesondere sollte Ihnen ein relativ aktueller Browser zur Verfügung stehen und Sie sollten ihn einigermaßen bedienen können. Oder lassen Sie uns das noch schärfer formulieren - für eine sinnvolle Programmierung in JavaScript (beziehungsweise jeder anderen Web-Technologie inklusive der Seitenbeschreibung durch HTML) sollten Sie verschiedene Browser zum Testen der Resultate zur Verfügung haben. Wir werden diesem Aspekt im Rahmen des Buchs große Aufmerksamkeit widmen und Quellen angeben, von wo Sie gegebenenfalls Browser beziehen können. Unsere Referenz-Browser werden hauptsächlich der Internet Explorer 5.0, der Netscape Navigator 4.7 und Opera 3.6 sein.

Was Sie abschließend noch benötigen, ist ein Programm, mit dem Sie HTML-Seiten und JavaScripte schreiben können. Dazu langt freilich jeder Texteditor, wie er in fast jedem Betriebssystem mitgeliefert wird. Selbst Notepad, ein extrem einfacher Editor aus dem Zubehör von Windows, reicht aus. Das Speichern und Öffnen von Dokumenten mit einem Editor sollte Ihnen halbwegs vertraut sein, aber auch das werden wir an Hand von einem Windows-Editor behandeln.

So, jetzt können Sie frisch ;-) ans Werk gehen und mit dem Erlernen von JavaScript beginnen.

Ralph Steyer

www.webscripting.de

1 Natürlich können Sie sich auch mehr als einen Tag Zeit für eine Lektion lassen.

2 Denken Sie nur an den Streit, ob die »so genannte« bzw. »sogenannte« neue Rechtschreibung sinnvoll ist oder die Sprache nur schwerer macht und ihrer Nuancen beraubt.

3 ;-)

4 Wobei - gerade bei neueren Techniken, die viel Rechenarbeit vom Browser erfordern - ein möglichst guter Rechner natürlich der Idealfall ist.

5 Nicht weil es das ideale Betriebssystem für das Internet ist, da sind viele andere Systeme mindestens gleichwertig oder besser.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 1

Die erste Woche liegt nun vor Ihnen. In dieser Woche wollen wir

- Grundlagen des Internets, des WWW und von Netzwerken allgemein erörtern,
 - die Bedeutung von HTML und Style Sheets klären,
 - Browser und Programme zur Erstellung von Webseiten und JavaScripten besprechen,
 - die Integration von Scripten und Style Sheets in Webseiten durchnehmen,
 - JavaScript-Grundlagen klären und erste Scripte erstellen,
 - JavaScript im Web in Vergleich zu weiteren Techniken einordnen und
 - die Fehlerreduzierung bzw. -behandlung und Lokalisierung samt der Optimierung von Webseiten, Scripten und Programmen im Allgemeinen durchsprechen.
-

Woche 1

Tag 1

Einführung & Grundlagen

Nochmals willkommen zu »JavaScript in 21 Tagen«. Vielleicht stellen Sie sich gerade die Frage: »Was kann ich mit JavaScript anstellen?«. In den nächsten drei Wochen wollen wir diese Frage beantworten.

Grundsätzlich werden Sie aber sicher schon eine Vorstellung davon haben, was JavaScript ist - eine an Java angelehnte, offene, plattformunabhängige Scriptsprache. Sie wurde von der Firma Netscape (<http://www.netscape.com>) in Kooperation mit Sun Microsystem (<http://www.sun.com>) entwickelt.



JavaScript wird auf Grund des ähnlichen Namens oft mit Java verwechselt. Die Sprache ist jedoch nur sehr weitläufig mit Java verwandt, obwohl bei der Entwicklung von JavaScript mit Sun der Erfinder von Java mit im Boot saß. Wir werden auf genauere Details später im Buch noch eingehen.



Neben JavaScript werden wir gelegentlich JScript erwähnen. Bei JScript handelt es sich um einen Ableger von JavaScript, welcher hauptsächlich der Unterstützung von JavaScripten im Internet Explorer der Firma Microsoft (<http://www.microsoft.com>) dient. Dort gibt es aus Lizenzgründen keinen direkten JavaScript-Interpreter. JScript stimmt in der Syntax in großen Bereichen mit JavaScript überein.

Grundsätzlich basiert jede Webseite auf HTML (HyperText Markup Language). So wichtig und grundlegend HTML jedoch für das Web ist - eine rein in HTML erstellte Webseite ist in ihrer Leistungsfähigkeit in alle Richtungen stark begrenzt und muss sich für viele Situationen supplementärer Technologien bedienen. Ob Style Sheets, Grafiken, Videos, Java-Applets, ActiveX-Controls oder Scripte - alle ergänzen im Rahmen einer Webseite HTML in gewisse Richtungen. Und alle verankern sich mittels HTML in einer Webseite, werden als Referenz einfach in den HTML-Code eingebunden oder direkt hineingeschrieben.

JavaScript ist in vielerlei Hinsicht die ideale Ergänzung zu HTML. JavaScript ist die wohl am weitesten unterstützte Scriptsprache für das World Wide Web, dazu noch leistungsfähig und dennoch recht einfach zu lernen. Mit einer Scriptsprache wie JavaScript kann man den eingeschränkten Client-Server-Prozess (was das ist, werden wir in dieser Lektion noch besprechen) einer »normalen« Internet-Verbindung im Rahmen des WWW (etwa mit HTML und CGI - Common Gateway Interface) mit Intelligenz aufwerten oder statische HTML-Seiten dynamisieren. Zudem erlauben JavaScripte die Programmierung von Clientprogrammen (Browser), optimieren, steuern oder kontrollieren Interaktionen, automatisieren Vorgänge, steuern und

beeinflussen andere ergänzende Technologien (wie sie gerade angeführt wurden) und noch vieles mehr. Obwohl JavaScript von relativ einfacher Struktur ist, erlaubt der potentielle Zugriff auf vorgefertigte Module (Objekte) die Erstellung von vielen komplexen Vorgängen.

Wir wollen am ersten Tag unserer JavaScript-Tour die Hauptsache - eben JavaScript - erst einmal etwas hinten anstellen und zuerst grundlegende Begriffe rund um JavaScript und vor allem deren Umgebung im Allgemeinen klären. Wundern Sie sich nicht, wenn die einführende Etappe recht umfangreich ist - sie beinhaltet viel grundlegende, aber recht eingängige Theorie und kann deshalb leicht in einem Tag bewältigt werden. Sie lernen in dieser ersten Lektion Folgendes:

- Grundlagen des Internets und des WWW
- Die Bedeutung von HTML und einige Hintergründe
- Die Bedeutung von Style Sheets
- Die Funktion von Browsern
- Vorstellung verschiedener Browser
- Vorstellung von Programmen zur Webseitenerstellung und JavaScript- Programmierung
- Die Funktion von Protokollen
- Die verschiedenen Dienste im Internet
- Der Client-Server-Mechanismus

Grundlagen des Internets und des WWW

Es scheint so, dass man kaum jemand in einem Buch des 21. Jahrhunderts den Begriff Internet noch erklären muss. Selbst Personen, welche keinen Videorecorder programmieren könnten (wobei das komplizierter sein kann, als ein JavaScript zu schreiben ;-)), reden über das Internet mit, nutzen es möglicherweise oder kaufen gar Aktien von DOT-COM-Firmen¹. Dass das Internet ein weltweites Netzwerk mit unzähligen Einsatzmöglichkeiten ist, hat man wahrscheinlich im hintersten Winkel der zivilisierten Welt bereits gehört. Internet als Begriff gehört mittlerweile zum normalen Sprachgebrauch. Das heißt aber noch lange nicht, dass alle, die über Internet reden, auch wissen, was sie da von sich geben oder was sich hinter in die Umgangssprache eingegangenen Fachbegriffen überhaupt verbirgt². Es ist einfach zu »hipp«, Begriffe aus der IT-Branche einzustreuen. An sämtlichen Stellen, wo so etwas Sinn machen könnte oder einfach nur gut klingt. Wer nachfragt, erntet vielfach ein mildes Lächeln statt Erklärung. Was aber wie gesagt oft nicht bedeutet, dass der Gefragte die Antwort überhaupt weiß.

Dies äußert sich vielfach an ganz einfachen Grundfragen. Wenn ich beispielsweise in der Einleitung zu meinen Internet-Schulungen die Frage nach dem Alter des Internets stelle, wird diese Frage meist falsch beantwortet. Das Alter des Internets wird überwiegend viel zu jung geschätzt. Oder die Frage, welche wichtigen Dienste die Säulen des Internet bilden. Schon der Begriff *Dienste* ist für viele Teilnehmer ein Rätsel. Und das, obwohl sie diese Dienste teilweise wie selbstverständlich nutzen.

Das Wissen über das Internet und dessen Grundlagen erscheint mir bei vielen Leuten wie ein riesiges Gebäude, das auf einem maroden Fundament steht und dessen tragende Säulen extrem morsch sind. Zwar ist eigentlich überall ein gewisses Basiswissen da, man kann irgendwie mit Dingen umgehen, aber wehe es wird nachgebohrt. Im Nu trifft man auf Hohlräume, und schnell bricht das gesamte Gebäude zusammen. Dies kann aber kaum als Vorwurf an die breite Masse der Internet-Anwender gewertet werden. Diese werden einfach von Unmengen von nicht näherer erläuterten Informationen und falsch eingesetzten Begriffen überschüttet. Es soll ein Vorwurf an diejenigen sein, welche immer wieder Begriffe und Hintergründe voraussetzen, die erklärungsbedürftig sind, oder die Fachbegriffe einfach einsetzen, ohne sie selbst zu verstehen. Wir wollen insoweit diesem Manko abhelfen, als wir die wichtigsten Begriffe - obwohl scheinbar mittlerweile allgemein bekannt - nochmals durchsprechen.

Klären wir als Erstes die oben zitierte Frage nach dem Alter des Internets. Das Internet geht von der Idee her

bis in die sechziger Jahre zurück. Und hat was mit Schweinen zu tun. Zumindest am Rande. Schweinen? Ja, aber im übertragenen Sinn ;-). Genau genommen mit der Invasion in der Schweinebucht in Kuba, mit der Exilkubaner mit Hilfe des CIA Fidel Castro stürzen wollten. Die Kubakrise, die drohende Stationierung von Atomwaffen auf Kuba durch die damalige UdSSR und der gesamte unselige kalte Krieg dieser Epoche bewirkte die Entwicklung eines Rechnernetzwerks, das bei einem potentiellen atomaren Erstschatz der UdSSR die USA zu einem Gegenschlag befähigen sollte. In den späten sechziger Jahren begann in den Vereinigten Staaten von Amerika eine Organisation mit Namen Advanced Research Projects Agency (ARPA) im Auftrag des US-Verteidigungsministeriums, ein solches Rechnersystem zu entwickeln. Kernziele des Systems sollten eine extrem hohe Ausfallsicherheit sein. Insbesondere sollte die Datenübertragung zwischen den einzelnen Rechnern unter allen Umständen funktionieren - auch beim Ausfall eines beliebigen Rechners im Netz und der Zerstörung eines Teils des Leitungssystems.

Wesentlicher Bestandteil war in Bezug auf die Ausfallsicherheit der Verbindungsstrukturen die Unterstützung von heterogenen Leitungsstrukturen. Von permanenten - und damit sehr schnellen - Standleitungen bis zu äußerst flexiblen und sicheren Telefonverbindungen - die unterschiedlichsten Techniken wurden einkalkuliert, um eine möglichst krisensichere Verbindung zu gewährleisten.

Zu Beginn (1969) bestand das ARPANET nur aus vier Computern, aber bereits 1972 waren ungefähr 50 militärische Forschungsanstalten und Rechner in Universitäten daran angeschlossen. In den Jahren 1973/74 bekam das Netz dann ein standardisiertes Übertragungsprotokoll, auf dem auch heute noch die meisten Vorgänge im Internet basieren - TCP/IP (Transmission Control Protocol/Internet Protocol).



Ein Protokoll kann als eine Vereinbarung verstanden werden, wie verschiedene unabhängige Einheiten (Personen, Tiere, Computer, Organisationen, beliebige Datengeräte usw.) so miteinander umgehen können, dass gewünschte Prozesse zuverlässig funktionieren. Dabei werden alle Schritte standardisiert, die für einen erfolgreichen Ablauf notwendig sind. Diese werden dann protokolliert und alle Beteiligten soweit wie notwendig bekannt gegeben.

Als Beispiel kann man die Kommunikation zwischen zwei Faxgeräten heranziehen, die wir einmal abstrakt durchspielen wollen. Das Kommunikationsprotokoll beim empfangsbereiten Fax könnte so aussehen:

- 1. Nehme ankommenden Anruf entgegen.*
- 2. Analysiere, ob Sprachanruf, Datenanruf oder Faxanruf.*
- 3. Bei Sprach- oder Datenanruf unterbreche die Verbindung.*
- 4. Frage beim anrufenden Fax verstandene Übertragungsprotokolle ab.*
- 5. Bestimme in Kooperation mit der Gegenstelle bestes gemeinsames Übertragungsprotokoll.*
- 6. Bestimme in Kooperation mit der Gegenstelle maximal mögliche Übertragungsrate.*
- 7. Warte auf die Übertragung der eigentlichen Daten.*

Die entsprechenden Schritte auf Seiten des anrufenden Faxgerätes könnten so aussehen:

- 1. Warte, bis Anruf angenommen wird.*
- 2. Analysiere, ob Gegenstelle ein Faxgerät.*
- 3. Bei fehlender Rückmeldung durch ein annehmendes Faxgerät unterbreche die Verbindung nach adäquater Zeit.*

4. Beantworte die Frage des angerufenen Fax nach den verstandenen Übertragungsprotokollen.
5. Bestimme in Kooperation mit der Gegenstelle bestes gemeinsames Übertragungsprotokoll.
6. Bestimme in Kooperation mit der Gegenstelle maximal mögliche Übertragungsrate.
7. Starte die Übertragung der eigentlichen Daten.

Die jeweils abwechselnd durchzuführenden Schritte definieren ein Protokoll, mit dem ein Datenaustausch zwischen Faxgeräten unterschiedlichster Art möglich ist (wobei das oben beschriebene Verfahren wie gesagt sehr abstrakt charakterisiert ist). Protokolle auf Netzwerkebene arbeiten genauso, wobei natürlich die jeweilige Situation zu berücksichtigen ist.

Signifikante Eigenschaft von TCP/IP ist die Möglichkeit mehrerer gleichzeitig und parallel aufgebauter Übertragungen. Sowohl mittels einer als auch mehrerer paralleler Leitungen. Das Protokoll wurde zudem so ausgelegt, dass es sich mit den verschiedensten Leitungsqualitäten und Leitungstypen zurechtkam und für die unterschiedlichsten Übertragungsgeschwindigkeiten einzusetzen war. Genau genommen ist TCP/IP kein einzelnes Protokoll, sondern die Erweiterung von IP (1973) um das 1974 auf IP aufgesetzte Erweiterungsprotokoll TCP, welches erst eine fehlerfreie Übertragung gewährleistet.



Das IP-Protokoll beinhaltet als ein zentrales Element ein Adresskonzept, mit dem Rechner im Internet (oder jedem anderen auf diesem Protokoll basierenden Netzwerk) eindeutig identifiziert werden können und das den Datenaustausch zwischen so lokalisierbaren Rechnern erlaubt. Es handelt sich um die so genannten IP-Nummern, welche derzeit³ aus vier Byte bestehen. Da in einem Byte 256 verschiedene Zeichen binär dargestellt werden können ($2 \text{ hoch } 8 = 256$), lässt sich in jedem der vier Byte eine Zahl zwischen 0 und 255 verschlüsseln. Durch Kombination dieser vier Byte lassen sich dann theoretisch $256 \text{ hoch } 4$ Adressen angeben. Allerdings werden durch Nummernkreise für verschiedene Netzwerkgrößen⁴ und reservierte Kombinationen die tatsächlich adressierbaren Rechner ziemlich reduziert. Gewöhnlich werden die IP-Nummern im Dezimalsystem dargestellt und jedes Byte wird mit einem Punkt abgetrennt. Eine fiktive IP-Nummer eines Rechners könnte also so aussehen:

123.210.111.145

Anfang der 80er-Jahre nahmen die nichtmilitärischen Rechner (und damit die Sicherheitsrisiken für militärische Rechner) im ARPANET so stark zu, dass die meisten Teile des militärischen Bereichs in ein eigenes Netz, das MILNET, ausgegliedert wurden. Die zivilen Teile (Forschung, Bildung, Gewerbe) bekamen den Namen Internet. Ende der achtziger-Jahre öffnete sich das Internet dann über die USA hinaus.

Das Internet ist über die Zeit sowohl technisch als auch inhaltlich extrem heterogen (d.h. uneinheitlich) gewachsen. Es ist also im besten Sinn des Wortes chaotisch oder anarchistisch (es gibt keine Kontrollinstitution - das Internet als Ganzes gehört niemanden). Die technische Heterogenität bedeutet vor allem, dass Rechner jeglicher Architektur mit jedem denkbaren Betriebssystem, das TCP/IP unterstützt, zusammengeschlossen werden können. Damit diese aber sich dann auch verstehen, mussten Protokolle festgelegt werden, die auf dem Transportprotokoll TCP/IP aufsetzen und einheitliche Funktionalitäten bereitstellen.



In der Internet-Sprache sind diese auf dem Transportprotokoll TCP/IP aufsetzenden Funktionalitäten die so genannten Dienstprotokolle, welche den verschiedenen Diensten im

Internet zugrunde liegen.

Diese Dienste sind die verschiedenen Funktionalitäten des Internets, welche sich über die Zeit im Netz der Netze gebildet haben. Die meisten sind schon in frühen Phasen des Internets entstanden. Etwa der Dienst Telnet (Name des Dienstprotokolls ist gleichfalls Telnet), mit dem via Internet fremde Computer ferngesteuert werden können. Oder der FTP-Dienst mit gleichnamigem Protokoll für den Dateien-Upload und -Download zwischen Rechnern über das Internet. Fast jeder Internet-Anwender nutzt Email, was ein weiterer eigener Dienst ist (mit verschiedenen zugrunde liegenden Protokollen), und auch vom Usenet bzw. den Newsgroups (Dienstprotokoll ist NEWS) haben viele bereits gehört. Der populärste Dienst im Internet ist jedoch erst Anfang der neunziger Jahre entstanden - das World Wide Web.

HTTP - das Protokoll des World Wide Web

HTTP (Hyper Text Transport Protocol) ist das bekannteste, aber auch eines der neuesten Protokolle im Internet. Es ist das Protokoll des World Wide Web. Wie bei FTP und TELNET etc. handelt es sich ebenfalls um ein Dienstprotokoll, welches auf TCP/IP aufgesetzt wird. Zu seinen herausragenden Eigenschaften gehört die Fähigkeit zur Übertragung von hochauflösenden Grafiken, Sound und Video.

Damit eröffnete es bei seiner Einführung dem Internet zum ersten Mal alle Möglichkeiten, solche Art von Informationen zu übertragen, die gerade auf normale Privatanwender eine große Faszination ausüben.

Gerade die Grafiken und damit verbunden die Möglichkeit einer grafischen Bedienungsfläche waren der Fortschritt, der die Bedienung und Navigation im Internet so vereinfachte, dass aus dem Tummelplatz für Technikfreaks und Mathematiker das Medium für die breite Masse wurde.

Mit HTTP werden wir uns später im Rahmen vom World Wide Web noch ausführlicher beschäftigen, denn im gesamten World Wide Web werden Objekte (HTML-Seiten, Grafiken, Sounds etc.) mit diesem Protokoll übertragen.

Die Adressierung im Internet

Jedes Netzwerk benötigt zum Datenaustausch zwischen Rechnern ein eindeutiges Adressierungskonzept. Also ein Verfahren, wie Informationen sicher und zuverlässig von einem Rechner auf den anderen gelangen können. In auf dem TCP/IP-Protokoll aufbauenden Netzwerken gibt es dazu, wie etwas weiter oben besprochen, die IP-Nummern. In der Praxis werden IP-Nummern von Anwendern beim Surfen im Internet (was streng genommen die Angabe von Rechneradressen ist oder zumindest beinhaltet) jedoch selten verwendet. Im Hintergrund werden die IP-Nummern zwar immer verwendet, der Anwender gibt jedoch meist symbolische Adressen ein, welche auf dem DNS-Konzept basieren. Grund dafür ist, dass IP-Nummern zwar hervorragend dazu geeignet sind, in kompakter und eindeutiger Form eine Adressierung von Rechnern vorzunehmen. Für Menschen jedoch sind diese abstrakten Zahlenkolonnen sehr lästig. Aus diesem Grund wurde den Zahlencodes zusätzlich ein eindeutiger Name zugeordnet und diese Zuordnung weltweit auf speziellen Name-Servern dokumentiert. Diese Zuordnung der eindeutigen Namen erfolgt über ein System von logisch und inhaltlich zusammengehörigen Rechnergruppen, das hierarchisch geordnet ist.



Die logisch zusammengehörenden Bereiche werden Domain genannt, woraus sich der Name für dieses Namenssystem ableiten lässt: Domain Name System, kurz DNS. Genau genommen ist das DNS-Konzept eine verteilte Datenbank, die über mehrere Internet-Rechner gestreut ist.

Genau wie bei den IP-Nummern werden die einzelnen Bestandteile eines DNS-Namens durch Punkte getrennt; die hierarchische Wertigkeit der Stufen ist jedoch von hinten nach vorne zu interpretieren.

- Der hinterste Teil eines solchen Aliasnamens stellt die größte logische beziehungsweise inhaltliche

Einteilung dar (die bzw. der⁵ so genannte Top-Level-Domain), etwa eine Nation oder eine Organisationsform.

- Der unter Umständen mehrteilige Mittelteil bezeichnet den Rechner.
- Der vorderste Teil bezeichnet direkt den einzelnen Server.



Der Begriff Server taucht an dieser Stelle das erste Mal auf. Er gehört dringend erläutert (auch wenn viele Leser schon mit diesem Begriff umgehen), denn kaum ein Terminus in der EDV wird so oft falsch oder ungenau erklärt wie Server. Insbesondere ist der Begriff im Rahmen von JavaScript-Programmierung und für das Verständnis von grundlegenden Internet-Vorgängen ganz wichtig, da das Internet als Client-Server-Prozess zu verstehen ist. Deshalb sollten die Begriffe Client, Server und Client-Server-Beziehung als Ganzes vollkommen klar sein.

Der Ausdruck Server leitet sich aus dem lateinischen Wort Servus ab und bedeutet Sklave oder Diener. Dieser Sklave bedient irgendwelche Klienten - englisch: Clients. Eine Beziehung zwischen Client und Server bedeutet also streng genommen nicht mehr, als dass irgendjemand oder irgendetwas eine Leistung bereitstellt, die irgendjemand oder irgendetwas Weiteres verwenden will. Die oft verwendete Gleichsetzung von einem Server mit einem Rechner (im Sinn von Hardware) ist deshalb schlampig und verhindert das sinnvolle Verwenden dieses Begriffs, wenn man Prozesse in einem Netzwerk beschreiben möchte. Zwar ist es in der Tat so, dass in einem Netzwerk einem Rechner dort in der Regel eine besondere Bedeutung zukommt. Er stellt eine Leistung zur Verfügung (Bereitstellen von Daten oder Programmen, Datensicherung, Bereitstellung von Hardware), die von anderen Rechnern genutzt werden. Er ist gewissermaßen der Diener (oder Sklave) der anderen Rechner und kann dann auch als Server bezeichnet werden. Genau genommen stellt aber nicht der Rechner als Hardware die Ressourcen bereit, sondern die entsprechende Software auf dem Rechner.

Die Gleichsetzung von Server und dem physikalischen Rechner stößt spätestens dann an seine Grenzen, wenn auf einem physikalischen Rechner mehrere Serverprogramme installiert sind oder ein einziges Serverprogramm über mehrere physikalische Rechner verteilt wird (was beides so der Regelfall, nicht die Ausnahme, ist). Ebenso lässt sich mit dem vereinfachten Hardware-Denkansatz nicht so einfach klarmachen, warum ein Rechner oder Programm einmal als Server, dann aber wieder als Client fungieren kann.

Wenn man im EDV-technischen Sinn sowohl beim Server als auch beim Client jeweils an Programme denkt, welche miteinander kommunizieren können und einfach bestimmte ihnen zugewiesene Aufgaben in dem logischen Client-Server-Prozess durchführen, erleichtert dieses Gedankenmodell das Verständnis von dem Client-Server-Prinzip erheblich. Dann wird auch klar, warum ein Programm in Beziehung zu einem Programm als Client, in Beziehung zu einem weiteren Programm jedoch gleichzeitig auch als Server fungieren kann. Ein ziemlich oft verwendetes Beispiel, wo Client und Server sogar gewöhnlich auf einem Rechner laufen (damit ist klar, dass der Server kein mit dem physikalischen Rechner gleichzusetzender Ausdruck sein kann), ist die Einbindung einer Excel-Tabelle in ein Word-Dokument. Excel dient in dieser Konstellation als Server für Word, Word ist dann der Client. Genauso kann zwischen Wordpad und Paint eine Client-Server-Verbindung aufgebaut werden, wenn eine Paint-Grafik in Wordpad eingefügt wird.

Statt Server spricht man bei der Bezeichnung eines Rechners, der eine bestimmte Funktionalität bereitstellt, besser von einem Host.

Serverprogramme werden in der Regel nach ihren Aufgaben aufgeteilt und oft auch benannt. Im

Internet/Intranet benennt man in diesem Zusammenhang die Serverprogramme nach den jeweilig betreuten Internet-Diensten als FTP-Server, News-Server, Mail-Server, WWW-Server usw.

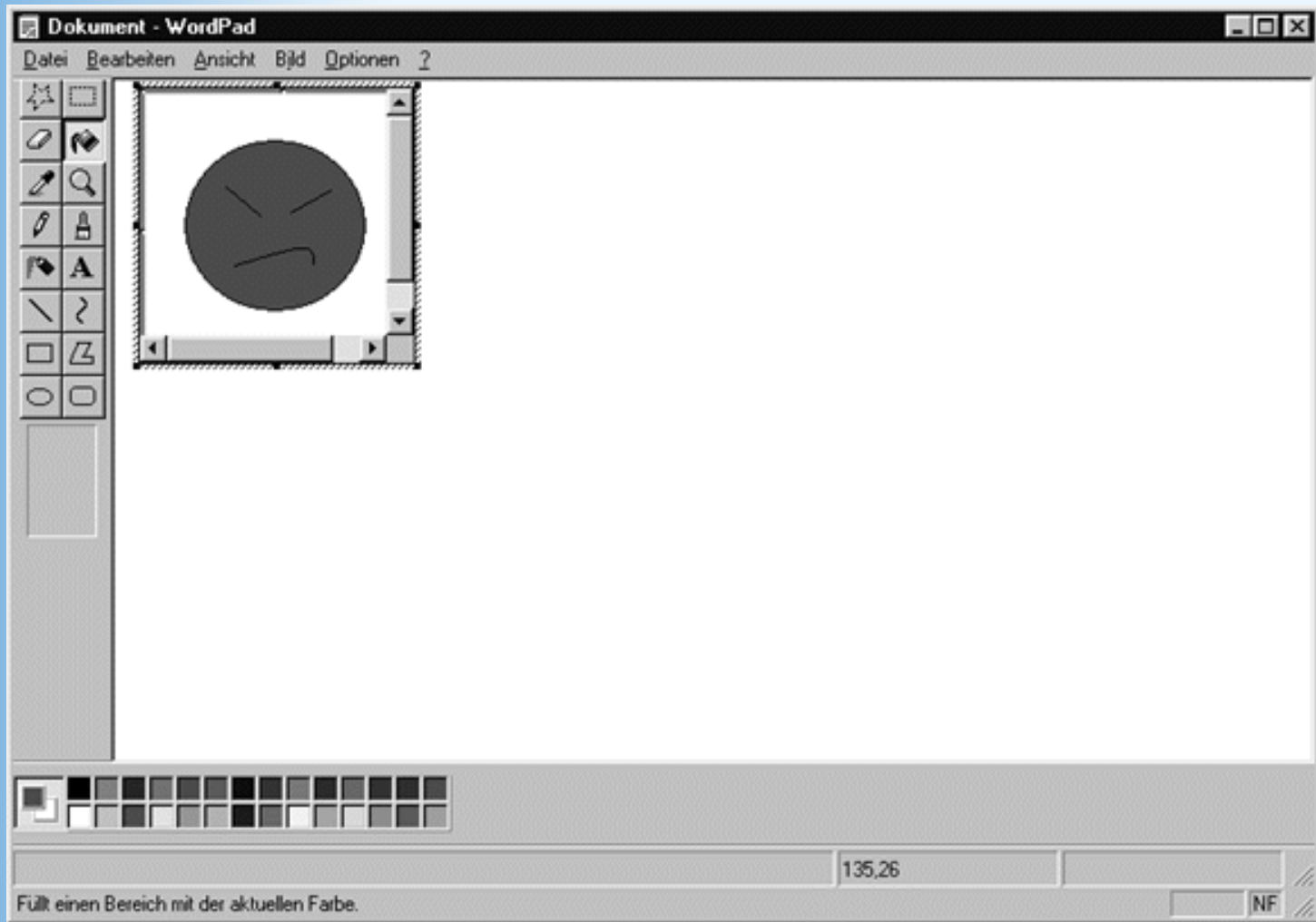


Abbildung 1.1: Eine Client-Server-Verbindung zwischen Paint und Wordpad auf einem Rechner

Die Top-Level Domain wird bei Ländern außerhalb der USA mit der jeweiligen Länderkennung belegt. Diese Ländercodes sind in der ISO- 3166-Norm festgelegt. Hier ein paar Beispiele:

Top Level	Land
at	Österreich
au	Australien
be	Belgien
ch	Schweiz
de	Deutschland
fr	Frankreich
nl	Niederlande
tr	Türkei
uk	Großbritannien

Tabelle 1.1: Ländercodes in der ISO-3166-Norm

Für die USA ist die Top-Level Domain **us** reserviert, aber dort wird ein anderes Konzept bevorzugt. Der

hinterste Teil des DNS-Namens wird für Rechner, die ihren Standort in den USA haben, nach einem anderen Regelwerk definiert (das Internet war ja am Anfang auf die USA beschränkt und da ist kein Hinweis auf die Nation notwendig gewesen). Es gibt in diesem nur in der USA gültigen Regelwerk (daher ist auch die Nation implizit klar) nur sechs (wesentliche) logische Bereiche, in die alle Server in den USA eingeordnet werden. Dies sind folgende Top-Level:

Top-Level	Abkürzung für	Bedeutung
com	commercial	Kommerzielle Unternehmen
edu	educational	Ausbildungsinstitutionen (Universitäten, Schulen ...)
gov	government	Regierungsbehörden
mil	military	Militärische Computer
net	network	Organisationen, die ein eigenes Netzwerk betreiben
org	organization	Nichtkommerzielle Organisationen

Tabelle 1.2: Top-Level Domain in den USA

Beispiele für gültige DNS-Namen sind:

www.mut.de
 www.webscripting.de
 www.rjs.de
 www.sun.com
 www.whitehouse.gov
 www.netscape.com



Die Registrierung eines Rechners in einem bestimmten Land bedeutet in keiner Weise, dass der Rechner dort auch physikalisch stehen muss. Sie können sich einen solchen DNS-Namen wie ein Nummernschild bei einem Auto vorstellen. Eine Erweiterung COM bedeutet nur, dass der Rechner in den USA für das Internet angemeldet ist. So als ob ein Auto in einer Zulassungsbehörde in den USA angemeldet wurde. Wo es dann herumfährt, hat damit überhaupt nichts zu tun. Die Folge davon ist, dass nicht direkt aus der Erweiterung COM geschlossen werden kann, dort nur Informationen und Angebote aus den USA zu finden. Im Gegenteil - auch immer mehr deutsche Firmen, die sich als Global Player verstehen (oder zumindest den Anschein erwecken wollen), lassen sich COM-Adressen reservieren.

Wie wir bereits diskutiert haben, verwendet intern das TCP/IP-Protokoll IP- Nummern zur Adressierung. Die DNS-Namen müssen durch spezielle Namens-Server erst in die zugehörigen IP-Nummern übersetzt werden. Dies erfolgt auf speziellen Domain-Name-Servern, die gemeinsam die DNS- Datenbank bilden. Sofern ein Aliasname nicht auf dem zuerst kontaktierten Server zuzuordnen ist, wird entsprechend der Namenshierarchie in einem komplexen Verfahren der Übersetzungsauftrag an andere Server weitergeroutet, bis die IP-Nummer vollständig ermittelt ist.

Uniform Resource Locator

Rechner selbst sind über die IP-Nummern beziehungsweise ihre DNS- Aliasnamen eindeutig adressiert. Damit ist aber noch nicht viel gewonnen. Wie auf Ihrem lokalen Rechner müssen auch im Internet alle Dateien auf dem Rechner adressiert werden können. Das bedeutet, es werden neben der Rechneradresse zusätzlich einige andere Angaben notwendig. Das sind zum einen die Pfad- und Dateiangaben (wie bei Ihrem

lokalen Rechner), zum anderen aber noch zusätzlich Angaben zum verwendeten Dienstprotokoll.

Der Name dieser vollständigen Internet-Adressen lautet URL und steht für Uniform Resource Locator, was übersetzt ins Deutsche ungefähr einheitliches Adressierungsschema für Objekte im Internet bedeutet. Einheitlich deshalb, weil mit einem URL sowohl die verschiedenen Dienste, wie WWW, FTP, TELNET usw. als auch Rechner selbst oder direkt Dokumente adressiert werden können.

Die exakte Schreibweise ist je nach Dienstprotokoll leicht unterschiedlich, sieht aber in der Regel schematisch folgendermaßen aus:

Dienstprotokoll://[DNS-Name]/[Pfad]/[Datei]

Unter Dienstprotokoll sind beispielsweise `http` oder `ftp` zu verstehen. Danach folgen fast immer der Doppelpunkt und zwei Schrägstriche (eine Ausnahme ist `mailto` zum Aufruf eines Email-Clients oder `news` zum Aufruf des News-Clients - dort folgen nach dem Doppelpunkt keine zwei Schrägstriche).

Die anschließende Angabe des DNS-Namens oder der alternativen IP- Nummer gibt den Server an. Da auf jedem Rechner eine ganze Reihe von verschiedenen Diensten parallel betrieben werden (üblich sind z.B. ein FTP- Server und ein HTTP-Server), kann man bei Bedarf eine zweite Information, den so genannten Port (eine Art Kanal - ähnlich den Kanälen beim Funkverkehr), angeben. Dieser wird dann mit Doppelpunkt abgetrennt dem DNS-Namen nachgestellt. Ports werden durch positive numerische Werte spezifiziert. In der Regel ist die Port-Angabe überflüssig, denn jeder Internet-Dienst besitzt einen Defaultport, der immer dann verwendet wird, wenn man keinen Port angibt⁶.

Die letzte Angabe in einem URL ist immer die konkrete Pfad- und Dateiangabe.



Beachten Sie, dass im Internet zur Trennung von Ebenen keine Backslashes (wie unter DOS oder Windows) genommen werden, sondern die normalen Slashes.



Wenn Sie in einem Browser beim Surfen im Internet eine Adresse im Adressfeld des Browsers eingeben, geben Sie einen URL an. Dabei wissen erfahrene Anwender, dass dort oft eine auf verschiedene Weise verkürzte Angabe möglich ist. Man kann etwa in der Regel das Protokoll weglassen. Dieses wird im Rahmen der Client-Server-Beziehung zwischen Browser und Server automatisch ergänzt (meist korrekt, aber nicht immer). Sogar den DNS-Namen kann man in vielen Fällen scheinbar verkürzen. So kann die Angabe eines zweiteiligen DNS-Namens der Art `rjs.de` durchaus funktionieren, obwohl die vollständige Internet-Adresse als `www.rjs.de` angegeben wird (was auch funktioniert - andere Angaben als `www` führen bei dieser nachfolgend noch genauer erläuterten Konstellation jedoch meist zu Fehlern). Dabei wird streng genommen aber gemogelt. Zweiteilige DNS-Namen sind nur dann möglich, wenn auf einem per DNS-Name adressierten Host nur ein einziges Serverprogramm läuft. Dann ist die Angabe `www` einfach überflüssig, weil das Serverprogramm bereits ohne diese Angabe klar ist.

Was ist aber in dem Fall vorgesehen, wenn ein Anwender bei einem URL keine Datei und keine Pfadangaben macht? Dann werden die Defaultseiten des jeweiligen Verzeichnisses in einem Browser angezeigt. Ein Vergleich mit einem großen Bürogebäude ist ganz passend:

Wenn man einen URL ohne Pfad- und Dateiangaben im Browser eingibt, ist es so, als wenn man sich beim Pförtner des Unternehmens meldet. Dieser leitet dann weiter, was bei einer Default-Webseite eines Servers die entsprechenden Hyperlinks sind. Gibt man bereits zusätzlich

ein Verzeichnis an, wird dessen Default-Webseite im Browser angezeigt (entspricht in etwa der Sekretärin der Abteilung oder so). Gibt man jedoch zusätzlich die gewünschte Datei an, kommt man als Surfer ohne Umwege zum gewünschten Ziel.

Beachten Sie bitte, dass Sie dieses unterstützende Verhalten des Browsers respektive des Servers bei der Verwendung von Links in HTML oder Adressierungsbefehlen in JavaScript teilweise nicht (!) zur Verfügung haben. Dort müssen Sie insbesondere die Protokolle unbedingt angeben.

HTML

In einem Atemzug mit dem Dienstprotokoll HTTP wird meist die Dokumentenbeschreibungssprache HTML (HyperText Markup Language) genannt, die darauf aufbaut. In diesem Buch steht zwar JavaScript im Mittelpunkt, aber ohne HTML-Grundlagen lässt sich kein JavaScript sinnvoll betreiben. Deshalb sollte man über die Grundzüge von HTML und bestimmte Verhaltensweisen von daraus erstellten Webseiten als JavaScript-Programmierer informiert sein. Vieles des notwendigen Hintergrundwissens erschließt sich, wenn man die Entwicklung von HTML verfolgt.

HTML ist eine Weiterentwicklung der Ende der Sechziger-Jahre von IBM entwickelten und derweil in der ISO-Norm 8779:1986 festgeschriebenen Sprache SGML (Structured Generalized Markup Language) zur abstrakten Beschreibung von Dokumenten auf Computerplattformen und dem internationalen standardisierten Datenaustausch. HTML ist speziell auf HyperText-Funktionalität (also anklickbare Querverweise bzw. Hyperlinks in einer Seite, die dann direkt einen neuen Inhalt anzeigen) ausgerichtet, aber das ist beileibe nicht alles. Zu einer so mit HTML beschriebenen logischen Dokumentenstruktur im Web können Texte, Überschriften, Bilder, Kapitel, Absätze, Aufzählungen, Tabellen, Referenzen auf multimediale Elemente (Sound, Video u.ä.) oder komplexe Objekte und Referenzen auf Scripte und Stilvereinbarungen gehören. HTML-Dateien bestehen (wie auch JavaScript-Quelltext) aus reinem Klartext. Dies bewirkt die Plattformunabhängigkeit von Webseiten (sofern keine plattformabhängigen Zusatztechniken referenziert werden).

HTML ist keine Programmiersprache wie JavaScript, sondern beschreibt nur die logischen Strukturen eines Dokuments. HTML fehlen wesentliche für eine Programmiersprache unabdingbare Elemente - insbesondere Variablen zum Zwischenspeichern von dynamischen Informationen und Kontrollflussanweisungen (darauf kommen wir bald noch genauer zu sprechen). Dementsprechend werden Webseiten mit HTML auch nicht programmiert. Wenn dies jedoch so behauptet wird, zeugt es entweder von Unkenntnis oder man möchte den diffizilen Unterschied nicht erläutern. Für Sie als potentiellen JavaScript-Programmierer ist es aber wichtig, den Unterschied zu verstehen (wir gehen noch auf Details ein). Er verdeutlicht auch gut die Situationen, wann bei bestimmten Problemen in einer Webseite HTML nicht ausreicht und zur Lösung auf JavaScript oder eine andere Web-Programmiersprache zurückgegriffen werden muss.



Da HTML ein Dokument nur abstrakt und plattformneutral beschreibt, muss diese Beschreibung irgendwann in die konkrete Darstellung der Webseite umgesetzt werden. Dies machen Browser, welche die plattformneutralen Anweisungen in einer Webseite analysieren und dann nach ihren Vorgaben auf ihrer jeweiligen Plattform umsetzen. Dieses Verfahren wird Interpretation genannt. Ein Browser ist also ein unter einem bestimmten Betriebssystem laufendes Programm, welches neutrale Anweisungen so interpretiert, wie die Plattform es gestattet. Er wird deshalb auch Interpreter genannt.



Auch für ein Programm oder ein Script gilt, dass es in der Regel erst einmal in Klartextanweisungen erstellt wird. Bevor daraus eine für einen Computer verständliche Folge von Anweisungen entsteht, muss der Klartext für den Prozessor übersetzt werden, auf dem es laufen soll (in Maschinencode). Es gibt also auch für diesen Fall den Mechanismus der Interpretation. Beim Anwender, der das noch nicht übersetzte Programm/Script als Quelltext erhält, erfolgt die dynamische Übersetzung zur Laufzeit durch ein entsprechendes Programm, welches den Quelltext interpretiert.

HTML kann man sich gut als ein Notenblatt oder eine ganze Partitur vorstellen. Der Browser ist das Orchester, das die Noten interpretiert und das Stück spielt. Der Vergleich passt übrigens ziemlich gut. Spinnen wir ihn einfach mal weiter. Stellen Sie sich eine anspruchsvolle, klassische Oper vor. Diese wird einmal von den Berliner Philharmonikern und das zweite Mal von dem Hintertupfinger Blasorchester interpretiert. Beide bedienen sich der gleichen Vorlage, die Interpretation wird jedoch sehr wahrscheinlich leicht⁷ unterschiedlich sein. Mit HTML-Seiten verhält es sich analog - jeder Browser kann die HTML-Anweisungen verschieden interpretieren. Zwar werden von den wichtigsten Browsern die meisten HTML-Anweisungen gleich oder sehr ähnlich interpretiert, aber es gibt genügend Abweichungen.

Denken wir den Vergleich mit den Berliner Philharmonikern und dem Hintertupfinger Blasorchester bei der Interpretation eines schwierigen Musikstücks noch einmal weiter. Ohne den Musikern des (fiktiven - jede Ähnlichkeit mit lebenden oder verstorbenen Personen und Orchestern :-)) Hintertupfinger Blasorchesters zu nahe treten zu wollen - bei diversen Stellen werden verschiedene Musiker garantiert überfordert sein. Was bleiben da für Möglichkeiten? Falsch spielen oder aufhören und erst wieder einsetzen, wenn man den Faden wiedergefunden hat. Letzteres ist genau das, was Browser tun, wenn sie einen HTML-Befehl in einer Webseite vorfinden, den sie nicht kennen und damit nicht interpretieren können. Sie ignorieren solche Anweisungen einfach. Das Verfahren ist im Web übrigens zu einem festen Prinzip geadelt worden - das Prinzip der Fehlertoleranz bei Browsern (vielleicht das wichtigste Element für den Erfolg des Webs - HTML kann wachsen, ohne dass neue Webseiten alte Browser abschießen).



Das Prinzip der Fehlertoleranz löst ein großes Problem beim Interpreterkonzept. Was soll passieren, wenn ein Interpreter eine Anweisung nicht kennt? Grundsätzlich veralten Interpreter sehr schnell. Wenn eine Interpretersprache einen neuen Befehl hinzugefügt bekommt, können die bis dahin entwickelten Interpreter diesen Befehl noch nicht kennen und ihn entsprechend nicht ausführen. Solche unbekannten Befehle bewirken also auf jeden Fall Probleme, die irgendwie zu lösen sind. Das Prinzip der Fehlertoleranz bei HTML mag zwar erst einmal nicht besonders positiv erscheinen, ist aber immer noch die beste Möglichkeit überhaupt. Das Ignorieren von unbekannten Anweisungen durch den Browser erlaubt die Auswertung von HTML-Dokumenten, soweit es irgend möglich ist. Selbst syntaktisch unkorrekte Dokumente können teilweise ausgewertet werden. Soweit Browser korrekte Anweisungen vorfinden, werden diese ausgeführt und angezeigt (darunter fällt bei HTML jede Klartextangabe außerhalb der spitzen Klammern, welche einen HTML-Befehl kennzeichnen). Falsche oder unvollständige Anweisungen werden ganz einfach ignoriert. Schlimmstenfalls können einige signifikante Informationen beim Anzeigen verloren gehen. Leider lässt sich dieses geniale Konzept nicht so einfach auf Scripte fortsetzen. Scripte werden nicht linear von oben nach unten abgearbeitet. Vielfache Abhängigkeiten in Scripten (Variablen, Funktionen, Objekte etc.) lassen es einfach nicht zu, bestimmte Anweisungen einfach zu ignorieren, denn sie könnten ja Grundlage für Folgeanweisungen sein.

All das zusammen bewirkt die unterschiedlichen Darstellungen von Webseiten in verschiedenen Browsern, wie sie sicher vielen Lesern schon aufgefallen sind.

HTML gibt es seit 1997 in der Version 4, welche auch heute noch aktuell ist. Ein Grund für die - in

Internet-Maßstäben gemessene - doch sehr lange Zeitspanne dürfte darin liegen, dass hinter der Normung von HTML das World Wide Web-Consortium (W3C - <http://www.w3.org>) steht.

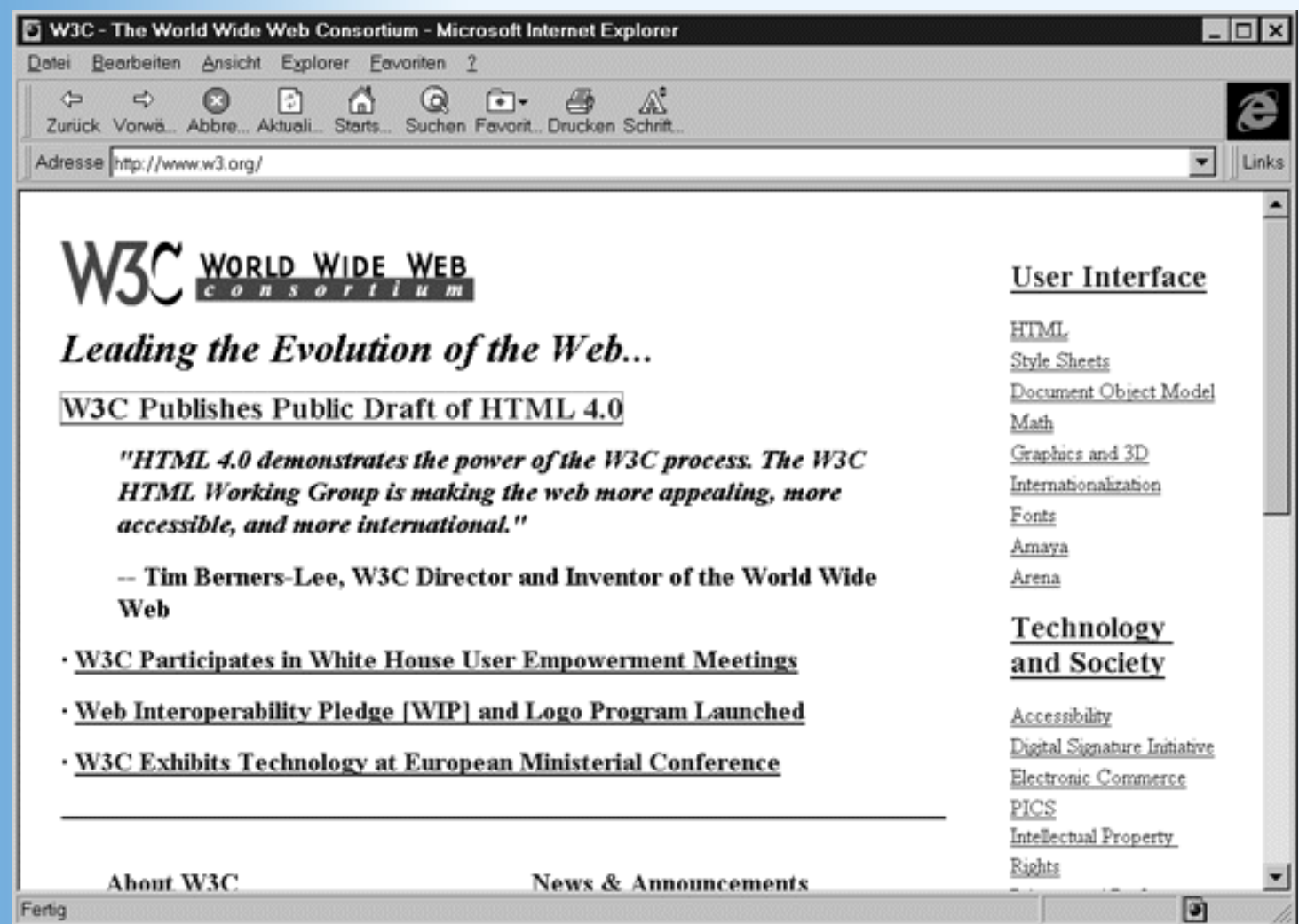


Abbildung 1.2: Die Webseiten des W3C.

Diese Organisation bemüht sich schon seit dem ersten Auftreten von HTML im Jahr 1991 um die Standardisierung der Sprache. Mit leider sehr geringem Erfolg und insgesamt wenig effektiv, was sich sicher nicht zuletzt aus der Zusammensetzung des W3C erklären lässt. In dieser Organisation sind Protagonisten des Internets zusammengeschlossen, die sich nicht die Butter auf dem Brot gönnen. Mittlerweile sind an dem W3C weltweit mehr als 130 Unternehmen beteiligt, von denen die meisten Konkurrenten auf dem Internet-Markt sind. Die damit verbundenen strategischen und politischen Entscheidungen bei der Sprachstandardisierung lassen die geringen Fortschritte bei einer einheitlichen Struktur verständlicher werden. Jede Interessensgruppe versucht Entwicklungen in die ihr genehme Richtung zu beeinflussen, weshalb ein allgemein akzeptierter Standard immer eine sehr schwere Geburt ist. Dabei muss man unbedingt betonen, dass das W3C keinerlei rechtliche Handhabe zur Durchsetzung von Beschlüssen hat, sondern auf die freiwillige Kooperation der beteiligten Stellen angewiesen ist (also in Bezug auf HTML sind das im Wesentlichen die Browser-Hersteller). Das hat zur Folge, dass jeder Browser-Hersteller einen beschlossenen Standard von HTML oder anderen Webtechniken umsetzen, ihn aber auch ignorieren oder um individuelle Befehle erweitern kann. Dies ist leider so auch nicht die Ausnahme, sondern die Regel. Fast alle Browser kennen jeweils nur einen Teil der zu einem Standard gehörenden Befehle, dafür bieten sie aber meistens Extrawürste, die dann konkurrierende Browser nicht verstehen. Oft werden solche Extrawürste so populär, dass viele Anwender sie für HTML-Standards halten oder die Technik über den Druck der Masse eine Anpassung des HTML-Standards erzwingt. Frames sind ein bekanntes Beispiel aus der HTML-Vergangenheit.

HTML (und die damit erstellten ersten WWW-Seiten) hat mit einem sehr einfachen 1.0-Standard

angefangen. In der ersten Variante dieser Sprache bestand zunächst nur die Möglichkeit, HyperText-Systeme im Internet aufzubauen, die neben einfachem Text und Grafiken insbesondere Hyperlinks, also Verweise auf andere Dokumente, enthalten konnten. Eine exakte Gestaltung von Strukturelementen wie Überschriften, Aufzählungen oder Kapiteln war in der Version 1.0 bereits vorgesehen; um das genaue Layout sollte sich bei der Darstellung jedoch jeder Browser selbst kümmern. Diese mageren Darstellungsmöglichkeiten genügten vielen Browser- Herstellern sehr schnell nicht mehr und die ersten Erweiterungen um eigene Befehle wurden integriert. Etwa durch den populären Browser Mosaic, den ersten WWW-Browser mit grafischer Oberfläche. Obwohl eine schnelle Reaktion auf die daraus resultierenden Internet-Entwicklungen für die Verhinderung einer Unmenge von Individuallösungen durch verschiedene Browser-Hersteller äußerst wichtig gewesen wäre, konnte erst im September 1995 der HTML-Standard 2.0 verabschiedet werden. Nicht zuletzt durch diese lange Zeitspanne zwischen Version 1.0 und 2.0 und die Trägheit des W3C (in Bezug auf das unglaublich dynamische Internet) hatten sich derweil verschiedenste Browser-spezifische Dialekte gebildet, an denen eine Standardisierung des WWW auch heute noch krankt. Dazu kam, dass schon im 2.0-Standard durch Unstimmigkeiten im W3C nur ein kleinster gemeinsamer Nenner realisiert werden konnte. Dieser 2.0-Standard enthielt dennoch schon eine ganze Menge Gestaltungsmöglichkeiten für das World Wide Web. Dazu gehörten unter anderem die Elemente der äußeren Dokumentenstruktur (zum Beispiel Kopfteil mit Meta- und Titelzeile, die Body-Struktur), verschiedene Überschriften, Hyperlinks, Hervorhebungen, Befehlsstrukturen zu einer flexiblen Grafikeinbindung, aber auch bereits Tabellen, Listen und Formulare. Nach der HTML-Stufe 2 gab es eine peinliche Konfusion mit einem Standard 3.0, der konsequent von den Browser-Herstellern ignoriert wurde. Erst anlässlich der fünften WWW-Konferenz vom 6. bis 8. Mai 1996 in Paris präsentierte das W3C endlich wieder eine Version, die zum aktuellen Sprachstandard von HTML wurde, die Version 3.2 vom 7. Mai 1996. Der 3.2-Sprachstandard unterstützte und standardisierte endlich Sprachelemente, die in der Praxis seit geraumer Zeit in Gebrauch waren. Etwa Tabellen mit der strukturierten Darstellung von Elementen, verschiedene Schriftgrößen und Schriftfarben, das Höher- und Tieferstellen von Zeichen, Textfluss um Grafiken herum, verweissensitive Grafiken ohne Serverkommunikation und vor allem die standardisierte Referenz auf Java- Applets und Scripte (was gerade in Hinsicht auf unser JavaScript-Thema von entscheidender Bedeutung ist). Leider galt aber auch bei der Verabschiedung des 3.2-Standards, dass sich die W3C-Teilnehmer auch diesmal wieder nicht auf die Standardisierung einiger Elemente einigen konnten, obwohl diese bereits sehr verbreitet waren. Dazu zählten insbesondere Frames, d.h. die Aufteilung des Anzeigebereichs eines Browsers in mehrere voneinander unabhängige Teilbereiche. Diese Technik wurde von Netscape entwickelt und deren Browser Navigator unterstützt. Obwohl auch andere Browser-Hersteller sich dem Druck der Popularität von Frames beugten und sie unterstützten, ließ sich in HTML 3.2 kein standardisiertes Verfahren festschreiben (erst HTML 4.0 hat die Technik offiziell eingeführt). Aber auch andere Techniken wie die Unterstützung von untergeordneten Fensterstrukturen, eine direkte Unterstützung von Scriptsprachen als Erweiterung eines HTML-Befehls, erweiterte Formulare, eine Notation von mathematischen Sonderzeichen und hardwareabhängige Darstellungen fanden keinen Eingang in HTML 3.2. Genau genommen wurde HTML 3.2 sogar gegenüber der unglücklichen HTML-3.0-Zwischenlösung um einige (in der Regel redundante) Befehle reduziert.

Anfang Juli 1997 wurde dann ein erster öffentlicher Entwurf für die nächste Version von HTML - die auch heute noch aktuelle Variante 4.0 - veröffentlicht. Neben einem neuen Tabellenkonzept und Formularerweiterungen betrafen wesentliche Neuerungen der neuen HTML-Version die Erweiterung der Aus- und Eingabemöglichkeiten und HTML. Darunter sind beispielsweise verschiedene Auflösungen, Farbuweisungen, flexible Ein- und Ausgabemöglichkeiten sowie Spracheingabe und akustische Informationen zu verstehen. Grundsätzlich sollte die Version 4.0 die Beschränkungen auf Maus und Tastatur (bisher auch nicht voll unterstützt) bei der Eingabe und dem Bildschirm bei der Ausgabe abschaffen.

HTML 4.0 wurde auch internationaler. Die so genannte Internationalisierung (engl.: internationalization) nach dem ISO/ IEC:10646-Standard bzw. I18N war ein Schritt, wie er bei den meisten Internet-basierenden Techniken notwendig wurde, um die Grenzen des bisher verwendeten ISO-Standards 8859-1 bei der Verwendung von Zeichen und Buchstaben zu sprengen. Der abwärtskompatible Umstieg vom bisher verwendeten ASCII-Code (Darstellung eines Zeichens in einem Byte, was 256 verschiedene Zeichen in einem Zeichensatz erlaubt) auf den neuen UNICODE mit zwei Byte erlaubt die Verwendung von vielen

neuen Zeichen in einer Webseite innerhalb eines Zeichensatzes (unabhängig von der zugrunde liegenden Plattform und deren Einstellungen). Die Darstellung von Sonderzeichen und landesspezifischen Zeichen erfolgt gegebenenfalls über so genannte Entities (eine Folge von Zahlen oder Zeichen, die dem Steuerzeichen & folgen).

Grundsätzlich hat das W3C auch mit HTML 4.0 wieder versucht, redundante Anweisungen zu eliminieren oder ähnliche Anweisungen zu einer einzigen Angabe zusammenzufassen. Dies betrifft insbesondere die Referenzierung von Multimedia-Objekten (Bilder, Videos, Java-Applets, ActiveX-Controls usw.) über nur eine einzige Anweisung - den `<OBJECT>`- Tag.

Für die Verbindung zu JavaScript ganz wichtig ist die Erweiterung von HTML um eine Reihe von neuen Schlüsselwörtern, die Voraussetzung für das Ausführen von Scripten bei bestimmten Ereignissen in Webseiten sind. Diese Schlüsselwörter können als Attribute in HTML-Anweisungen verwendet werden. Dazu zählen `onLoad`, `onMouseOver`, `onClick` usw. (so genannte Event-Handler). Bezüglich der Script-Verwendung sind unter HTML 4.0 die Voraussetzungen für die Einbindung erheblich erweitert worden. Und zwar unabhängig von einer zu verwendenden Sprache: Sowohl JavaScript als auch VBScript, Tcl und Perl lassen sich verwenden.

Style Sheets

HTML erlaubt, wie wir besprochen haben, keine vollständige Kontrolle über das Aussehen von Webseiten. Um diesem Manko abzuhelpen, gibt es mit Style Sheets (Stilvereinbarungen) eine ergänzende Technologie, die versuchen soll, die teils sehr unterschiedlichen Interpretationen von Gestaltungsanweisungen für Webseiten auf verschiedenen Plattformen zu vereinheitlichen. Entgegen sehr frei interpretierbaren HTML-Empfehlungen an den Browser kann über Style Sheets das Layout von Webseiten (zumindest in der Theorie) recht genau festgelegt werden (wenn der Browser die Style Sheets versteht und sich an vorgegebene Standards hält). Style Sheets eröffnen daneben die Möglichkeit der strikten Trennung von Gestaltungsbefehlen und Informationsträgern, d.h. es kann eine klare Aufteilung von Struktur und Layout erreicht werden.

Wie auch Scripte gehören Style Sheets nicht zu HTML, sondern sind eine eigenständige Technik, die in HTML verankert werden kann und diese Technik ergänzt. Style Sheets bezeichnen keine bestimmte Sprache (wie etwa HTML oder JavaScript), sondern nur eine Technik. Es gibt diverse Sprachen zum Definieren von Style Sheets. Cascading Style Sheets (CSS) sind ein vom W3C (World Wide Web Consortium) akzeptierter Sprachstandard zur Darstellung und Formatierung von HTML-Seiten und werden von den meisten modernen Browsern - zumindest im Ansatz - unterstützt. Cascading Style Sheets gibt es mittlerweile in der Version 2, aber die aktuellen Browser unterstützen maximal die Version 1 (und das meist nur unvollständig beziehungsweise mit abweichenden Varianten - das gleiche Dilemma wie bei HTML).

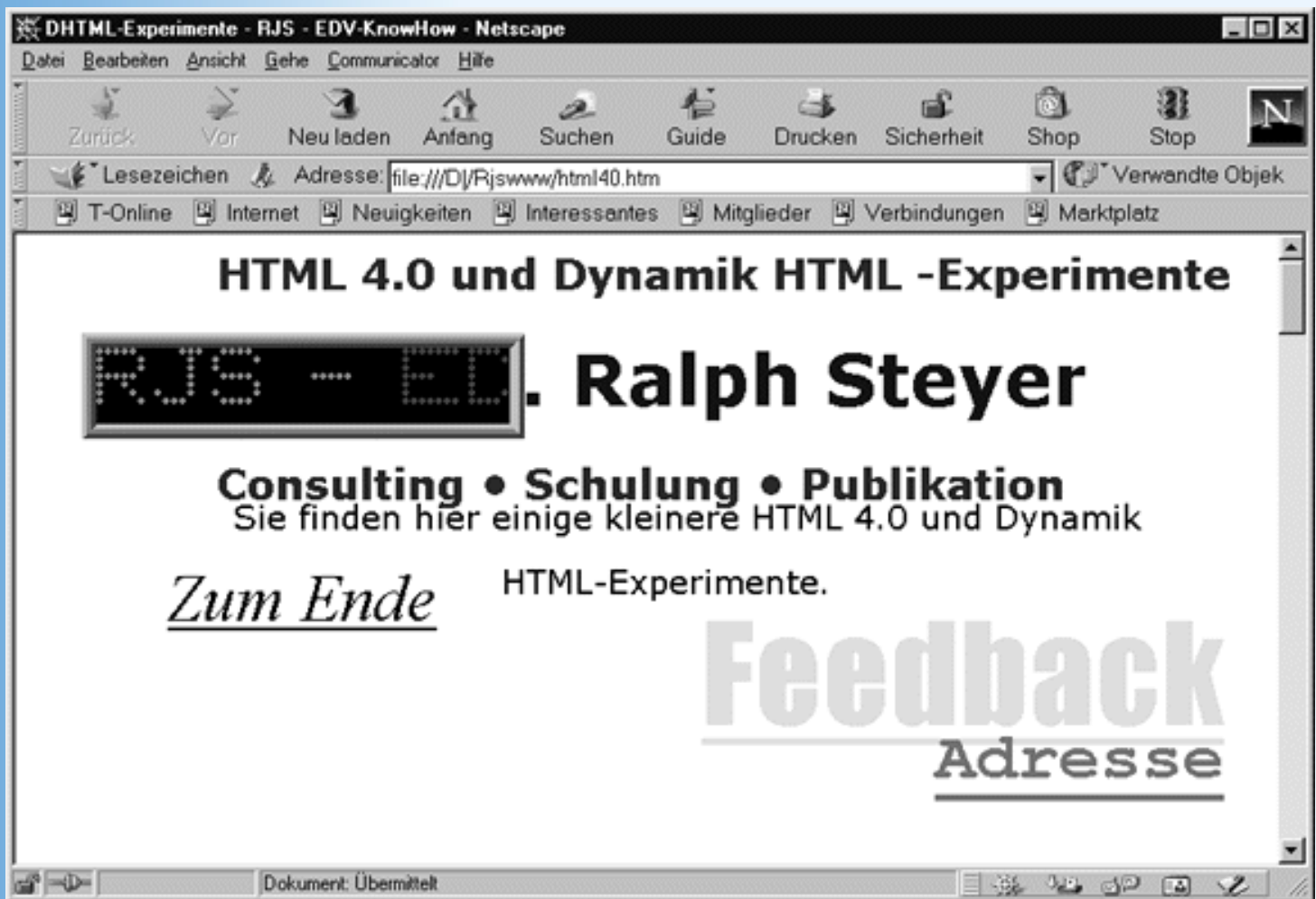


Abbildung 1.3: Eine Seite mit CSS-Style-Sheets

Verschiedene Browser

Wie bereits angesprochen, werden Webseiten beim Anwender durch Browser interpretiert und angezeigt. Zeitgemäße Browser sind zudem in der Lage, neben der Anzeige von Webseiten auch andere Internet-Dienste wie FTP, Email oder Newsgroups direkt zu benutzen oder eine automatische Verbindung zu einem ergänzenden Programm (etwa einem Email-Client) aufzubauen. Wenn man in dem Eingabefeld einer URL einen anderen Internet-Dienst eingibt, wird der zurückkommende Datenblock dem Dienst entsprechend aufbereitet. Selbst der Zugriff auf das lokale Verzeichnissystem des eigenen Rechners funktioniert, sodass ein Browser wie der »normale« Arbeitsplatz unter Windows zu verwenden ist (etwa mit dem Protokoll `file`, z.B. `file:///C:/` für die Anzeige des Inhalts der Festplattenpartition C:).

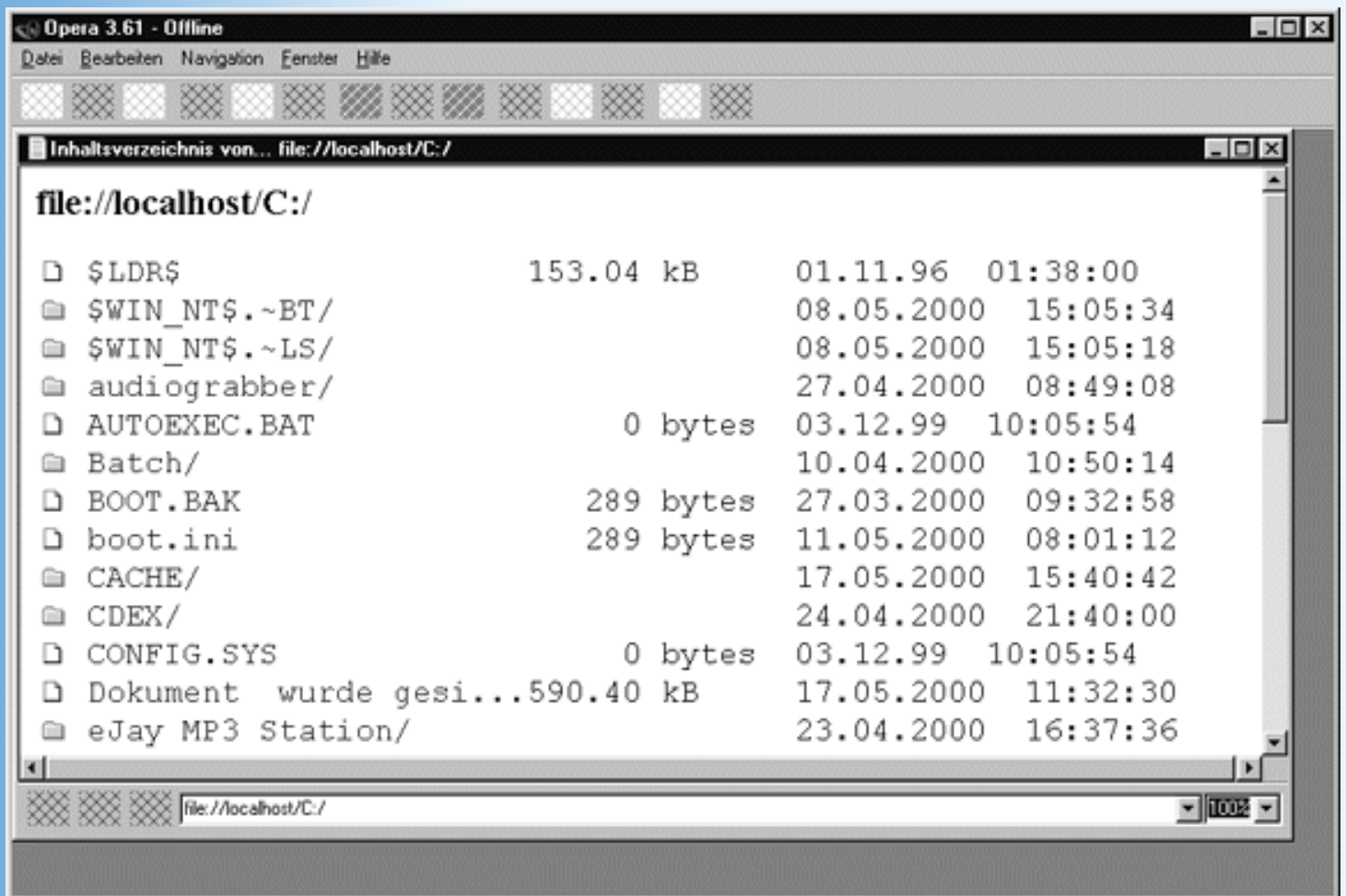


Abbildung 1.4: Die Darstellung der lokalen Festplatte im Opera-Browser

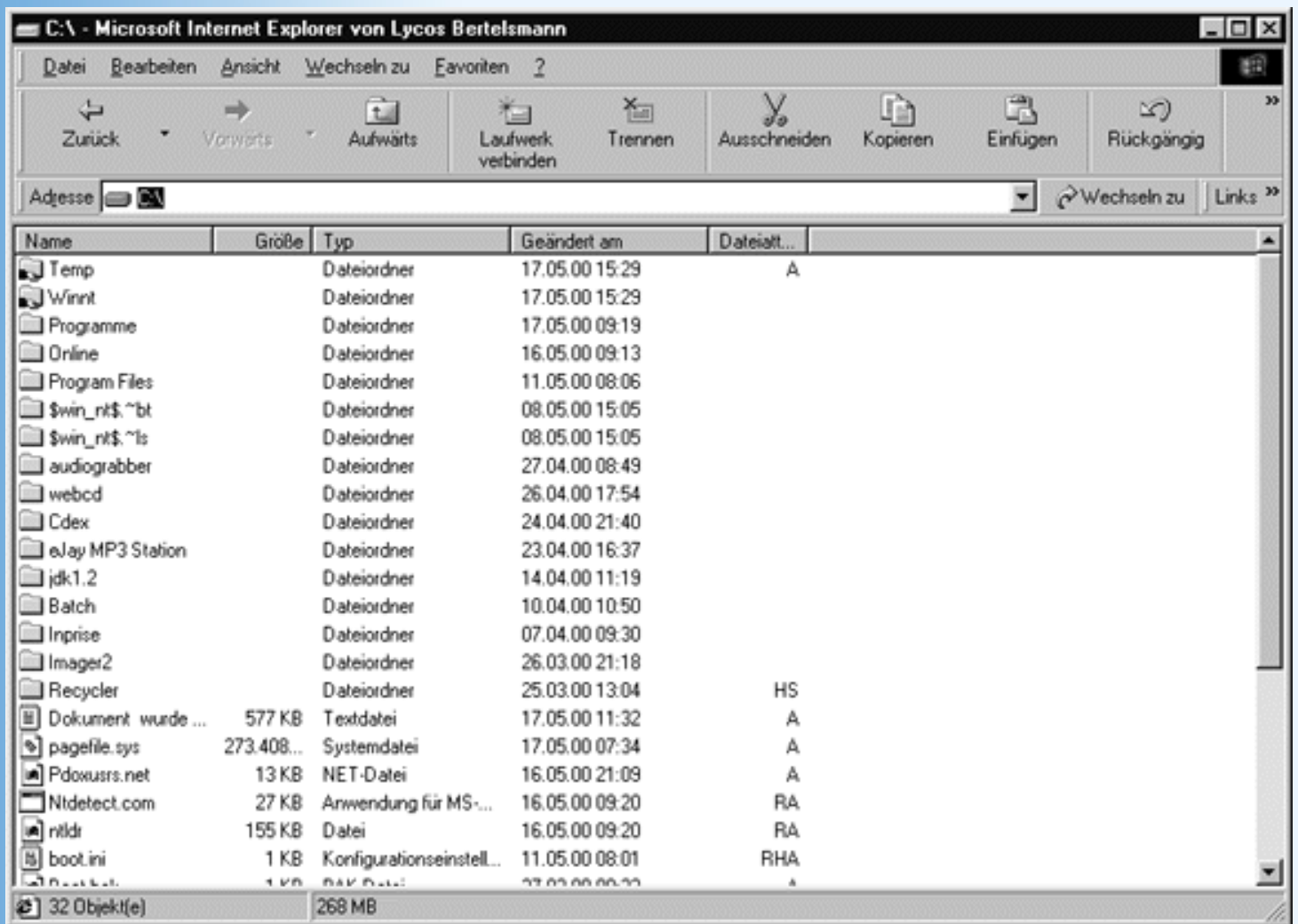


Abbildung 1.5: Die Darstellung der lokalen Festplatte im Internet Explorer bei gleicher Eingabe von `file:///C:/` - die enge Verzahnung mit Windows zeigt sich deutlich

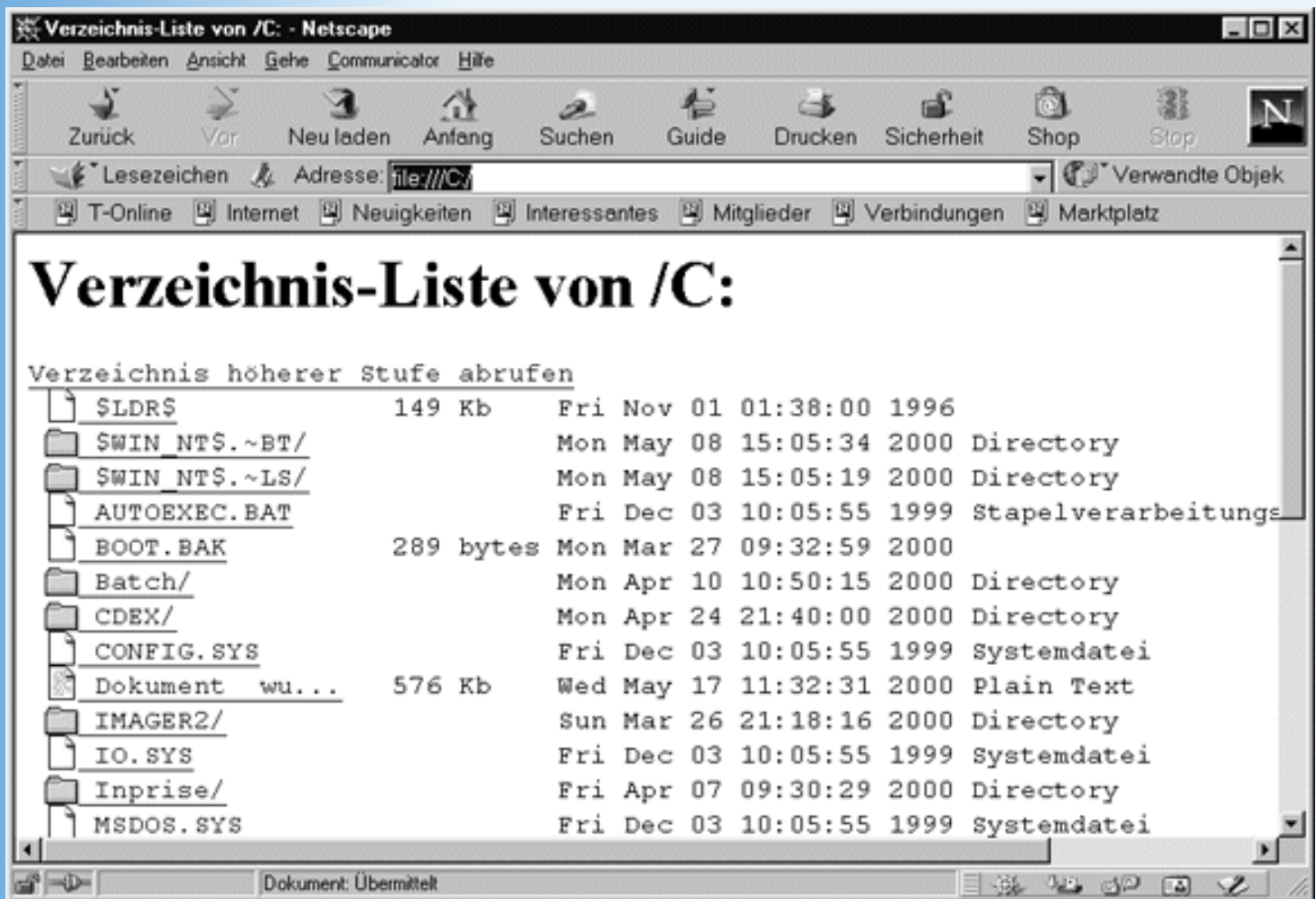


Abbildung 1.6: Die Darstellung der lokalen Festplatte im Netscape Navigator

Ein Browser kümmert sich neben den offensichtlichen Aktionen auch um Hintergrundaktionen. Insbesondere darum, Daten über das Internet zu empfangen und zu senden. Dabei sollte man sich die Art der Datenübertragung im Internet klar machen. Das Internet basiert auf paketorientiertem Datenaustausch, was zentraler Aspekt von TCP/IP ist. Dies bedeutet, jede über das Netz zu übertragende Datei wird bei Bedarf in kleinere Datenblöcke bzw. Datenpakete zerlegt, die dann einen Strom von einzelnen Datenpaketen bilden. Nichtsdestoweniger wird jedes einzelne Datenpaket als in sich geschlossene, vollständige und in Hinsicht auf den Transport im Netz vollkommen unabhängige Dateneinheit vom Sender zum Empfänger transportiert. Auf zentralen Zwischenstationen - so genannten Routern - werden die Datenpakete in die unter den aktuellen Bedingungen optimale Richtung weitergeleitet. Die etwas geschraubte Formulierung »unter den aktuellen Bedingungen optimale Richtung« bedeutet, dass der Weg der einzelnen Datenpakete nicht unbedingt fest vorgegeben ist. Es kann durchaus vorkommen, dass ein Strom von Datenpaketen auseinander gerissen wird und die einzelnen Datenpakete unterschiedliche Wege nehmen (bei Überlastung von Verbindungen wird von Routern bei Bedarf automatisch eine Alternativverbindung gewählt). Es kann sogar vorkommen, dass hintere Datenpakete der Kolonne auf Ausweichrouten geschickt werden und dabei die vorderen Datenpakete überholen.

Ein Browser muss nun auf so eine Situation reagieren können. Wenn eine in mehrere Datenpakete zerlegte Webseite bei ihm ankommt, müssen die Datenpakete so zusammengesetzt werden, dass das Datenpaket mit den ersten Zeichen der Seite zuerst ausgewertet und verwendet wird, danach das zweite usw. Dies muss unabhängig davon erfolgen, in welcher Reihenfolge die Datenpakete beim Browser ankommen. Relevant darf nur die Reihenfolge sein, wie eine Webseite zerlegt und abgeschickt wurde. Um dieses zu gewährleisten, beinhaltet die Kombination TCP/IP und HTTP (neben Adressangaben) Informationen darüber, wie groß der gesamte zu empfangende Strom von Datenpaketen ist, welche Position ein jedes einzelne Datenpaket in dem Strom einnimmt und wie eine Webseite wieder zusammengesetzt ist.

Sobald die ersten Datenpakete auf dem lokalen Rechner eintreffen, fängt der Browser an, diese zu

interpretieren. Alle Datenpakete, die in chronologischer Reihenfolge eintreffen, können bereits ausgewertet und verwendet werden (auch wenn der Rest noch fehlt). Alle diejenigen Datenpakete, welche zu früh eintreffen, werden in einem Puffer gespeichert und erst dann weiter verwendet, wenn die Lücken davor geschlossen sind. Dies ist auch der Grund, warum sich beispielsweise eine Webseite im Browser oft einige Zeilen weit aufbaut, dann ewig unverändert bleibt und hernach schlagartig der gesamte Rest erscheint. In diesem Fall hat einfach in der Mitte des Stroms ein Paket gefehlt, der Rest war aber schon da. Sobald der Nachzügler eingetroffen ist, können dann auch die bereits im Puffer gespeicherten Datenpakete vom Browser verwendet werden.

Wichtige Browser

Schauen wir uns einige der bekanntesten Browser nun an. Der Browsermarkt wurde lange Zeit von einem Browser dominiert - dem Browser Navigator von Netscape. In den neunziger Jahren verwendeten bis zu 90% der Internetsurfer ihn in den verschiedensten Versionen. Dies hat sich gegen Ende des Jahrzehnts durch erhebliche Anstrengungen von Microsoft (sowohl technisch als auch in der Vertriebsstruktur) geändert. Dessen Browserfamilie, die Internet Explorer, hat - nicht zuletzt durch die standardmäßige Integration in Windows - erhebliche Marktanteile gewonnen. Beide Browser dürften derzeit und wahrscheinlich auch noch längere Zeit die gleiche Bedeutung haben, obwohl es Mitte 2000 eine Untersuchung gab, wonach der Internet Explorer in Deutschland von 94% aller Anwender genutzt wird (wobei - wie bei allen Statistiken - die Aussagefähigkeit sehr in Frage gestellt werden kann - immerhin ist der Browser ja sowieso in Windows integriert, und damit wird fast jeder Anwender zumindest das Symbol einmal angeklickt haben - die Unabhängigkeit der Studie obendrein außer Acht gelassen).

Beide Browser sollten auf jeden Fall zur Verfügung stehen, wenn man Webseiten erstellen möchte, und vor allen Dingen, wenn JavaScript- Programmierung ansteht. Es gibt aber zu den beiden Marktführern noch eine ganze Anzahl von Alternativen, von denen hier insbesondere eine interessante vorgestellt werden soll - Opera.

Der Netscape Communicator

Der Navigator war wie gesagt lange Zeit der mit Abstand wichtigste Browser im WWW und obwohl er diese Position mittlerweile mit dem Internet Explorer teilen muss, ist er immer noch eines der Referenzsysteme schlechthin, wenn man Webseiten erstellen und vor allem JavaScripts oder Java-Applets programmieren möchte. Gerade bei JavaScript- Programmierung muss man den Browser des JavaScript-Erfinders natürlich auf dem Rechner haben. Es gibt ihn für die verschiedensten Betriebssysteme.

Der Navigator wurde erstmals mit der Versionsnummer 2.0 veröffentlicht, denn er ist der direkte Nachfolger des legendären Mosaic-Browsers (dessen Entwickler ist Mitbegründer von Netscape). Für dieses Buch wird der Navigator in der Version 4.7 als Referenzsystem verwendet⁸.

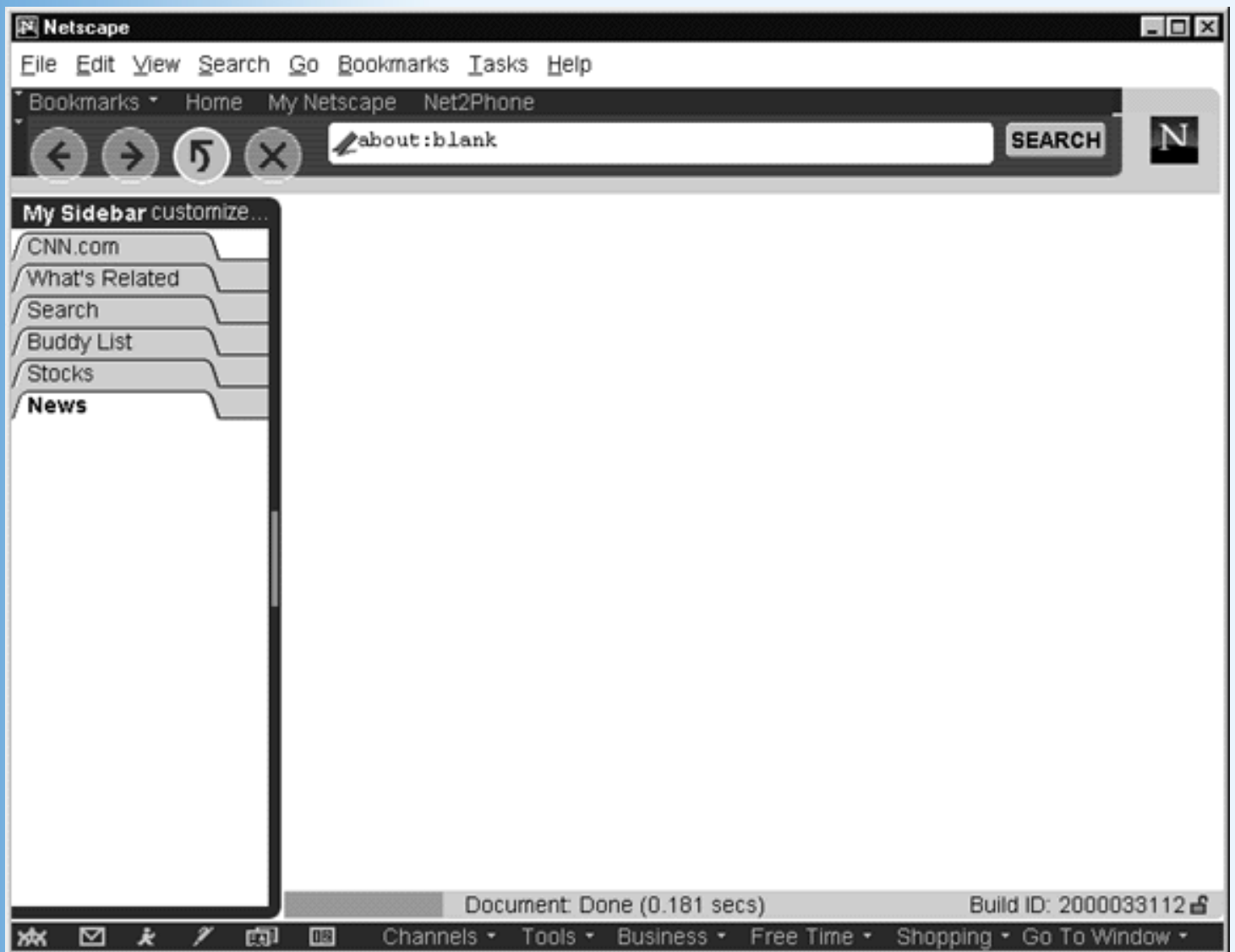


Abbildung 1.7: Der Netscape Navigator 6.0 mit einem erheblich veränderten Design

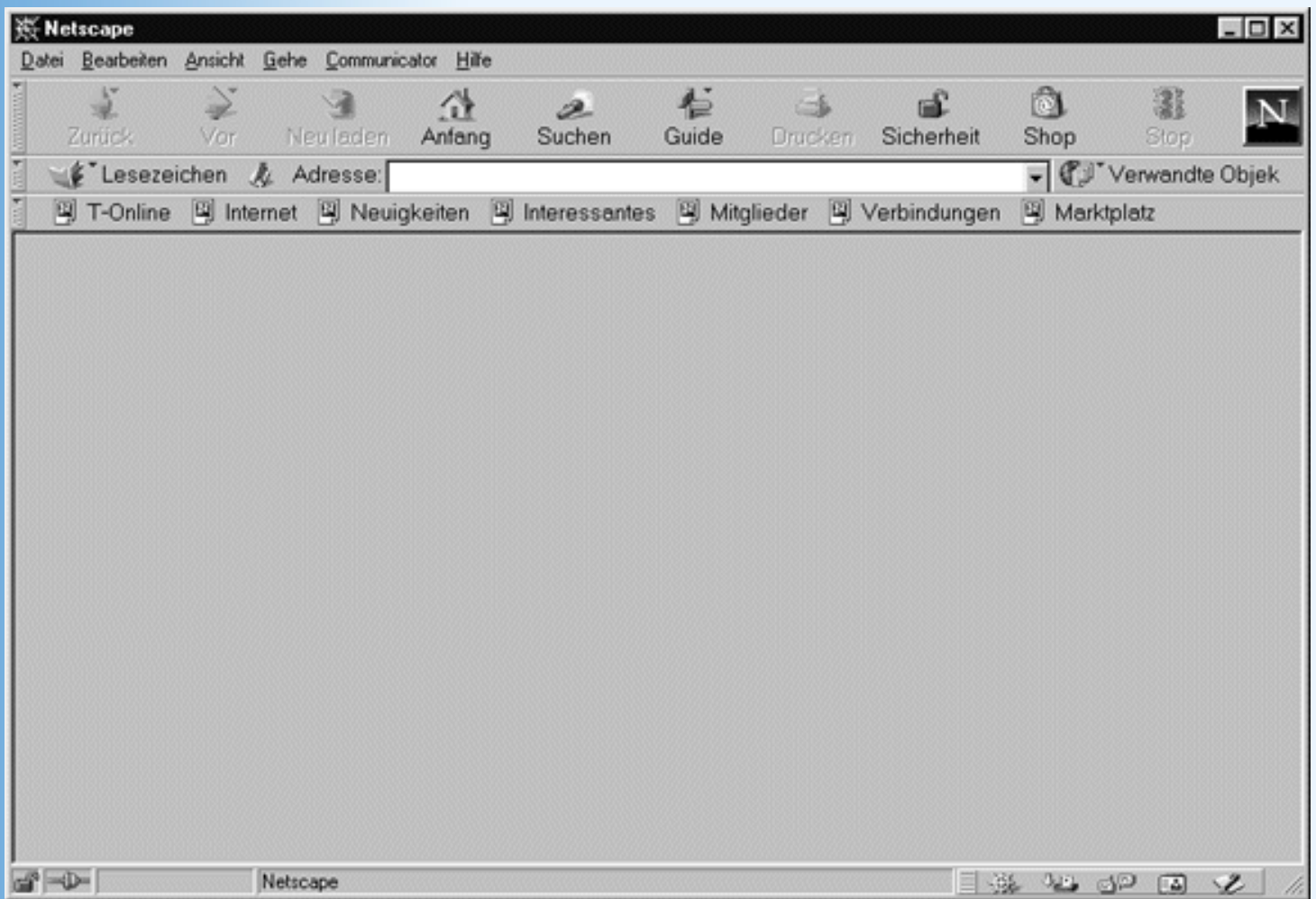


Abbildung 1.8: Der Netscape Navigator 4.7

Der Navigator bietet so alles, woran sich Websurfer über die Zeit gewöhnt haben:

- Automatische Vervollständigung von Webadressen
- Java-Unterstützung
- JavaScript-Unterstützung - für genauere Details sei auf spätere Lektionen in diesem Buch verwiesen
- Unterstützung von Suchaktionen im Internet
- Lesezeichen, um schnell zu häufig besuchten Seiten zurückzukehren
- Filterfunktionalität, um gewisse Inhalte auszublenden
- Sicherheitsfunktionalitäten zur Verschlüsselung und Entschlüsselung, Kennwörter und Schlüssel, Sicherheitseinstellungen, digitale Zertifikate und Unterschriften und Java-Applet-Einstellungen

Der Browser Navigator ist seit der Version 4.0 nur noch als Bestandteil des Communicator-Programmpakets (dieses gibt es in verschiedenen Versionen) zu haben, welches aus einer Reihe von Einzelprogrammen besteht, die sich gegenseitig ergänzen. Dazu zählen grundsätzlich mit dem Netscape Messenger ein Email- und News-Client auf HTML-Basis sowie zum Verfassen und Bearbeiten von Webseiten ein integrierter HTML-Editor - der Netscape Composer. Daneben gibt es im Communicator ein Konferenz- und Chat-Tool (Netscape Conference), ein Tool zur Unterstützung von Channels bzw. Informationsdiensten im WWW (Netcaster) sowie in den umfangreicheren Versionen des Communicator Fernkonfigurations- und Zugriffsmöglichkeiten auf verschiedenste Elemente (AutoAdmin, Administration Kit, AutoConfig, AutoUpdate, IBM Host On- Demand für Zugriffe auf einen IBM 3270-Host) und einen Terminplaner (Netscape Calendar), der in Verbindung mit dem Netscape Calendar Server eine Lösung für die unternehmensweite und plattformübergreifende Arbeitsgruppenplanung bereitstellt.

Eine Besonderheit des Communicator ist die Möglichkeit, Benutzerprofile für verschiedene Anwender zu erstellen. Wenn der Communicator zum ersten Mal gestartet wird, muss sogar ein solches Benutzerprofil erst

einmal erstellt werden, bevor mit dem Programm gearbeitet werden kann (was viele Anwender als sehr lästig empfinden). Diese Benutzerprofile ermöglichen es, dass mehrere Benutzer Communicator auf jeweils individuelle Weise für den Internet-Zugriff einsetzen. Für jeden Anwender werden darin Einstellungen, Lesezeichen und Mail-Nachrichten separat verwaltet.

Ein großer Nachteil des Communicator, aber auch des Internet Explorer, ist dessen Ressourcenbedarf beim Anwender. Sowohl die Größe der Installationsdatei (über 20 Megabyte bei der Version 4.7) als auch die hinterher auf dem Rechner des Anwenders geforderten Ressourcen sind indiskutabel (je nach installierten Komponenten 40 Megabyte und mehr auf der Festplatte und ein nicht zu langsamer Prozessor).



Die Version 6 geht (zumindest in der Betaversion) bei der Installation einen etwas anderen Weg als die Vorgängerversionen - nur ein Teil der zu installierenden Daten befindet sich in der Installationsdatei. Der Rest wird bei der Installation bzw. dem ersten Start direkt aus dem Internet nachgeladen. Ein bei einer kostenlosen Internet-Verbindung sicher interessanter, aber in Deutschland - trotz neuester Preissenkung - teurer Weg⁹.

Vor allem bedenklich an dieser Technik ist aber der Sicherheitsaspekt. Egal ob rational begründet oder nicht - psychologisch ist eine Online-Verbindung bei der Installation von Software sicher ein nicht unerhebliches Problem. Es ist fraglich, ob vorsichtige Anwender eine solche Online-Verbindung bei der Installation von Software akzeptieren. Was Verbindungen zum Internet gekoppelt mit automatischen Aktionen so alles bewirken können, hat der Virus »I love you« ja deutlich gemacht. Leider tendiert auch der Communicator prinzipiell bei vielen Funktionalitäten zu einer permanenten Online-Verbindung, wobei eigentlich die Tendenz - wenigstens in Bezug auf Sicherheit und Kostenersparnis - in die andere Richtung gehen sollte.

Beziehen kann man den Communicator in den jeweils aktuellen Versionen über CD-Beilagen diverser Computerzeitungen oder aus dem Internet von der Homepage der Netscape Corporation, <http://www.netscape.com> bzw. <http://www.netscape.com/de/>.

Der Microsoft Internet Explorer

Der hauptsächlich unter dem Windows-Betriebssystem (es gibt ihn aber auch für andere Betriebssystem-Versionen, etwa für den Mac) verfügbare Internet Explorer von Microsoft ist der große Gegenspieler des Netscape Communicator. Zum Zeitpunkt der Bucherstellung ist gerade die Version 5.01 des Internet Explorer von Microsoft aktuell, die als Referenzsystem für dieses Buch verwendet werden soll¹⁰.



Abbildung 1.9: Der Internet Explorer 5.01

Während die ersten Versionen des Internet Explorer im Web mehr oder weniger als Witz aufgefasst wurden, steht er seit der Version 4.0 dem jeweilig aktuellen Communicator im Leistungsumfang kaum nach. Die Experten streiten wie bei einem Religionskrieg, welcher der bessere Browser ist.

Wie auch der Navigator kennt der Internet Explorer die automatische Vervollständigung von Webadressen¹¹, Unterstützung von Suchaktionen im Internet, Unterstützung, um schnell zu häufig besuchten Seiten zurückkehren¹², Filterfunktionalität, um gewisse Inhalte auszublenden, und diverse Sicherheitsfunktionen, um den Computer vor potentiell unsicheren Dateien und Programmen zu schützen und die von Ihrem Computer freigegebenen Informationen einzuschränken. Auch der Internet Explorer unterstützt Java und die Zertifizierung von Java-Applets, aber kein JavaScript. Stopp - keine Panik! Fehlende JavaScript-Unterstützung heißt nicht, dass der Internet Explorer keine JavaScripts versteht - er muss sie sich quasi nur vorlesen lassen. Der Internet Explorer darf JavaScript nämlich nicht direkt verstehen, er darf nur eine Abwandlung davon kennen - JScript. Bei JScript handelt es sich um einen Microsoft-Ableger von JavaScript, womit Microsoft Zahlungen von Lizenzgebühren an Sun/ Netscape umgeht, welche bei der direkten Implementation von JavaScript in den Internet Explorer fällig wären. JScript besitzt im Wesentlichen die gleiche Struktur wie JavaScript und versteht die gleiche Syntax. Deshalb kann in der Regel von der JavaScript-Syntax direkt auf die JScript-Syntax geschlossen werden (und umgekehrt). Allerdings ist der Befehlssatz der beiden Sprachen in einigen Details nicht vollständig deckungsgleich. Insbesondere bietet JScript viele Erweiterungen, welche nur im Internet Explorer oder verwandten Microsoft-Techniken unterstützt werden. Man kann zwar Scripte direkt in JScript erstellen, aber meist wird JavaScript vorgezogen. Dabei gibt es das Problem, dass durch die nicht vorhandene direkte Implementation von JavaScript der Internet Explorer sämtliche JavaScripte in einer Webseite in JScripte übersetzen muss. Erst diese werden dann ausgeführt. Dies funktioniert zwar halbwegs zuverlässig, aber Fehler sind nicht ganz auszuschließen. Prinzipiell hat JScript im Web nicht die Bedeutung wie JavaScript. Der Internet Explorer versteht JavaScript wie gesagt weitgehend, der Navigator von Netscape und viele andere Browser dagegen kein JScript¹³.

Eine Erweiterung der Leistungsfähigkeit des Internet Explorer gegenüber dem Navigator ist die Unterstützung von VBScript, einer von Microsoft aus Visual Basic entwickelten und weitgehend auf das Windows-Betriebssystem mit INTEL-basierenden Prozessoren optimierte Scriptsprache. Zwar wurde VBScript einige Zeit als ernst zu nehmende Konkurrenz zu JavaScript gesehen, aber da die Sprache fast vollständig auf den Internet Explorer als Clientprogramm beschränkt ist (ohne Zusatzmodule streiken der Netscape Navigator und dazu kompatible Browser bei VBScript), verbauen sich Webseitenersteller mit dem

Einsatz dieser Sprache den Weg zu den meisten Kunden. Dies hat die Akzeptanz von VBScript wenig gefördert. Obwohl VBScript durch seine Optimierung im Hinblick auf die WINTEL-Plattform dort als leistungsfähiger als JavaScript gesehen wird, wird auch das eher als Gegenargument zu VBScript gesehen, denn die Sprache eröffnet einige Missbrauchsmöglichkeiten und Sicherheitslücken (gerade in Verbindung mit den äußerst umstrittenen ActiveX-Controls, die der Internet Explorer als einziger wichtiger Browser unterstützt).

Der Begriff Internet Explorer bezeichnet nicht nur den Microsoft-Browser, sondern eine Programmfamilie, die wie beim Communicator Mail-, Newsreader- und einige weitere Internetanruf-Programme integriert hat. Microsoft liefert als Gegenstück zu dem Netscape Messenger Outlook Express eine abgespeckte Version von Outlook als Mail- und Newsclient aus. Audio- und Videokonferenzen werden mittels Netmeeting realisiert. Die Audiokonferenzen werden nach H.323 (Internet-Telefonie) durchgeführt. Statt einer Videokonferenz mit nur zwei Teilnehmern wie beim Communicator kann Netmeeting auch mehr Teilnehmer verwalten. Daneben sind ein Chat-Tool, ein Programm zum Erstellen von Grafiken und ein weiteres Programm zum Dateitransfer in Netmeeting enthalten.

Auch Microsoft integriert in das Explorer-Paket einen HTML-Editor zum Erstellen von Webseiten. FrontPage tritt in einer Light-Variante gegen den Composer des Netscape Communicator an.

Dem Internet Explorer muss wie dem Netscape-Produkt der gigantische Ressourcenhunger (von 6,5 bis zu 46 Megabyte bei Vollinstallation der Version 5.01) angekreidet werden, aber beim Explorer gibt es zudem noch das Problem, dass neuere Versionen des Explorer Vorgängerversionen überschreiben und Webprogrammieren damit unumgängliche Testmöglichkeiten rauben. Außerdem greift der Internet Explorer bei der Installation grundsätzlich sehr tief in das Betriebssystem ein. Mittels der unter Windows 95 eingeführten Active-Desktop-Technologie müssen Informationen nicht mehr zwingend innerhalb des Browsers bleiben, sondern können direkt in den Desktop integriert werden¹⁴. Ziel von Microsoft ist eine enge Verzahnung von dem Betriebssystem und lokalen Anwendungen mit dem Internet. Der Internet Explorer soll mit diesem Prozess langfristig den Windows Explorer überflüssig machen (und natürlich damit erst recht Browser von anderen Firmen). Die Vorteile der Active-Desktop-Technologie sind die Möglichkeit einer persönlichen Gestaltung des Desktops und der einheitliche Zugriff auf lokale Daten, Daten aus dem Netzwerk und Daten aus Intranet oder Internet. Die enge Verzahnung mit dem Betriebssystem schmeckt sicher nicht jedem Anwender - es ist einfach nicht mehr transparent, welche Zugriffe auf das lokale System bei einer Online-Verbindung möglich sind und welche Informationen beim Surfen die lokale Station verlassen - und schon gar nicht den Konkurrenten. Die letzten Urteile durch amerikanische Gerichte werden Microsoft wohl zwingen, die Verzahnung zwischen Betriebssystem und Browser wieder zu lösen. Da der Internet Explorer mit Windows ausgeliefert wird, hat ihn jeder Windows-Anwender zwingend installiert (nur vielleicht nicht die neueste Version). Zu beziehen ist die jeweils neueste Variante des Explorer von der Internet-Homepage von Microsoft (<http://www.microsoft.com>) oder als aktuelle CD-Beilage in diversen Computerzeitingen.

Der Opera-Browser

Die wohl interessanteste Alternative zu den beiden gerade beschriebenen Browsern dürfte der äußerst stabil laufende Opera-Browser der norwegischen Firma Opera Software sein, den wir in der Version 3.6 als drittes Referenzsystem verwenden wollen¹⁵. Den Browser gibt es in einer 16- und einer 32-Bit-Version für MS Windows (3.1, 95, 98, NT3.51 und NT4.0). Er läuft aber auch unter OS/2 und der Mac-Plattform unter Virtual PC. Linux, BeOS, Amiga und Psion sollen bald folgen.

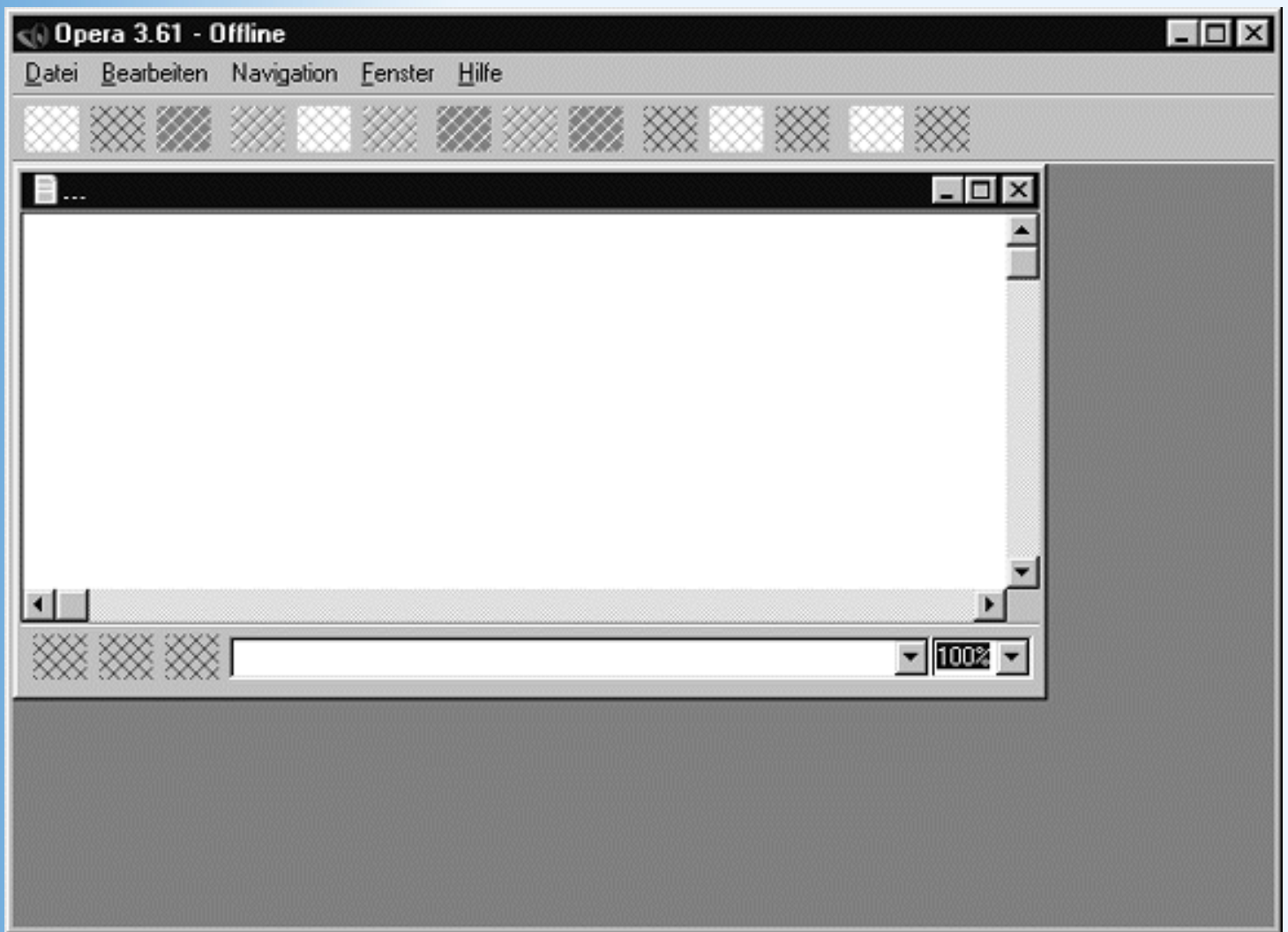


Abbildung 1.10: Der Opera 3.6

Um bei diesem Browser mit dem größten Nachteil für eine höhere Verbreitung anzufangen: Er ist im Gegensatz zu den beiden Massenbrowsern nicht kostenlos (Preis ca. 35 Dollar). Zu beziehen ist dieser Browser jedoch in einer 30-Tage-Testversion unter [http:// www.opera.com/de/](http://www.opera.com/de/) oder wie die beiden anderen Browser als Beilage in etlichen Computerzeitingen.

Dafür bietet diese norwegische Browser-Alternative jedoch einige erhebliche Vorteile gegenüber den Konkurrenten, weshalb gerade immer mehr Profis auf diesen Browser umsteigen. Insbesondere ist Opera extrem ressourcensparend (nur 1,2 Mbyte in der Version 3.6 - die Installationsroutine passt damit auf eine Diskette, was sonst kaum noch ein Programm schafft) und baut Internet-Seiten bedeutend schneller auf als seine fettleibigen Rivalen. In diversen Tests im Sommer 2000 hat sich gezeigt, dass Opera bei einigen (besonders bei komplexeren) Webseiten bis zu viermal schneller ist als der Navigator (Zweitschnellster). Opera wurde, im Gegensatz zu anderen Browsern, von Grund auf neu entwickelt und basiert nicht auf altem 16-Bit-Code. Insbesondere ist Opera unabhängig von Win32. Gerade bei älterer Hardware¹⁶ zeigen sich die Geschwindigkeitsvorteile, aber auch auf schnellen Rechnern funktionieren Datei-Downloads, Cache-Zugriffe und die Darstellung von Webseiten mit rechenintensiven Anweisungen extrem schnell. Daneben bietet der Browser eine intelligente und umfangreiche, wenngleich etwas gewöhnungsbedürftige Befehlssteuerung mit Maus und Tastatur.

Die Installation von Opera greift so gut wie gar nicht in das Betriebssystem ein (nur eine Datei - `opera.ini` - wird in das Windows-Verzeichnis geschrieben) und ist absolut unkompliziert. Dies prädestiniert den Browser für die Verbreitung von Software mit einer unter HTML erstellten Oberfläche. Der mitgelieferte Browser kann sogar direkt von einer CD gestartet werden, ohne unerwünschte Installationen beim Anwender vornehmen zu müssen.

Die Hersteller von Opera schreiben sich auf die Fahne, den in anderen Browsern zu findenden überflüssigen Ballast erheblich reduziert und dafür intelligente Zusatzfunktionen ergänzt zu haben. Opera unterscheidet sich bereits im Design in einigen wichtigen Details von anderen Browsern. Etwa befindet sich die Statuszeile oben und die Adresszeile unten (entgegen der Positionierung bei den Rivalen). Nach einer Eingewöhnung erweisen sich die meisten Veränderungen als echte Verbesserungen. Opera bietet zudem zahlreiche Einstellungsvarianten, die den Browser äußerst flexibel anpassen lassen. Selbstverständlich gibt es die von den anderen Browsern gewohnten Features (so lange nach Einschätzung der Opera-Leute sinnvoll) wie Lesezeichen (obwohl auch diese leistungsfähiger sind als bei den Kontrahenten - sie erlauben das gleichzeitige Hinzufügen mehrerer Bookmarks), aber die Vorteile von Opera liegen woanders. So bietet Opera beispielsweise einfachste Möglichkeiten, das Laden von Grafiken auf Wunsch zu aktivieren oder zu deaktivieren um Ladezeiten einzusparen (Echtzeit-Grafikumschaltung). Und nicht nur Grafiken können deaktiviert werden. Opera unterstützt zwar auf der einen Seite alle gängigen Multimedia-Formate, aber alle Multimedia-Elemente können auf einfachste Art und Weise jederzeit abgestellt werden, inkl. GIF-Animationen, JavaScript, Tabellen, Frames, Grafiken allgemein, Cookies, Referrer Logging, Videos, Sound in Webseiten und PlugIns. Statt sich über überfrachtete Webseiten mit endlosen Ladezeiten ärgern zu müssen, hat der Anwender totale Kontrolle über die Anzeige von Webseiten¹⁷. Ebenso können dynamisch per Tastenkombination in Webseiten gewählte Farbkombinationen durch die Standardanwendereinstellung ersetzt oder wieder zurückgeschaltet werden. Genauso können mit Tasten Schriften in Webseiten einfach vergrößert und verkleinert werden (das können auch andere Browser), und zwar - wenn gewünscht - mitsamt den integrierten Grafiken (also eine Zoom-Funktion, und so etwas kennt weder der Navigator noch der Internet Explorer).

Etwas gewöhnungsbedürftig für Browserverhältnisse sind die im Opera verwendeten Mehrfachfenster. Dies ist eine andere Fenstertechnik, als herkömmliche Browser verwenden. Opera wird auch dann nur einmal in den Arbeitsspeicher geladen, wenn mehrere Webseiten angezeigt werden sollen (in anderen Browsern wird für jedes parallel angezeigte Dokument eine neue Kopie des Browsers geladen). Die verschiedenen Fenster werden in Opera so geöffnet wie beispielsweise Word-Dokumente in den Versionen vor Word 2000. Dies ermöglicht dem Anwender erheblich mehr Kontrolle über die einzelnen geladenen Webseiten. Damit können mehrere Fenster gleichzeitig geöffnet und überlagert, gekachelt oder im Vollbildmodus angezeigt werden.



Abbildung 1.11: Gekachelte Fenster im Opera



Die Mehrfachfenstertechnik des Opera kann bei Seitenreferenzen unter JavaScript Schwierigkeiten machen. Ohne zu weit vorzugreifen zu wollen: Es gibt in JavaScript die Möglichkeit, mit einer `open()`-Anweisung neue Seiten im Browser anzeigen zu lassen. Opera kann da in einigen Versionen recht eigen reagieren, weshalb solche Konstrukte unbedingt damit getestet werden sollten. Leider muss man bei Opera aber auch an anderen Stellen mit Problemen bei JavaScript (und auch Java) rechnen, was der vielleicht größte Kritikpunkt beim Opera-Browser ist. Die Probleme sollten aber in der Finalversion 4.0 beseitigt sein.

Ein Highlight von Opera ist die Offline-Arbeit. Wenn man mit dem Browser offline geht, werden nicht einfach alle Verbindungen gekappt, sondern Opera fragt nach, ob die geladenen Fenster gesichert werden sollen.

Im Bereich der Sicherheit hat Opera die vollen 128-Bit- Verschlüsselungsprotokolle nach SSL 2 und 3 sowie TLS 1.0 integriert. Java wird bis zur Version 4.0 nur über ein Java PlugIn von Sun Microsystems unterstützt, die Version 4.0 hat Java dann komplett implementiert¹⁸. Insbesondere hält sich Opera aber strikt an die von Sun veröffentlichten Vorgaben für Java.

In Bezug auf Style Sheets hält sich Opera strikt an den W3C-Standard für CSS1 und unterstützt diesen nach eigenen Angaben fast vollständig. Dies ist keine Selbstverständlichkeit, wie sowohl der Internet Explorer, aber auch der Navigator zeigen (beide halten sich nicht vollständig an die W3C- Empfehlungen). Etwas schmalbrüstig zeigt sich Opera bedauerlicherweise in der Unterstützung von JavaScript. Bis zur Version 4

wird nur JavaScript 1.1 unterstützt, ab der Version 4 soll JavaScript 1.3 funktionieren, und die bei vorherigen Versionen vorhandenen JavaScript-Probleme sollen behoben sein¹⁹. Ebenso wird die Mail- und News-Unterstützung erst ab der Version 4 konkurrenzfähig sein (bisher kann Opera Emails versenden, jedoch nicht empfangen, und der eingebaute Newsreader funktioniert nur online).

Einige Alternativen

Der NCSA Mosaic ist der Urahn aller grafischen Browser und wird seit einiger Zeit nicht mehr weiterentwickelt (dessen Nachfolger ist wie gesagt der Navigator). Er kann deshalb weder die neuen HTML-Standards anzeigen noch Webscripte ausführen. Trotzdem ist dieses Programm bei Anwendern noch sehr beliebt, denn der Browser ist schnell und läuft auf den einfachsten Rechnern mit minimaler Ausstattung. Gerade diese Verbreitung und die mangelnden Fähigkeiten machen ihn zum idealen Testbrowser, um zu sehen, wie sich ein Projekt in einem Browser mit minimalen Fähigkeiten darstellt.

Ein erwähnenswerter Browser ist ebenso HotJava, den die Firma JavaSoft - ein Ableger des Java-Erfinders und JavaScript-Mitentwicklers Sun - vollständig in Java geschrieben hat. Durch die Erstellung des Browsers in Java bietet er die optimale Integrationsmöglichkeit für Java-Applets in HTML-Seiten. Ebenso ist er als Java-Applikation plattformunabhängig und kann direkt auf allen 32-Bit-Betriebssystemen eingesetzt werden, wo Java unterstützt wird. Durch die nicht vorhandene Beschränkung auf das Windows-Design zeichnet er sich gegenüber seinen Konkurrenten auch durch ein anderes Styling aus. Leider ist der Browser recht langsam und hat sich nicht allzu weit verbreitet.

Tools zur Webseitenerstellung und JavaScript-Programmierung

Bei der Erstellung von Programmen und auch Webseiten stellt sich die Frage, welche Programme man dazu überhaupt braucht? Dass Sie als zukünftige Webseitenersteller und JavaScript-Programmierer verschiedene Browser benötigen, haben wir ja gerade durchgesprochen. Aber welche weiteren Tools benötigen Sie noch? Wenn man der Fraktion der Hardcore- Programmierer folgt - für HTML oder JavaScript keine. Dieses Lager der Webprogrammierer verwendet nur einen Texteditor, wie er beispielsweise in Windows schon dabei ist (unter Zubehör). Selbst dieser Minimaeditor ohne jeglichen Komfort langt für alles, was Sie mit HTML und JavaScript machen können. Im Ernst. Hardcore-Programmierer geben jeden Befehl Buchstabe für Buchstabe über die Tastatur ein. Da sowohl HTML als auch JavaScript ausschließlich aus Klartext bestehen, benötigt man nach der Erstellung des Quellcodes auch kein weiteres Programm, welches den Quelltext in Maschinencode übersetzt. Dies übernimmt ja der Interpreter - sprich der Browser - auf der jeweiligen Plattform.

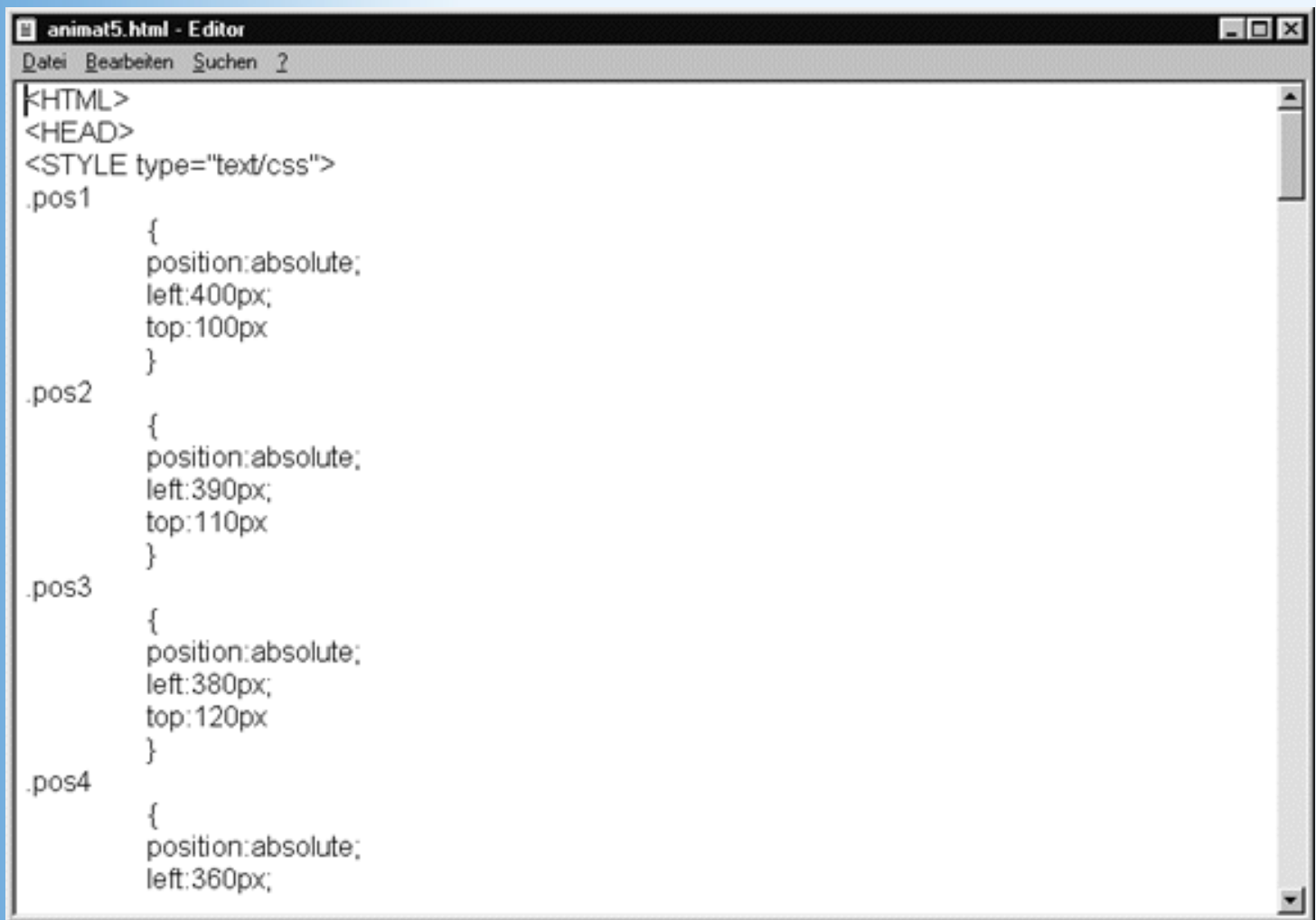


Abbildung 1.12: Der Windows-Einfachsteditor Notepad

Das direkte Eingeben aller Befehle Buchstabe für Buchstabe über die Tastatur hat einiges für sich. Etwa die vollständige Kontrolle über alle Befehle einer Sprache (auch die neuesten Befehle) und Anweisungen in einem Dokument (keine Menüstruktur oder sonst etwas schränkt einen ein), eine schnelle Eingabe der Befehle (wenn man fix mit der Tastatur ist und die Befehle kennt), keinerlei Kosten für irgendwelche Programme oder Einarbeitungszeit in ein solches Tool (englisch: Werkzeug, also ein Hilfsprogramm, welches bestimmte Arbeiten erleichtern soll).

Nur - so gnadenlos Buchstabe für Buchstabe ohne Hilfe durch Tools wird die ganze Geschichte nicht sonderlich komfortabel sein. Und Sie werden viel Zeit verlieren, indem Sie Dinge manuell eingeben müssen, die mit guten Tools leichter und schneller gehen. Ebenso ist die Suche von Fehlerquellen ohne Hilfsmittel sehr mühsam. Grundsätzlich werden die Resultate von professionellen Hardcore-Programmierern zwar nicht schlechter sein als die Ergebnisse der meisten Tools zur Erstellung von Webseiten. Im Gegenteil - normalerweise viel besser als die von Tools, wenn diese zu viel automatisch machen. Man hat jedoch viel Schreibarbeit (insbesondere bei standardisierten Elementen meist überflüssig), muss Befehle samt Parameter genau kennen, eventuelle spezielle Zeichenkombinationen umständlich eingeben²⁰ und permanent mit Schreib- und Flüchtigkeitsfehlern rechnen.

Es erscheint also sinnvoll, sich von Tools ein bißchen helfen zu lassen - sowohl in Hinsicht auf HTML als auch JavaScript. Dabei sollte aber gleich voran geschickt werden, dass die Hilfe der meisten Tools bezüglich HTML recht umfangreich werden kann, für JavaScript jedoch in der Regel ziemlich beschränkt bleibt.

Welche Form von Hilfsprogrammen und welche konkreten Tools sollten Sie dann aber verwenden? Diese Frage kann nicht allgemein beantwortet werden, denn jeder Webseiten-Ersteller und JavaScript-Programmierer hat seine persönlichen Vorlieben, Stärken und Schwächen.

Die Webeditoren in den Browser-Paketen

Auf jeden Fall haben Sie schon mal die Tools zur Verfügung, die in modernen Browser-Paketen integriert sind. Im Fall vom Internet Explorer ist das Frontpage Express, im Fall vom Netscape Communicator der Composer.

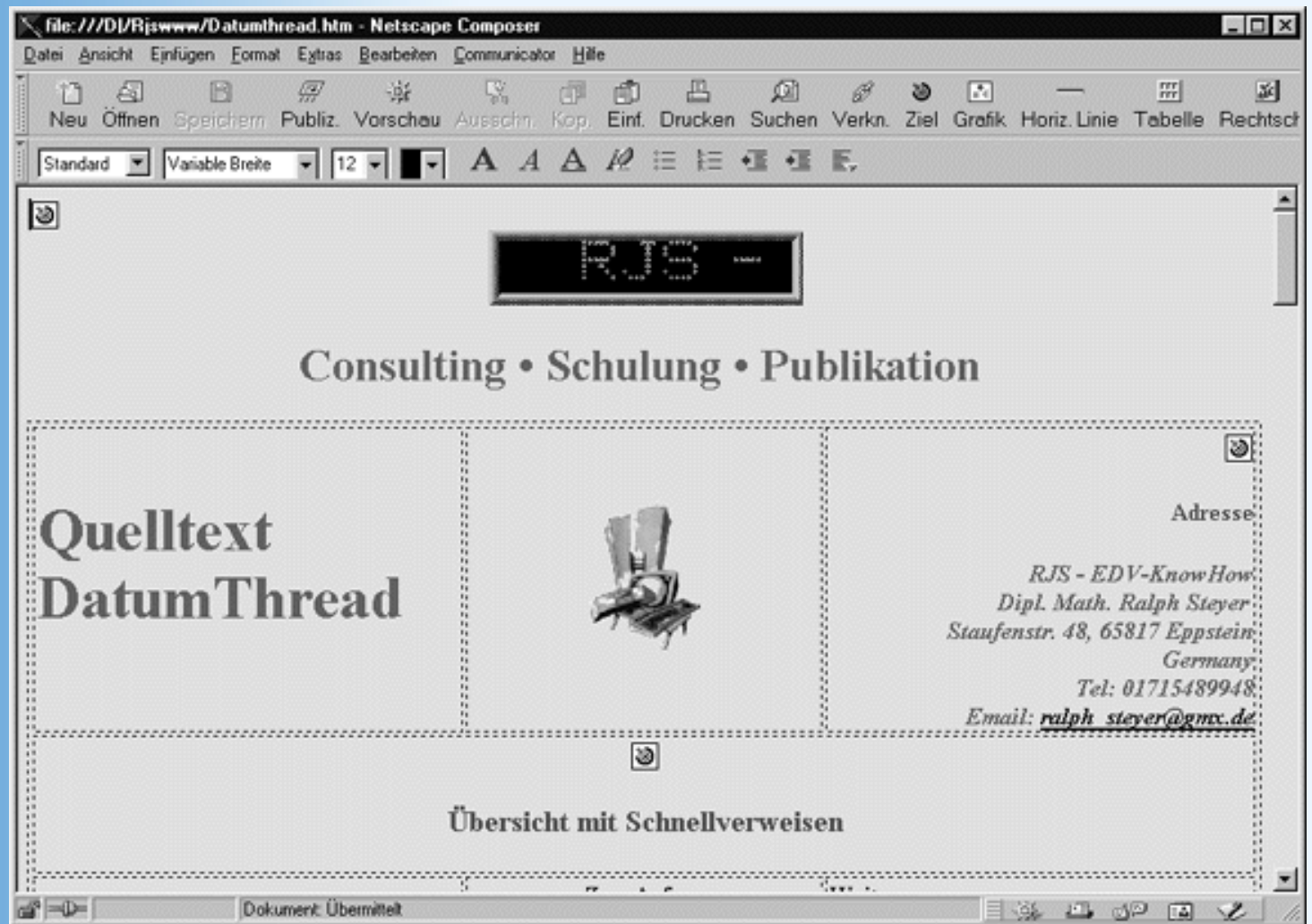


Abbildung 1.13: Eine Webseite im grafischen Erstellungsmodus des Composers

Diese integrierten Programme zur Erstellung von Webseiten werden jedoch von vielen Profis nicht sonderlich geschätzt. Das liegt an mehreren Gründen, die wir im Folgenden durchsprechen wollen. Steigen wir in die Erklärung etwas grundsätzlicher ein.

Texteditoren mit sinnvollen Zusatzfunktionen

Besonders sinnvoll und auch bei Hardcore-Programmierern meist noch akzeptiert sind Texteditoren mit ein paar sinnvollen Zusatzfunktionen: Etwa dass sie mehrere Dateien gleichzeitig öffnen lassen, »Suchen und Ersetzen« in mehreren Dokumenten in einem Schritt erlauben oder einige Bestandteile einer Sprache (etwa HTML) kennen und damit gewisse Vorgänge unterstützen. Etwa das Maskieren von Sonderzeichen oder Hilfestellung für die Übersichtlichkeit durch farbliche Kennzeichnung von bekannten Befehlen. Gerade dies als Syntaxhighlighting bezeichnete Verfahren, welches einen Quelltext vor der Anzeige durch einen Syntaxparser analysieren lässt, vereinfacht eine weniger fehlerträchtige Programmierung erheblich.



Ein Parser ist ein Mechanismus zum Identifizieren von zu einer Sprache gehörenden

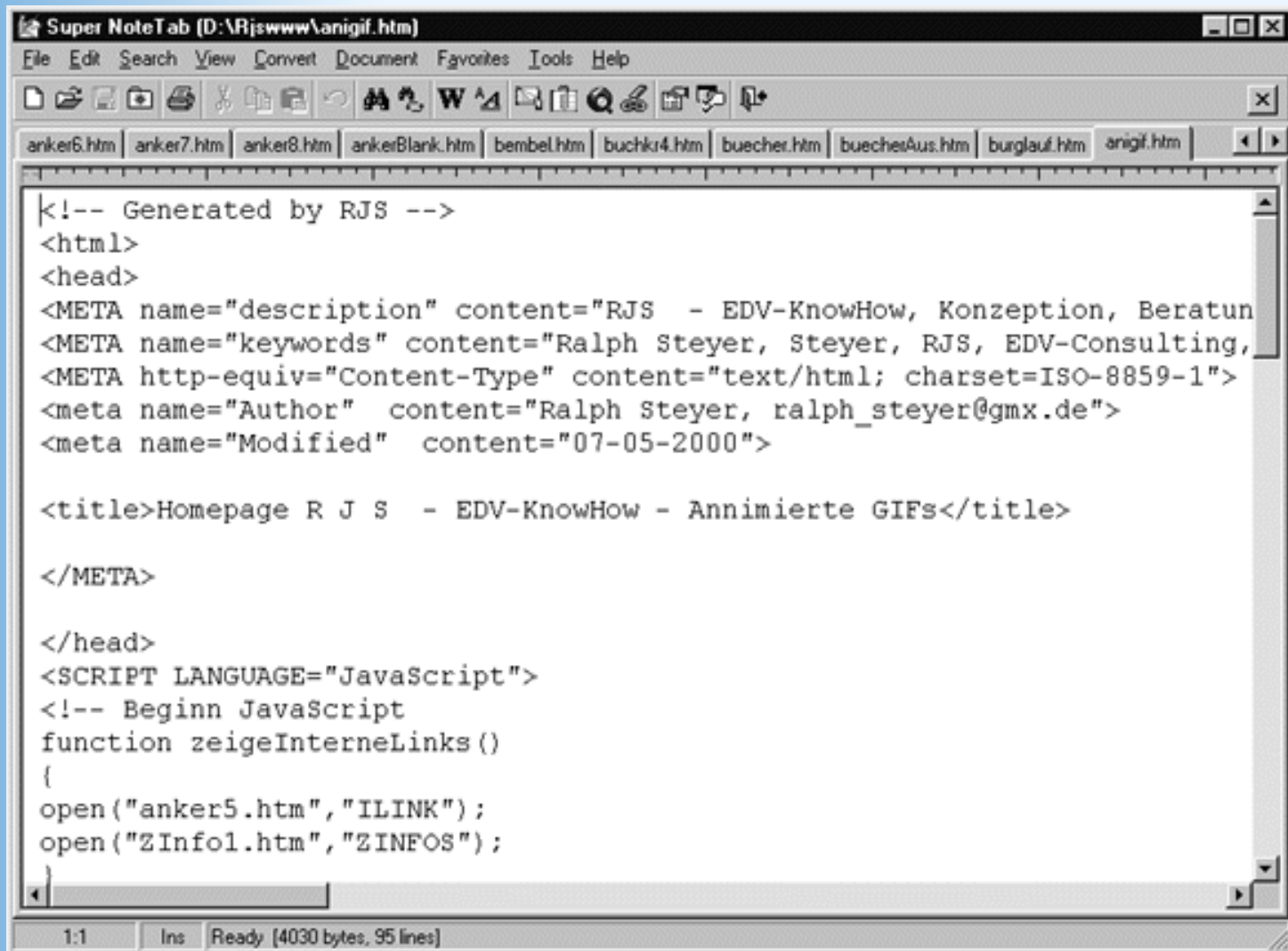
Ausdrücken.

Abbildung 1.14: *Super NoteTab, ein Editor, der mehrere Dokumente gleichzeitig verwalten kann*

Wenn luxuriöse Texteditoren dann noch bei Standardvorgängen wie Suchen und Ersetzen von Textpassagen leistungsfähige Bonbons anbieten (etwa das automatische Ersetzen von Textpassagen in allen geöffneten Dokumenten, wie es beispielsweise der Freeware-Editor Super NoteTab - <http://www.unige.ch/sciences/terre/geologie/fookes/> - kann), hat man sich schon einige Arbeit gespart, ohne Vorteile der Hardcore- Programmierung aufzugeben.

Etwas mehr Hilfestellung geben HTML-Editoren und direkt HTML- erstellende Programme (die oft auch Grundzüge von JavaScript kennen). Solche Programmtypen bieten die Befehle einer Sprache direkt an - etwa durch Menüs oder Symbolleisten, wo der Anwender diese auswählen kann (auch mit der Maus). Sie werden je nach der Auswahl und dem Programm dann in verschiedenartiger Form im Dokument dargestellt. Obwohl hier bereits die Grenze für viele Verfechter der Hardcore-Programmierung überschritten ist, bieten solche Tools Hilfestellungen in vielen Situationen. Zumal gute Tools dieser Art verschiedene Ansichten eines Webdokumentes bieten - etwa eine Vorschau, einen grafischen Erstellungsmodus und vor allem eine Ansicht des eigentlichen HTML- und JavaScript-Codes.

Wizards, Office-Programme und WYSIWYG-Editoren

Die generelle Ablehnung von weitergehenden HTML-Tools durch viele Experten rührt daher, dass es leider auch zahlreiche schlechte Vertreter von HTML-Editoren und Tools zur Webseitenerstellung gibt. Zumindest aus Sicht derjenigen, die HTML und/oder JavaScript beherrschen und die Hintergründe des Webs verstehen.

Vollkommen indiskutabel sind für halbwegs in die Materie involvierte Anwender so genannte Wizards für Webseiten, die einem Anwender vollständig die Erstellung eines Dokuments abnehmen und nur auf Grund einiger Basiseingaben eine HTML-Seite erstellen. Diese Resultate werden langweiliger Einheitsbrei sein, der keinerlei gesteigerten Ansprüchen genügen kann. Dazu ist der erzeugte Quelltext nie optimal. Dies gilt auch für Wizards, die von Online-Anbietern als Webseiten für ihre Kunden ins Netz gestellt werden (so etwas wie die Homepage für 39 Pfennige mit Homepage-Baukasten). Die Verwendung von Wizards bleibt meist nur absoluten Laien vorbehalten und scheitert auf jeden Fall, wenn Scripte ins Spiel kommen.

Aber auch viele andere Programme, welche Dokumente in HTML-Code abspeichern können, verschleiern vollständig, dass damit HTML-Seiten für das heterogene Web erstellt werden. Anwender werden dabei meist vom eigentlichen HTML-Source ferngehalten. Es handelt sich bei solchen Quellcode-fernen Programmen entweder um reine Office-Programme (etwa Word oder Excel ab der Version 7.0), deren resultierende Seite einfach im HTML-Format gespeichert wird. Wenn der resultierende HTML-Quelltext in einem Texteditor analysiert wird, erkennt der Profi unzählige Stellen, die verbessert werden könnten²¹.

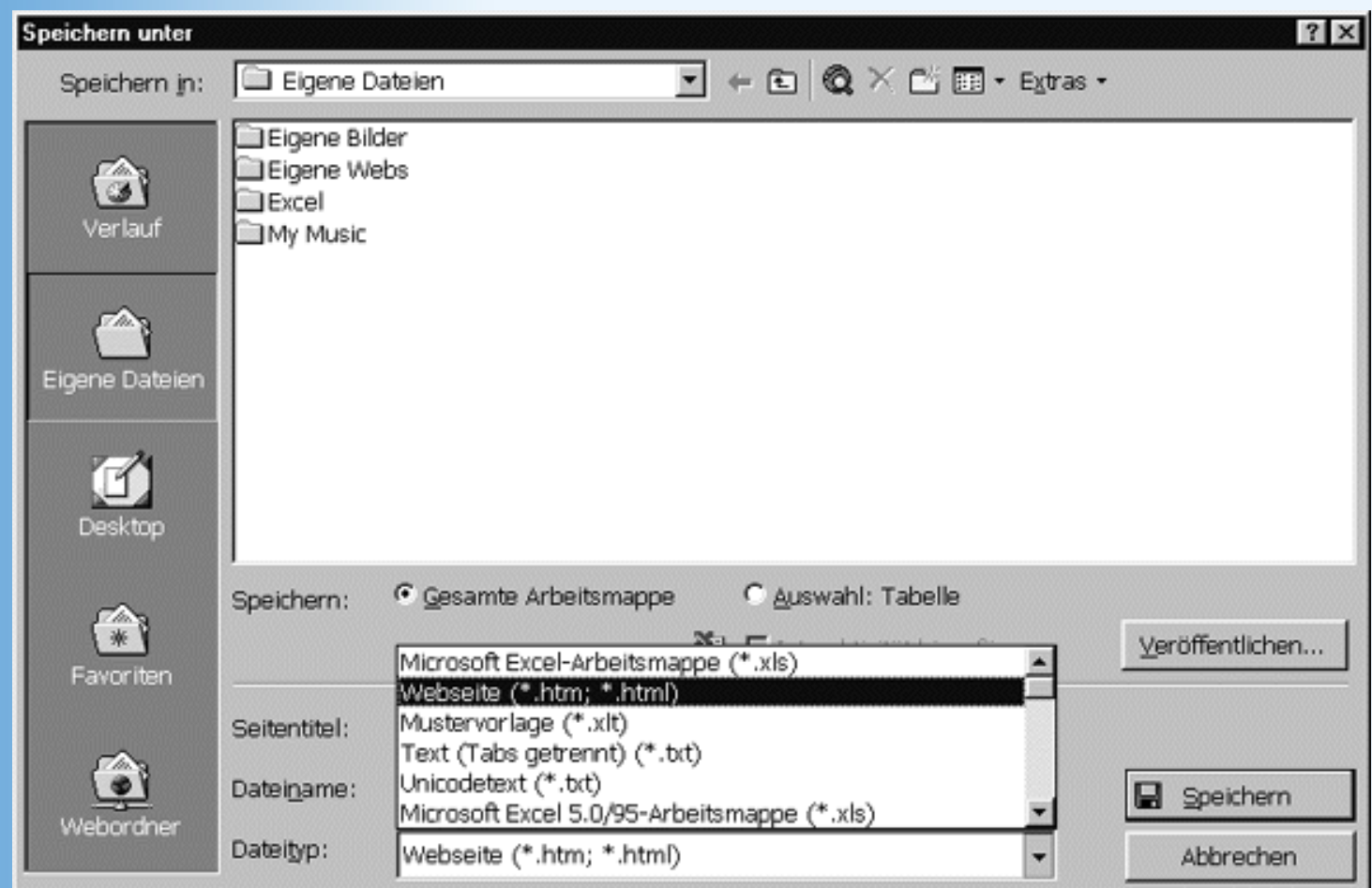


Abbildung 1.15: Ein Office-Dokument einfach als Webseite gespeichert

Die anderen von vielen Experten abgelehnten Programme sind solche Tools, welche sich von der Bedienung wie eine Textverarbeitung zeigen und dem Anwender suggerieren, eine Webseite lasse sich wie ein Textdokument für den Drucker erstellen. Dazu zählen leider auch die meisten der in den Browser-Paketen integrierten Tools (zumindest in den meisten bisherigen Versionen - die neuesten integrierten Tools sind etwas besser). Solche Programme arbeiten meist mit einem grafischen Erstellungsmodus, der den Zauberzustand WYSIWYG (What you see is what you get) verspricht (wie auch ein Office-Programm). Ein solcher WYSIWYG-Zustand ist bei der Erstellung von Webseiten für das Internet jedoch reine Theorie. Eine Vorgaukelung einer Situation, die einen unerfahrenen Webprogrammierer in trügerische Sicherheit wiegt und Probleme verschleiern. Der Reiz und die Leistungsfähigkeit des Internets beruhen doch darauf, dass Anwender mit den unterschiedlichsten Plattformen zusammengeschlossen sind. Was soll dann das Weismachen eines bestimmten Aussehens, wenn die meisten Anwender die Seite nicht so sehen werden? Kaum ein Anwender wird die vollkommen identische Konstellation wie der Ersteller der Webseite haben²².

Dies würde einen übereinstimmenden Browser samt identischer Versionsnummer, eine vollkommen deckungsgleiche Konfiguration des Browsers, die kongruente Bildschirmauflösung, die gleiche Anzahl Farben, das analoge Betriebssystem, die gleiche Monitorgröße und den korrespondierenden Rechnertyp voraussetzen. Eine mehr als unwahrscheinliche Voraussetzung. Um dennoch eine halbwegs realistische Vorschau bieten zu können und das Resultat auf den unterschiedlichsten Plattformen oder zumindest den Rechnern einer Familie (etwa PCs mit Windows-Betriebssystemen) nicht zu extrem von dem - scheinbaren - Endergebnis beim Erzeuger der Webseite abweichen zu lassen, werden alle denkbaren Techniken zum Bändigen des Internets herangezogen. Style Sheets, massenhafter Einsatz von HTML- Befehlen, sehr neue Befehle, eventuell Plattform-beschränkte Anweisungen usw. Es wird versucht, nahezu jedes einzelne Zeichen unter Kontrolle zu bekommen. Statt sich die fließende und freie Darstellung von Seiten im Web zunutze zu machen, wird mit brutaler Gewalt versucht, Webseiten wie Papierdokumente zu behandeln (entgegen der ursprünglichen Philosophie des Webs). Anstelle auf den Wellen des Webs wie mit einem Surfbrett zu reiten, versucht man, die Wellen des Web-Meers einzubetonieren, um trockenen Fußes an den Strand zu laufen. Der erzeugte HTML-Code wird deshalb sehr oft schlecht sein, weil er unnütz groß ist, aber oft dennoch nicht funktioniert. Der Anwender wird durch längere Übertragungszeiten verärgert, obwohl es überflüssig ist. Dazu kann kommen, dass der Einsatz von Browser-beschränkten Befehlen in anderen Browsern zu erheblichen Fehldarstellungen führt. So werden Microsoft-Tools unter Umständen Befehle verwenden, die der Navigator nicht versteht, und Netscape- Programme erzeugen HTML-Code, mit dem der Internet Explorer nichts anfangen kann. Daneben kann es vorkommen, dass Befehle automatisch gewählt werden, obwohl es bessere Befehle (etwa weiter abwärtskompatible Befehle oder mit kürzerem Code) mit gleicher Funktionalität gibt. Wenn dann noch ein Tool überhaupt keinen direkten Zugriff auf den Quelltext erlaubt, sollte ein solches Programm vom ambitionierten Anwender nur als Ergänzung für Teilaufgaben gesehen werden. Außerdem sollte solcher Code auf jeden Fall nachträglich in einem Editor analysiert und optimiert werden (darauf gehen wir am Tag 7 genauer ein).

Die Erstellung von Webseiten unter der Voraussetzung, bei der Erstellung bereits eine exakte Vorschau dessen zu sehen, was die Anwender sehen werden, drängt mir den Vergleich mit einem Auto auf, das ursprünglich für eine Geschwindigkeit von 100 km/h konzipiert wurde (das Web- und HTML-Konzept) und dessen Motor nun mit allen möglichen Tuning- Maßnahmen aufgebohrt wird. Hauptsache, der Wagen schafft die 200 km/ h. Ob Bremsen, Fahrwerk oder Sicherheitsausstattung dem noch gerecht werden, ist egal. Dem (unbedarften) Kunden zeigt man dann aber nur, wie schnell das Auto plötzlich ist, und verschleiert alle Hinweise auf Sicherheitsprobleme. Jedoch der Vergleich macht das Problem deutlich - wehe, wenn was schiefgeht. Führt die Kiste auf einer Autobahn mit absolut ebener Strecke ohne Schlaglöcher, Kurven, Hindernissen und ähnlichen Problemen (sprich, der Besucher hat eine Plattform, die kompatibel zu der des Webseiten-Erstellers ist), wird es wahrscheinlich gutgehen. Aber wehe, es kommt eine Kurve oder ein Hindernis (ein Besucher mit einer anderen Plattformeinstellung), dann knallt es. Wenn also das Webkonzept in einer Hinsicht optimiert wird, müssen auch Vorkehrungen für die Situationen geschaffen werden, wo die Optimierung für Probleme sorgt. Zumal - im Web hat man so gut wie nie so etwas wie eine freie Autobahn vor sich, sondern eine permanent verstopfte Straße, wo Hindernisse die Regel sind.



Office-Programme erzeugen zwar in Bezug auf die Dateigröße und Kompatibilität einen außergewöhnlich schlechten HTML-Code, aber man muss ihnen zugute halten, dass sie nicht nur das Web im Auge haben. HTML ist nicht gleich HTML. HTML für das Web sollte ohne Zweifel möglichst kleine Dateien mit maximaler Plattformunterstützung gewährleisten. Das unter den Office-Programmen erzeugte HTML muss jedoch das leisten, was die Hausdateiformate der Programme sonst gewährleisten. Eine Excel-Datei, die in HTML gespeichert ist, darf auf PC-Ebene gegenüber dem Programm-eigenen XLS-Format keine Informationen verlieren. Dies erzwingt viel HTML-Code und Stilvereinbarungen, die im Web überflüssig oder gar schädlich sind. Auf Einzelplatzrechnern, homogenen Netzwerken oder Intranets ist es hingegen durchaus sinnvoll. Nur dann sollte man klar sagen, dass es nun mal

keine Eier-legende-Woll-Milch-Sau gibt - auch wenn die Hersteller von Office-Programmen ihre Produkte als solche an den Kunden verkaufen wollen.

Aber die Verwendung von Tools beinhaltet noch einige weitere Klippen. Da sind erst einmal potentielle Kosten, denn einige Tools sind nicht kostenfrei. Das sollte aber kein Argument sein, denn Freewareprogramme oder in Browser integrierte Erstellungs-Tools bekommen Sie umsonst und deren Leistung langt allemal (oder sind sogar bereits überfrachtet). Die Fallstricke sind andere.

Programme können naturbedingt nur den Standard einer Sprache kennen, der zum Zeitpunkt ihrer Erstellung realisiert war. Wenn dieser erweitert wird, muss es eine Möglichkeit zur Eingabe von den dann notwendigen neuen Befehlen geben (oder auch sonst nicht direkt im Programm unterstützten Anweisungen). Dummerweise bieten viele Programme nicht diese Möglichkeit, neue Befehle hinzuzufügen oder sie anderweitig in den Quellcode zu integrieren. Dies ist gerade Programmen ohne direkten Quellcode-Kontakt ein gravierender Mangel.

Ein anderes Problem, das lange Zeit insbesondere Frontpage angekreidet wurde (in neueren Versionen ist der Mangel beseitigt), ist die ungewollte Codeveränderung durch Tools. Das bedeutet, eine auf einem anderen Weg erstellte Webseite wird dabei beim Laden in ein Tool vollkommen verändert (auch wenn in dem betroffenen Bereich keine Veränderungen vorgenommen werden). Gut und komprimiert erstellter HTML-Code wird nach den Vorstellungen des Tools umgebaut und meist schlechter und größer. Insbesondere nutzen viele Tools Webseiten, um Eigenwerbung für den Hersteller des Tools in der Seite unterzubringen (in Form einer Metainformation, die wir bei HTML behandeln). Dahingegen werden Möglichkeiten zur sinnvollen Nutzung von Metainformationen (Hintergrundinformationen über die Webseite) in der Regel nicht bereitgestellt (auf Metainformationen gehen wir am Tag 3 ein).

Content-Management-Systeme

Ein bei diesen bisherigen Ausführungen vollkommen außer Acht gelassenes Thema ist jedoch das, was mit dem schrecklichen neudeutschen Unwort »Content Management« bezeichnet wird. Dies bezeichnet die - möglichst transparente - Erstellung und vor allem Verwaltung von Inhalten und Webseiten. Insbesondere, wenn Web-Projekte sehr umfangreich²³ sind und sich in sehr kurzen Zeitabständen aktualisieren. Professionelle Content- Management-Systeme (CMS) können Kosten im fünf- bis sechstelligen Bereich verursachen und sind für Privatanwender sowieso, aber auch für kleinere und mittlere Webpräsentationen überdimensioniert. Umfangreichere Webeditoren bieten heutzutage oft bereits ein Seitenmanagement für kleinere bis mittlere Projekte. Das beinhaltet vom Layouten der Webseiten (mit allen gerade beschriebenen Faktoren) bis hin zum Publizieren insbesondere weitergehende Features wie automatische Linkkontrolle, übergeordnetes, grafisches Erstellen von Navigationsstrukturen, Verzeichnis- und Dateiverwaltung, Vorlagenerstellung, Einbindung von unterschiedlichsten Elementen, Assistentenunterstützung, die diesen Namen auch verdienen, Datenbankverbindungen oder Unterstützung von verteilter Arbeit mit mehreren Personen in größeren Projekten. Meist stellen diese Programme mehrere Fenster bereit, in denen dann die verschiedenen Vorgänge parallel durchgeführt werden können.

Wir wollen, obwohl diese Systeme in der Regel mehrere hundert DM kosten, ein paar professionelle Webeditoren zumindest kurz vorstellen.

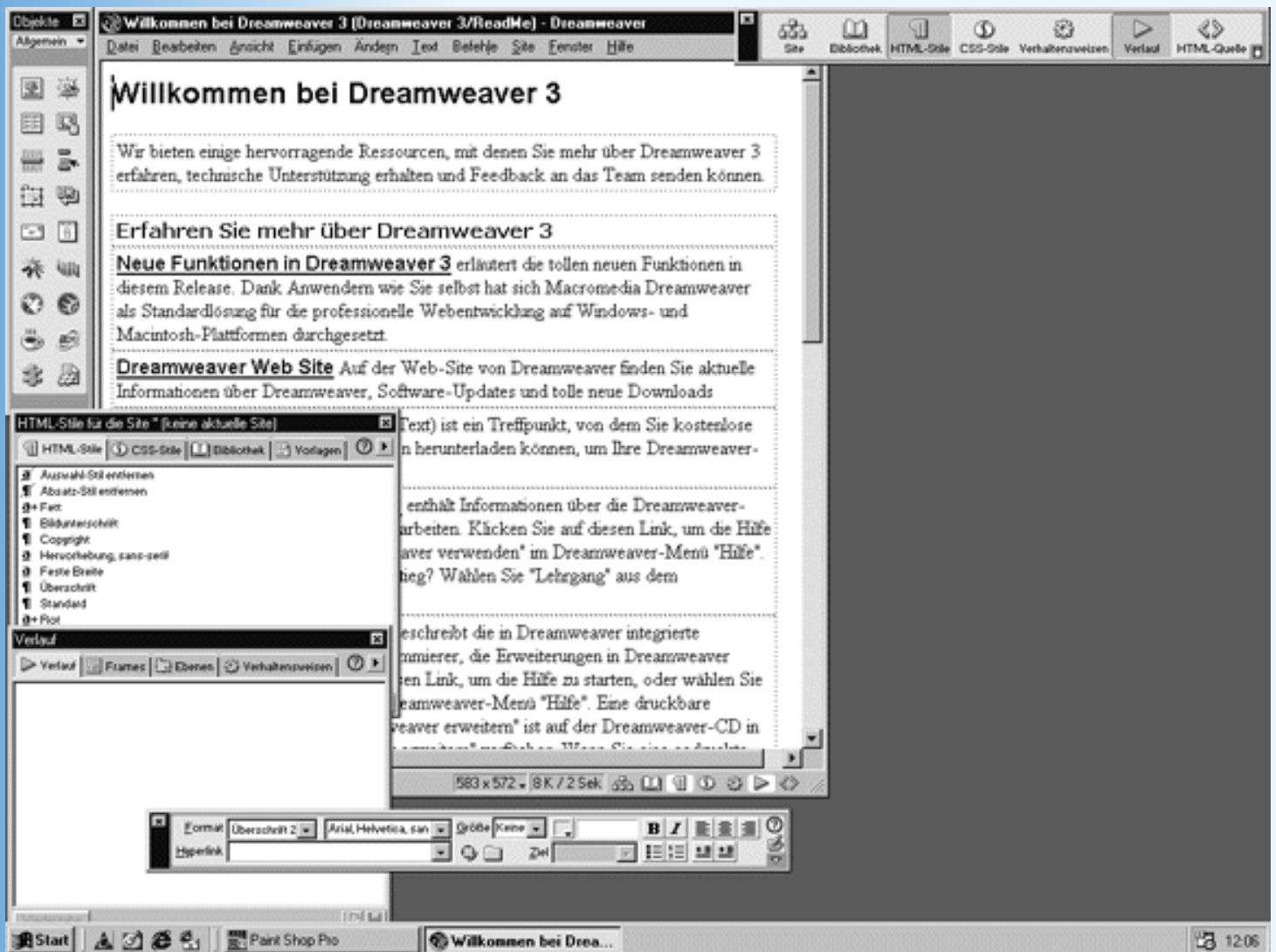


Abbildung 1.16: Dreamweaver 3.0

Macromedia Dreamweaver 3.0 ist einer der beliebtesten professionellen Webeditoren, dessen Stärken im Designbereich (Style Sheets, dynamische Elemente, Positionierungen - und das weitgehend Browser-unabhängig) liegen. Die Verwaltung von Struktur und Inhalt von Webprojekten ist nicht so weit gediehen.

Ein weiteres sehr populäres Tool ist Microsoft Frontpage 2000. Dessen Stärke ist eindeutig das Management von großen Seiten sowie die Bereitstellung von diversen Assistenten. Die Pflege von Inhalt ist jedoch unnötig kompliziert, da extern erstellte Inhalte nur schwer in das - dort Web genannte - Projekt zu integrieren sind. Außerdem werden HTML- Seiten Microsoft-typisch mit einer Unzahl von Formatierungsanweisungen versehen, die sich dann schwer abstellen lassen, wenn es überflüssig ist. Die nachträgliche Entfernung ist sogar fast unmöglich.

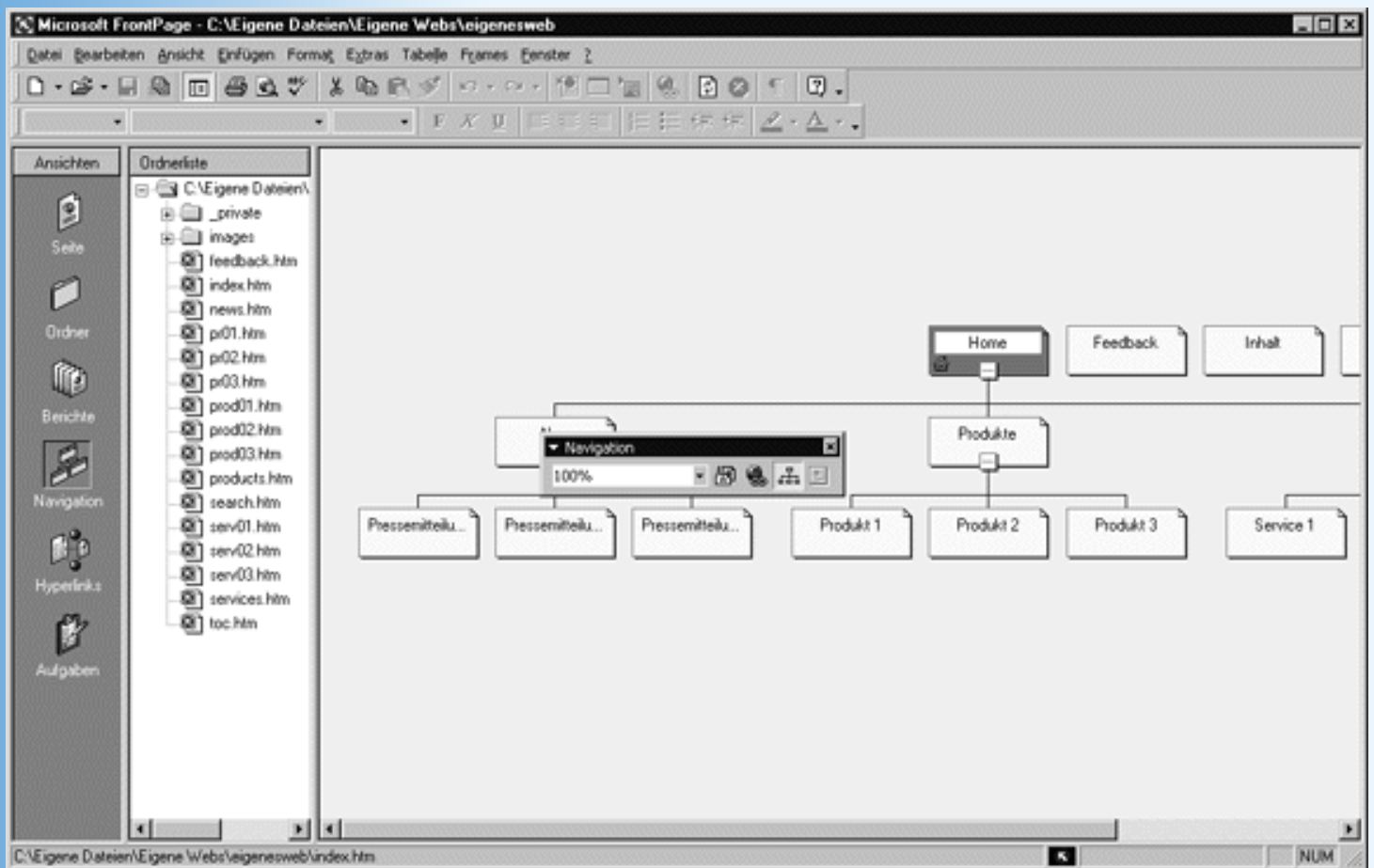


Abbildung 1.17: Frontpage 2000

Netobjects Fusion 5.0 ist ein weiteres - sehr teures - Tool, das für statische Seiten ideale Verwaltungsmöglichkeiten besitzt. Schwächen zeigt es mit dynamischen und häufig zu aktualisierenden Inhalten; dafür gibt es zahlreiche Schnittstellen zu externen Elementen wie ASP (Active Server Pages) und Datenbanken.

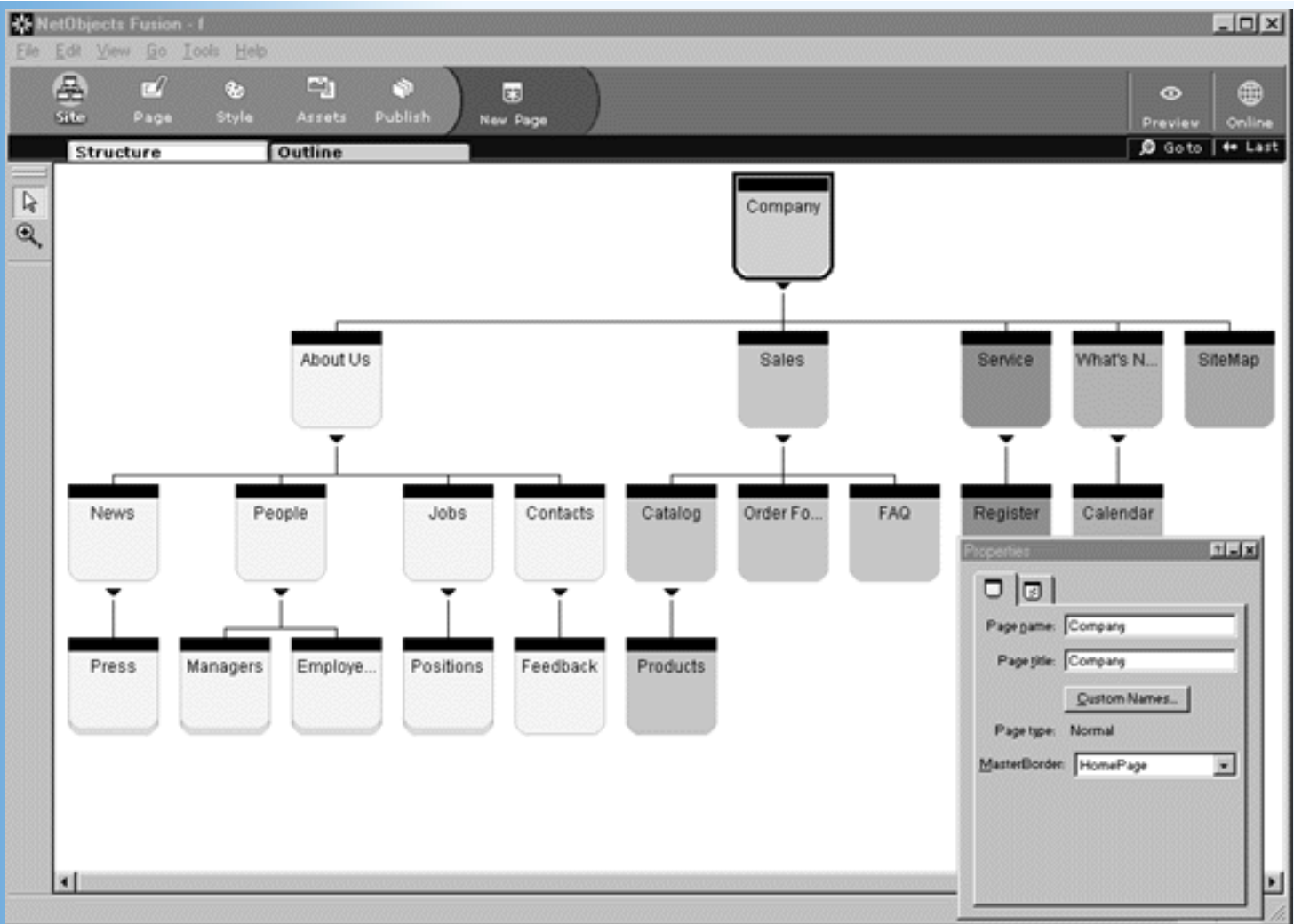


Abbildung 1.18: Fusion 5.0.

Hot Metal Pro 6.0 von der Firma Softquad ist eine weitere, verbreitete Alternative. Er ist einer der schon am längsten auf dem Markt befindlichen Webeditoren und gilt als recht ausgewogen, aber sehr ressourcenhungrig und gelegentlich instabil.

Grundsätzlich muss man bei diesen kommerziellen Webeditoren festhalten, dass sie wie gesagt recht teuer und oft kompliziert und überdimensioniert sind. Im professionellen Bereich bei größeren Webprojekten haben sie sicher ihre Bedeutung, im privaten Bereich und bei kleineren Projekten kann man sich das Geld sicher sparen. Zumal es Freeware-Editoren gibt, die diesen Tools kaum nachstehen.

Zwei gute Freeware-Webeditoren

Auch auf die Gefahr hin, andere gute Freeware-Webeditoren zu vergessen, sollen zwei leistungsfähige HTML-Programme angegeben werden, die zeigen, wie Webseitenersteller auf der einen Seite durch viele nützliche Tools sinnvoll unterstützt, auf der anderen Seite nicht durch Fernhalten vom Quellcode und anderen Gängeleien behindert werden. Beide Programme dürften sogar für Hardcore-Programmierer noch akzeptabel sein, da sie keinerlei Code verändern und ziemlich guten Code erzeugen. Dazu löst der Einsatz eines der beiden Tools ein grundsätzliches Problem, was man bei der Arbeit ohne Hilfsmittel hat - die sinnvolle Verwaltung und Strukturierung mehrerer zusammengehörender Dateien in Projekten (eben das, was mit dem grässlichen Begriff Content-Management beschrieben wird). Beide Programme unterstützen solche bei größeren Webprojekten sehr sinnvolle Verwaltung, nötigen einen Anwender jedoch nicht dazu, wie es etwa Frontpage versucht, wo die von einem Projekt losgelöste Bearbeitung einzelner Seiten ziemlich umständlich ist.

Die eine Empfehlung ist das australische Programm *1st Page 2000* von der Firma Evrsoft (<http://www.evrsoft.com>), welches sehr stark im Bereich Webseitenoptimierung ist, verschiedene

über HTML hinausgehende Techniken ansatzweise unterstützt (etwa CGI, Perl, SSI, ASP und natürlich JavaScript) sowie zahlreiche vorgefertigte Schablonen in verschiedenen Techniken bereitstellt. Insbesondere erzeugt das Programm - Zitat aus der Hilfe - »keinen unerwünschten Junk-Code wie die meisten WYSIWYG- Editoren«, sondern bietet eine flexible Programmierumgebung für den Anwender mit maximalen Einstellungs- und Optimierungsmöglichkeiten.

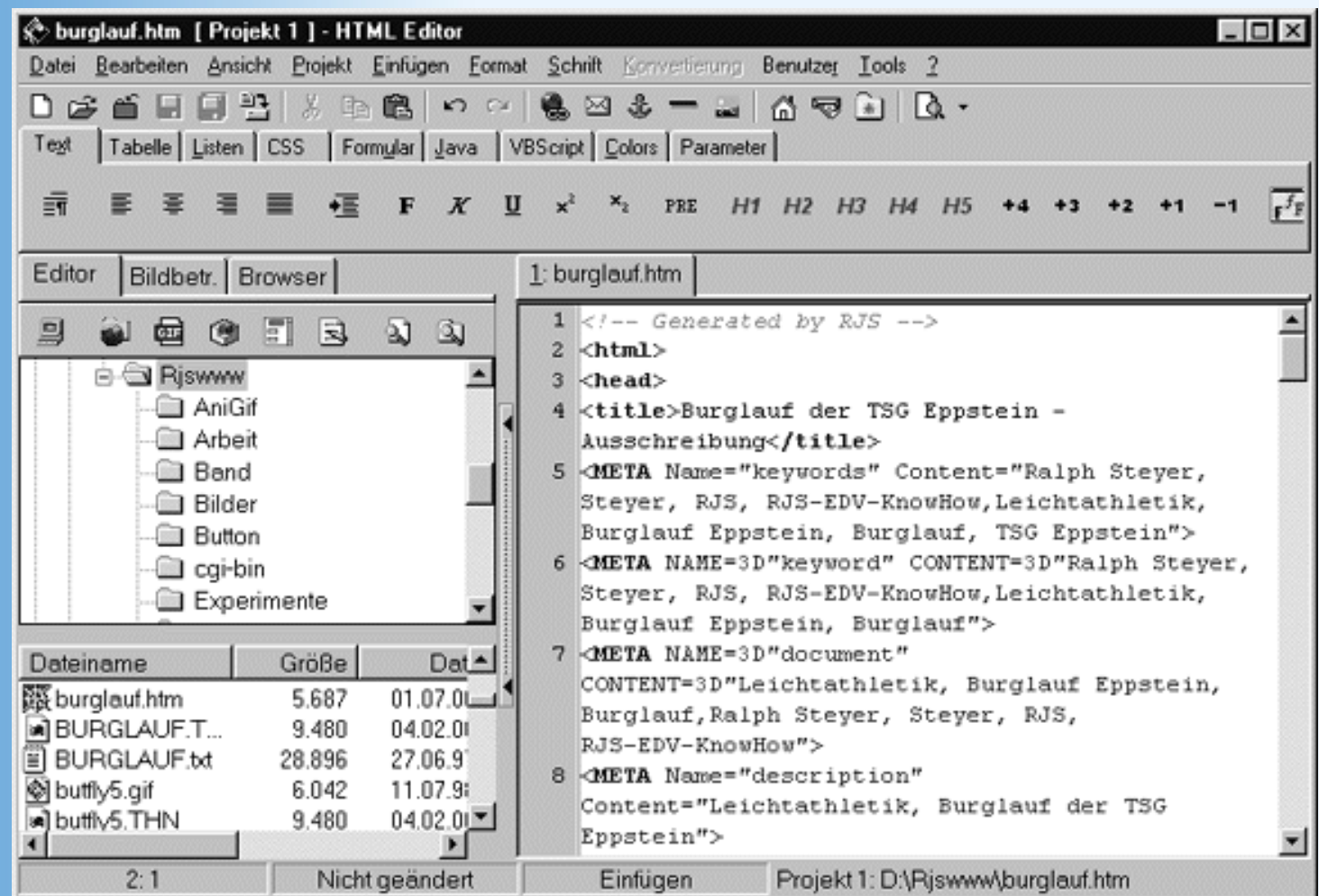


Abbildung 1.19: Der kostenlose HTML-Editor Phase 5

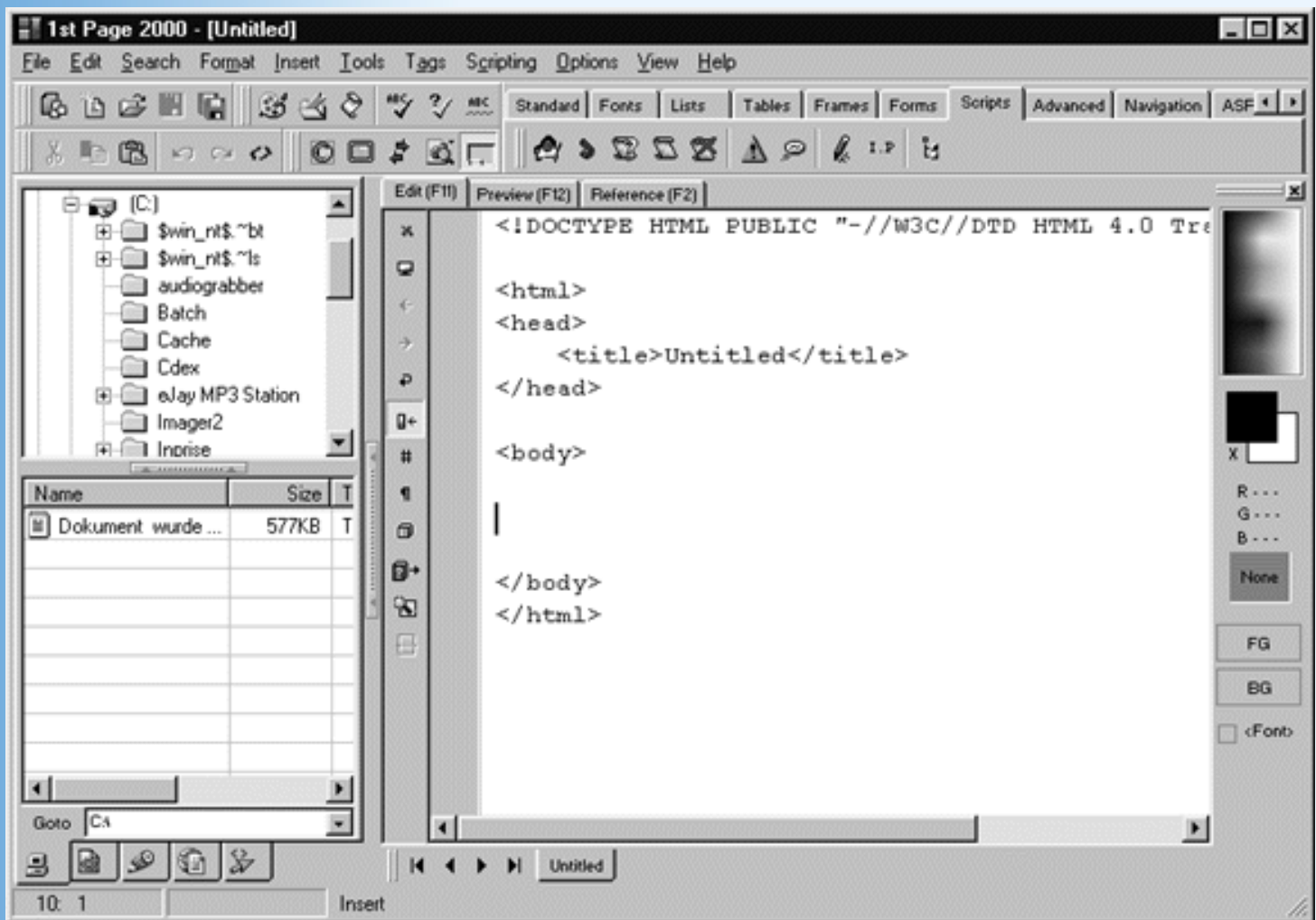


Abbildung 1.20: Der Freeware-HTML-Editor 1st Page 2000

Der zweite Vorschlag ist der deutsche HTML-Freeware-Editor *Ulli Meybohms HTML EDITOR Phase 5* (<http://www.meybohm.de>). Der Editor beinhaltet verschiedene Syntaxparser, welche die Darstellung von Quelltexten in verschiedenen Sprachen (JavaScript, HTML, SQL, Perl usw.) durch Syntaxhighlighting erheblich verbessert. Befehle, Parameter und Text werden durch unterschiedliche Darstellungsfarben voneinander abgehoben. Reservierte Wörter werden fett dargestellt. Der Editor erlaubt weiterhin die Darstellung von Zeilennummern, was erheblich bei der Fehlersuche in Scripten und Quelltexten hilft. In der unteren Statuszeile werden Statusinformationen des Editors angezeigt, zum Beispiel die Position des Cursors. Dazu kann der Anwender Befehle über Tastatur, Schaltflächen oder Menüs eingeben und bekommt auf Wunsch HTML- Container und Schablonen generiert.



Sie finden Phase 5 auf der Buch-CD.

Script-Editoren und Entwicklungstools für Programmiersprachen

Die bisherigen Überlegungen betreffen die Tools, welche im Schwerpunkt HTML-Seiten erstellen sollen. Was ist aber mit Script-Editoren und Tools für komplexere Sprachen? Nun, die meisten umfangreicheren HTML-Tools beherrschen Grundzüge von erweiternden Techniken. Da JavaScript eine der wichtigsten Erweiterungen ist, zählt diese Technik fast immer dazu. Allerdings beschränkt sich die jeweilige Unterstützung meist auf die Integration in Webseiten und wenige ausgewählte Anweisungen.

Es gibt auch Tools, die sich auf die weitergehende Unterstützung von JavaScript spezialisiert haben. Der gegenüber HTML komplexere Aufbau von Scriptsprachen lässt die Unterstützung durch geeignete Tools

sogar angeraten erscheinen. Aber obwohl bei einigen Script-Editoren von deren Herstellern versprochen wird, dass auf die Kenntnisse der Sprache verzichtet werden kann, zeigt sich immer wieder, dass dies nicht stimmt. Grundlegende Kenntnisse der gewünschten Sprachen sind immer notwendig, und leider fängt bei echten JavaScript-Tools auch das Geldausgeben an. Wir werden deshalb - und vor allem wegen des Lerneffekts - auf die Verwendung von JavaScript-Tools verzichten.

Bei »richtigen« Programmiersprachen wie C/C++, PASCAL oder Visual Basic kommen Sie ohne vollständige Tools nicht aus. Insbesondere wenn Sie keine Programme auf Interpreterbasis erstellen wollen. In diesem Fall benötigen Sie mindestens einen Compiler, also ein Programm, welches den in einem Editor erstellten Quelltext erst in Maschinencode übersetzt. Richtige Programmiersprachen sind außerdem kompliziert. Sie benötigen viele Funktionalitäten, die nur durch eine vollständige Programmierumgebung zur Verfügung gestellt werden.

Fazit

Trotz einiger wirklich empfehlenswerter Tools werden wir im Laufe des Buchs dennoch auf die explizite Verwendung von bestimmten Programmen verzichten. Dies hat mehrere Gründe:

- Kosten verursachende Tools sollen vollkommen außer Acht gelassen werden, zumal es meist fast gleichwertige, aber kostenlose Alternativen gibt - zumindest wenn die Projekte nicht zu extrem ausufern.
- Nicht alle Anwender werden den gleichen HTML-Editor gut finden, und die Verwendung eines bestimmten Editors wäre Zwang.
- Nicht HTML oder die Erstellung von allgemeinen Webseiten steht bei diesem Buch im Vordergrund, sondern JavaScript. JavaScript wird bei vielen Tools nur rudimentär unterstützt und deshalb lohnt es sich für unsere Beispiele kaum, ein Tool explizit zu verwenden.
- Es ist einfach effektiver vom Lerneffekt her, wenn man Befehle von Hand ohne irgendwelche Unterstützung eingibt. Wenn Befehle durch irgendwelche Hilfen vorgegeben werden, wird der Lerneffekt geringer. Die bei der späteren Arbeit unerwünschten Fehler sind beim Lernen eher nützlich. Man wird gezwungen, über Dinge nachzudenken, um Fehler zu finden, und damit lernt man das meiste.

Weitere sinnvolle Programme

Neben Browsern, einem geeigneten Texteditor und einem optionalen HTML-Editor gibt es noch weitere Programme, welche für die Erstellung von Webseiten im Allgemeinen sinnvoll (wenn nicht zwingend) sind. Dies sind im Wesentlichen Grafik- und Animationsprogramme sowie Tools zum Upload von den fertigen Seiten ins Internet.

Grafik und Animation

So gut wie keine Webseite kommt ohne Grafiken aus und auch aus JavaScript heraus kann eine Verbindung dazu aufgebaut werden. Sei es nur der Start eines Scripts über eine Grafik, sei es eine Script-gesteuerte Animation, sei es die Manipulation von Grafiken über Scripte. Ebenso finden sich in Webseiten immer mehr Animationen unterschiedlichster Art. Teilweise mit JavaScript über geschickte Steuerung von Grafiken realisiert, aber auch immer öfter, indem unterschiedlichste Techniken und Formate zum Einsatz kommen.

Unabhängig davon, dass der Einsatz von Grafiken und Animationen auf vielen Webseiten übertrieben wird - ein Webseitenersteller sollte einen kleinen Fundus an den wichtigsten Programmen aus dem Grafik- und Animationsbereich zur Verfügung haben. Zumal die meisten Programme für den normalen Gebrauch Freeware oder zumindest Shareware sind. Eine Ausnahme bildet das recht teure Macromedia Flash, das in letzter Zeit bei vielen Animationen im Web zum Einsatz kommt und mit dem Flash-Format mittlerweile

einen Animationsstandard im Web etabliert hat, der sich schon einer recht breiten Unterstützung erfreut²⁴.

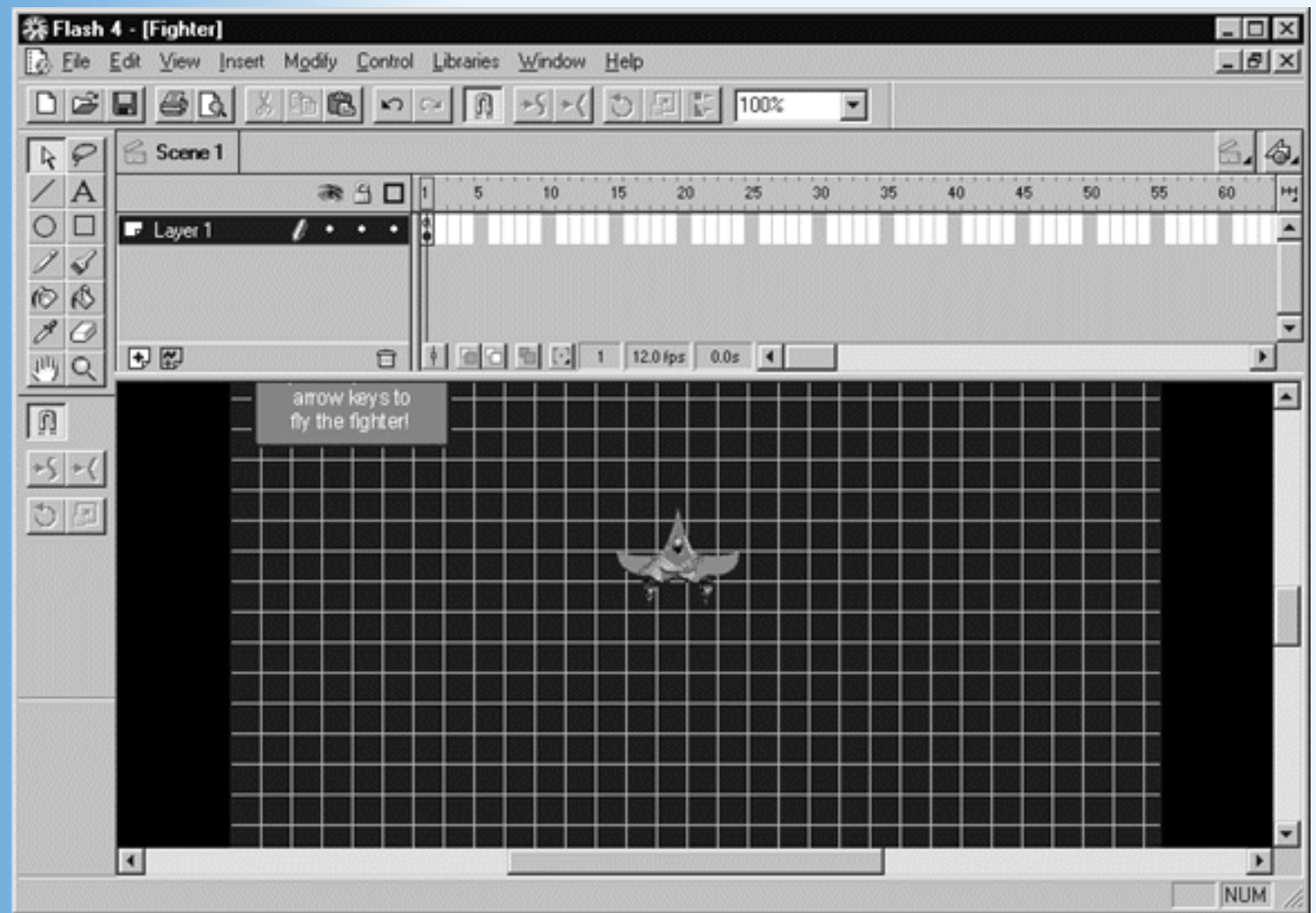


Abbildung 1.21: Flash

Aber wenn Sie JavaScript beherrschen, können Sie sich ein solches Animationsprogramm auch sparen - das meiste werden Sie am Ende des Buchs auch über JavaScript und Grafiken selbst programmieren können. Und das, ohne dass auf Zielrechnern eine Flashunterstützung vorhanden sein muss.

Ansonsten gehören normale Grafikprogramme auf fast jedem Rechner zur Grundausstattung. Die Palette ist so umfangreich, dass kaum alle sinnvollen Programme aufgeführt werden können. Das Grafikprogramm sollte nur auf jeden Fall in der Lage zur Verarbeitung der üblichen Internet-Grafikformate .GIF und .JPEG sein, eine möglichst große Anzahl von anderen Bildformaten importieren können und Konvertierungen bezüglich der Farbanzahl beherrschen. Ideal ist, wenn eine flexible Reduktion der Dateigröße möglich ist, was ja im Web eine oft notwendige Aktion ist.



Das beliebte Grafikprogramm Paint Shop Pro von Jasc (<http://www.jasc.de>) wurde Mitte 2000 als kostenlose Vollversion in der Version 4.15 freigegeben. Das Programm beinhaltet eine direkte Scanner-Einbindung, über 30 Grafik-Import- und -Export-Formate, eine große Anzahl an Effekten und zahlreiche Farbmanipulations- und -Reduktions-Befehle, was ideal für die Erstellung von Webgrafiken ist.

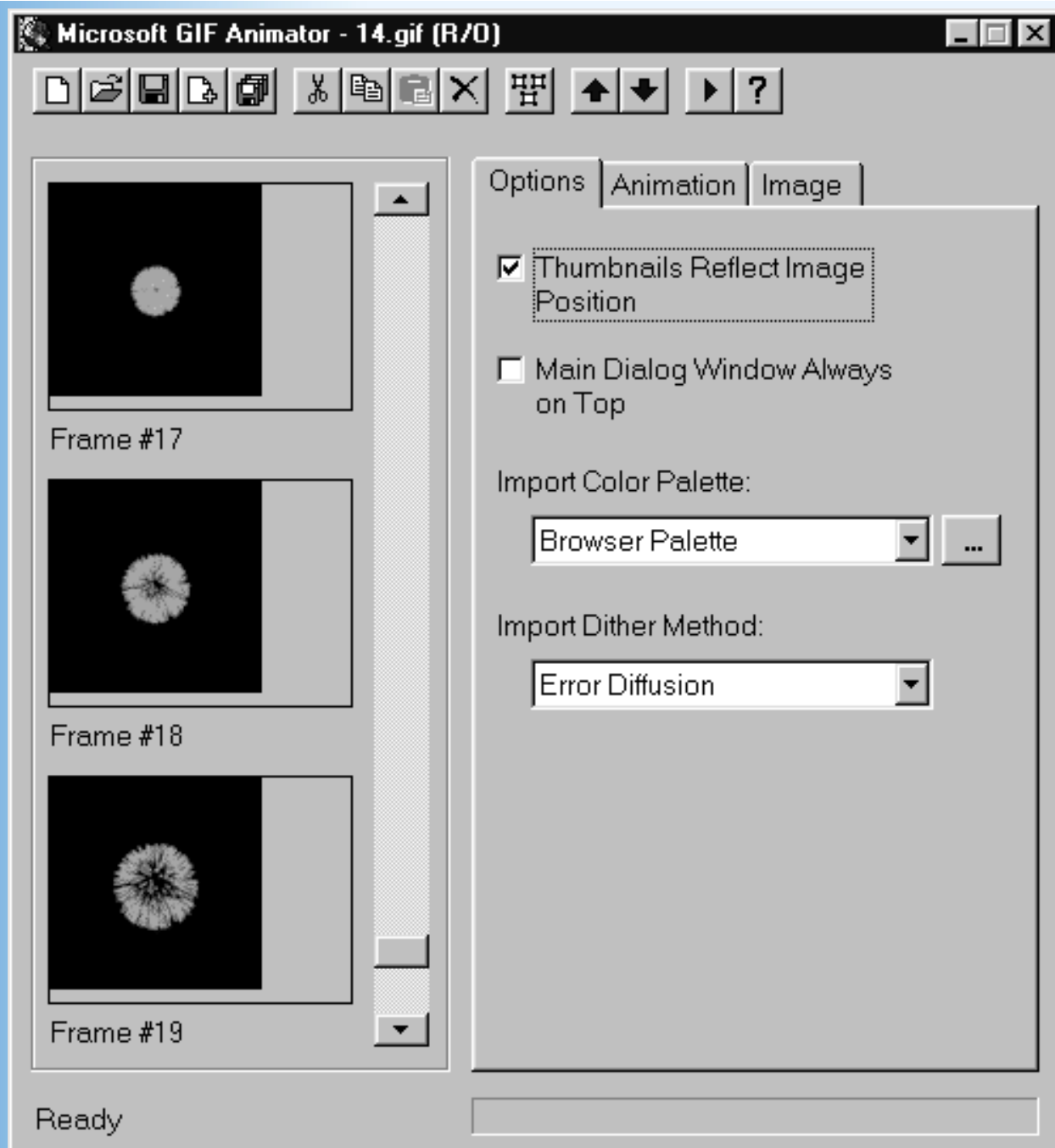


Abbildung 1.22: Der GIF-Animator

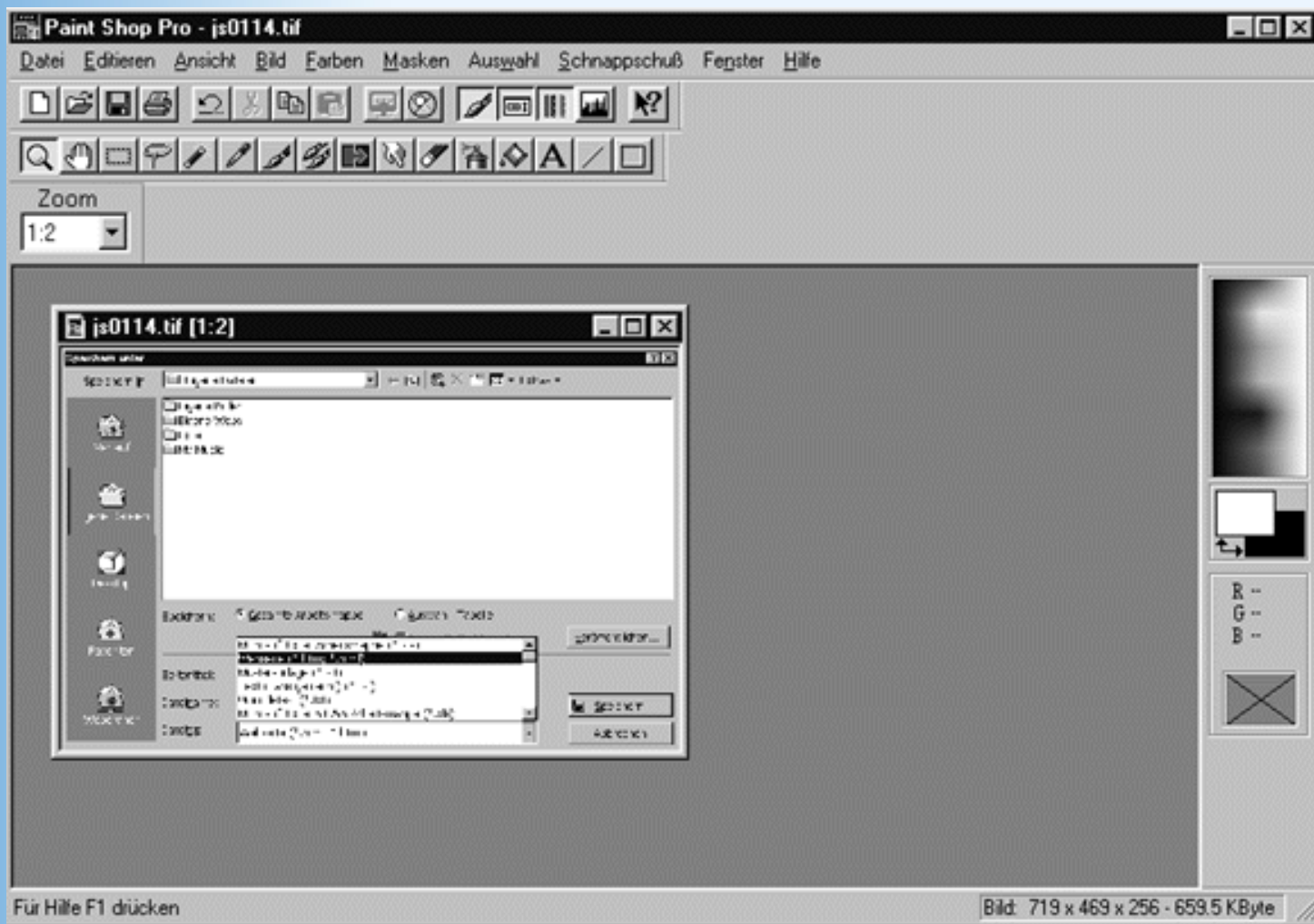


Abbildung 1.23: Die kostenlose Vollversion von Paint Shop Pro

Animierte GIFs gehören im Web zu den Klassikern von Animationen. Viele einfache Animationen lassen sich bereits mit kostenlosen Grafikprogrammen realisieren, zumal animierte GIFs in vielen Fällen bessere Lösungen sind als zusätzliche Techniken voraussetzende Animationen. Zwei interessante Vertreter für Programme zur GIF-Animation sind der kostenlose Microsoft GIF Animator oder das preisgünstige GIF Construction Set von Alchemy Mindworks Inc. (<http://www.mindworkshop.com/alchemy>). Beide Programme erlauben das Zusammensetzen einzelner GIFs zu einer fertigen Animation. Per Maus oder Menü werden die Bilder an die gewünschte Position gebracht sowie Zusatzdaten wie die Wiederholungsrate, der Transparenzmodus, die Bildwechselgeschwindigkeit, die Größe usw. eingegeben.

Wenn sich im Laufe einer Webprojektphase eine Vielzahl von Grafiken ansammeln, ist ein Programm zur Verwaltung, zum Sortieren und für die Vorschau der in einem Projekt/Verzeichnis enthaltenen Grafikdateien sehr nützlich. Die meisten teuren Grafikprogramme bieten das natürlich, aber auch preiswerte Shareware und teilweise sogar Freeware-Programme bieten diese Features. Aus der Masse der Angebote wollen wir nur Grafik Workshop von Alchemy Mindworks Inc. erwähnen (<http://www.mindworkshop.com/alchemy/gww.html>). Das Programm bietet alles zum Anschauen von Grafikdateien in zahlreichen Grafikformaten, eine gute Sortierfunktion und die Möglichkeit, den Bildern einen Kommentar zu verpassen.

Upload-Programme

Neben Grafikprogrammen benötigen Sie unbedingt ein Programm, welches den Upload der fertigen HTML-Seiten samt dazugehöriger Dateien ermöglicht, denn grundsätzlich müssen alle Informationen, Bilder, Scripte etc., die im World Wide Web öffentlich zugänglich gemacht werden sollen, irgendwie von dem lokalen Rechner auf einen über das Internet erreichbaren Webserver geladen werden. Dazu können im einfachsten Fall Upload-Wizards verwendet werden, im allgemeinen Fall FTP-Clients.

Die meisten Internet-Dienstanbieter bieten Webspeichermöglichkeiten für die Webseiten ihrer Kunden an. Die großen Online-Dienste bieten in ihrer Zugangssoftware integrierte Module in Form von Wizards zum Upload an, die dann aber normalerweise auf den Kontakt zu dem jeweiligen Online- Dienst beschränkt sind. Daneben haben sie oft noch andere Schwächen (etwa ungenügender Bedienkomfort bei AOL oder kein direktes Management von Client und Server parallel), weshalb eine zusätzliche Software meist notwendig wird.

Ein recht komfortabler und brauchbarer (und überdies kostenloser) Upload- Wizard für den Kontakt zu beliebigen Servern ist Microsofts Web- Publishing-Assistent. Sie erhalten ihn von den üblichen Microsoft-Seiten. Der Assistent stellt ein einfaches Verfahren für das Übertragen von beliebigen Daten auf einen beliebigen Webserver zur Verfügung. Dies funktioniert damit u.U. auch dann, wenn ein direkter FTP-Kontakt per FTP-Client nicht möglich ist. Der Assistent unterstützt neben dem Upload per FTP-Protokoll auch HTTP Post oder CRS. Das ist beispielsweise für CompuServe-Kunden wichtig, wo kein direkter FTP-Upload möglich ist oder aber auch für AOL-Kunden, die nur bei bestimmten Konstellationen mit einem FTP-Client arbeiten können.



Abbildung 1.24: Microsofts Web-Publishing- Assistent

Grundsätzlich ist die Flexibilität eines Wizards jedoch nicht mit der eines echten FTP-Clients vergleichbar. So ist die Wartung eines etwas umfangreicheren bestehenden Webprojekts auch mit dem Web-Publishing-Assistenten mehr oder weniger eine Zumutung, wenn einzelne Dateien auf dem Server aus dem Projekt gelöscht, verschoben oder umbenannt werden sollen. Entweder muss man alle Dateien pauschal löschen und alles neu hochladen oder in alphabetischer Reihenfolge für jede Datei (Grafik, Webseite, Videodatei, externe Scriptdatei, externe Style Sheetdatei, etc.) bestätigen, ob sie gelöscht werden soll oder nicht. Da kommen leicht dreistellige Dateianzahlen zusammen, wo jedes Mal Löschen oder nicht ausgewählt werden muss. Und zurück gibt es in dem Dialog nicht. Bei Fehlern muss man wieder alles durchlaufen. Wenn die Möglichkeit eines direkten FTP-Kontaktes besteht, sollte auf jeden Fall mit einem solchen FTP-Client gearbeitet werden.

Zwei populäre FTP-Clients sind CuteFTP (<http://www.cuteftp.com>) und WS_FTP95 (<http://www.ipswitch.com>).

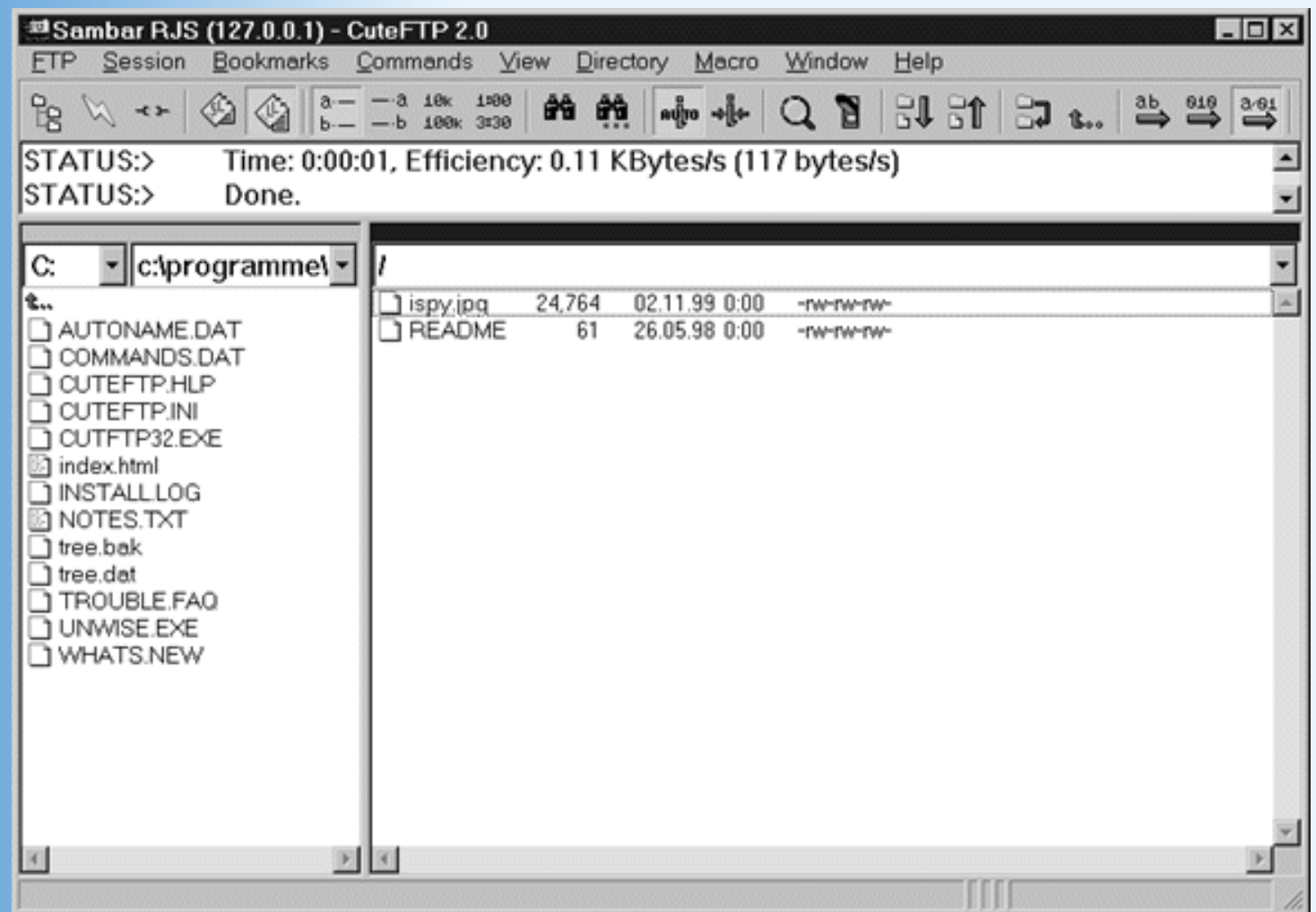


Abbildung 1.25: Der FTP-Client CuteFTP

Beide Clients bieten die Möglichkeit des vom Windows-Dateimanager/- Explorer gewohnten Dateimanagements. Über zwei Fenster werden Client und Server nebeneinander dargestellt. Man hat im Rahmen seiner Rechte sowohl beim Client als auch Server Zugriff auf alle relevanten Daten wie Dateigröße, Datum, Verzeichnisstrukturen usw.

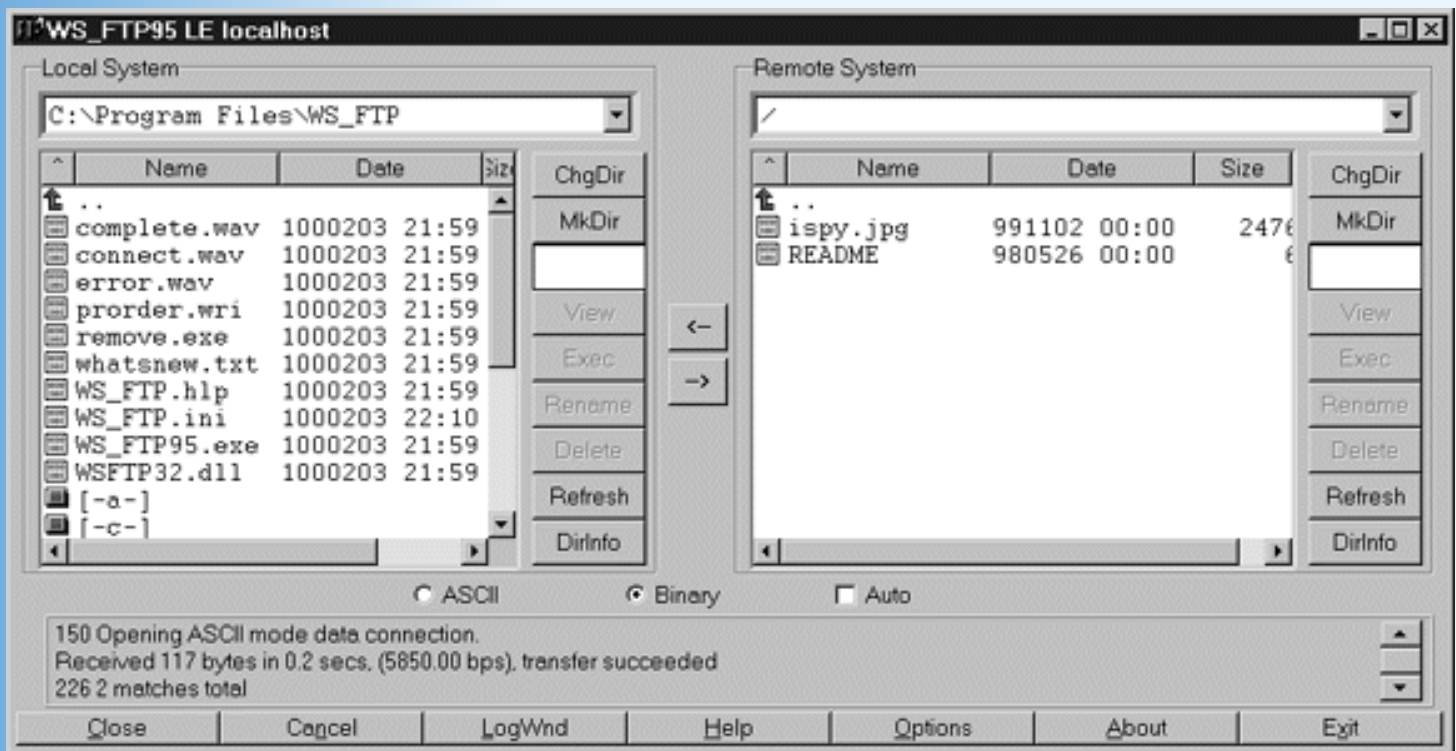


Abbildung 1.26: Der FTP-Client WS_FTP95 - links der Clientrechner, rechts das Serververzeichnis

Der Up- und Download von Dateien funktioniert einfach per Drag&Drop. Ebenso lassen sich auf dem Server im Rahmen der zugeordneten Rechte Verzeichnisse erstellen, Dateien gemeinsam markieren und manipulieren (löschen, verschieben, downloaden) usw. - wie bei der Arbeit auf dem lokalen Rechner.

Auch die Kontaktaufnahme zu einem Server ist über die Clients ein Kinderspiel. IP-Nummer, UserId und Passwort (bei frei zugänglichen Servern auch anonymer Zugang, aber dann funktioniert so gut wie nie ein Upload) müssen eingegeben werden. Ansonsten langen meist die Grundeinstellungen wie automatische Hosterkennung.

Zusammenfassung

Das Internet ist ein gigantisches Netzwerk, welches die unterschiedlichsten Fähigkeiten und Informationen bereitstellt. Sie haben in dieser Lektion einige wichtige Hintergründe zum Internet, seinen Protokollen und seinen verschiedenen Diensten - insbesondere dem World Wide Web - erfahren. Besonders beachtet bei dieser Lektion wurde die Bedeutung von HTML (noch gänzlich ohne auf die Syntax einzugehen) sowie das überall im Web zu findende Interpreter-Prinzip, welches sowohl bei HTML, aber auch bei JavaScript zum Tragen kommt. Des Weiteren stand die Funktion von Browsern (mit einer Vorstellung verschiedener Browser) sowie der im Internet überall zu findende Client-Server-Mechanismus im Mittelpunkt unserer Ausführungen. Daneben sollten Sie eine Vorstellung davon haben, welche Programme Sie im Laufe Ihrer JavaScript-Karriere wirklich benötigen, wo Vor- und Nachteile von bestimmten Programmen liegen und worauf Sie erst einmal verzichten können.

In den nächsten Tagen erfahren Sie Näheres zu HTML und JavaScript und wie Sie Scripte in HTML-Seiten integrieren können.

Fragen und Antworten

Frage:
Ist jeder Browser JavaScript-fähig?

Antwort:

Nein, aber die meisten. Insbesondere neuere Varianten können so gut wie immer JavaScript direkt umsetzen oder - wie der Internet Explorer - zumindest die Sprache verstehen. JavaScript ist damit ein Quasistandard bei Scriptsprachen im Web und hat gegenüber allen anderen Scriptsprachen - etwa VBScript - eine erheblich höhere Verbreitung. Allerdings sollte man immer wieder betonen, dass sich auch in neueren Browsern JavaScript vom Anwender deaktivieren lässt. Außerdem kann man sich nicht darauf verlassen, dass auch die neusten Versionen von JavaScript verstanden werden.

Frage:

Ist Interpretation die einzige Möglichkeit, aus Quelltext lauffähige Arbeitsschritte auf dem Computer zu machen?

Antwort:

Nein, es gibt ebenso einen Vorgang, der Compilierung genannt wird. In diesem Fall wird der Quelltext bereits vom Entwickler eines Programms oder Scripts direkt übersetzt und dann diese in lauffähigen Programmcode übersetzte Datei an den Anwender weitergegeben. Unter Windows sind das in der Regel Dateien mit der Erweiterung .exe. Compilierung hat gegenüber Interpretation einige Vorteile, aber diesen stehen natürlich auch einige Nachteile gegenüber (sonst gäbe es nicht beide Varianten).

Interpretation hat sich als ideale Lösung für den Fall bewährt, wenn unterschiedliche Plattformen zu bedienen sind. Etwa wie sie im Internet mit seiner extrem heterogen Struktur und den unterschiedlichsten dort zusammengeschlossenen Rechnertypen mit ihren unzähligen Betriebssystemen vorkommen. Diese Heterogenität erzwingt gemeinsame Nenner zur Verständigung, und da sind Klartextanweisungen ideal, denn diese werden auf nahezu jeder Plattform verstanden. Solche neutralen Befehle werden in einem gemeinsamen Zeichensatz (ASCII oder neuerdings oft UNICODE) dargestellt und erst auf der jeweiligen Zielpattform (also beim Client) in konkrete Prozessorbefehle übersetzt (interpretiert)²⁵. Das Programm zur Übersetzung der Anweisungen in plattformspezifischen Code läuft auf der jeweiligen Plattform und kennt die konkreten Prozessorbefehle. HTML als auch JavaScript sind Beispiele für so eine Konstellation. Sowohl die HTML als auch JavaScript-Anweisungen bestehen aus Klartext, der auf jeder Rechnerplattform verstanden werden. Über einen Interpreter - den Browser - werden die Anweisungen in die jeweiligen Prozessorbefehle übersetzt und ausgeführt. Ein weiterer Vorteil der Interpretation ist die Reduzierung der Menge der zu übertragenden Daten. Man überlässt dem Anwender einige Freiheiten und hat zudem bereits in den Interpreter einige Funktionalitäten eingebaut, die nicht jedes Mal zu übertragen sind.

Compilierung ist dagegen sinnvoller, wenn direkt für eine Plattform (etwa Windows) entwickelt wird und es auf Performance ankommt. Wenn man ein vollständig in Maschinencode für einen bestimmten Prozessor übersetztes Programm (compiliert) unter einer anderen Plattform laufen lassen möchte, wird dies in der Regel nicht funktionieren, aber das spielt dann ja keine Rolle. Dafür ist es in Hinsicht auf eine Plattform optimiert und muss vor allem nicht zur Laufzeit erst übersetzt werden. Dies verbessert das Laufzeitverhalten gegenüber interpretierten Anwendungen erheblich.

Frage:

Wo können IP-Nummern und DNS-Namen registriert werden?

Antwort:

Das Internet kommt ohne eine obere Instanz aus. Dies bedeutet, es gibt keine Organisation oder Behörde, die im Internet so etwas wie eine gesetzgebende Funktion ausüben kann. Mit eigentlich nur einer Ausnahme - die Vergabe von IP-Nummern respektive zugeordneten DNS-Namen. Dies kann nicht willkürlich erfolgen, sondern muss rein technisch zentral verwaltet werden. Verantwortlich für die eindeutige Vergabe und Verwaltung von DNS-Namen im Internet ist eine Organisation namens InterNIC (<http://www.internic.net>) bzw. deren nationale Ableger wie das Deutsche Network Information Center (DE-NIC - <http://www.denic.de>). Diese Organisation zur Vergabe und Verwaltung von eindeutigen Adressangaben samt ihrer Unterorganisationen sind wohl die einzigen Instanzen im Internet, welche die Möglichkeit haben, verbindliche Vorschriften zu vergeben. Wenn eine Firma, eine Organisation oder auch eine Privatperson eine IP-Nummer für einen Server oder einen DNS-Namen für eine Homepage

beantragen möchte, kann dies direkt bei dem Network Information Center erfolgen. Allerdings wenden sich die meisten Interessenten nicht direkt an diese Organisation, sondern an einen Zwischenhändler, der für (bei Privatpersonen ziemlich kleines) Geld die Beantragung übernimmt²⁶.

Frage:
Hat jeder Rechner im Internet einen DNS-Namen?

Antwort:
Nein. Nur eine IP-Nummer ist notwendig, ein DNS-Name ist optional.

Frage:
Verstehen der Netscape Navigator und der Opera die JavaScript- Konkurrenzsprache VBScript?

Antwort:
Nein. Nicht ohne Zusatzmodule oder gar nicht.

Frage:
Verstehen alle Browser Style Sheets?

Antwort:
Nein. Nur die neueren, und diese verwenden Style-Sheet- Anweisungen sehr unterschiedlich.

Frage:
Unterstützen der Netscape Navigator und der Opera ActixeX-Controls?

Antwort:
Nein²⁷.

Frage:
Sollte man auf Flash-Formate bei Animationen vertrauen?

Antwort:
Nein, denn trotz der recht großen Verbreitung wird die Masse der Anwender keine Flash-Unterstützung installiert haben und viele Anwender akzeptieren auch kein nachträglich installiertes PlugIn. Eine Flash-Animation ist nur unter dem Aspekt »Nice-to-have« zu sehen, darf aber nie eine ausschließliche Informationsquelle sein.

Workshop

Für diesen Workshop benötigen Sie verschiedene Browser. Mindestens sollte eine halbwegs aktuelle Version des Internet Explorer und des Netscape Communicator installiert sein. Wenn es geht, noch eine Version des Opera-Browsers. Sofern Sie noch keine verschiedenen Browser zur Verfügung haben, installieren Sie die fehlenden Browser (jeweils eine neuere Version).



Die parallele Installation von mehreren Browsers verschiedener Hersteller auf einem Rechner sollte mittlerweile kein Problem mehr sein. Es gibt eigentlich nur eine kleine Unschönheit zu beachten - jeder Browser versucht, sich selbst als Standardbrowser für alle Webseiten und einige zugehörige Dateiformate einzurichten. In der Regel wird damit der zuletzt installierte Browser zu Ihrem Standardbrowser (wenn Sie es bei der Installation nicht verneinen). Ein Doppelklick auf eine zugeordnete Datei (normalerweise lokal vorhanden) startet dann diesen Browser und lädt die Datei. Die Folge ist, dass sich vorher installierte Browser beim direkten Start über den Programmaufruf beschweren, sie wären nicht mehr der Standardbrowser und ob

Sie das nicht vielleicht doch nicht lieber ändern wollten. Diese ungefragten, lästigen und eigentlich ziemlich unverschämten Nachfragen können Sie ein für alle Mal deaktivieren, wenn Sie das entsprechende Kontrollfeld zur Erinnerung bei jedem neuen Start im Rückfragefenster ausschalten.

Leider ist es jedoch nicht so einfach, auf einem Rechner mehrere Versionen desselben Browsers zu installieren. In der Regel überschreibt die Installation einer neuen Version die alte Variante. Dieser für den Laien sinnvolle Mechanismus erschwert dem Webseitenersteller die Arbeit erheblich, denn man sollte - wie sich aus den Ausführungen dieses Kapitels ergibt - auch alte Browser zum Testen von Webseiten und Scripten zur Verfügung haben. Da helfen nur verschiedene Rechner oder gegeneinander abgeschottete Partitionen auf einem Rechner. Beachten Sie jedoch, dass die Installation einer Browserversion Schwierigkeiten machen kann, wenn dort bereits eine neuere Abart davon installiert ist. Selbst eine Deinstallation der neueren Variante vor einer Installation der Altversion funktioniert nicht immer. Es können Reste der Neuversion zurückbleiben, die eine Installation einer älteren Variante verhindern (etwa in der Registry).

Ihre Aufgaben in diesem Workshop sollen folgende sein:

- Machen Sie sich mit den Fähigkeiten und den Einstellungsmöglichkeiten der verschiedenen Browser vertraut.
- Starten Sie mehrere Browser parallel und laden Sie dieselben Seiten in die nebeneinander angeordneten Browser. Wenn Sie mit einer Internet-Verbindung arbeiten, können Sie direkt aus dem Internet eine Seite in mehrere Browser laden. Sie müssen nur den gleichen URL in jedem Adressfeld angeben. Versuchen Sie, Unterschiede in den Darstellungen zwischen den verschiedenen Browsern zu finden. Wenn Sie einen Testfall für die unterschiedlichen Darstellungen von Webseiten in Browsern ein Beispiel suchen, besuchen Sie beispielsweise die Internet-Adresse `http://www.rjs.de/html4.htm` und laden sie in die verschiedenen Browser.
- Besorgen Sie sich geeignete Grafikprogramme und zur Unterstützung unserer zukünftigen Erstellung von HTML- und JavaScript-Seiten einen zweckmäßigen Texteditor sowie als Ergänzung einen brauchbaren HTML- Editor wie etwa Phase 5. Sie finden nützliche Freeware-Programme beispielsweise unter `http://www.freeware.de`. Machen Sie sich mit der Programmbedienung vertraut.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Welches ist der meist genutzte Dienst im Internet?
2. Aus welcher Sprache sind Webseiten aufgebaut?
3. Was ist ein Server?
4. Gehören Style Sheets zu HTML?
5. Welchen Style-Sheet-Standard unterstützen die meisten modernen Browser zumindest im Ansatz?
6. Was ist Unicode?
7. Sind JScript und JavaScript identisch?
8. Was ist ein URL?
9. Zu welcher HTML-Version gehören Frames?

10. Wie alt ist das Internet?



Synonym für Firmen, die mit und im Internet Geld verdienen (wollen). Die Abkürzung ergibt sich auf Grund des letzten Teils des DNS-Namens (darauf kommen wir gleich zu sprechen).

Insbesondere viele Politiker und Entscheider der Wirtschaft, von Behörden oder der Justiz outen sich schnell als Internet-Dünnbrettbohrer, die nur nachplappern statt verstehen.

Die Anzahl der freien IP-Nummern geht zur Neige, weshalb eine abwärtskompatible Erweiterung des Adressierungskonzepts auf 128 Bit diskutiert wird.

Es gibt im Internet drei so genannte Topologien, welche die maximale Anzahl der innerhalb eines lokalen Netzwerks gegenüber dem Internet adressierbaren Rechner festlegt. Das Klasse-C-Netz beinhaltet die kleinste Anzahl von IP-Nummern (das letzte Byte der IP-Nummer wird für die interne Adressierung bereitgestellt). Das Klasse-B-Netz verwendet die zwei hinteren Byte für eine Adressierung im lokalen Netz. Das Klasse-A-Netz verwendet für die interne Adressierung die drei letzten Byte der IP-Nummer.

Einer der vielen Fachbegriffe, wo sich Fachleute und Experten der deutschen Rechtschreibung um die korrekte Schreibweise balgen.

Der Defaultport für einen HTTP-Server ist 80, für einen FTP-Server 21.

Vorsicht: Ironie ;-)

Zum Zeitpunkt der Bucherstellung liegt zwar bereits die Version 6.0 in einer Betaversion vor (aus Marketinggründen ist die Versionsnummer 5 einfach übersprungen worden - ein Schachzug gegenüber dem Internet Explorer, der zeitgleich in der Version 5.5 als Betaversion auf den Markt kam), aber wir wollen auf die Verwendung von Betaversionen als Referenzsystem verzichten. Bei Bedarf wird die Version 6 allerdings herangezogen, denn sie wurde gegenüber den Vorgängern erheblich unstrukturiert (sowohl im Design als auch im Programmkern selbst).

Die Möglichkeit der Nutzung einer Flatrate (d.h. Pauschalkosten pro Monat ohne Zeitbeschränkung) kann die Sache jedoch sogar in dem Internet-Entwicklungsland Deutschland ;-) für Privatanwender interessant machen.

Wie beim Communicator gilt, dass es zur Zeit bereits eine Betaversion einer neueren Variante gibt - die Version 5.5, welche jedoch nicht als Referenzsystem verwendet werden soll. Dies gilt umso mehr, als zum Zeitpunkt der Bucherstellung verfügbare Tests wenig sinnvolle Neuerungen, dafür aber umso mehr Fehler der Betaversion (insbesondere bei der Unterstützung von CSS, was das Haupt-Highlight der neuen Version werden soll) dokumentierten.

Teilweise sogar in bedenklicher Perfektion, denn sogar Passwörter und UserIds in Formulareingaben lassen sich bei entsprechender Freigabe durch den Anwender vervollständigen, was Spionage und Missbrauch Tür und Tor öffnet.

Dummerweise in der Windows-Terminologie Favoriten und nicht wie beim Navigator Lesezeichen genannt.

Für die direkte Programmierung in JScript spricht nur die Konstellation, dass man bewusst nur für den Internet Explorer programmieren und zudem die speziellen Features nutzen möchte, die JScript auf Grund seiner Windows- und Internet-Explorer-Optimierung bietet.

Die Version 5.5 geht dahingehend noch weiter und erlaubt beispielsweise die Verlagerung von Menüs aus dem Browserfenster hinaus oder auch die Überlagerung von Frames innerhalb des Anzeigebereichs.

Auch beim Opera-Browser ist während der Erstellung des Buchs eine neue Modifikation - die Version 4.0 - in einer Betavariante verfügbar. Für diese gilt wie beim Navigator und beim Explorer, dass wir auf Betaversionen als Referenzsystem verzichten. Wichtig ist jedoch, dass erst die Version 4 Unterstützung für JavaScript 1.3 bietet.

Die Hardwareanforderungen sind mit einem 386SX mit 8Mbyte Arbeitsspeicher wirklich sehr niedrig.

Kleine Anmerkung am Rande: Interessant, dass mittlerweile das Weglassen oder die Möglichkeit des Abschaltens von Features als Highlight gewertet wird ;-). Das sollten sich Programmhersteller mal zu Herzen nehmen.

Was in der verfügbaren Betaversion noch nicht funktionierte.

Noch nicht in der Betaversion.

Etwa gibt es in HTML einen Vorgang, den man Maskieren von Zeichen nennt. Dabei werden landesspezifische Zeichen (etwa die deutschen Umlaute ä, ö, ü oder das ß) in ein Format gebracht, damit sie international verstanden werden. Dies erfolgt über die Eingabe von einer längeren Zeichenfolge, die recht kompliziert ist und in einem reinen Editor bei jeder Verwendung eines solchen Zeichens eingegeben werden muss.

In Bezug auf Dateigröße und damit Übertragungszeiten sind die Dateien sogar grottenschlecht. Beachten Sie dazu den Tag 7, an dem wir dazu einige Experimente machen.

Was WYSIWYG beim Ersteller zu einer sinnvollen Angelegenheit für den Anwender machen könnte.

Umfangreich ist natürlich sehr relativ, aber man sollte sich vor Augen halten, dass bei großen Firmen tausende bis hunderttausende von Webseiten mit bis in die Millionen gehenden Bildern und anderen Dateien zusammenkommen können. Aber auch kleiner Sites sprengen oft die 100-Webseiten-Grenze.

Breite Unterstützung im Vergleich zu anderen Formaten - die Mehrzahl der Anwender wird es dennoch nicht verwenden können, und auf die Unterstützung des Flash-Formats kann man sich auf keinen Fall verlassen.

Mit dem Nachteil, dass Befehle und Anweisungen in einem Klartext-Script oder -Programm recht leicht durch einen Anwender zu lesen sind (wenn er sich mit der verwendeten Sprache auskennt), da Interpretercode kaum zu schützen ist.

In den meisten Computerzeitschriften liegt permanent Werbung von solchen Anbietern bei.

Gott sei Dank.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 1

Tag 2

Das Wesen von JavaScript

Nach dem doch sehr grundlegenden und theoretischen ersten Tag wollen wir nun zu JavaScript als Hauptthema kommen. JavaScript ist wie bereits angedeutet die ideale Ergänzung zu HTML, wenn es gilt, statische HTML- Seiten zu dynamisieren, den umgebenden Browser zu programmieren oder auf Objekte vielfältiger Natur zugreifen zu können. Wir wollen am zweiten Tag Folgendes behandeln:

- Was ist JavaScript und was kann man damit machen?
- Was sind ergänzende Techniken wie XML, XSL, Perl oder CGI?
- Einige einfache Beispiele.
- Server-seitige Techniken versus Client-seitige Techniken.
- JavaScript versus HTML.
- JavaScript versus VBScript.
- JavaScript versus Java.
- JavaScript und CGI.

Was ist JavaScript?

Als das WWW entstand, gab es nur HTML, um Webseiten zu beschreiben. HTML ist das Rückgrat des WWW und es wird wohl auch in Zukunft so bleiben. HTML ist nicht das ganze Web, aber ohne HTML gibt es überhaupt kein Web. Keine (!) der anderen Technologien (Grafiken, Videos, Sounds, Scripte, ActiveX-Controls, Java-Applets usw.) ist notwendig im Web. Das Web an sich funktioniert - wenn es sein muss - auch nur mit reinem HTML. Jede der anderen Technologien benötigt HTML, um sich darin zu verankern. Ob es ein Script in einer Webseite ist - das Script wird Bestandteil einer HTML-Seite sein oder sich zumindest darin verankern. Ob es ein Java-Applet ist - es ist im Web nur im Rahmen einer Referenz in einer HTML-Seite lauffähig. Analog verhält es sich mit anderen Techniken.

Ein Vergleich aus der realen Welt kann die Bedeutung von HTML verdeutlichen - ein Vergleich mit einem Haus. HTML ist dabei das Mauerwerk samt Fundament und Dach. Ohne einen Verputz außen und Tapeten innen ist das Haus zwar recht hässlich, aber es hält den Regen ab, schützt vor Kälte, man kann Dinge einräumen, es lässt sich abschließen und was so ein unausgestattetes Haus noch so leistet. Grafiken samt ihrer Spielarten könnte man mit dem Verputz, den Tapeten und auch den Bildern an der Wand vergleichen. Ohne das Mauerwerk sinnlos, aber darauf verankert eine ästhetische Verfeinerung. Sonstige Multimedia- Techniken wie Sound und Animationen stehen für Radios, Videos und Fernsehgeräte. Scripte und aktive Programme wie Java-Applets oder ActiveX-Controls stehen in unserem Vergleich beispielsweise für Zusatztechniken wie Telefon, Fax oder Computer, aber auch allgemeine Bequemlichkeiten wie elektrische Garagentüröffner, Kühlschränke, Geschirrspüler, Waschmaschinen, Kaffeemaschinen¹ etc. Alles Dinge, die das Wohnen in einem Haus komfortabel machen, die Funktionalität erweitern, es bequemer und besser machen. Aber allesamt sind ohne das Haus, worin sie eingebaut werden, woher sie wichtige Dinge wie Strom beziehen, absolut nutzlos.

Genauso würde das Web viel an Faszination und Bequemlichkeit verlieren, wenn man auf ergänzende Techniken verzichten würde. Die Möglichkeiten von reinem HTML stoßen sehr bald an Grenzen und werden mehr und mehr durch verschiedene Zusatztechnologien erweitert. Teilweise basieren diese Erweiterungstechnologien direkt auf HTML, etwa Style Sheets.



Auch XML (Extensible Markup Language) ist ein Begriff, der in der letzten Zeit oft in Zusammenhang mit HTML genannt wird. Genau genommen hört man immer öfter, dass HTML von XML in Kürze abgelöst werden soll. XML hat wie HTML seine Wurzeln in SGML, nur eine andere Aufgabe. Streng genommen kann XML all das, was HTML auch kann, und mehr, aber das ist beileibe nicht alles. XML als Erweiterung von HTML zu sehen, fasst das Potential von XML nicht richtig.

XML ist eine so genannte Metasprache, mit deren Hilfe eine eigene, neue Dokumentenbeschreibungssprache definiert werden kann. Auch XSL (Extensible Stylesheet Language) ist eine verwandte in Verbindung zu HTML zu sehende Erweiterung. Im Grunde ist XSL das für XML, was CSS für HTML ist - eine Style-Sheet-Sprache, um visuelle Angaben in einem Dokument umzusetzen (dies nennt man eine physische Auszeichnung). XML kümmert sich nur um inhaltliche und strukturelle Definitionen eines Dokuments - eine so genannte semantische Auszeichnung.

Die Behauptung, dass XML und XSL HTML bald ablösen werden, muss trotz der Leistungsfähigkeit kritisch hinterfragt werden. Mit beiden Techniken lässt sich die Leistungsfähigkeit von HTML zwar erheblich erweitern, aber XML und XSL sind nicht ganz so einfach zu lernen. Zudem lässt die Unterstützung von XML/XSL durch geeignete Interpreterprogramme ziemlich zu wünschen (in den neusten Browser-Versionen von Microsoft und Netscape sind erste Ansätze vorhanden). Es scheint so, dass XML und XSL zwei der vielen Web-Technologien sind, um die erst viel Wind gemacht wird, die dann aber trotz ihrer unbestrittenen Vorteile nicht so recht aus den Startlöchern kommen. Und selbst wenn - sie werden keine direkte Ablösung von HTML sein, sondern nur eine Ergänzung, deren Haupteinsatzgebiet wahrscheinlich die Business-to-Business-Anwendungen im Rahmen des E-Commerce (also dem elektronischen Datenaustausch zwischen Unternehmen zur Abwicklung von Geschäftsvorgängen) sein werden.

Eine der wichtigsten Erweiterungen von HTML sind die Scriptsprachen, von denen es zahlreiche Versionen gibt. Beispielsweise Perl (Practical Extraction and Report Language), VBScript oder eben JavaScript. Dabei muss man zwischen Server-seitigen und Client-seitigen Scriptsprachen unterscheiden, worauf wir etwas weiter unten genauer noch eingehen. Das im Internet praktizierte Client-Server-Prinzip erlaubt die Aufteilung von Funktionalität auf beide beteiligten Partner. Sowohl der Client als auch der Server können über Scripte programmiert werden. Es existieren sowohl auf der Clientseite wie auf Serverseite die unterschiedlichsten Scriptsprachen. Viele davon lassen sich sowohl auf Clientseite wie auf Serverseite einsetzen oder haben zumindest definierte Schnittstellen auf beiden Seiten. Auf den meisten Servern stehen Perl-Interpreter zur Verfügung. Auf Clientseite werden im Web im Wesentlichen zwei (genau genommen drei) Sprachen unterstützt. Das sind eben JavaScript bzw. JScript sowie VBScript - letztere weitgehend begrenzt auf den Internet Explorer.

Bei den Scriptsprachen handelt es sich immer um Interpretersprachen, die von irgendeinem Interpreter interpretiert werden müssen. Diesen Interpreter stellt das umfassende Programm bereit, in das Scripte geladen werden. Im Fall von JavaScripten beim Anwender ist das der Browser, der einen internen JavaScript-Interpreter (bzw. einen JScript-Interpreter im Fall vom Internet Explorer) bereitstellt.

JavaScript

JavaScript ist eine an Java angelehnte, offene, plattformunabhängige Scriptsprache. Entwickelt wurde JavaScript von Netscape in einer Kooperation mit Sun Microsystem sowie einigen weiteren Partnern. Ursprünglicher Einsatzzweck von JavaScript war die Erweiterung des Navigator 2.0 um Steuerungsmöglichkeiten für diesen Browser aus einer Webseite heraus. Diese mit dem Navigator 2.0 1995 erstmals veröffentlichte Scriptsprache wurde zuerst Mocha, dann LiveScript und letztendlich JavaScript (um eine Anlehnung an Java zu verdeutlichen) genannt.

JavaScript erlaubt die Notation von Scripten direkt in eine HTML-Seite (oder zumindest die Referenz darauf - was das genau bedeutet, folgt in späteren Lektionen). Diese werden dann von einem Interpreter ausgeführt, der von dem Browser automatisch aufgerufen wird, wenn er den entsprechenden Befehl dazu in der Webseite vorfindet. JavaScript- Anweisungen sind damit eine reine Klartexterweiterung des HTML-Codes und nur als eingebundener Bestandteil eines HTML-Gerüsts zu verwenden.

JavaScript ist seit seiner Entstehung durch mehrere Versionszyklen gegangen. Derzeit gibt es JavaScript als Core JavaScript (also purer Syntaxstandard) bereits in der Version 1.4, aber die ist noch nicht als Standard im Web akzeptiert und von Browsern unterstützt, sodass die Version 1.3 als relativ weit unterstützte Fassung unsere Referenzversion sein soll. Auf Serverseite gibt es derzeit JavaScript in der Version 1.5, wobei man festhalten sollte, dass serverseitiges JavaScript sich in einigen Details von clientseitigem unterscheidet. Das soll für uns aber keine Rolle spielen - wir bleiben sowieso auf Clientseite.



Auch die Version 1.3 ist noch nicht in allen Browsern integriert. Beispielsweise unterstützt der Opera-Browser bis zur Version 3.6 (die bis Mitte 2000 aktuelle Version) sogar nur JavaScript 1.1. Dies ist aber für Einsteiger kein sonderliches Problem, denn bereits die Leistungsfähigkeit von JavaScript 1.1 erlaubt, viele interessante Funktionalitäten zu programmieren. Im Anhang finden Sie eine Auflistung, welcher Browser welche Version von JavaScript bzw. JScript unterstützt. Grundsätzlich muss zudem beachtet werden, dass für nahezu alle Sprachzyklen von JavaScript kein einziger Browser den offiziellen Standard vollständig unterstützt! Es gibt immer bestimmte Konstellationen, wo es Probleme geben kann. Tests auf verschiedenen Browser sind deshalb leider unumgänglich - insbesondere bei Anweisungen ab JavaScript 1.2.

VBScript

Eine zeitweise als großer Gegenspieler von JavaScript gefeierte Scriptsprache ist die von Microsoft aus Visual Basic entwickelte Sprache VBScript. Der Ursprung in Visual Basic sollte von vornherein vielen Einsteigern das Erlernen von VBScript erleichtern, denn Basic- bzw. Visual- Basic-Grundkenntnisse sind weit verbreitet. VBScript wurde gegenüber seinem Vorfahren erheblich vereinfacht, gilt aber durch seine weitgehende Beschränkung auf das Windows-Betriebssystem mit INTEL-basierenden Prozessoren und dem Internet Explorer als Clientprogramm und damit einhergehende Optimierung im Hinblick auf die WINTEL-Plattform dort als leistungsfähiger als JavaScript.

Die Optimierung der plattformabhängigen Leistungsfähigkeit scheint jedoch nicht den gewünschten Erfolg mit sich zu bringen. Die Akzeptanz von VBScript ist beileibe nicht so groß wie die von JavaScript. Dies kann man auf mehrere Gründe zurückführen:

- Wenn man VBScript einsetzt, muss man sich klar sein, dass die wegen der Optimierung auf die WINTEL-Plattform mit dem Internet Explorer als Browser reduzierte Kundschaft ein Alternativprojekt erzwingt, wenn man nicht einen großen Teil der Besucher verprellen will². Diesen erheblichen Mehraufwand leisten sich nur wenige Entwickler.
- VBScript ist in relativ kurzer Zeit bei Microsoft als eine sehr plattformnahe Sprache entwickelt worden. Während JavaScript organisch gewachsen ist und sich über Jahre hinweg in der Praxis als sicher und ungefährlich für den Anwender bewährt hat, beinhaltet VBScript zahlreiche Trickmöglichkeiten, aber auch Missbrauchmöglichkeiten und Sicherheitslücken (gerade in Verbindung mit den äußerst umstrittenen ActiveX-Controls). Von JavaScript werden insbesondere ActiveX-Controls nicht bzw. kaum unterstützt, was von vielen Webprogrammierern und Anwendern - je nach Standpunkt - sogar als ein gravierender Vorteil gesehen wird.
- Ein Argument gegen die allgemeine Akzeptanz von VBScript ist dessen Erfinder. Ob gerechtfertigt oder nicht - Microsoft ist im Internet unbeliebt. An diesem Unternehmen scheiden sich auf jeden Fall die Geister. Sei es Neid, sei es berechtigte Kritik - sowohl die vielen Konkurrenten (besonders im Internet agierende Firmen), aber auch viele Anwender lehnen alles ab, was von Microsoft kommt. Ob gut oder schlecht - das spielt dabei oft keine Rolle. Gerade für viele Anwender gilt, dass sie einerseits zu der Verwendung von Microsoft-Produkten wie Windows und MS Office gezwungen werden, andererseits aber, wenn es irgend geht, Alternativen suchen (die Erfolge von Linux und BeOS, aber auch dem Navigator oder Opera belegen solche Versuche).
- Trotz der Optimierung sind sich JavaScript und VBScript im Leistungsumfang und der Zielrichtung recht ähnlich. Zumindest in allen für eine Scriptsprache sinnvollen Bereichen. VBScript ist teilweise auf PCs bereits zu mächtig geworden, als dass es für eine Scriptsprache noch sinnvoll wäre. Viele Experten kreiden mit Scripten aufgepeppten Webseiten an, dass normalerweise keine sinnvollen Aktionen programmiert, sondern nur überflüssige und lästige Spielereien die Seite verschandeln³. Mit seinen Möglichkeiten verleitet VBScript natürlich noch mehr zu solchen Spielereien. Auf der anderen Seite ist VBScript dann doch wieder gegenüber Programmiersprachen wie Java so stark beschränkt, dass wirklich über die Scriptfähigkeiten von JavaScript hinausgehende ernsthafte Aktionen nicht möglich sind. Die Mehrleistung von VBScript gegenüber JavaScript hängt also irgendwie zwischen allen Stühlen. Scriptsprachen können sich einfach nicht anmaßen, all das zu können, was eine vollständige Programmiersprache zu leisten vermag. Warum auch?
- JavaScript ist in keiner Weise komplizierter zu lernen als VBScript und ist bereits länger auf dem Markt.

JavaScript und Java

JavaScript wird als eine an Java angelehnte, offene, plattformunabhängige Scriptsprache bezeichnet. Dies und der im Namen vorkommende Begriff Java führt dazu, dass JavaScript oft mit Java verwechselt oder als Ersatz gesehen wird. Dies ist allerdings falsch. Java ist viel leistungsfähiger als jede Scriptsprache (und die meisten Programmiersprachen), jedoch auch viel komplizierter zu lernen. Für Java gibt es ganz andere Anwendungen als für Scriptsprachen wie JavaScript.

Java ist eine weitgehend betriebssystem- und hardwareunabhängige, objektorientierte Programmiersprache beziehungsweise ganze Entwicklungsplattform, die zwar einen Schwerpunkt im Bereich Internet/ Intranet hat (mit den Java-Applets), den Haupteinsatzzweck aber im Bereich der datenbasierenden Konsumerelektronik finden soll (die Programmierung von allen Geräten des täglichen Lebens, die mit einem Prozessor ausgestattet sind oder in Zukunft ausgestattet werden können - vom Videorecorder bis zum Kühlschrank). Daneben gilt Java als sehr sicher und geeignet für Anwendungen, die in einer heterogenen Netzwerkumgebung ablaufen sollen. Viele Aktionen im Bereich des Online-Bankings, des elektronischen Zahlungsverkehrs oder sonstiger sensibler Datenkommunikation beruhen auf Java.

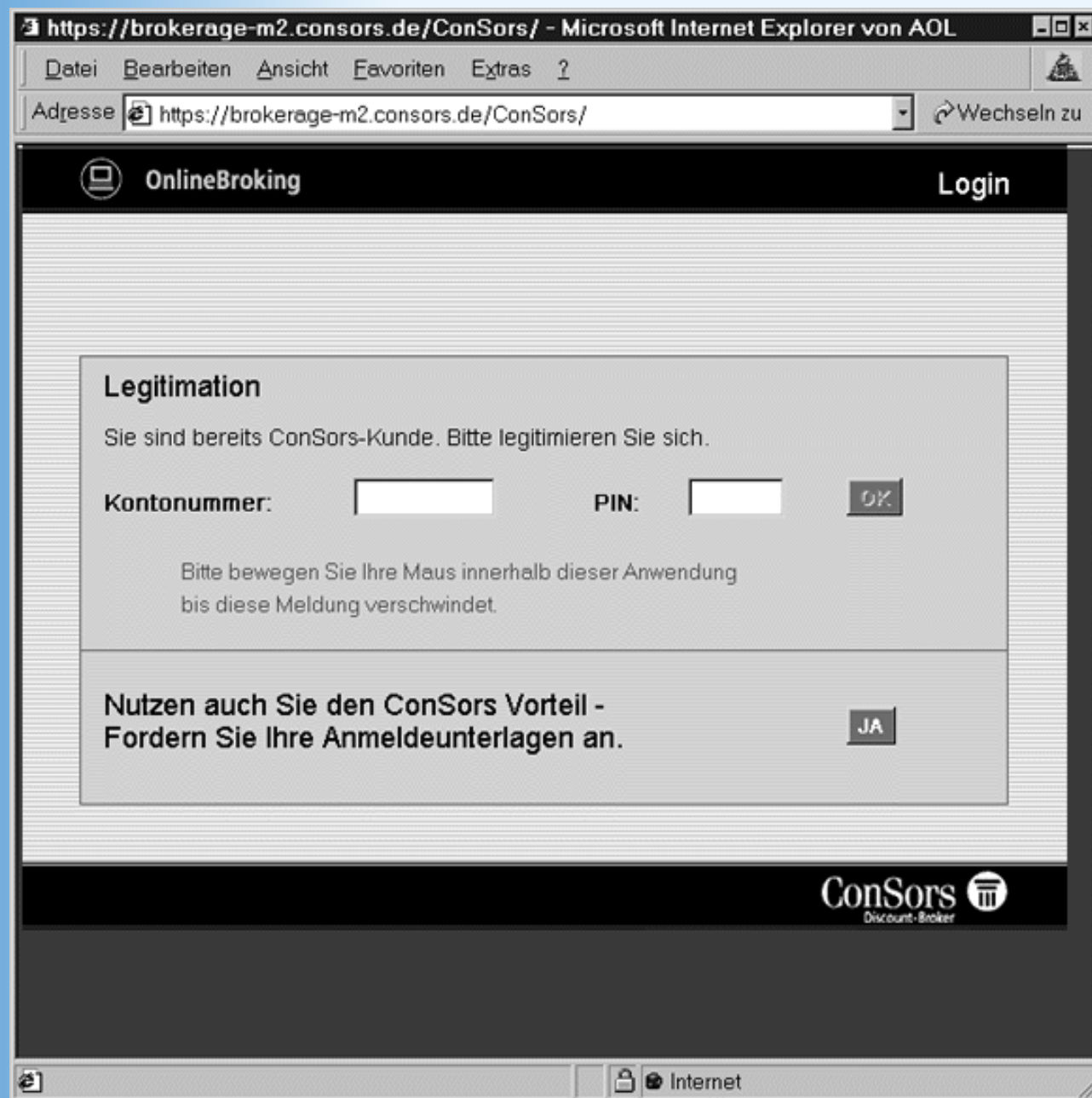


Abbildung 2.1: Ein Java-Applet, welches im Bereich Online-Broking verwendet wird

Mit Java lassen sich vollständige Programme entwickeln (im Gegensatz zu Scripten, die nur bereits fertige Programme steuern und nutzen). Sowohl in Form von Java-Applets, die mit HTML-Anweisungen in eine Webseite integriert werden und auch nur dort lauffähig sind als auch als vollständig eigenständige Applikationen, die im Gegensatz zu den Applets keinen Container wie einen Browser mehr benötigen.

Entwickelt wurde Java von Sun Microsystems als eine Sprache, die von der Syntax C/C++ sehr ähnlich ist. Java-Quellcode wird durch den Java- Compiler in ein plattformunabhängiges Bytecode-Format übersetzt, das dann auf der Zielmaschine durch die so genannte Java Virtual Machine (JVM) interpretiert wird.



Die Java Virtual Machine ist eine Interpreterschnittstelle zwischen dem Java-Bytecode auf der einen und der zugrunde liegenden Hardwareplattform und dem Betriebssystem auf der anderen Seite. Die JVM muss auf jedem Rechner bzw. programmierbaren Gerät vorhanden sein, welcher Java-Anwendungen ausführen möchte. Sie realisiert die Umsetzung des Bytecodes in ausführbare Prozessorbefehle. Die Java Virtual Machine kennt die Spezifika der jeweiligen Hardware, auf der sie läuft, und kann so auf die Eigenheiten eines bestimmten Prozessorstyps und der restlichen Hardware Rücksicht nehmen.

Die offizielle Definition von Sun Microsystems für Java lautet folgendermaßen:

Java: eine einfache, objektorientierte, dezentrale, interpretierte, stabil laufende, sichere, architekturneutrale, portierbare und dynamische Sprache, die Hochgeschwindigkeits-Anwendungen und Multithreading unterstützt.

Was kann man mit JavaScript tun?

JavaScript erweitert eine Webseite auf vielfältigste Art und Weise. Insbesondere kann der Besucher einer Seite in einen Dialog mit der Webseite einbezogen werden, statt sich die Seite nur anzuschauen. Dies können bewusste Dialoge sein - etwa Rückfragen nach einem Passwort oder nach dem Namen des Anwenders, aber auch die Kontrolle von Eingabe in einem Webformular. Aber auch Hintergrundaktionen, die beispielsweise eine Art Dialog mit dem Browser des Anwenders oder seiner Plattform führen - etwa die automatische Abfrage des Browserstyps und entsprechende Reaktion darauf, das Lesen und Schreiben von Cookies, die Abfrage der History des Browsers, die Abfrage der Systemzeit usw. Mit JavaScript kann man auch Berechnungen durchführen, beispielsweise eine Umrechnung zwischen DM und EURO, die Summierung der Gesamtkosten einer Bestellung einzelner Artikel oder die Ermittlung der optimalen Größe eines anzuzeigenden Bildes. Grundsätzlich lassen sich mit JavaScript auch Vorgänge automatisieren, welche ein Anwender sonst in mehreren Schritten manuell durchführen müsste (etwa das gleichzeitige Laden von mehreren Webseiten auf einen Mausklick, was beispielsweise bei einer Framestruktur oft sinnvoll ist).

JavaScript beinhaltet einen internen Befehlssatz, eine Syntax und eine definierte Struktur, mit denen diverse Objekte rund um den Browser kontrolliert werden können. Ein kontrollierbares Objekt ist beispielsweise der Browser selbst. Aber auch andere Objekte lassen sich mit Scripten beeinflussen. JavaScript kann beispielsweise zur Kontrolle von Bestandteilen einer Webseite, Java-Applets und anderen Objekten verwendet werden. Eigentlich ist es so, dass Client-seitige Scriptsprachen im Wesentlichen den relativ dummen Browser intelligenter machen. In modernen Browsern ist die Nutzung einer oder mehrerer Scriptsprachen standardmäßig integriert. So gut wie immer wird JavaScript verstanden.

Grundsätzlich kann man die Hauptfunktion von Scriptsprachen, die im Rahmen einer Webseite verwendet werden, so beschreiben: Sie erlauben eine Verlagerung von solcher Funktionalität zu dem Browser, die bei einer reinen HTML-Seite von einem Anwender mehrere zusätzliche Aktionen erfordern würde oder auf dem Server abgearbeitet werden müsste. Solche Client-seitigen Scripte erlauben einem Webbrowser, auf eine intelligente Weise mit Situationen, die sonst eine Aktion auf dem Webserver erforderlich machen würden, umzugehen, ohne erst eine Rückfrage beim Server durchzuführen. Wesentlicher Effekt ist damit eine Beschleunigung vieler Vorgänge, da der Browser wie gesagt keine Anfrage an den Server schicken und die Antwort nicht abwarten muss. Um diesen Positiveffekt genauer zu verstehen, muss man sich das Alternativverfahren unter reinem HTML verdeutlichen. Der konventionelle Weg einer eventuell erforderlichen Kommunikation zwischen dem Besucher einer Webseite und dem Server, wo die Seite gespeichert ist, erfolgt meist über eine Technik, die CGI heißt.



CGI ist die Abkürzung für »Common Gateway Interface«, was übersetzt »Allgemeine Zugangsschnittstelle« bedeutet und einen Prozess zur Datenkommunikation auf einem Server beschreibt. CGI ist eine Schnittstelle für einen Client zu einem Serverprogramm, über welche die Übergabe von Parameterwerten an Programme oder Scripte auf einem Server erfolgen kann. Dort werden die übergebenen Werte verarbeitet und bei Bedarf wird eine Antwort (meist ein HTML-Dokument) an den Client generiert. CGI ist keine Programmiersprache, sondern ein Protokoll zur plattformübergreifenden Standardisierung von so einem Datenaustausch. CGI-Scripte können in jeder Sprache erstellt werden, die Scripte erstellen kann und von der auf dem jeweiligen Server ein passender Interpreter vorhanden ist. Die meistverbreitete Sprache zur Erstellung von Scripten ist Perl (die Sprachspezifikation ist unter <http://www.perl.com> zu finden), aber auch nahezu alle anderen Sprachen eignen sich, wenn sie Scripte erstellen können und auf dem Zielserver ein passender Interpreter vorhanden ist (dazu zählt auch JavaScript).

Allgemein stellt CGI einen Mechanismus zur Verfügung, bei dem auf dem Client erfasste Daten (beispielsweise über ein HTML-Formular) zu einem Server geschickt werden können. Dieser leitet diese Daten zu einem

Server-seitigen Programm/Script weiter, wo sie verarbeitet werden. Anschließend geht die Antwort zurück zum Client.

Ein klassischer Prozess über das Client-Server-Prinzip mittels CGI und einem Browser als Clientprogramm läuft nach folgendem Muster ab:

1. Aktionen, die im Rahmen einer im Browser geladenen Webseite ausgelöst werden (etwa der Klick auf eine Schaltfläche in einer Webseite im Rahmen eines Formulars), müssen ungeprüft zu dem Server zurückgeschickt werden.
2. Auf dem Server werden sie verarbeitet.
3. Anschließend wird das daraus ermittelte Ergebnis als Antwort zu dem Client zurückgeschickt.

So eine Vorgehensweise hat zahlreiche Nachteile.

- Das Laufzeitverhalten ist schlecht. Bis eine Antwort wieder beim Client ist, kann viel Zeit vergehen.
- Belastung der Kommunikationswege.
- Belastung des Servers bei gleichzeitigem Brachliegen der Client- Ressourcen.

Gerade bei fehlerhaften Daten wird so viel Zeit und Kapazität verbraucht. Ein Beispiel ist etwa ein Formular, in dem die Eingabe eines numerischen Wertes (etwa Geldbetrages) erfolgen soll. Wenn nun ein Anwender dort statt Zahlen Buchstaben einträgt, hat man unter HTML keinerlei Kontrollmöglichkeit. Die fehlerhaften Daten werden ungeprüft zum Server geschickt, der leitet sie weiter an seine zugeordnete Prüfungsroutine und erst das Server-seitige Programm bemerkt den Fehler. Es generiert eine Fehlermeldung, die dann als Antwort zurück zum Browser geschickt wird. Keine ideale Vorgehensweise - insbesondere, wenn die ganze Kommunikation mehrfach rund um die Welt geschickt werden muss.

Sehr sinnvoll wäre es, wenn der Fehler bereits vor dem Abschicken der Daten beim Client bemerkt werden würde. Solche Funktionalität, die auch ohne Serverkontakt funktioniert, kann mit JavaScript auf den Client übertragen werden. Etwa wie in unserem Beispiel eine Typüberprüfung eines Eingabefeldes.

Erste JavaScript-Beispiele

So langsam ist es Zeit, einige JavaScript-Beispiele durchzuspielen. Dabei wollen wir bewusst auf die exakte Erklärung der genaueren Hintergründe verzichten, um Sie nicht noch länger auf die - theoretische - Folter spannen zu müssen. Folgen Sie einfach den Anweisungen und tippen Sie den jeweiligen Quelltext der Beispiele ab.

Starten Sie als Erstes einen Editor. Dies kann sogar der im Windows- Zubehör mitgelieferte Editor Notepad sein (Sie finden ihn in der Regel unter dem Startmenü, **Programme, Zubehör**).



Eine gute und sicher bequemere Alternative für einen reinen Texteditor ist beispielsweise der Editor Super NoteTab, den Sie als Freeware unter <http://www.unige.ch/sciences/terre/geologie/fookes/> finden.

Beispiel 1:

Geben Sie im Editor den nachfolgenden Quelltext ein (beachten Sie bitte unbedingt Groß- und Kleinschreibung - auch wenn es teilweise nicht relevant ist):



Bitte trennen Sie bei den Beispielen in Hochkommata eingeschlossene Texte - so genannte Strings oder Zeichenketten - nicht durch Zeilenumbruch. Auch wenn dies im abgedruckten Quelltext aus Satzgründen so notwendig ist - im Editor sollten Sie das vermeiden. Dies gilt für alle Quelltexte im Buch.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function schreibeSeite()
{
var datum = new Date();
```

```

var tag = datum.getDate();
var monat = datum.getMonth() + 1;
var jahr = 2000 + (datum.getYear()%100);
var name = prompt(
"Geben Sie bitte Ihren Namen ein", "");
document.write('<HTML>');
document.write('');
document.write('<HEAD>');
document.write(
'<title>Meine erste JavaScript-Homepage</title>');
document.write('</HEAD>');
document.write('<BODY>');
document.write(
'<H1 align=center>Herzlich Willkommen</H1>');
document.write('<H3 align=center>');
document.write(name);
document.write('</H3>');
document.write('<H4 align=center>');
document.write('Heute, am ');
document.write(tag);
document.write('.');
document.write(monat);
document.write('.');
document.write(jahr);
document.write(' wartet auf Sie hier wieder ein spannendes Programm</H4>');
document.write('</BODY>');
document.write('</HTML>');
}
</SCRIPT>
</HEAD>
<BODY onLoad="schreibeSeite()">
</BODY>
</HTML>

```

Speichern Sie die Datei unter dem Namen ErsteSeite.htm und starten Sie diese dann mit einem Doppelklick darauf in ihren Standardbrowser (oder eine Ihnen sonst genehme Art).

Das Beispiel (wie auch die folgenden) zeigt die dynamische Erstellung einer Webseite mittels JavaScript, in der Informationen berücksichtigt werden, die erst zum Zeitpunkt des Ladens der Seite bekannt sein können. Das sind in diesem Fall das aktuelle Datum und der Name, mit dem sich ein Besucher vor dem Laden der Webseite identifiziert. Diese Informationen werden in Variablen zwischengespeichert. Zuerst sollte ein Eingabefenster erscheinen, in dem der Anwender aufgefordert wird, seinen Namen einzugeben.



Variablen gehören zu den absolut grundlegenden Elementen einer jeden Programmier- oder Scriptsprache. Ohne Variablen kann man nicht - sinnvoll - programmieren. Wer bereits in irgendeiner Sprache Programme oder Scripte erstellt hat, wird sie kennen. Aber auch aus der Schulmathematik dürfte vielen Lesern zumindest der Begriff geläufig sein.

Technisch gesehen bezeichnen Variablen Stellen im Hauptspeicher, in denen irgendwelche Werte temporär gespeichert werden können. Dazu haben sie einen Namen (einen Bezeichner), über den sie später im Programm oder Script wieder verwendet werden können sowie einen so genannten Datentyp (eine zugeordnete Information, was für eine Art von Information in der Variablen gespeichert wird, also beispielsweise eine Zahl oder ein Text).

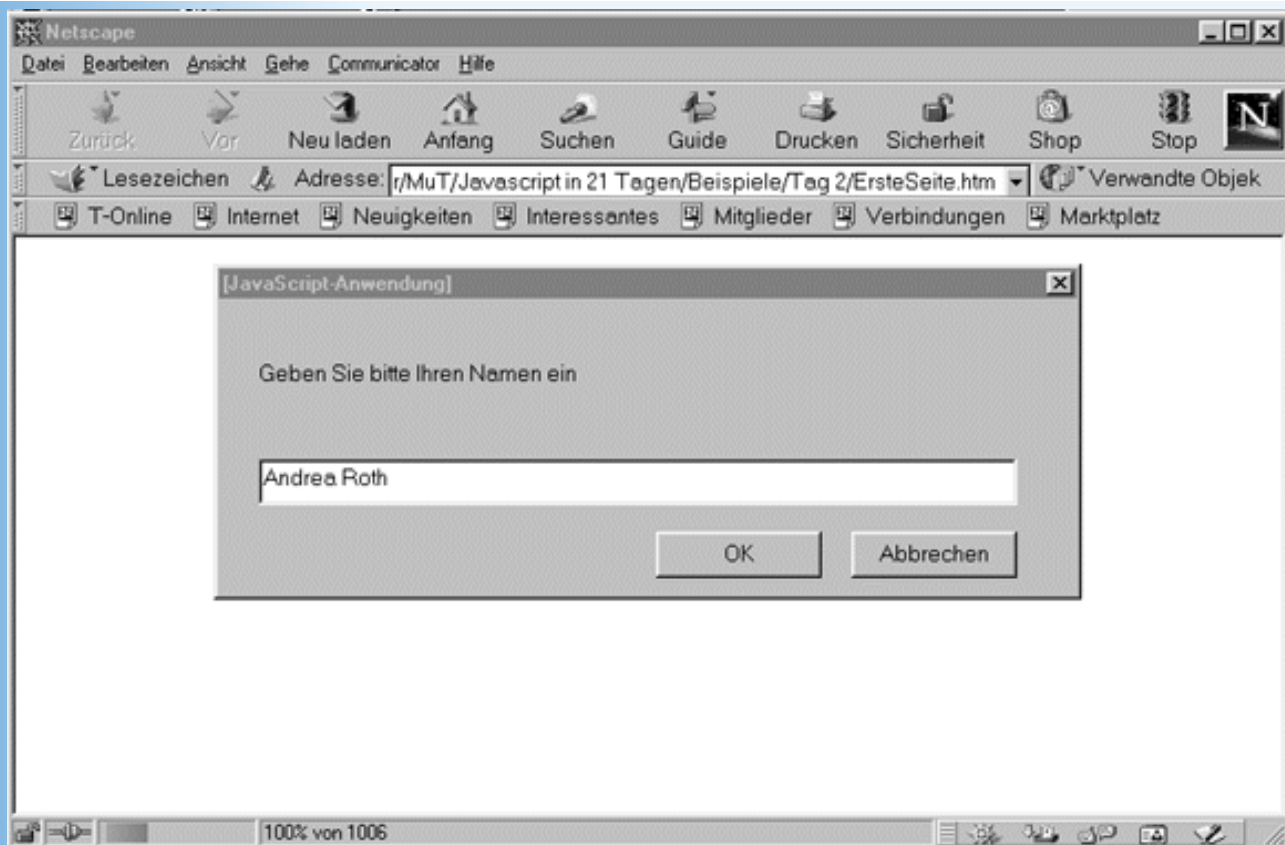


Abbildung 2.2: Die Eingabe vor dem Aufbau der Seite

Der Name der Variablen wird anschließend verwendet, indem mit dieser Angabe, dem aktuellen Tagesdatum und einigen vorgefertigten Textbausteinen dynamisch eine Webseite geschrieben wird. Dabei werden die jeweils notwendigen HTML-Befehle ebenso per JavaScript in die Seite geschrieben.



Abbildung 2.3: Die dynamisch mit JavaScript aufgebaute Seite mit spezifischen Informationen



Diese oder eine ähnliche Technik wird allgemein von Serverprogrammen verwendet, wenn dynamische Webseiten zu erzeugen sind. Gute Beispiele dafür sind Suchmaschinen, die auf Grund einer Eingabe (eines oder mehrerer Suchbegriffe) dynamisch eine neue Webseite mit den jeweils spezifischen Ergebnissen erzeugen.

Bei dem Beispiel verwenden wir eine JavaScript-Funktion, die beim Laden der Webseite aufgerufen und ausgeführt wird.



Eine Funktion können Sie sich erst einmal als die Zusammenfassung von mehreren Anweisungen vorstellen, die mit einem Namen (einem Bezeichner) versehen wird und über den die Gruppe von Anweisungen wieder aufgerufen werden kann. Eine Funktion wird auch erst bei einem solchen Aufruf ausgeführt. Dort, wo sie definiert wird, ist sie gegen eine Ausführung abgeschottet. Wir werden den Begriff später noch genauer verfolgen.



Der Opera-Browser kann in einigen Versionen bei diesen Beispielen Probleme machen. Dies ist aber ein spezifisches Browser-Problem und kein Fehler in den Scripten.

Beispiel 2:

Das nächste Beispiel demonstriert, wie man unter JavaScript Berechnungen durchführen kann. Es soll nur eine kleine Kalkulation durchgeführt werden - eine Zeitberechnung, wie lange es dauert, wenn man mit einer gewissen Geschwindigkeit eine bestimmte Distanz zurücklegt. Nennen Sie die Datei Wegberechnung.htm:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function wegBerechnung()
{
var entfernung = prompt("Geben Sie bitte die Entfernung in Kilometer ein, die Sie
zurücklegen wollen","");
var geschwindigkeit = prompt("Wie hoch ist Ihre Durchschnittsgeschwindigkeit in
Km/h?","");
var ergebnis = entfernung/geschwindigkeit;
document.write('<HTML>');
document.write('');
document.write('<HEAD>');
document.write('<title>Wegberechnung</title>');
document.write('</HEAD>');
document.write('<BODY>');
document.write('<H1 align=center>Wegberechnung</H1>');
document.write('<H4 align=center>');
document.write('Für eine Strecke von ');
document.write(entfernung);
document.write(' Kilometer benötigen Sie bei einer Durchschnittsgeschwindigkeit von
');
document.write(geschwindigkeit);
document.write(' Km/h ');
document.write(ergebnis);
document.write(' Stunden.</H4>');
document.write('</BODY>');
document.write('</HTML>');
}
</SCRIPT>
</HEAD>
<BODY onLoad="wegBerechnung()">
</BODY>
```

</HTML>



Beachten Sie bitte, dass dies nur ein ganz einfaches Beispiel ist. Es ist weder gegenüber einer Division durch 0 noch fehlerhafte Eingaben wie Buchstaben abgesichert. Auch wenn das Ergebnis kleiner 1 ist, wird die führende Null nicht angezeigt.

Beim Laden der Seite wird zuerst eine Eingabe der Entfernung abgefragt. Diese Information wird in einer entsprechenden Variablen gespeichert.

Abbildung 2.4: Die zurückzulegende Entfernung in Kilometern

Danach geben Sie die Durchschnittsgeschwindigkeit ein. Auch diese Information wird in einer weiteren Variablen gespeichert.



Abbildung 2.5: Die dynamisch mit JavaScript aufgebaute Seite mit den berechneten Daten

Abbildung 2.6: Die Durchschnittsgeschwindigkeit

Aus beiden Werten wird die Zeit berechnet, welche für die Strecke benötigt wird (mit dezimalem Anteil der Minuten). Diese Information wird in einer dritten Variablen gespeichert und in der nun dynamisch generierten Webseite verwendet.

Beispiel 3:

Mit JavaScript kann man einiges über die Plattform des Besuchers herausbekommen - das erste Beispiel fragt ja bereits die in dem Computer eingestellte Systemzeit ab. Das dritte Beispiel (Plattform.htm) soll nun den Browser des Anwenders abfragen und den verwendeten Browser in der Webseite anzeigen.



Beachten Sie, dass die hier abgefragten Informationen nicht von jedem Browser bzw. jeder Plattform vollständig und teilweise nicht einmal korrekt angegeben werden.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function spionage()
{
document.write('<HTML>');
document.write('<HEAD>');
document.write('<title>Wer bist Du</title>');
document.write('</HEAD>');
document.write('<BODY>');
document.write('<H1 align=center>Big Brother is watching you</H1>');
document.write('<H4 align=center>');
document.write('Herzlich willkommen ');
document.write('</H4>');
document.write('<H4 align=center>');
document.write('Ich weiss bereits einiges über Sie');
document.write('</H4>');
document.write('<P>Sie verwenden folgenden Browser: ');
document.write(navigator.appCodeName + ', ' + navigator.appName + ', '
+ navigator.appVersion);
document.write('<P>Ihre eingestellte Sprache ist ');
document.write(navigator.language + '.');
document.write('<P>Ihr Betriebssystem ist ');
document.write(navigator.platform + '.');
document.write('</P>');
document.write('</BODY>');
document.write('</HTML>');
}
</SCRIPT>
</HEAD>
```



```
<BODY onLoad="spionage()">  
</BODY>  
</HTML>
```

Das Beispiel kommt diesmal vollkommen ohne Variablen aus, sondern verwendet die direkte Auswertung von Informationen, die jeder Browser über sich preisgibt.

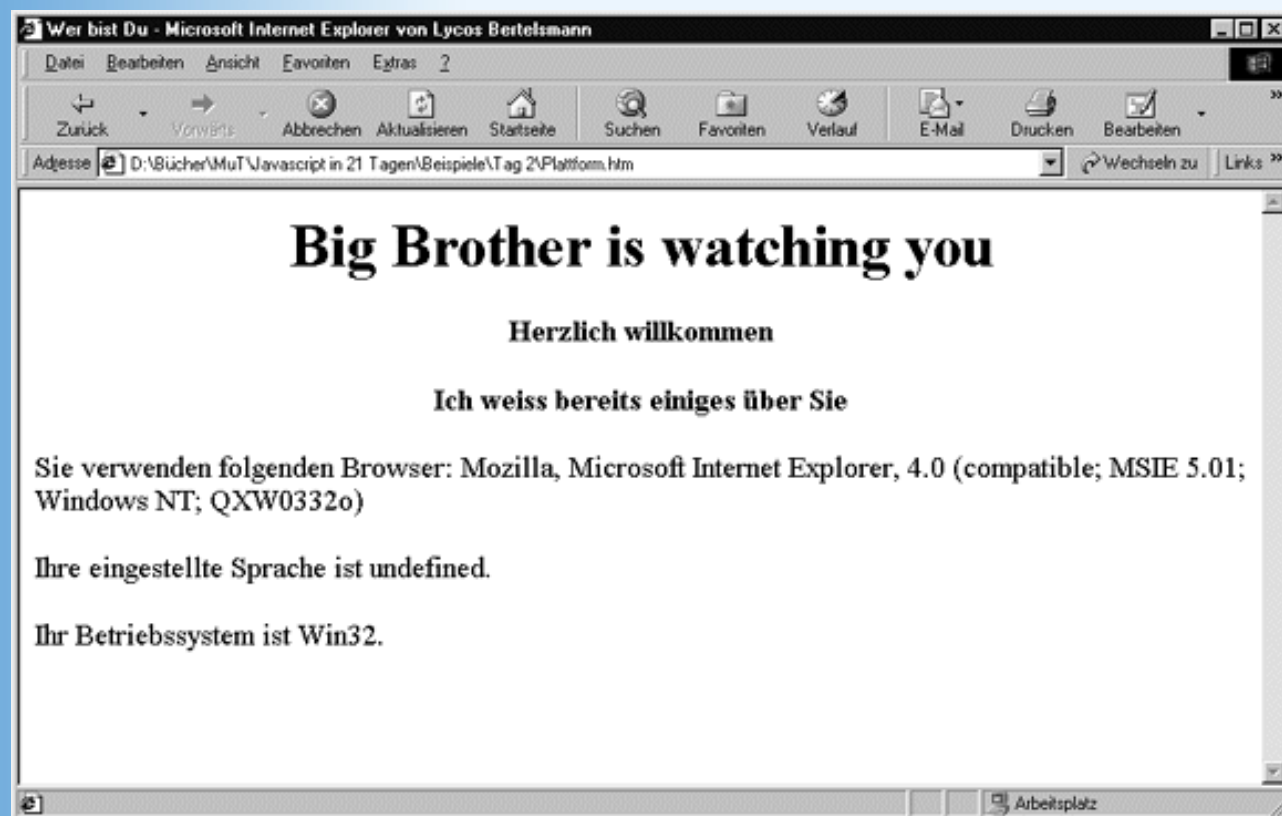


Abbildung 2.7: So outet sich der Internet Explorer 5.01 unter Windows NT 4.0



Abbildung 2.8: Der Netscape Navigator 6.0 in der englischen Version unter Windows NT 4.0



Abbildung 2.9: Das gibt der Netscape Navigator 4.7 unter Windows NT 4.0 bekannt

Beispiel 4:

Das dritte Beispiel (Plattform.htm) verwendet wieder eine JavaScript- Funktion, die beim Laden der Webseite automatisch geladen wird. Mit der hier verwendeten Form des Scriptaufrufs hat der Opera-Browser unter Umständen Probleme, wie bereits etwas weiter oben angedeutet. Deshalb wollen wir das Beispiel geringfügig umbauen, sodass dieses Problem umschifft wird (dies ist keine Lösung für jeden Fall, sondern nur für diese spezielle Situation). Wesentliche Änderung ist

die direkte Notation der Scriptanweisungen direkt im BODY-Bereich der Webseite (ohne Funktionsaufruf). Bauen Sie das dritte Beispiel wie folgt um (Name der Datei Plattform2.htm):

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write('<HTML>');
document.write('<HEAD>');
document.write('<title>Wer bist Du</title>');
document.write('</HEAD>');
document.write('<BODY>');
document.write('<H1 align=center>Big Brother is watching you</H1>');
document.write('<H4 align=center>');
document.write('Herzlich willkommen ');
document.write('</H4>');
document.write('<H4 align=center>');
document.write('Ich weiss bereits einiges über Sie');
document.write('</H4>');
document.write('<P>Sie verwenden folgenden Browser: ');
document.write(navigator.appCodeName + ', ' + navigator.appName + ', '
  + navigator.appVersion);
document.write('<P>Ihre eingestellte Sprache ist ');
document.write(navigator.language + '.');
document.write('<P>Ihr Betriebssystem ist ');
document.write(navigator.platform + '.');
document.write('</P>');
document.write('</BODY>');
document.write('</HTML>');
</SCRIPT>
</BODY>
</HTML>
```



Abbildung 2.10: Das gibt der Internet Explorer 5.01 unter Windows 98 bekannt.

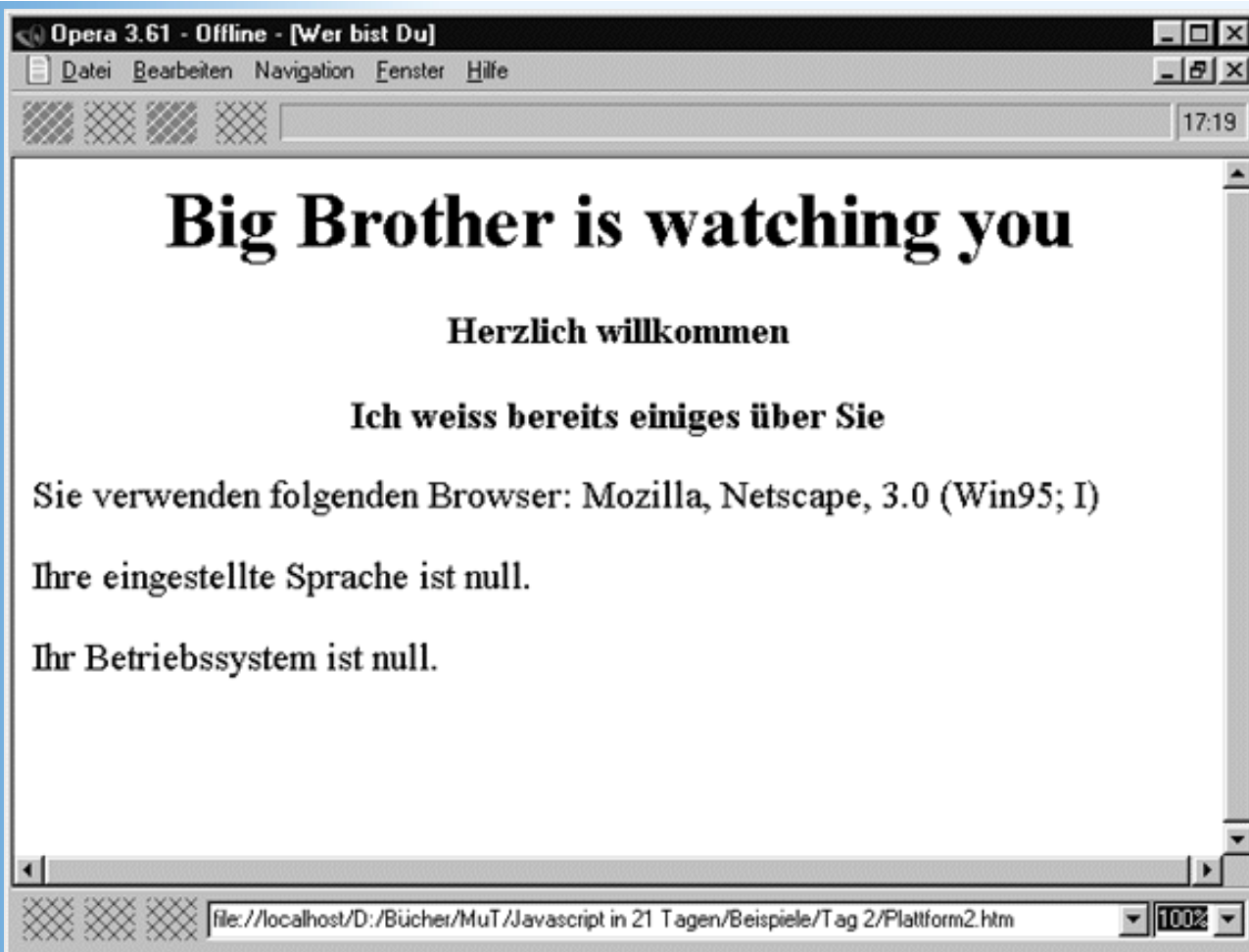


Abbildung 2.11: Opera 3.6 unter Windows NT 4.0

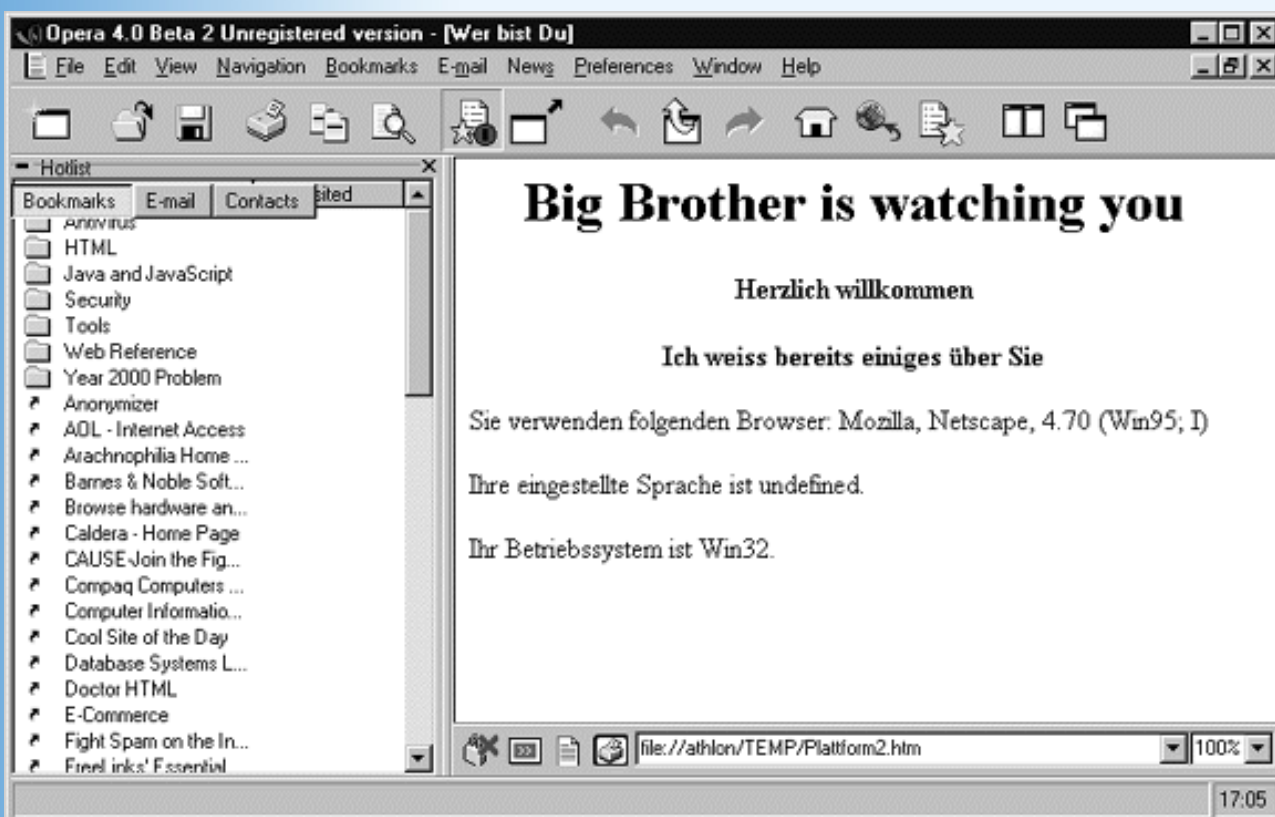


Abbildung 2.12: Opera 4 unter Windows 98

Das Beispiel gibt die gleichen Informationen aus wie Beispiel 3, nur funktioniert es auch unter dem Opera Browser. Der gibt sich allerdings als Netscape Browser aus (der Grund ist, dass er weitgehend dazu kompatibel ist) und geizt auch - gerade in den Version vor 4.0 - mit korrekten Informationen.

Client-seitige versus Server-seitige Scripte

Bei Scripten (wie auch vielen anderen Techniken in einem Netzwerk) muss man grundsätzlich den Ort berücksichtigen, wo sie ausgeführt werden. Im Wesentlichen kann man beim Ort zwischen dem Server und dem Client unterscheiden. Was Client-seitige Scripte sind und was man damit machen kann, haben wir ja schon gesehen. Aber was leisten nun Server-seitige Scripte und wozu gibt es sie überhaupt? Dies klarzumachen ist um so wichtiger, da wir uns in dem Buch schwerpunktmässig auf Client-seitige Scripts konzentrieren. Es sollte klar sein, was Server-seitige Scripte zu leisten vermögen, was deren Einsatzgebiete sind und wo die Grenzen von Client-seitigen Scripten liegen.

So elegant die Verlagerung von Funktionalität auf den Client über Client-seitige Scripte ist - es lassen sich nicht sämtliche Funktionalitäten vom Server auf den Client verschieben. Das liegt einmal an der relativen Einfachheit von Scriptsprachen, zum anderen in der Natur der Sache. Wenn man alle Funktionalität beim Client lösen könnte, würde ja keine Server-Client-Logik für den Internet-Prozess benötigt werden. Insbesondere lassen sich mittels Client-seitigen Scripten in der Regel keine Aufgaben lösen, die auf dem Server Aktionen wie Schreiben oder Lesen von Dateien notwendig machen. So etwas ist nahezu immer dann notwendig, wenn mehrere Clients auf einen Server zugreifen und dies irgendwie im Rahmen einer Webseite protokolliert werden soll.

Als Beispiel kann man sich einen Webseitenzähler vorstellen, der überwacht, wie oft eine Webseite auf einem Server aufgerufen wird, ein Gästebuch oder einen Online-Shop mit Datenbank, wo Bestelldaten aufgenommen werden. Solche Informationen müssen zwingend auf dem Server geführt werden, und man muss für Aktivitäten so gut wie immer auf Server-seitige Scripte oder Programme zurückgreifen.

Server-seitige Scripttechnologien sind etwa CGI oder auch Server-seitige JavaScripte. Auch Techniken wie LiveWire (eine Technologie des Netscape-Webservers), ASP (Active Server Pages - eine Technologie für Microsoft-Webserver) oder Java-Servlets (serverseitige Java-Applets, die auf dem Server durch die dortige virtuelle Javamaschine ausgeführt werden) gehören in diese Welt.

Server-seitige Techniken sind auf jeden Fall sehr interessant und leistungsfähig. Sie unterliegen jedoch diversen Einschränkungen, die ihren objektiven Nutzen für viele Privatanwender erheblich reduzieren. Erstes Problem - der Webserver, auf dem sie laufen sollen, muss über einen Interpreter für die Sprache verfügen, mit der das Server-seitige Script erstellt wurde (was natürlich auch auf Clientseite gilt - dort ist jedoch die Bereitstellung eines solchen Interpreters weniger ein Problem, da ihn für eine Sprache wie JavaScript jeder moderne Browser bereitstellt). Dies kann man zwar vielfach bei einer weit verbreiteten Sprache wie etwa Perl voraussetzen, jedoch bei anderen Sprachen ist das oft nicht gegeben. Für die gleiche Funktionalität muss unter Umständen mehr als ein Script geschrieben werden.

Daneben gibt es noch weitere potentielle Probleme. Besonders beschränkend für viele Webseitenersteller erweist sich, dass zahlreiche Provider die Ausführung von Server-seitigen Fremdscripthen nicht gestatten. Zwar stellen sie oft eigene Scripte für den allgemeinen Gebrauch bereit. Den Upload (und/oder das Ausführen) von selbst geschriebenen Scripten, die ihren Server steuern sollen, unterbinden sie jedoch meist. Für den Webprogrammierer, der seine Seiten bei einem solchen Provider platzieren möchte (insbesondere zählen dazu auch die großen Provider AOL, T-Online und CompuServe), ist eine Beschäftigung mit dem Erstellen von den teilweise doch recht komplizierten Server-seitigen Prozessen also oft ein wenig frustrierend - die Resultate können nicht in natura verwendet werden.

Die Einrichtung eines eigenen Webservers wie Sambar (ein frei verfügbarer Webserver für Windows - nicht mit Samba⁴ unter Linux zu verwechseln) als Alternative ist zwar unproblematisch und auch aus verschiedensten Gründen sehr zweckmässig, dahingegen die Bereitstellung einer permanenten Verbindung zum Internet (geeignete Hardware, eigene feste IP-Nummer, Standleitung, Einrichtung einer zuverlässigen Firewall, etc.) sehr aufwendig und kostenintensiv. Der Webserver wird also meist nur im eigenen Intranet oder sogar nur zusammen mit dem Clientprogramm auf einem einzigen Rechner laufen.

Zusammenfassung

Sie haben an dem heutigen Tag einen ersten Eindruck von JavaScript sowie verwandten oder ergänzenden Technologien wie HTML, VBScript, Java, XML, XSL, Perl oder CGI gewonnen. Insbesondere haben Sie mit einigen Beispielen bereits praktisch JavaScript-Programmierung geübt. Ein Augenmerk galt dem Unterschied zwischen Server-seitigen und Client-seitigen Techniken, wobei insbesondere damit die Möglichkeiten von JavaScript auf Clientseite deutlich gemacht werden sollten, aber auch wo dazu ergänzende Vorgänge auf dem Server unumgänglich werden.

Morgen wollen wir uns um HTML-Grundlagen und vor allem die Einbindung von JavaScripten in Webseiten kümmern.

Fragen und Antworten

Frage:

Es wird im Buch gelegentlich von Scriptsprachen und dann wieder Programmiersprachen gesprochen. Ist das identisch?

Antwort:

Es kommt darauf an, wie diese Begriffe verwendet werden. Wir wollen sie unterschiedlich einsetzen. Scriptsprachen sind solche Sprachen, die im Wesentlichen zur Programmierung einer Umgebung (etwa eines Browsers, aber auch des Betriebssystems wie mit WHS - Windows Host Scripting - unter Windows) verwendet werden. In der Regel erweitern sie die sonst in der Umgebung bereits vorhandenen Möglichkeiten und/oder automatisieren bestimmte Vorgänge, die ein Anwender auch Schritt für Schritt manuell durchführen könnte. Programmiersprachen sind Sprachen, mit denen eigenständige Programme erstellt werden, die nicht nur hauptsächlich ihre Umgebung über eine Schnittstelle beeinflussen können und vor allem keine solche interpretierende Umgebung benötigen. Dazu zählen Sprachen wie Java, C/C++, Delphi, aber auch Visual Basic und sogar Basic selbst. Die Unterscheidung ist irgendwo etwas willkürlich, denn streng genommen benötigen auch in Programmiersprachen erstellte Programme eine Umgebung (zumindest das Betriebssystem oder bei Java die JVM), innerhalb der sie agieren können. Auch die Unterscheidung in kompiliert und interpretiert greift nicht, denn Basic-Programme sind interpretiert. Dennoch - das Charakteristikum, vornehmlich die Schnittstelle einer interpretierenden Umgebung steuern und Vorgänge automatisieren zu können, beschreibt eine Scriptsprache recht gut.

Frage:

Sind Client-seitige oder Server-seitige Scripte wichtiger?

Antwort:

Für normale Anwender ohne uneingeschränkten Zugang sicher Client-seitige Scripte. Zumal sie einfacher zu lernen sind. Bei Kontrolle über einen Server kann man dies nicht mehr so pauschal sagen. Es kommt auf die Aufgabenstellung an.

Workshop

Experimentieren Sie mit den in diesem Kapitel besprochenen Beispielen ein wenig herum. Versuchen Sie Teile davon so zu verändern, dass die Scripte noch sinnvoll laufen, aber echte Unterschiede zu den Beispielen sichtbar sind.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Ist XML eine Ablösung für HTML?
2. Welche Version von JavaScript ist auf Serverseite aktuell?
3. Ist VBScript oder JavaScript weiter verbreitet?
4. Läuft VBScript im Opera-Browser?
5. Welchen JavaScript-Standard unterstützt der Opera-Browser 3.6?
6. Sind Java und JavaScript eng verwandt?
7. Wofür ist CGI die Abkürzung?
8. Welche Sprache wird hauptsächlich zur Erzeugung von Server-seitigen Scripten eingesetzt?

Hätte sich der Internet Explorer im Web so durchgesetzt, wie es Microsoft sicher gerne sehen würde, wäre das Problem natürlich kleiner.

Sie werden das natürlich viel besser machen.

Ohne das »r«.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 1

Tag 3

Wichtige HTML- Grundlagen

Wenn Sie mit JavaScript programmieren wollen, kommen Sie auf Seiten des Clients nicht umhin, diese Scripte irgendwie in eine Webseite zu integrieren. Dazu gibt es gewisse Techniken, die zu HTML zählen. Ohnehin ist es für die Erstellung von Webscripthen mittels JavaScript unabdingbar, dass Sie sich mit HTML-Grundlagen auskennen. Sie müssen nicht jede Anweisung kennen, aber zumindest das Konzept verstehen. Am dritten Tag wollen wir uns also genauer mit HTML-Grundlagen beschäftigen. Die Verbindung zu JavaScript wird uns dabei erst einmal nur am Rande interessieren. Am darauf folgenden Tag können wir dann zu der konkreten Integration von Scripten in Webseiten kommen. Darüber hinaus sollen dann auch Referenzen auf Style Sheets zur Sprache kommen, denn dieser Vorgang weist viele Parallelen dazu auf.

HTML-Grundlagen

Wir wollen am Anfang noch mal die wichtigsten Eckdaten von unseren bisherigen Ausführungen zu HTML zusammenfassen: HTML steht für »Hyper Text Markup Language« und bezeichnet eine Dokumentbeschreibungssprache, mit der die logischen Strukturen eines Dokuments im WWW beschrieben werden. HTML-Dateien bestehen immer aus reinem Klartext ohne irgendwelche Formatierungen, wodurch HTML-Dokumente plattformunabhängig bleiben. HTML ist damit zwangsweise eine Interpretersprache, die interpretiert werden muss. Dazu verwendet man Browser. HTML verfügt im Gegensatz zu Programmier- und Scriptsprachen weder über Kontrollstrukturen in Form von Bedingungen, Sprüngen oder Schleifen noch über Variablen. Von daher können rein mit HTML beschriebene Seiten keinerlei Aktivitäten aufweisen (von referenzierten Ergänzungstechniken wie animierten GIFs oder so abgesehen) - sie sind statisch.

Die HTML-Steueranweisungen

HTML-Befehle bestehen aus Klartext-Steueranweisungen, die wiederum aus so genannten Tags aufgebaut sind. Ein Tag beginnt immer mit einer geöffneten spitzen Klammer

<

und endet mit der geschlossenen spitzen Klammer

> .

In so einem abgeschlossenen Bereich wird ein HTML-Schlüsselwort notiert, das den konkreten Befehl spezifiziert. Ein Tag sieht also von der Struktur her immer so aus:

<Anweisung>

Wenn die Anweisung ein gültiges Schlüsselwort ist, wird der Interpreter (d.h. der Browser) die Anweisung im Inneren des Klammerpaars ausführen. Kennt er sie nicht, wird die Anweisung nach dem Prinzip der Fehlertoleranz ignoriert. Damit können jederzeit neue Anweisungen im WWW eingeführt werden, ohne dass in der Anfangsphase die alten Browser beim Laden einer Seite mit neuen Anweisungen abstürzen.

Grundsätzlich unterscheidet man bei Tags zwischen einem einleitenden Tag und einem beendenden Tag, der die Wirkung einer vorher erteilten Anweisung wieder aufhebt. Denken Sie beispielsweise an eine Anweisung, die ab einem bestimmten Zeichen allen nachfolgenden Text fett darstellen soll (das wäre die HTML-Anweisung). Jedes nachfolgende Zeichen wird fett dargestellt, bis die Wirkung wieder aufgehoben wird (im Fall von fetter Textdarstellung mit). Beide Tags sehen fast identisch aus, außer dass dem beendenden Tag vor der Anweisung ein Slash (Schrägstrich) / vorangestellt wird. Beispielsweise gehört zu dem einleitenden Tag

<H1>

der beendende Tag

</H1>

oder zu

<CENTER>

der beendende Tag

</CENTER>.

Beide Tags zusammen bilden einen so genannten Container. Dies bedeutet, die im einleitenden Tag angegebene HTML-Anweisung (etwa eine Formatierung) wirkt sich auf sämtliche Dinge aus, die sich im Inneren des Containers befinden und von der Anweisung beeinflusst werden können. Dies wird in vielen Fällen ein Text sein, es kann sich aber auch um Grafik oder andere Multimedia-Objekte handeln. Oder die Kennzeichnung eines Bereichs, indem Scripte oder Style Sheets notiert werden, wie wir bald sehen werden.

Nicht alle HTML-Steueranweisungen werden über Container realisiert. Es gibt auch eine Anzahl von Tags, wo kein beendender Tag notwendig ist. Etwa die Referenz auf eine Grafik. Oder das Einfügen eines Zeilenumbruchs. Daneben kann es Situationen geben, wo der Ende-Tag zwar vorgesehen ist, er aber in der Praxis ohne negative Auswirkung weggelassen werden kann. Etwa wenn ein nachfolgender Start-Tag eine widersprüchliche Anweisung auslöst, die die vorherige Anweisung deaktiviert. Auf eine solche »automatische« Deaktivierung sollte man sich aber nicht verlassen. Zumal im Fall von unbedingt notwendigem Ende-Tag ein Fehlen fatale Folgen haben kann. Dagegen ist die Angabe eines eigentlich nicht unter HTML definierten und damit überflüssigen Ende-Tags nie ein Problem. Für den Browser ist das einfach eine Anweisung, die er nicht kennt und er wird sie nach dem Prinzip der Fehlertoleranz ignorieren.

Viele HTML-Steueranweisungen sind erst dann sinnvoll einzusetzen, wenn sie genauer spezifiziert werden. Nicht jeder, denn beispielsweise ein Zeilenumbruch ist durch eine einfache Anweisung - die Steueranweisung
 - vollständig beschrieben (ohne weitere Parameter und auch ohne Ende-Tag). Aber nicht alle Anweisungen sind so eindeutig. Es gibt viele Steueranweisungen, wo man genauere Angaben machen muss. Etwa wenn eine Grafik referenziert werden soll. Neben dem Befehl, dass eine solche Referenz erfolgt, muss für dieses Beispiel auch noch angegeben werden, welche Datei genau das ist. Diese genauere Spezifikation erfolgt durch so genannte Parameter, die bei Bedarf den einleitenden Tag - durch Leerzeichen abgetrennt - erweitern. Dabei gibt es zwei Formen von Parametern:

1. Parameter mit einer Wertzuweisung.
2. Parameter, die bereits einen Wert repräsentieren.

Parameter mit einer Wertzuweisung bekommen über das Gleichheitszeichen = den entsprechenden Wert zugeordnet. Dies kann ein Text oder eine Zahl sein oder auch eine beliebige andere Form haben. Ein Tag mit einem Parameter mit einer Wertzuweisung sieht schematisch so aus:

<Anweisung Parameter=Wert>

Beispiel:

Diese Anweisung referenziert eine Grafik in der Webseite.

Parameter, die bereits einen Wert repräsentieren, brauchen bloß durch ein Leerzeichen abgetrennt in den Einleitungs-Tag geschrieben werden. Sie fungieren als Schalter. Wenn sie angegeben werden, wird eine Eigenschaft aktiviert, fehlen sie, ist die jeweilige Eigenschaft deaktiviert. Ein Tag mit einem Parameter, der bereits einen Wert repräsentiert, sieht schematisch so aus:

<Anweisung Parameter>



Es wird immer nur der einleitende Tag durch Parameter erweitert. Beim beendenden Tag wird niemals (!) ein Parameter angegeben.

Viele Befehle lassen sich über mehr als einen Parameter spezifizieren. Diese werden dann einfach durch ein Leerzeichen getrennt aufgelistet.

Beispiel:

```
<BODY bgcolor=white text=red link=yellow>
```

Die Reihenfolge der Parameter spielt in der Regel keine Rolle.



HTML unterscheidet im Gegensatz zu JavaScript nicht (!) zwischen Groß- und Kleinschreibung. Jede HTML-Anweisung kann sowohl klein als auch groß geschrieben werden. Ebenso die Parameter. Sogar das Mischen von Groß- und Kleinbuchstaben innerhalb eines Worts ist erlaubt und hat keine Auswirkung (und ist sogar bei Parametern, die aus mehreren Worten zusammengesetzt sind, äußerst sinnvoll - etwa so: `onMouseOver`). Jeder kann nach seiner eigenen Vorliebe glücklich werden, es sei denn, in einem Projekt mit mehreren Beteiligten werden bestimmte Konventionen verbindlich beschlossen. Viele HTML-Programmierer tendieren dazu, das gesamte Befehlswort innerhalb der spitzen Klammern immer groß zu schreiben, andere schreiben es konsequent klein. Dies gilt auch für die Konventionen in HTML-Tools. Einige schreiben Steueranweisungen immer klein, andere immer groß und manche lassen es sogar einstellen (etwa der Editor Phase 5). Sehr gut lesbar ist Quelltext, wenn Befehle vollständig groß und zugehörige Parameter vollständig klein geschrieben werden. Wie auch immer - es dient ausschließlich der Übersichtlichkeit und ist ohne irgendwelche Nebeneffekte.

Dagegen dürfen aus mehreren Einzelworten zusammengesetzte Parameter niemals getrennt werden. Auch, wenn die Einzelworte in dem zusammengesetzten Begriff jeweils mit einem Großbuchstaben beginnen. Etwa bei `onClick`.



Zwar hat Groß- und Kleinschreibung innerhalb einer Anweisung keine Auswirkungen, aber es gibt natürlich auch in HTML einige gefährliche Fehler. Dazu zählen die falsche Verwendung der spitzen Klammern zum Kennzeichnen eines HTML-Befehls. Beispielsweise werden Klammern genau falsch herum verwendet (etwa so `> Anweisung <`). Oder es wird die einleitende Klammer vergessen. In diesem Fall erkennt der Browser nicht den Beginn des Befehls und wird die nachfolgende Anweisung als anzuzeigenden Text interpretieren. Schlimmer noch ist, wenn eine schließende Klammer vergessen wird. In diesem Fall wird auch der Text nach der eigentlichen Anweisung vom Interpreter als Parameter zu dem Befehl hinzugezählt. Das Resultat kann unterschiedlich aussehen, aber in jedem Fall wird das eigentlich nach der Anweisung darzustellende Element nicht angezeigt - es ist Bestandteil der Anweisung, die durchgeführt werden soll. Der Fehler kann sich über mehrere Zeilen im Quelltext erstrecken. Grundsätzlich dürfen die spitzen Klammern auch nicht als Teil eines Textes (etwa einer Formel) verwendet werden. Falls man sie in einem Text darstellen muss, müssen sie durch eine Kombination von Zeichen verschlüsselt werden (darauf gehen wir noch ein).

Ein weiteres Problem tritt gerade in Verbindung mit Scripten und Style Sheets auf: Wenn ein Abschluss-Tag notwendig ist und er vergessen wird, wirkt eine Anweisung möglicherweise länger weiter, als sinnvoll. Das kann bei einer Script-Einbindung (worauf wir gleich zu sprechen kommen) so weit gehen, dass die gesamte nachfolgende Webseite als Teil des Scriptcontainers gesehen wird. Wahrscheinliche Folge wird sein, dass weder das Script ausgeführt noch die Webseite überhaupt angezeigt wird. Das gilt analog bei Style-Sheet-Referenzen.

Die Verschachtelung von Containern

Container können verschachtelt werden, das heißt zwischen einem einleitenden und einem abschließenden Tag können beliebige - sinnvolle - andere Steueranweisungen stehen. Dies ist sogar zwingend, da insbesondere die vollständige HTML-Seite einen einleitenden und einen abschließenden Tag benötigt und alle weiteren Steueranweisungen darin verschachtelt werden müssen.

Bei der Verschachtelung von Tags sollte aber die Reihenfolge der Auflösung eingehalten werden! Das bedeutet, wenn ein Container weitere Container enthält, sollte dessen Ende-Tag vor dem Ende-Tag des zuerst geöffneten Containers notiert werden. Container werden also in umgekehrter Reihenfolge der Einleitungs-Tags geschlossen.

HTML-Grundstrukturen

HTML-Interpreter verhalten sich bei der Auswertung von Dokumenten größtmöglich fehlertolerant. Ein Klartext wird ohne irgendeine Struktur von einem Browser ausgewertet werden können. Ohne die geringsten HTML- Steueranweisungen im Inneren. Dann fehlen zwar sämtliche gestalterischen Elemente bei der Darstellung - die reine Information ist jedoch sichtbar. Es gibt nur eine einzige Regel, die aus einem einfachen Klartext gegenüber dem Betriebssystem ein HTML-Dokument macht - die Endung sollte .HTM, .HTML oder .SHTML lauten. Und selbst diese Minimalbedingung setzen Browser nicht voraus - Sie können jede Textdatei, gleich welcher Endung, einfach in den Browser laden. Er wird sie darstellen. Dabei wird jedoch die Fehlertoleranz des Browsers erheblich strapaziert, und bei der Verwendung komplexerer Techniken wird das nicht mehr funktionieren. Wir wollen es dennoch mit zwei kleinen Beispielen durchspielen, die sich bewusst dieser Fehlertoleranz bedienen.

Beispiel 1:

Geben Sie im Editor den nachfolgenden Quelltext ein. Dabei lassen Sie bitte erst einmal Zeilenschaltungen und Leerzeilen aus. Warum, das werden Sie bald sehen:

Was da steht, ist vollkommen egal. Es sollte nur genug Text sein, damit man sehen kann, wie sich die Datei im Browser verhält. Also munter Unsinn hingeschrieben, auf dass sich die Datei mit Zeichen füllt.

Speichern Sie den Text bitte unter dem Dateinamen Text.txt ab. Es handelt sich also um eine reine Klartextdatei, was auch der Arbeitsplatz unter Windows mit einem entsprechenden Symbol bestätigt.



Abbildung 3.1: Die Datei wird vom Betriebs- system als reiner Text eingeordnet

Wenn Sie aber die Datei in einen beliebigen Browser laden, wird der Text dennoch angezeigt.



*Dateien können auf vielfältige Weise vom lokalen Rechner in einen Browser geladen werden. Der Doppelklick auf eine HTML-Datei lädt diese automatisch in dem zugeordneten Standardbrowser. Da wir in dem Beispiel jedoch eine Textdatei laden wollen, funktioniert das nicht. Sie können aber mit dem **Datei**-Menü und dort dem **Öffnen**-Befehl die Datei laden oder noch einfacher - zumindest unter Windows - die Datei per Drag&Drop einfach mit der Maus auf das Browser-Symbol oder in das geöffnete Browserfenster ziehen. In beiden Fällen wird die Datei im Browser geladen.*

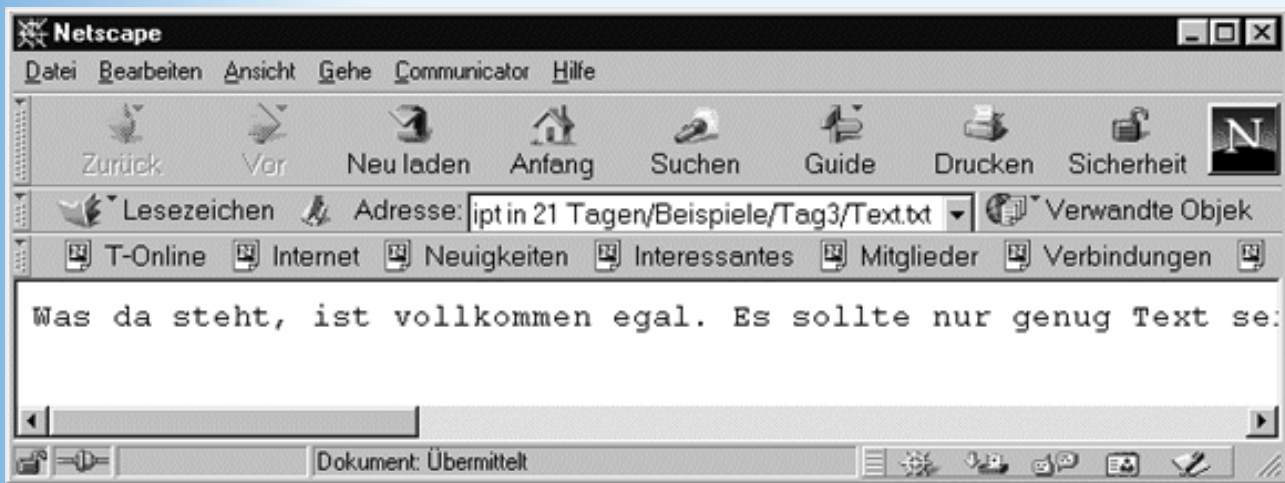


Abbildung 3.2: Der Text wird ohne Zeilenumbrüche angezeigt

Nun benennen Sie die Datei einfach um in Text.html. Das können Sie beispielsweise erreichen, indem Sie im Arbeitsplatz mit der rechten Maustaste auf die Datei klicken und aus dem Kontextmenü den entsprechenden Befehl auswählen. Alternativ markieren Sie die Datei und wählen in dem **Datei**-Menü den **Umbenennen**-Befehl aus.

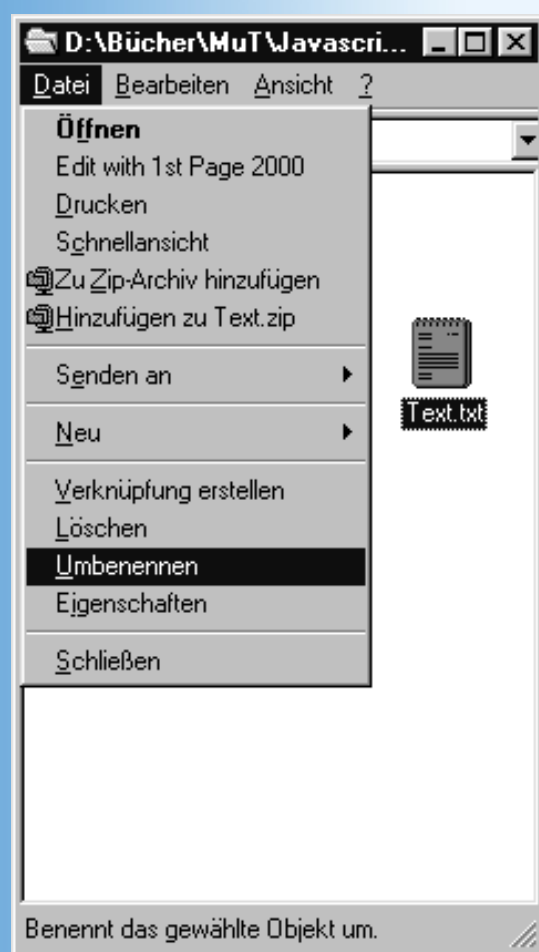


Abbildung 3.3: Umbenennen der Datei

Die Datei wird anschließend vom Betriebssystem über ein entsprechendes Symbol als Webseite dargestellt.



Abbildung 3.4: Die Datei wird als Webseite erkannt

Starten Sie die als Webseite symbolisierte Datei mit einem Doppelklick in einem Browser. Beachten Sie das Resultat. Was fällt Ihnen auf?



Abbildung 3.5: Der Text wird mit Zeilenumbrüchen und anderer Schrift angezeigt

Im Gegensatz zu dem Fall, wo eine mit .txt gekennzeichnete Textdatei in den Browser geladen wurde, behandelt der Browser diese über die Dateieindung als Webseite gekennzeichnete Datei offensichtlich viel differenzierter. So wird etwa automatisch der Text umbrochen und in einer anderen Schrift mit definierter Hintergrundfarbe, Textfarbe und Textgröße dargestellt. Ohne irgendwelche HTML-Anweisungen im Inneren. Warum dies so ist, werden wir einige Absätze weiter beim Grundgerüst der Webseite abermals genauer veranschaulichen. Die Textformatierung lässt sich jedoch bereits jetzt einfach erklären - es sind Voreinstellungen des Browsers, wie er jede als Webseite erkennbare Datei darstellt. Aber was ist mit dem Zeilenumbruch? Warum hat der Browser Umbrüche verwendet und wo? Verändern Sie bitte einmal die Größe des Browserfensters.

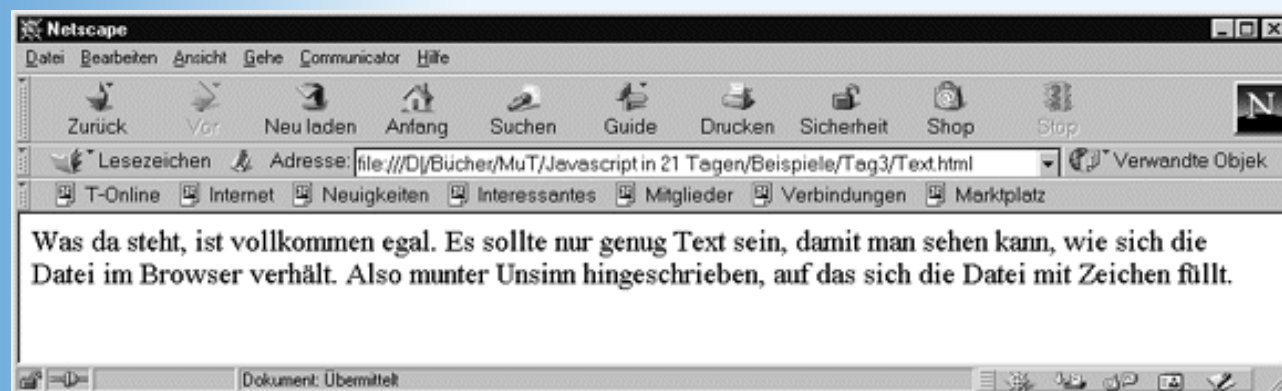


Abbildung 3.6: Veränderte Zeilenumbrüche bei anderer Fenstergröße

Die Umbrüche werden sich an anderen Stellen befinden.

Nun bearbeiten Sie bitte die Datei Text.html, indem Sie beliebig Zeilenumbrüche zwischen den einzelnen Worten einfügen. Auch durchaus mehrere hintereinander, um Leerzeilen zu erzeugen. Speichern Sie die Datei unter Text2.html und laden Sie diese dann in einen Browser. Sind die Zeilenumbrüche da, wo Sie diese per Zeilenschaltungstaste des Editors im Text

eingefügt haben? Vermutlich nicht, denn sowohl Zeilenschaltungen als auch Leerzeilen, die per Editor eingegeben werden, sollten durch Browser ignoriert werden. Dafür müssen Zeilenschaltungen in Form von expliziten HTML-Anweisungen verwendet werden. Auch wenn Sie an verschiedenen Stellen im Text Leerzeilen und Zeilenschaltungen mit der Datenfreigabetaste im Editor eingefügt haben, werden sie nicht dargestellt. Größenveränderungen des Browsers werden sich jedoch auswirken. Der Text wird immer in der Breite automatisch der Größe des Browsers angepasst. Damit sind Browser in der Lage, darzustellenden Text immer an die zugrunde liegende Plattform optimal anzupassen. Weder die Größe des Browserfensters, die Browser-spezifischen Einstellungen, noch die Auflösung des Bildschirms werden die Darstellung des Textes so beeinflussen, dass er über den seitlichen Anzeigebereich hinaus geht.

Das ist aber kein Nachteil oder gar Problem von HTML oder Browsern. Ganz im Gegenteil - das ist eines der Highlights von HTML. Quellcode kann für den Ersteller der Webseite übersichtlich gestaltet werden und er muss sich dabei nicht darum kümmern, wie der Text beim Anwender aussieht. Ob er in jeder Auflösung zu sehen ist, in jeder Fenstergröße, in jeder eingestellten Schrift usw. Darum kümmert sich der Browser beim Anwender - der Ersteller der Webseite ist von der Last befreit. Ohne entsprechende Gegenmaßnahmen über eigene HTML-Befehle wird Text immer automatisch der Größe des Browsers angepasst. Damit haben wir bereits die Grundphilosophie von HTML im Wesentlichen beschrieben: Der Ersteller stellt Informationen bereit, wie diese jedoch konkret beim Anwender aufbereitet werden, überlässt man dem Browser. Dieses geniale Konzept erlaubt es, schnell Informationen zu publizieren und diese mit minimaler Zusatzinformation über das Netz zu verteilen. Wir werden darauf im Rahmen des Buches an verschiedenen Stellen genauer zurückkommen. Grundsätzlich muss man sich aber unter HTML klar werden, dass es zahlreiche Layoutfaktoren gibt, die ein Ersteller einer Webseite nicht unmittelbar kontrollieren kann. So wird Text mit vordefinierter Hintergrundfarbe, Textfarbe und Textgröße innerhalb eines Browsers dargestellt, wenn die Webseite keine eigenen Angaben macht (Voreinstellungen des Browsers). Dies sind aber Darstellungsbedingungen, die der Designer einer Webseite niemals beeinflussen kann, sondern von der Plattform des Betrachters abhängen. Auf diese Faktoren kann also über HTML kein Einfluss genommen werden, sie müssen jedoch unbedingt im Gesamtkonzept berücksichtigt werden.

Das Grundgerüst einer HTML-Seite

Reiner Klartext kann jederzeit in einem Browser dargestellt werden. Auch ohne dass die Datei als Webseite über ihre Endung identifiziert ist. Aber erst mit einer solchen Identifikation wird eine Datei vom Browser entsprechend den Regeln für Webseiten behandelt. Und obwohl wir gerade gesehen haben, dass eine HTML-Seite dann auch ohne HTML-Befehle im Inneren vom Browser sinnvoll darstellbar ist, wird normalerweise immer ein Grundgerüst aus HTML-Anweisungen benötigt. Fehlt dieses, reagieren Browser nämlich wieder mit dem Prinzip der Fehlertoleranz und »denken sich die fehlenden Grundanweisungen hinzu«. Das funktioniert aber nur so lange, wie keine komplexeren Dinge hinzukommen wie etwa Style Sheets oder Scripte. Verlassen Sie sich nicht darauf, dass ein fehlendes Grundgerüst schon keine Probleme macht.

Ein Webseitengrundgerüst definiert den Anfang und das Ende einer HTML- Seite und unterteilt sie in verschiedene Bereiche. Im Wesentlichen gibt es einen später im Anzeigebereich des Browsers sichtbaren Bereich und einen dort nicht angezeigten Bereich. Schauen wir uns einmal ein solches Gerüst an. Eine HTML-Seite beginnt immer mit der Anweisung

```
<HTML>
```

und endet mit

```
</HTML> .
```

Dies ist der äußerste Container einer HTML-Seite. Den im Anzeigebereich des Browsers unsichtbaren Bereich nennt man »Header« (Kopf) einer Webseite, den sichtbaren Bereich »Body« (Körper).

In dem der einleitenden HTML-Anweisung direkt folgenden Header werden allgemeine Hintergrundinformationen über die Seite zusammengefasst. Im Wesentlichen werden diese von Suchmaschinen verwendet. Damit ist dieser Teil einer Seite besonders wichtig, damit sie überhaupt jemals besucht wird und/oder der Inhalt der Seite vor einem Besuch richtig bewertet werden kann. Aber auch die Einbindung von Scripten und Style Sheets wird oft im Header vorgenommen. Ein Header wird mit der Anweisung

```
<HEAD>
```

begonnen und mit

```
</HEAD>
```

wieder geschlossen.

Der Body enthält sämtliche Informationen, die der WWW-Browser einem Anwender auf dem Bildschirm anzeigen soll. Er beginnt mit der Anweisung

<BODY>

und endet mit

</BODY>.

Damit sieht das minimale Grundgerüst einer normalen HTML-Datei so aus:

<HTML>

<HEAD>

</HEAD>

<BODY>

</BODY>

</HTML>

Wenn man sich das Grundgerüst jetzt in einen Browser lädt, wird man nicht viel sehen. Das sind aber die Anweisungen, die ein Browser »dazu denkt«, wenn man sie wie in unserem ersten Beispiel weglässt. Allerdings darf nun nicht der Eindruck entstehen, dass das »Hinzudenken« immer ohne Probleme funktioniert. Das erste Beispiel war nur von so einfacher Natur, dass es nicht zu Problemen für einen Browser kommen konnte. Prinzipiell können Sie nicht auf ein solches Grundgerüst verzichten. Zumindest nicht auf die Angabe von <HTML> und <BODY>. Der Header kann nur unterbleiben, wenn die Seite keinerlei Hintergrundinformationen beinhaltet.



Start- und Ende-Tags von Kopfbereich und Body der Webseite müssen sich vollständig innerhalb des umgebenden HTML-Containers befinden. Der Kopfbereich muss unbedingt geschlossen werden, bevor der Körper der Webseite geöffnet wird. Sie dürfen also auf keinen Fall verschachtelt werden.

HTML-Kommentare

Kommentare sind ein äußerst wichtiger Bestandteil einer Webseite. Prinzipiell versteht man unter einem Kommentar einen Bestandteil eines Quelltexts, der von dem ausführenden Programm (Interpreter-Fall) oder dem übersetzenden Programm (Compiler-Fall) ignoriert wird. Er kann in der Regel an jeder Stelle innerhalb eines Quelltexts stehen. Dies bedeutet, Kommentare haben keinerlei Auswirkung auf die konkrete Abarbeitung eines Quelltexts. Für eine HTML-Seite hat das die Folge, dass ein HTML-Kommentar im Darstellungsbereich eines Browsers nicht angezeigt wird und auch sonst für den HTML-Interpreter keine Wirkung hat. HTML-Kommentare dürfen sogar vor dem Beginn des Grundgerüsts stehen. Dennoch sind Kommentare von elementarer Bedeutung für Webseiten. Es gibt unter anderem folgende Rechtfertigungen:

1. Das vielleicht offensichtlichste wenngleich nicht wichtigste Argument für die Verwendung von Kommentaren ist die Tatsache, dass man damit für eine bessere Lesbarkeit des Quelltexts sorgen kann. Gerade bei späterer Überarbeitung ist eine solche Dokumentation im Quelltext selbst oft sinnvoll - vor allem bei größeren Projekten.
2. Mit Kommentaren lassen sich im HTML-Quelltext Hintergrundinformationen unterbringen, die bei der Darstellung der Seite nicht direkt sichtbar sein sollen.
3. Zeitweise lassen sich über Kommentare bestimmte Teile eines Programms oder Dokumentes ausblenden. Wenn Sie diese Teile wieder benötigen, entfernen Sie einfach die Zeichen für den Kommentar wieder. Dies ist im Fehlerfall eine sehr effektive Möglichkeit, um Fehler zu lokalisieren (darauf gehen wir am siebten Tag noch genauer ein).
4. HTML-Kommentare sind ein wichtiger Schlüssel zur Verwendung von Scriptsprachen und Style Sheets in HTML-Seiten, ohne durch deren Verwendung inkompatiblen Browsern Probleme zu bereiten. Darauf kommen wir morgen genauer zu sprechen.

Im Prinzip kann jeder Tag, der eine nicht für den Interpreter bekannte Anweisung enthält, implizit als Kommentar verstanden werden. Die Anweisung selbst und alle der Anweisung folgenden Zeichen bis zu der beendenden Klammer werden nach dem Prinzip der Fehlertoleranz ignoriert. Diese indirekte Kommentartechnik hat aber diverse Schwachstellen und sollte eigentlich nicht verwendet werden. Zumal es explizite HTML-Anweisungen gibt, die für Kommentare in HTML verwendet werden. Ein HTML-Kommentar beginnt mit einem Ausrufezeichen und zwei Strichen am Anfang (<!--) und endet mit zwei Strichen am Ende des Tags (-->). Alles, was in dem Bereich dazwischen eingeschlossen ist (der Bereich kann sich über beliebig viele Zeile erstrecken), wird als HTML-Kommentar gesehen und vom HTML-Interpreter ignoriert. Beispiel:

<!--

Hier kann beliebiger Text stehen.

Er wird vom HTML-Interpreter ignoriert

-->

Es gibt nun in HTML einen besonderen Kommentar - die Anweisung **DOCTYPE**. Diese Kommentarform wird etwa so aussehen:

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.0 Transitional//EN">
```

Dabei handelt es sich um einen Verweis auf die jeweilige DTD-Datei des Dokumentes. DTD steht dabei für »Document Type Definition«, was übersetzt Dokumententyp-Definition bedeutet. Darin werden alle Besonderheiten, welche ein Dokument bezüglich der Struktur und des Erscheinungsbildes haben darf, beschrieben. Konkret heißt dies, die HTML-Version wird angegeben (im obigen Beispiel HTML 4.0). Im Prinzip muss jedem SGML-Dokument (so auch einer HTML-Seite) eine solche DOCTYPE- Anweisung vorangestellt werden. Andererseits benötigt sie kein Browser wirklich und sie wird sogar von den meisten Browsern ignoriert bzw. hat keine weiteren Konsequenzen für die Darstellung der HTML-Datei. Die meisten im Netz zu findenden Seiten werden nur dann solche Anweisungen vorangestellt haben, wenn sie von einem Tool automatisch bei der Generierung gesetzt werden.

Details zum Header

Behandeln wir nun einige Details zum Header. Wir werden nicht alle dort verfügbaren HTML-Anweisungen angeben (dazu gibt es spezielle HTML- Literatur), aber die wichtigsten. Obwohl dort keine im Anzeigebereich des Browsers sichtbaren Informationen untergebracht werden, ist der Header dennoch der in der Regel wichtigste Bestandteil der Webseite. Der Header ist eine ideale Stelle für die Referenzierung von Scripten und Style Sheets, aber auch der Garant, dass eine Webseite auf unterschiedlichen Plattformen halbwegs korrekt angezeigt wird, und vor allem, dass eine Webseite überhaupt jemals Besucher bekommt. Die dort notierten Hintergrundinformationen sind nämlich die Hauptquelle für Suchmaschinen auf der Suche nach neuen Webseiten und deren korrekter Klassifizierung. Daneben kann man mit Header-Befehlen auch Serveraktionen auslösen und einige automatische Aktionen durch den Browser auslösen.

Eine der wichtigsten dieser Angaben im Kopfteil ist der Titel der Webseite, welche in den Tags

```
<TITLE>
```

und

```
</TITLE>
```

eingeschlossen wird.



Der Tag muss in der englische Schreibweise notiert werden. Die Ähnlichkeit zu dem deutschen Wort Titel führt leicht zu Fehlern.

Der Titel einer Webseite hat nun verschiedene Auswirkungen. Er erscheint beispielsweise beim Browser in der Titelzeile. Darüber kann also unter anderem eine im Browser angezeigte Seite identifiziert werden. Ein weiterer Effekt der Titelzeile ist, dass sie der Browser bei Lesezeichen und beim Anzeigen der bereits besuchten Dateien (Historie) verwendet. Der wichtigste Zweck der Titelzeile beruht freilich auf der Tatsache, dass diese Information von Suchmaschinen zum Bewerten und Einsortieren von Webseiten verwendet wird. Der Text in der Titelzeile ist auch das, was nach einer Suchabfrage durch den Anwender bei einem Treffer als Beschriftung des in der Suchmaschinenwebseite zurückgegebenen Hyperlinks angezeigt wird.

Innerhalb des Headers einer Webseite können noch weitere Informationen untergebracht werden. Zum Beispiel Hintergrundinformationen über den Autor, das Erstellungsdatum oder den verwendeten Zeichensatz - so genannte Metainformationen. Diese stehen auswertenden Programmen wie Suchmaschinen zur Verfügung und helfen bei der Dokumentation der Seite. Grundsätzlich werden solche Metainformationen über den <META>- Tag realisiert, der über einige standardisierte Attribute spezifiziert wird und auch noch allgemeinere Bedeutung hat. Die Werte für ein Attribut werden mit einem Gleichheitszeichen zugewiesen. Innerhalb eines Headers können beliebig viele <META>-Elemente verwendet werden. Es gibt nur einen Start- Tag, ein Ende-Tag ist nicht vorgesehen. Als Attribute des <META>- Elementes stehen im Wesentlichen folgende Angaben zur Verfügung:

Attribut	Beschreibung

lang	Mit dieser Angabe werden zwei Attribute abgekürzt, die im Zusammenhang mit dem Attribut content zur Angaben lang (Abkürzung für Language - Sprache) und dir für die Textfließrichtung (ltr für left-to-right - links nach rechts - oder rtl für right-to-left - rechts nach links) verwendet werden können.
http-equiv	Über das Attribut können HTTP-Headerinformationen spezifiziert werden, die Aktionen auf dem Server- oder Client-Programm auslösen können.
name	Die Spezifizierung des Namens der Metainformation. Es wird in Verbindung mit dem content-Attribut eingesetzt. Diese beiden Informationen bilden zusammen ein Wertepaar, das auch nur in Verbindung Sinn macht. Es gibt keine verbindliche Liste von erlaubten oder verbotenen Werten für dieses Attribut, sprich es ist frei zu wählen. Es gibt allerdings einige Werte, die allgemein verwendet werden. Wir geben gleich ein paar Beispiele an.
content	Die mit dem Attribut name assoziierte Information, die dieses Attribut genauer spezifiziert. Es gibt keine verbindliche Liste von erlaubten oder verbotenen Werten für dieses Attribut, d.h. auch dieses Attribut ist frei zu wählen.
schema	Spezifiziert die Art des Inhaltes über ein Schema, das dessen korrekte Darstellung in einer Seite gewährleistet. Dies kann zum Beispiel bei Datumsangaben nützlich sein, denn diese können - je nach landesspezifischer Schreibweise - mehrdeutig sein. Eine Datumsangabe im Format »10-9-00«; kann sowohl den 9. Oktober 2000 bedeuten als auch den 10. September 2000 - je nachdem, ob erst der Tag oder erst der Monat genannt wird. Der schema-Attributwert »Month-Date-Year« macht die Datumsangabe eindeutig.

Tabelle 3.1: Attribute des <META>-Elements

Schauen wir uns als Erstes einige Beispiele für die Verwendung von dem name-content-Wertepaar an.

Beispiel für die Angabe des Autors einer Webseite:

```
<META name="Author" content="Ralph Steyer">
```

Das name-content-Wertepaar wird auch oft für Eigenwerbung des Tools verwendet, mit dem eine Webseite erstellt wurde. So schreibt der Netscape Composer 4.7 Folgendes in den Header:

```
<meta name="GENERATOR" content=
"Mozilla/4.7 [de]C-CCK-MCD QXW03234 (WinNT; U) [Netscape]">1
```

Microsofts Frontpage setzt neben der Zeile des Generators

```
<meta name="GENERATOR"
content="Microsoft FrontPage 4.0">
```

auch noch eine Programm-Id²:

```
<meta name="ProgId"
content="FrontPage.Editor.Document">
```

Eine der wichtigsten Anwendungen für so ein name-content-Wertepaar ist die Bereitstellung einer Beschreibung für eine Webseite. Das ist das, was ein Anwender unter der per <TITLE>-Tag definierten Hyperlinkbeschriftung als Beschreibung einer Seite sieht, wenn er diese Seite als Ergebnis einer Suchabfrage in einer Suchmaschine angezeigt bekommt. Dazu wird als Wert von name der englische Begriff für Beschreibung (description) gesetzt und für content die konkrete Beschreibung der Seite, welche die Suchmaschine anzeigen soll.

Beispiel:

```
<META name="description" content=
"Die spannendste Seite zwischen in ganz Kleinhinterdorf.">
```

Eine weitere für Suchmaschinen extrem wichtige Metainformation ist das name-content-Wertepaar, wo als Wert von name der englische Begriff für Schlüsselworte (keywords) gesetzt wird. Diese dann per content angegebenen Schlüsselworte sind von Ihnen gewählte Worte, nach deren Eingabe als Suchbegriff in der Suchmaschine Ihre Webseite als Treffer gefunden werden soll. Mehrere Schlüsselworte werden durch Kommata getrennt.

Beispiele:

```
<META name="keywords" content=
"Umsonst, Billig, Kostenlos, Preiswert, Gratis">
```

Eine weitere Metainformation, die mit dem name-content-Wertepaar angegeben wird, ist das Copyright.

Beispiel:

```
<META name="copyright" content="2000 RJS">
```

Ebenso standardisiert ist das Erstellungsdatum einer Webseite über den Wert date für das Attribut name.

Beispiel:

```
<META name="date" content="2000-08-06T09:49:37+00:00">
```

Auch die Dokumentation der hinter einer Webseite stehenden Organisation kann angegeben werden:

```
<META name="organization" content="RJS EDV-KnowHow">
```

Neben den unter dokumentarischem Aspekt im Wesentlichen für Suchmaschinen angelegten Metainformationen kann man mit den HTTP-Headerinformationen in Zusammenhang mit dem http-equiv-Attribut die unterschiedlichsten Aktionen auf Server oder Client auslösen. Immer wenn Dokumente über das Hypertext Transfer Protocol (HTTP) zwischen Server und Client ausgetauscht werden, können damit verschiedene Aktionen beeinflusst werden. Wir wollen exemplarisch drei davon besprechen.

Die wichtigste Aktion dürfte die automatische Weiterleitung eines Besuchers auf eine andere Webseite sein. Dazu dient das refresh-Attribut, dessen Wert als ersten Eintrag die Wartezeit (in Sekunden) und anschließend - per Semikolon abgetrennt - das Ziel der Umleitung in Form eines URL enthält. Das werden wir in einem praktischen Beispiel testen, wobei wir gleichzeitig die gerade besprochenen dokumentarischen Metainformationen einsetzen wollen und einen Kommentar in der Praxis zeigen.

Beispiel 2:

Geben Sie im Editor den nachfolgenden Quelltext ein. Dazu sollte die Datei unter dem Namen start.html im gleichen Verzeichnis gespeichert werden, wo sich auch die im ersten Beispiel erstellte Datei Text.html befindet.



Bei unseren Beispielen können Sie sich viel Arbeit sparen, wenn Sie eine Grundgerüstdatei erstellen und jedes Mal als Schablone verwenden, indem Sie diese in den Editor laden oder via Zwischenablage in ein leeres Dokument einfügen und diese Schablone dann füllen.

Die Zwischenablage bedienen Sie übrigens so:

- 1. Text markieren*
- 2. Markierten Text in die Zwischenablage legen (zum Beispiel mit der Tastenkombination (STRG)+(C))*
- 3. Ziel aufsuchen*
- 4. Einfügen (beispielsweise mit der Tastenkombination (STRG)+(V))*

```
<HTML>
<HEAD>
<TITLE>
Meine Willkommenseite, von der direkt weitergeleitet wird
</TITLE>
<META name="Author" content="Ralph Steyer">
<META name="description" content="Die Seite leitet automatisch auf Text.html
weiter.">
<META name="keywords" content="Steyer, RJS">
<META name="copyright" content="2000 RJS">
<META name="date" content="2000-08-06T09:49:37+00:00">
<META name="organization" content="RJS EDV-KnowHow">
<!--
Die nachfolgende Anweisung ist die Weiterleitung. Sie erfolgt nach 3 Sekunden
-->
<META http-equiv="refresh" content="3;URL=Text.html">
</HEAD>
<BODY>
```

Herzlich Willkommen! Sie werden in drei Sekunden weitergeleitet.

</BODY>

</HTML>

Anschließend laden Sie bitte die Datei in einen Browser.



Abbildung 3.7: Die Startdatei heißt Sie willkommen

Nach drei Sekunden wird automatisch die referenzierte Folgeseite geladen. Über diese Anweisung können Sie eine beliebige Ressource automatisch bei Aufruf einer Seite nachladen lassen. Sie müssen nur deren URL (lokal oder aus dem Internet) angeben. Wenn Sie als Zeitspanne eine 0 angeben, wird die Ressource unmittelbar nachgeladen. Damit lassen sich auch ohne den Einsatz von Scripten (so ein Vorgang ist beispielsweise eine der leichtesten Aufgaben für ein JavaScript) Weiterleitungen realisieren.



Einige Browser unterstützen die Weiterleitung von Seiten auf diese Art und Weise nicht. Deshalb sollte auf jeden Fall auf der weiterleitenden Seite auch die Verzweigung per Hyperlink vorgesehen sein.

Eine weitere interessante Anwendung einer HTTP-Headerinformation ist das Setzen eines Ablaufdatums für ein HTML-Dokument. Damit kann gewährleistet werden, dass ein Browser bei Überschreitung des Datums nicht weiter die alte Version des Dokumentes lädt (etwa aus dem Cache oder von einem Proxy). Dazu dient der Wert expires für das Attribut http-equiv. Das Datum in der content-Wertzuzuweisung wird in einem gültigen HTTP-DATE-Headerformat (etwa nach der ISO8601-Datumsformatnorm) eingegeben. Das nachfolgende Beispiel verwendet ein solches Datumsformat:

```
<META http-equiv="expires" content="Tue, 17 Aug 2000 12:00:00 GMT">
```

Über das Attribut http-equiv kann auch die Zeichenkodierung eines HTML-Dokumentes festgelegt werden. Das nachfolgende Beispiel spezifiziert die Zeichenkodierung eines HTML-Dokumentes nach der Norm ISO-8859-5, wie sie in Westeuropa üblich ist:

```
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-5">
```

Alternativ findet man auch oft die nachfolgende Spezifizierung:

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
```

Die Hoffnung ist bei Verwendung dieser Metainformationen, dass die interpretierenden Browser bei der Darstellung von Text den richtigen Zeichensatz verwenden und auch Sonderzeichen wie ö oder ä sinnvoll darstellen. Das funktioniert aber nur beschränkt, weshalb man solche Sonderzeichen zur Sicherheit noch verschlüsseln sollte (darauf gehen wir am fünften Tag ein).

Wie am Beginn des Abschnittes angedeutet, ist der Header eine ideale Stelle für die Einbindung von Style Sheets und Scripten. Da dies jedoch nicht zwingend im Header erfolgen muss und zudem für unser Buch ein entscheidendes Thema ist, sollen diese Thematiken gesondert behandelt werden.



Die meisten Tools unterstützen den sinnvollen Aufbau des Headers einer Webseite nur mangelhaft oder

missbrauchen ihn höchstens für Eigenwerbung. Bessere Tools erlauben dessen gründlichen Aufbau, aber fast allen Hilfsprogrammen ist gemein, dass die Bedeutung des Headers nicht richtig klar gemacht wird. Gerade Anwender ohne Hintergrundwissen glauben oft, mit der Gestaltung des sichtbaren Bereichs einer Webseite sei diese fertig. Hinterher sind sie verwundert, wenn nur zwei Besucher pro Monat auf der Webseite vorbeikommen und einer davon ist man selbst, wenn man den Webcounter kontrolliert.

Der BODY einer Webseite

Der Body-Bereich einer Webseite umfasst sämtliche (!) im Anzeigebereich des Browsers sichtbaren Informationen einer Webseite. Innerhalb des von dem Start- und Ende-Tag gebildeten Containers sind sämtliche Daten einzufügen, die dort angezeigt werden sollen: Vom reinen Text über Grafiken bis hin zu komplexeren Objekten wie Java-Applets. Alles Sichtbare wird dort angegeben - auf Details kommen wir noch.

Der <BODY>-Tag besitzt eine ganze Reihe von Parametern, die ihn bei Bedarf erweitern und das globale Aussehen eines Dokumentes bestimmen. Es gibt da zum einen diverse Farbangaben, welche die globalen Farben von verschiedenen Elementen regeln:

- Die Hintergrundfarbe über das Attribut bgcolor.
- Die Angabe der Textfarbe über das Attribut text.
- Die Angabe, welche Farbe ein Hyperlink haben soll, über das Attribut link.
- Die Angabe, welche Farbe ein aktiver Hyperlink beim Anklicken haben soll, über das Attribut alink.
- Die Angabe, welche Farbe ein bereits besuchter Hyperlink haben soll, wenn ein Anwender wieder zu einer Seite zurückkehrt und einen Link sieht, den er bereits besucht hat, über das Attribut vlink.

Sämtliche Parameter können beliebig angeordnet und auch beliebig verwendet oder weggelassen werden. Ein vollständiger <BODY>-Tag mit allen Farbangaben hätte dann folgende Struktur:

<BODY bgcolor=[farbe] text=[farbe] link=[farbe] vlink=[farbe] alink=[farbe]>

Dabei kann die Angabe der Farbe in Hochkommata eingeschlossen oder auch ohne Hochkommata notiert werden. Wie auch alle innerhalb der Webseite individuell zu setzenden Farbangaben kann man unter HTML Farben auf zwei Arten angeben:

- Angabe eines RGB-Wertes (RGB = Rot/Grün/Blau-Wert) für eine Farbe in hexadezimaler Form.
- Angabe eines expliziten Farbnamens in englischer Schreibweise.



Das Hexadezimalsystem bedeutet eine Darstellung von 16 Werten in nur einem Zeichen. Dabei sind die Zahlen von 0 bis 9 identisch zum dezimalen Zahlensystem. Die dezimale Zahl 10 wird hexadezimal durch den Buchstaben A, 11 durch B, 12 durch C, 13 durch D, 14 durch E und 15 durch F dargestellt. Die hexadezimale Zahl 10 entspricht also dezimal dem Wert 16. Im Anhang finden Sie eine kleine Umrechnungstabelle zwischen hexadezimalen und dezimalen Zahlen.

Bei der Angabe der RGB-Werte in Hexadezimalform müssen die Farben aus den drei Grundfarben Rot, Grün und Blau über ihre jeweiligen Anteile (in der additiven Farbmischung) zusammengestellt werden. Dabei ist jede hexadezimale Farbdefinition sechsstellig und hat das Schema

#XXXXXX

Hinter dem Gatter # folgen zuerst zwei Stellen für den Rotanteil, dann zwei Stellen für den Grünanteil und die letzten beiden Stellen für den Blauanteil. Die Angabe der RGB-Werte in Hexadezimalform ist Browser-unabhängig und es stehen im Prinzip ca. 16,7 Millionen Farbnancen zur Verfügung (das ergibt sich daraus, das bei 6 Stellen und jeweils 16 verschiedenen Werten dort 16 hoch 6 Kombinationsmöglichkeiten vorhanden sind = 16.777.216). Die nachfolgende Tabelle gibt die hexadezimale Farbkodierung für 16 Standardfarben an, welche auch bei einer Bildschirmfarbeinstellung von nur 16 Farben korrekt angezeigt werden sollten:

Farbe	Rotwert	Grünwert	Blauwert
Blau	00	00	FF
Dunkeltürkis	00	80	80
Gelb	FF	FF	00
Grau	80	80	80

Grün	00	80	00
Hellblau	00	FF	FF
Hellgrün	00	FF	00
Lila	80	00	80
Marineblau	00	00	80
Olivgrün	80	80	00
Pink	FF	00	FF
Rostbraun	80	00	00
Rot	FF	00	00
Schwarz	00	00	00
Silber	C0	C0	C0
Weiß	FF	FF	FF

Tabelle 3.2: Farbcodes.

Nachteile der hexadezimalen Farbangabe sind, dass man sich mit dem hexadezimalen Zahlensystem auskennen und einzelne Farben über die jeweiligen Rot-, Grün- und Blauanteile zusammenstellen muss. Dazu kommt, dass 16,7 Millionen Farbnuancen im Web oft keinen Sinn machen, denn beileibe nicht alle Rechner sind mit einer entsprechender Grafikkarte ausgestattet, und vor allen Dingen haben viele Anwender eine niedrigere Farbtiefe eingestellt (was die Rechnerleistung verbessert). Deshalb gibt es die Möglichkeit der direkten Eingabe von standardisierten Farbnamen, die weitgehend den normalen englischen Farbbezeichnungen entsprechen. Die in der hexadezimalen Schreibweise gerade notierten 16 Standardfarben können auch so angegeben werden:

Farbe	Farbangabe
Blau	blue
Dunkeltürkis	teal
Gelb	yellow
Grau	gray
Grün	green
Hellblau	aqua
Hellgrün	lime
Lila	purple
Lilaton	fuchsia
Marineblau	navy
Olivgrün	olive
Rostbraun	maroon
Rot	red
Schwarz	black
Silber	silver
Weiß	white

Tabelle 3.3: Farbnamen

Natürlich gibt es nicht für 16,7 Millionen Farbnuancen einen Namen, aber mehr als diese 16 Farbnamen. Viele davon sind jedoch nicht standardisiert und in der Darstellung Browser-abhängig. Wenn der Browser einen Farbnamen überhaupt nicht kennt, wird er die Farbe nicht darstellen und eine Defaultfarbe verwenden. Das gilt auch bei hexadezimalen Angaben, wo die in der Wertigkeit naheliegendste Farbe gewählt wird, wenn die exakte Abstufung nicht dargestellt werden kann.

Schauen wir uns nun für den <BODY>-Tag einige vollständige Farbspezifizierungen an, wobei in den verschiedenen

Beispielen hexadezimale Angaben mit direkten Farbangaben gemischt, mal Hochkommata verwendet werden und mal nicht und die Reihenfolge der Parameter vertauscht wird.

Eine Variante sieht so aus:

```
<BODY bgcolor=white text=black link=red alink=yellow vlink=white>
```

Eine alternative Darstellung:

```
<BODY text="#000000" bgcolor="#FFFFFF" link="#FF0000" alink=yellow vlink=blue>
```

Eine Webseite kann zu den globalen Farbangaben ein gekacheltes Hintergrundbild verwenden, das den gesamten Anzeigebereich des Browsers ausfüllt. Diese Angabe erfolgt über eine weitere Erweiterung des <BODY>-Tags, mit dem Parameter background. Mit dieser Erweiterung laden Sie jedes Mal, wenn Ihre Webseite in einen Browser geladen wird, das Hintergrundbild von dem angegebenen Ort. Hintergrundbilder überdecken die Hintergrundfarbe, aber bei transparenten Bildern kann die Hintergrundfarbe durchscheinen. Falls die Anzeige von Grafiken im Browser deaktiviert ist, kommt sowieso die Hintergrundfarbe zum Tragen.

Bevor wir ein Hintergrundbild konkret einbinden können, müssen ein paar Hintergründe zu den im WWW erlaubten Bildformaten sowie der Weg der Referenzierung darauf behandelt werden. Diese Ausführungen gelten wie die Ausführungen zu den Farben dann analog bei Grafikreferenzen an einer lokalen Stelle in der Webseite.

Im WWW gibt es zwei Bildformate, die in jedem Browser verstanden werden:

- Das GIF-Format (Dateierweiterung .GIF), welches auf den Online- Anbieter CompuServe zurückgeht und 256 Farben darstellt. GIF- Dateien werden schichtweise aufgebaut, was dazu führt, dass der Besucher einer Webseite bereits recht schnell ein Bild sieht, allerdings in niedriger Qualität. Mit jedem ankommenden Datenpaket wird Schicht für Schicht das Bild in der Qualität verbessert. Eine Grafik im GIF-Format kann mit transparentem Hintergrund erstellt werden und es lassen sich Animationen in Form so genannter animierter GIFs erstellen.
- Das JPEG-Format (allgemeine Dateierweiterung .JPEG und DOS- Dateierweiterung .JPG) kann 16,7 Millionen Farben darstellen und wird zeilenweise im Browser aufgebaut. Das JPEG-Format eignet sich besonders für Grafiken mit sehr großer Farbtiefe (etwa Fotos). Animationen lassen sich mit diesem Format nicht direkt darstellen (aber durchaus mit Hilfe von Scripten).



Es haben sich in der letzten Zeit zwar verschiedene weitere Formate etabliert, aber es gilt immer noch, dass die Mehrzahl der Browser sie nicht oder nur mit einem Zusatzmodul anzeigen kann. Man sollte darauf verzichten, wenn man nicht ohne Notwendigkeit die Zielgruppe seiner Webseite radikal verkleinern will.

Beide Grafikformate haben ein paar Gemeinsamkeiten. Es handelt sich bei beiden um Pixelformate (Beschreibung von jedem Bildpunkt mit allen Angaben), keine Vektorformate (Berechnung von grafischen Formen), und beide sind hochkomprimierte Bildformate.

Die konkrete Einbindung von Hintergrundbildern erfolgt, indem dem Attribut background im <BODY>-Tag der URL einer Bilddatei zugewiesen wird. Dieser hätte dann folgende Struktur:

```
<BODY background="[URL der Grafik]">
```

Der URL ist dabei allgemein zu verstehen. Das bedeutet, die Grafikdatei kann sich auf dem gleichen Rechner befinden, aber auch auf einem beliebigen anderen - möglichst per Netz erreichbaren - Rechner. Schauen wir uns einige Beispiele an.

Beispiel für die Referenz von einer Bilddatei mit Namen raindrop.gif, die sich auf dem WWW-Server www.rjs.de befindet:

```
<BODY background="http:// www.rjs.de/raindrop.gif">
```

Beispiel für die Referenz von einer Bilddatei mit Namen raindrop.gif, die sich im gleichen Verzeichnis wie die Webseite selbst befindet:

```
<BODY background="raindrop.gif">
```

Beispiel für die Referenz von einer Bilddatei mit Namen raindrop.gif, die sich in einem Unterverzeichnis (bilder) des Verzeichnisses befindet, wo die Webseite selbst gespeichert ist:

```
<BODY background="bilder/raindrop.gif">
```



Wenn Sie für die Gestaltung von Webseiten geeignete Grafiken benötigen, können Sie diese mit jedem qualifizierten Grafikprogramm erstellen und einfach in einem der beiden passenden Formate speichern. Aber das ist vielfach nicht notwendig. Auf fast jedem Rechner werden sich bereits hunderte, wenn nicht tausende von GIF- und JPEG-Dateien befinden (über die Installation von verschiedenen Programmen oder Internetkontakte auf den Rechner gelangt usw.). Sie müssen sich nur passende suchen und für Ihre Zwecke verwenden. Dies funktioniert beispielsweise unter Windows wie folgt:

1. Wählen Sie im **START**-Menü den Eintrag **SUCHEN - DATEIEN/ORDNER...**
2. Wählen Sie im Folgedialog den Namen ***.GIF *.JPG *.JPEG**.
3. Wählen Sie Ihre Festplatte oder gleich den ganzen Arbeitsplatz aus (achten Sie bitte darauf, dass **UNTERGEORDNETE ORDNER EINBEZIEHEN** aktiviert ist).
4. Klicken Sie auf **STARTEN**.

Sie müssen anschließend nur noch eine passende Grafikdatei auswählen.



Bei der Verwendung von fremden Grafiken sollten Sie aber beachten, dass nicht alle Grafiken ohne Verletzung der Urheberrechte zu verwenden sind. Falls Sie sich nicht sicher sind, verzichten Sie auf deren öffentlichen Einsatz im Web. Es gibt genügend freie Grafiken, die Sie ohne Probleme einsetzen können. Zahlreiche CD-Beilagen in Computerzeitschriften, Beispielgrafiken in Grafikprogrammen oder Sammlungen im Internet (einfach mit Suchmaschinen suchen) bieten mehr als genug Auswahl.

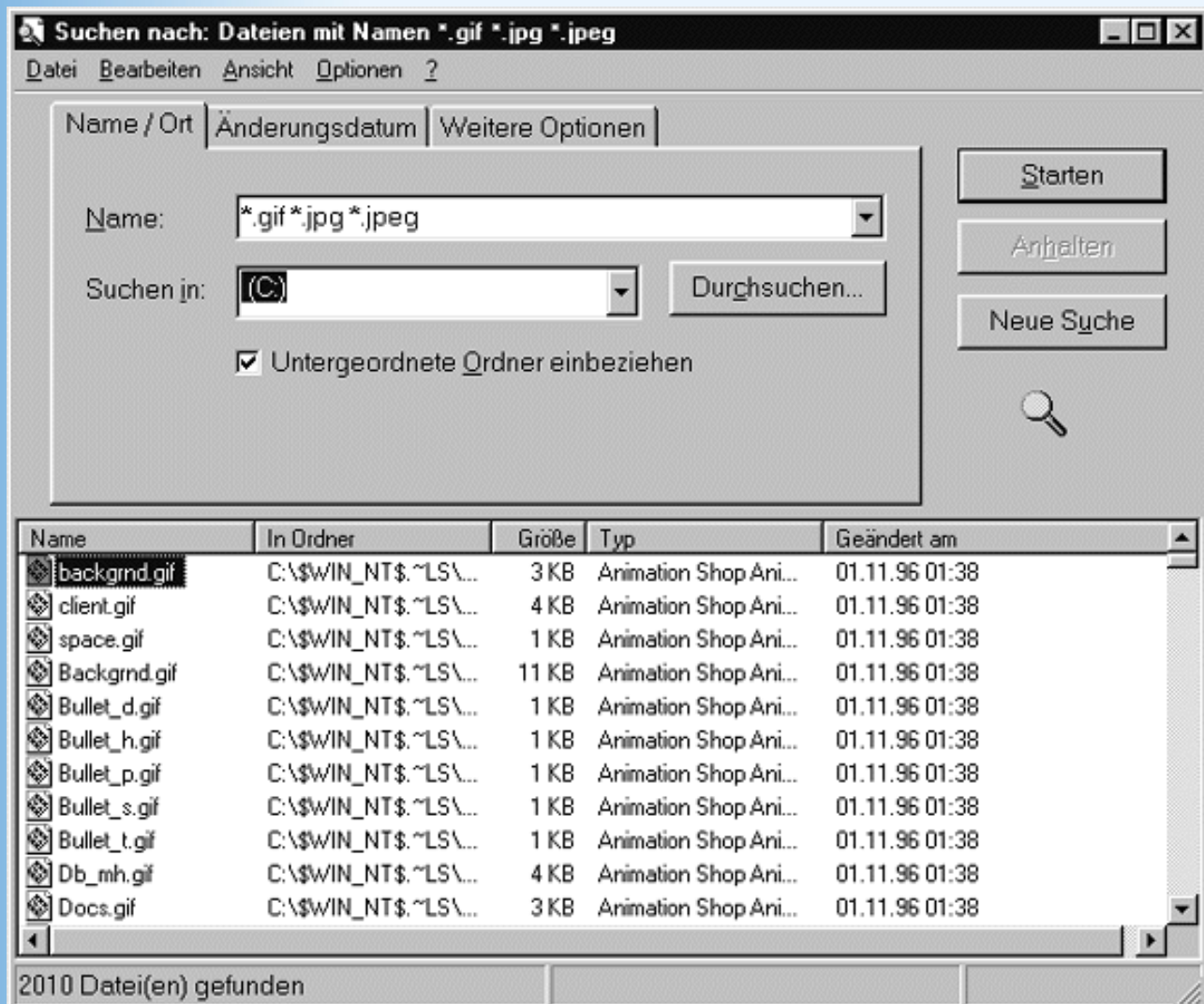


Abbildung 3.8: *Bereits über 2000 Treffer auf einer Festplatte bei der Suche.*

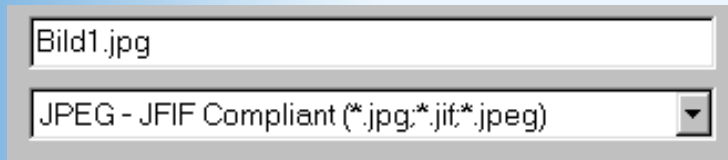


Abbildung 3.9: *Beim Speichern im Grafikprogramm eines der beiden passenden Formate auswählen*

Wir wollen nun die Verwendung von den globalen Farbangaben und einer Hintergrundgrafik in einem vollständigen Beispiel durchspielen. Dabei greifen wir bezüglich ein paar HTML-Anweisungen vor, die aber bald erklärt werden.

Beispiel 3:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY background="smile.gif" bgcolor=cyan text=blue link=green alink=yellow
vlink=white>
<H1 align=center>Herzlich Willkommen!</H1>
<B>
<A href="http://rjs.de/index.html">
Das ist ein Hyperlink</A>
</B>
</BODY>
</HTML>
```



Abbildung 3.10: Zwar kein guter Kontrast, aber es sind deutlich die Effekte zu sehen.

Zusammenfassung

Wir haben heute wichtige Grundlagen für die Erstellung von Webseiten und vor allem die Hintergründe und Grundstrukturen von HTML behandelt. Insbesondere sollte klar sein, dass HTML aus Klartext besteht, der eine Reihe von Steueranweisungen (Tags) kennt. Ein Tag sieht von der Struktur her immer so aus:

<Anweisung>

Ein Browser interpretiert eine solche Anweisung, wenn er sie kennt. Ansonsten wird sämtlicher Text innerhalb der Klammern ignoriert. Es gibt zwei Formen von Tags:

- Einen einleitenden Tag.
- Einen beendenden Tag.

Der einleitenden Tag eröffnet eine Anweisung, während der beendende Tag sie wieder ausschaltet. Beide Tags sehen fast identisch aus, außer einem Zeichen, mit dem der beendende Tag zusätzlich beginnt - dem Slash (Schrägstrich) /. Die beiden

Tags bilden einen Container. Container können verschachtelt werden. Einleitende Tags können auch ohne beendenden Tag vorkommen. Viele Tags sind erst dann sinnvoll einzusetzen, wenn sie über erweiternde Parameter genauer spezifiziert werden. Dabei gibt es zwei Formen von Parametern:

- Parameter mit einer Wertzuweisung.
- Parameter, die bereits einen Wert repräsentieren.

Eine Textdatei wird erst durch ein Grundgerüst und eine für das Betriebssystem als Webseite erkennbare Dateieindung (in der Regel die Endung .HTM, .HTML oder .SHTML) vom Browser so behandelt, wie es für eine Webseite sinnvoll ist. Das korrekte Grundgerüst einer Webseite beinhaltet zwei streng getrennte Bereiche:

- Einen im Anzeigebereich des Browsers unsichtbaren Bereich - den Header einer Webseite.
- Einen im Anzeigebereich des Browsers sichtbaren Bereich - den Body einer Webseite.

Fragen und Antworten

Frage:

Kann ein Browser auch andere Dateien als Klartexte anzeigen?

Antwort:

Ja. So können beispielsweise GIF- und JPEG-Dateien direkt in einem Browser geöffnet werden.

Frage:

Muss man alle HTML-Befehle kennen, wenn man mit JavaScript programmieren möchte?

Antwort:

Nein. Wichtiger ist es, die Philosophie von HTML und dem Web zu verstehen. Diejenigen Befehle, die für einen JavaScript-Programmierer unabdingbar sind, wurden bereits heute oder werden im Laufe des Buchs noch behandelt. Prinzipiell gilt auch in HTML wie in fast allen Sprachen (auch menschlichen Sprachen), dass man mit 10% des Wortschatzes 90% der Standardsituationen meistern kann.

Frage:

In Zusammenhang mit HTML-Anweisungen liest man in den offiziellen Dokumentationen häufig den Begriff »deprecated«. Was bedeutet das?

Antwort:

Eine neue Version einer Sprache bringt nicht nur neue Elemente mit sich, sondern kennzeichnet auch einige Sprachbestandteile der Vorgängerversion als überholt (deprecated). Für diese Elemente gibt es im Rahmen des neuen Sprachstandards eine andere Lösung, die statt dessen verwendet werden soll. Jedoch ist die Verwendung von als deprecated deklarierten Sprachelementen keinesfalls verboten. Die veralteten Sprachelemente werden auch von neuen Browsern weiter verstanden, denn diese sind abwärtskompatibel. Es ist sogar noch seltsamer, aber die Verwendung eines als deprecated erklärten Elementes wird so gut wie nie ein Fehler sein. Die Verwendung des ablösenden neuen Sprachelements in den ersten Monaten nach Verabschiedung einer Sprache jedoch schon, denn bei der überwiegenden Anzahl der Anwender ist dessen Unterstützung noch nicht garantiert. Es kann Monate dauern, bis endlich genügend Anwender neue Browser verwenden, die damit klarkommen. Bei einigen HTML-Elementen wird auch die Empfehlung des W3C bei den Anwendern überhaupt nicht akzeptiert und konsequent auf die als deprecated gekennzeichneten Elemente gesetzt. Der <APPLET>-Tag zum Einbinden von Java- Applets ist so ein Befehl, der eigentlich nicht mehr verwendet werden soll, dessen Ablösungs-Tag aber so gut wie kein Anwender nutzt.

Workshop

Laden Sie die Datei Text.html in den Netscape Navigator. Gehen Sie dann in das Menü **Bearbeiten** und wählen dort den Menüpunkt **Einstellungen...** aus.

Sie gelangen darüber zu diversen Konfigurationsmöglichkeiten des Browsers. Wählen Sie für diesen Workshop die Einstellung **Schriftart** aus und verändern Sie die Schriftgröße und die verwendeten Schriftarten. Sie geben damit Default-Schriftarten an, wie Texte in einer Webseite dargestellt werden.

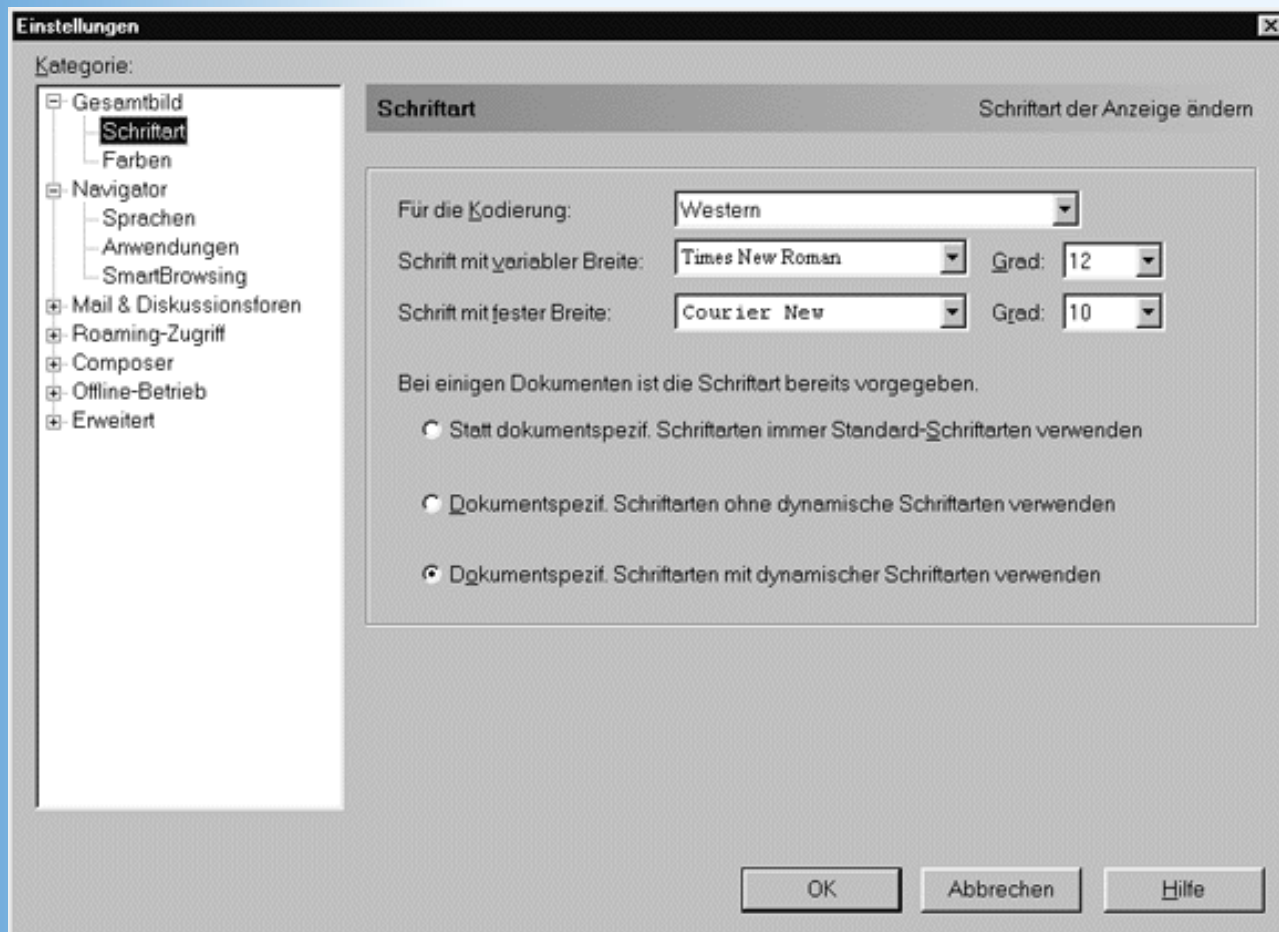


Abbildung 3.11: Die Konfigurationsmöglichkeiten des Netscape Navigator

Bestätigen Sie die Auswahl und betrachten Sie die Veränderungen in der Darstellung der Datei. Experimentieren Sie mit mehreren verschiedenen Einstellungen.

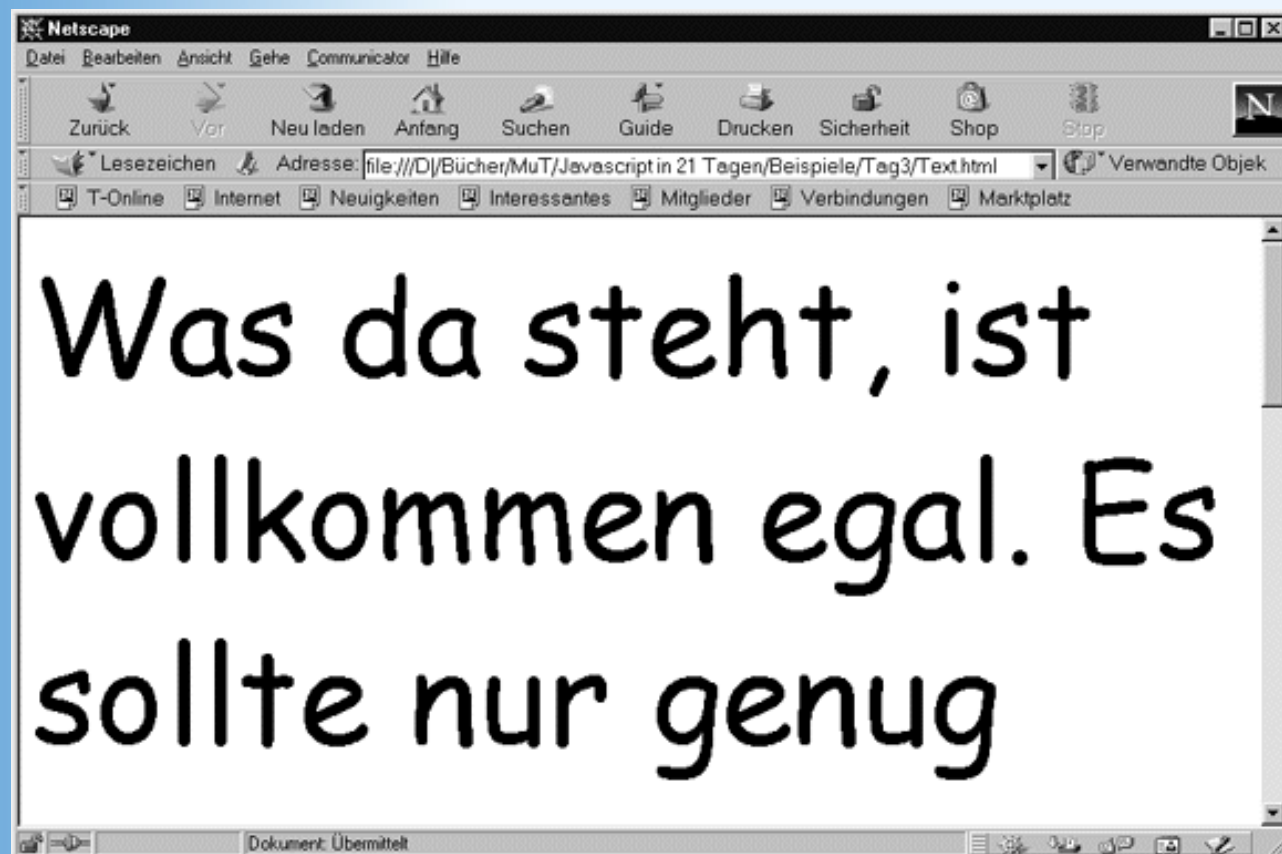


Abbildung 3.12: Veränderter Schrifttyp und Schriftgröße

Danach aktivieren Sie bitte einmal das **Ansicht**-Menü und wählen Schrift verkleinern und Schrift vergrößern mehrfach

hintereinander aus. Damit wird eine Art Zoom-Effekt erzeugt, der sich aber nur auf Texte in einer Webseite auswirkt. Grafiken und ähnliche Elemente bleiben davon unberührt. Das können Sie leicht mit einer Webseite aus dem Internet nachvollziehen.

Sie finden in den Konfigurationsmöglichkeiten unter den Schriftarten auch Konfigurationsmöglichkeiten für Farben. Experimentieren Sie auch hier mit mehreren verschiedenen Einstellungen.

Wenn Sie wollen, können Sie auch einmal die Auflösung ihrer Bildschirmdarstellung verändern.

Der gesamte Workshop soll Ihnen ein Gefühl dafür vermitteln, dass es bei der Darstellung von Inhalten einer Webseite extrem auf die beim Anwender vorzufindende Plattform und die Einstellungen des Browsers ankommt.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Ist HTML eine Programmiersprache?
2. Kann man auf das Grundgerüst einer Webseite verzichten?
3. Werden beim Ende-Tag Parameter angegeben?
4. Muss man sich bei HTML an Groß- und Kleinschreibungsregeln halten?
5. Kann man mit HTML absolute Gestaltungsangaben machen?

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

1

Beachten Sie, dass der Composer alle HTML-Anweisungen klein schreibt.

2

Damit bloß keine Werbefläche ungenutzt bleibt ;-).

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 1

Tag 4

Einbindung von Scripten und Style Sheets in Webseiten

Sheets in Webseiten

Gestern haben Sie wichtige Grundlagen zum Erstellen einer Webseite kennen gelernt. Dazu zählen der Aufbau von Steueranweisungen in HTML und das Grundgerüst einer Webseite. Heute wollen wir zu der Integration von JavaScript in Webseiten kommen. Darüber hinaus sollen auch Referenzen auf Style Sheets zur Sprache kommen, denn dieser Vorgang läuft sowohl technisch als auch logisch ziemlich identisch ab. Es gibt bei beiden Vorgängen die Möglichkeit,

- die Quelltexte direkt in einen Container in einer Webseite zu notieren,
- eine externe Datei mit der Webseite zu verbinden und
- eine Inline-Anwendung zu schaffen.

Wir werden in diesem Tagesabschnitt einige HTML-, Style-Sheet- und JavaScript-Anweisungen verwenden, die heute noch nicht näher besprochen werden. Das holen wir aber ab morgen und dann sukzessive nach. Wir benötigen diese Anweisungen nur, damit die Scripte und Style Sheets nicht in Verbindung mit einer leeren Webseite agieren und Sie die hier zu demonstrierenden Effekte nicht sehen können.

Die Einbindung von Scripten in eine Webseite

Im Prinzip ist es schon sehr lange möglich, in HTML Scripte zu integrieren. Aber erst seit HTML 3.2 können Scripte über standardisierte Wege in Webseiten eingebunden werden. Sie werden dazu entweder über genormte Schritte als Klartext direkt in HTML-Seiten notiert oder dort auf normierte Weise über eine externe Datei referenziert. Dazu gibt es im Wesentlichen den standardisierten HTML-Tag `<SCRIPT>`, der den Beginn eines Script-Containers definiert. Das Innere des Containers wird gegenüber dem HTML-Interpreter geschützt, während der Tag selbst samt seiner Attribute noch Bestandteil des HTML-Codes ist. Im Inneren des Containers wird bei einem entsprechend Script-fähigen Browser der Script-Interpreter aktiv.

Man kann sich das so vorstellen (ein Vergleich mit einer Firma): Wenn eine Webseite geladen wird, werden die jeweiligen Zeichen so lange von oben nach unten von der HTML-Abteilung bearbeitet, bis der `<SCRIPT>`-Tag entdeckt wird. Daraufhin wird der Fall an die zuständige Script-Abteilung weitergeleitet, die den Vorgang so lange bearbeitet, bis das Ende des `<SCRIPT>`-Containers erreicht wird. Anschließend übergibt die Script-Abteilung den Vorgang wieder an die HTML-Abteilung, welche den Hergang weiter bearbeitet.

Eine Script-Referenz kann im Prinzip an jeder Stelle in der Webseite notiert werden, so lange sie innerhalb des äußersten HTML-Containers - dem vom `<HTML>`-Tag gebildeten Block - steht. Am Beginn des Body-Teils der Seite, am Ende des Body, mitten drin oder gar zwischen Header und Body. Meist wird sie jedoch im Header der Seite oder direkt dahinter zu finden sein. Das hat mehrere Gründe. Einmal ist es bezüglich einer Wartung leichter, wenn man die Informationen zusammen verwaltet, die logischen zusammengehören. Logisch gehören Scripte nicht zu den darstellbaren HTML-Elementen. Deshalb werden bei einer Platzierung im Header oder direkt dahinter Layout und Hintergrundaktionen besser getrennt. Nebenbei wird das Auffinden von Scripten erleichtert, wenn Code immer zentral außerhalb des meist recht umfangreichen Body notiert wird. Das entscheidende Argument ist jedoch der Ladevorgang für eine Webseite - von oben nach unten.

Um ein wenig vorzugreifen: Scriptschritte können direkt in den Scriptcontainern notiert werden. Solcher Code wird ausgeführt, wenn die entsprechenden Anweisungen beim Laden der Webseite vorgefunden werden. Der Code kann damit auch nur genau einmal ausgeführt werden. Anwendungen für eine solche Vorgehensweise sind hauptsächlich Initialisierungen oder dynamische Generierungen von statischen Webseiten.

Der andere Weg zur Notation von Scripten ist der, dass Anweisungen über Funktionen zusammengefasst und dann gemeinsam über einen Namen aufgerufen werden. (Es müssen nicht unbedingt mehrere Anweisungen zusammengefasst werden. Es kann auch nur eine Anweisung in einer Funktion notiert werden.) So als Funktion gekennzeichnete Code wird erst dann ausgeführt, wenn an irgendeiner anderen Stelle im HTML-Code der Name der Funktion aufgerufen wird. Funktionen sind im gesamten Dokument verfügbar und können bei Bedarf immer wieder aufgerufen werden.

Wenn nun eine JavaScript-Funktion beim Laden einer Webseite direkt am Anfang des Body benötigt und sie dort aufgerufen wird, ihre konkrete Programmierung aber erst am Ende der Webseite zu finden ist, kann es zu Problemen kommen. Besonders im Internet, wo die Übertragung einer Webseite immer wieder ins Stocken kommen kann. Deshalb notiert man die Programmierung einer Funktion besser vor deren Aufruf. (Obwohl es wirklich nicht zwingend ist. Wenn der Quelltext da ist, findet der Interpreter bei einem Aufruf auch im Quelltext weiter unten notierte Funktionen.)

Der <SCRIPT>-Tag kann ohne irgendwelche Attribute verwendet werden. Etwa so:

```
<SCRIPT>
... irgendwelche Script-Anweisungen
</SCRIPT>
```

In diesem Fall wird der HTML-Interpreter beim Antreffen des Beginn-Tags die Verantwortung an den Default-Scriptinterpreter des Browsers abgeben. In der Regel ist das ein JavaScript-Interpreter, aber darauf kann man sich nicht verlassen. Man sollte sich immer verdeutlichen, dass ein Browser verschiedenste Script- Interpreter aufrufen kann, sofern sie ihm zur Verfügung stehen. Um die verwendete Scriptsprache explizit anzugeben, wird der <SCRIPT>-Tag um das Attribut language (Sprache) erweitert, welches bestimmt, um welche Scriptsprache es sich in dem nachfolgenden Container handelt.

Die Syntax

```
<SCRIPT language="JavaScript">
```

informiert den Browser, dass innerhalb des Containers ein JavaScript folgt. In einigen Browserversionen kann sogar der ursprüngliche Name von JavaScript - LiveScript - stehen. Dies ist aber selten sinnvoll und wird nicht von allen Browsern unterstützt. Genauso ist die Angabe von JScript zu sehen.

Was aber gelegentlich Sinn macht, ist die Angabe einer konkreten Version von JavaScript. Dies ist zweckmäßig, um Fehlern mit inkompatiblen Browsern vorzubeugen. Wenn einfach nur die Sprache allgemein spezifiziert wird, wird jeder Browser, der diese Sprache versteht, das Script abarbeiten. Sofern dann dort Befehle aus einer Sprachversion genutzt werden, die der Browser nicht unterstützt, kann es zu Komplikationen kommen. Etwa die Kombination JavaScript-Befehle der Version 1.2 und als Browser der Internet Explorer 3.0 oder JavaScript-Befehle der Version 1.3 und Netscape Navigator 3. Oder explizite JavaScript-1.2-Befehle in Verbindung mit dem Opera 3.6. Über die explizite Versionsangabe kann verhindert werden, dass die betroffenen inkompatiblen Browser die Scripte ausführen. Bei JavaScript werden die Versionen so angegeben:

```
<SCRIPT language = "JavaScript1.0">
<SCRIPT language = "JavaScript1.1">
<SCRIPT language = "JavaScript1.2">
<SCRIPT language = "JavaScript1.3">
```

Wenn man das Verhalten mit verschiedenen Script- Containern für verschiedenartige Browser koppelt, kann man sehr qualifiziert differenzieren. Wir werden im Rahmen unserer Beispiele auf die explizite Angabe der JavaScript-Version verzichten, aber ein Beispiel zum Differenzieren von Browsern und der JavaScript-Version zeigen.



Die Versionsnummer muss ohne Leerzeichen angefügt werden! Es spielt jedoch keine Rolle, ob Javascript oder JavaScript (meist gewählte Variante) oder javascript als Wert genommen wird, d.h. Groß- und Kleinschreibung sind hier egal. Das ist klar, wenn man sich vergegenwärtigt, dass diese Anweisungen noch zu HTML zählen. Erst in den konkreten im Container notierten Anweisungen wird Groß- und Kleinschreibung eine Rolle spielen - dort gilt JavaScript-Syntax.



Für den Internet Explorer kann man die Syntax `<SCRIPT language= "JScript">` angeben, wenn explizit die Microsoft-JavaScript-Variante verwendet werden soll. Dann aber grenzt man die Netscape-Anwender aus, was selten notwendig ist. Man kann allerdings dieses Verhalten nutzen, um Netscape- und Microsoft-Welten zu trennen, was oft wegen der Inkompatibilitäten sinnvoll ist. Wir werden darauf zurückkommen.

Man ist bei der Angabe einer Scriptsprache wie gesagt nicht auf JavaScript oder Abarten davon beschränkt. Jede im Browser unterstützte Scriptsprache lässt sich nach dem gleichen Schema referenzieren. Die Angabe `<SCRIPT language="VBScript">` legt beispielsweise VBScript als Scriptsprache in dem Container fest. Die konkrete Syntax für den Script-Container sieht also schematisch immer so aus:

```
<SCRIPT language="[Sprache]">
... irgendwelche Script-Anweisungen
</SCRIPT>
```

Der `<SCRIPT>`-Container beinhaltet also im Inneren JavaScript-Anweisungen, mit denen ein HTML- Interpreter nichts anfangen kann. Wenn ein Browser JavaScript versteht, wird von diesem deshalb der Vorgang an den JavaScript-Interpreter weitergegeben, aber nicht-JavaScript-fähige Browser haben mit dem in dem Script-Container notierten Code Probleme. Zwar wird der `<SCRIPT>`-Tag samt Abschluss-Tag auf Grund des Prinzips der Fehlertoleranz von diesen ignoriert. Da er jedoch nicht verstanden wird, werden die darin notierten Script-Befehle als HTML-Code interpretiert (der Browser weiß ja nichts von Scripten). In der Regel heißt dies, die Scriptanweisungen werden als Klartext verstanden, der einfach in den Anzeigebereich des Browsers geschrieben wird. So gut wie nie wünschenswert. Deshalb steht normalerweise als Vorsichtsmaßnahme bei einem Script-Container direkt nach dem einleitenden `<SCRIPT>`-Tag ein öffnendes HTML-Kommentarzeichen (`<!--`) - siehe dazu Tag 3. Dieser HTML-Kommentarcontainer wird unmittelbar vor dem abschließenden Tag `</SCRIPT>` mit dem entsprechenden HTML-Kommentarendzeichen (`-->`) wieder geschlossen. Dadurch steht der gesamte Script- Code innerhalb eines HTML-Kommentars. Dies ist wie gesagt nicht zwingend, aber sicherer für den Fall, dass ein Browser die Seite lädt, der keine JavaScripte interpretieren kann.

Der Netscape Navigator reagiert beim Einschluss von Scripten in HTML-Kommentare in manchen Versionen ziemlich gewöhnungsbedürftig. Der innerhalb des `<SCRIPT>`-Containers aktive Script-Interpreter kann mit der Zeichenkombination für das Ende eines HTML- Kommentars nichts anfangen. Genau genommen erkennt er nicht, dass es eine HTML-Anweisung ist, sondern interpretiert -- als JavaScript-Operator (diesen Operator gibt es) und erzeugt damit einen Fehler. Eine unverständliche Eigenheit, denn der Internet Explorer kommt mit dieser Situation hervorragend zurecht. Für den Navigator muss das Ende-Zeichen des HTML- Kommentars vor dem Script-Interpreter mit JavaScript- Kommentarzeichen versteckt werden (das lernen wir noch genauer kennen). Die Anweisung vor dem Ende des `<SCRIPT>`-Containers sieht dann so aus:

```
//-->
```

Das auch gegen JavaScript-unfähige Browser geschützte Grundgerüst entsprechend so:

```
<SCRIPT language="[Sprache]">
<!--
... irgendwelche Script-Anweisungen
//-->
</SCRIPT>
```



Die Problematik betrifft nicht die Browser, wo JavaScript nur deaktiviert ist. Diese werden den Container schon erkennen, nur die darin enthaltenen Anweisungen nicht ausführen.

Für die JavaScript-unfähigen Browser sollte man normalerweise dem `<SCRIPT>`-Container einen weiteren Container folgen lassen:

```
<NOSCRIPT>
Sorry, diese Seite verwendet JavaScript!
</NOSCRIPT>
```

Der Text (der natürlich frei wählbar ist) wird dabei explizit nicht in HTML-Kommentare eingeschlossen. Hier ist die Situation ja eine andere. JavaScript-fähige Browser werden den Tag erkennen und den darin enthaltenen Inhalt ignorieren. JavaScript-unfähige Browser werden den Tag genauso wenig kennen, wie sie den `<SCRIPT>`-Tag kennen. Das ist aber genau

so sinnvoll. Dies bedeutet doch, dass er ignoriert und nachfolgender Text einfach in der Webseite angezeigt wird. Bingo! Das soll ja genau so sein.

Führen wir nun ein kleines Beispiel durch, das ein Script verwendet. Die Anweisung wird direkt beim Laden der Webseite ausgeführt.

Beispiel 1:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language ="JavaScript">
  alert(navigator.appName);
</SCRIPT>
<NOSCRIPT>
Sorry, diese Seite verwendet JavaScript!
</NOSCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Browser.html ab und schauen Sie sich die Datei im Browser an. Experimentieren Sie mit verschiedenen Browsern.

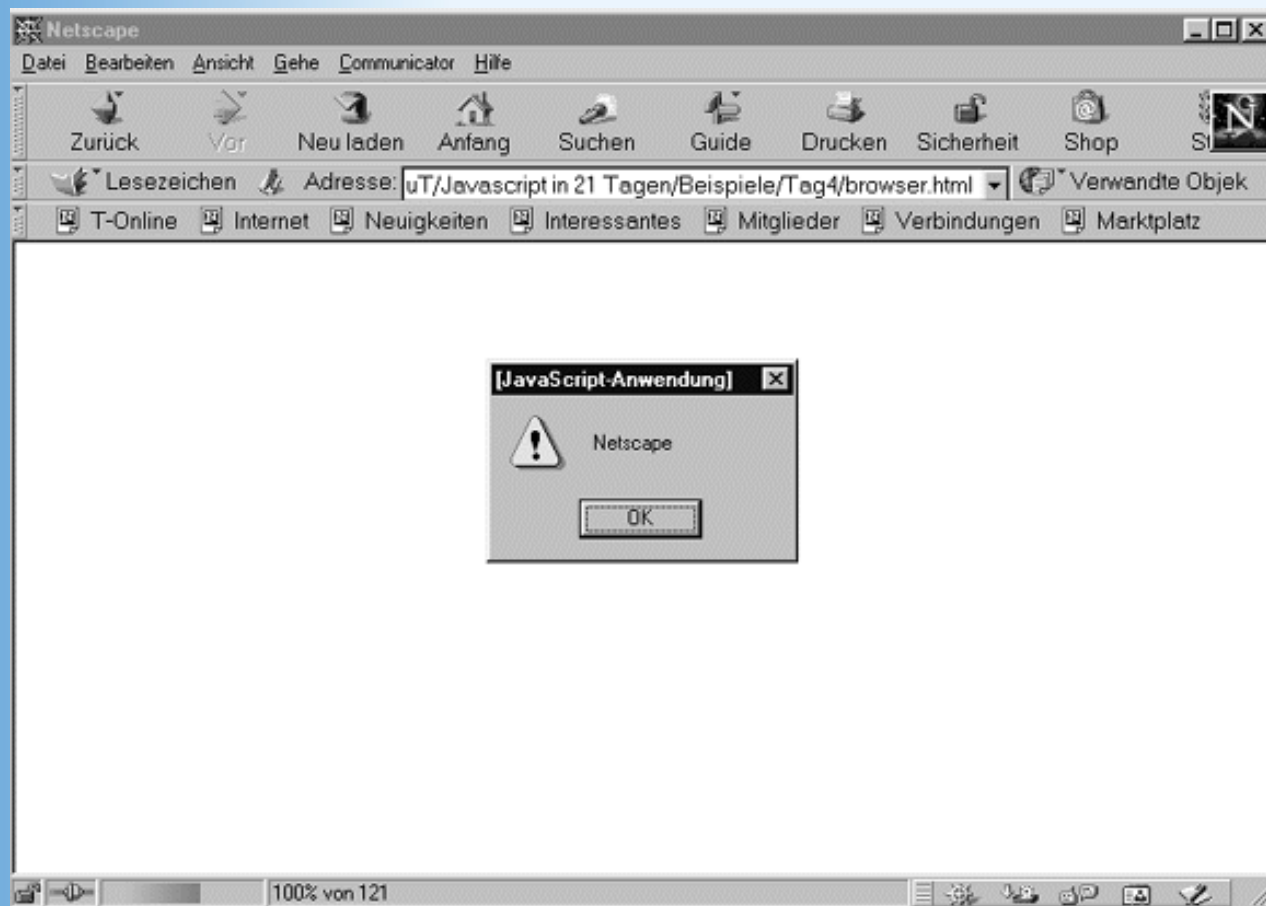


Abbildung 4.1: Die Anzeige im Navigator beim Laden der Webseite

Die alert()-Anweisung öffnet ein Mitteilungsfenster, wo die Browserversion angezeigt wird. Diese wird über navigator.appName abgefragt.

Ein weiteres Beispiel soll die alert()-Anweisung in einer Funktion »einpacken« (die näheren Hintergründe zu Funktionen bzw. Methoden folgen an späteren Tagen) und diese dann erst bei einem Klick auf eine Schaltfläche in der Webseite aufrufen (auch dieses wird demnächst einen großen Raum einnehmen). Beim Laden der Seite wird die Funktion nicht ausgeführt, was viel mehr Flexibilität erlaubt.

Wir sollten nun die Seite einmal testen, wenn keine JavaScript-Unterstützung vorhanden ist. Laden Sie diese in einen älteren Browser oder deaktivieren Sie die JavaScript-Unterstützung in ihrem Browser. Im Netscape Navigator deaktivieren Sie beispielsweise die JavaScript-Unterstützung unter **Bearbeiten, Einstellungen..., Erweitert**.



JavaScript-fähige Browser, bei denen die JavaScript-Unterstützung nur deaktiviert ist, werden sich bezüglich des <NOSCRIPT>-Tags genauso wie JavaScript-unfähige Browser verhalten. Sie können das gerne testen. Wir werden in den folgenden Beispielen bis auf wenige Ausnahmen auf den <NOSCRIPT>-Container verzichten.

Beispiel 2:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function testeBrowser()
{
    alert(navigator.appName);
}
</SCRIPT>
<BODY>
<FORM>
<INPUT type="button" value="Prüfe" onClick="testeBrowser()">
</FORM>
</BODY>
</HTML>
```

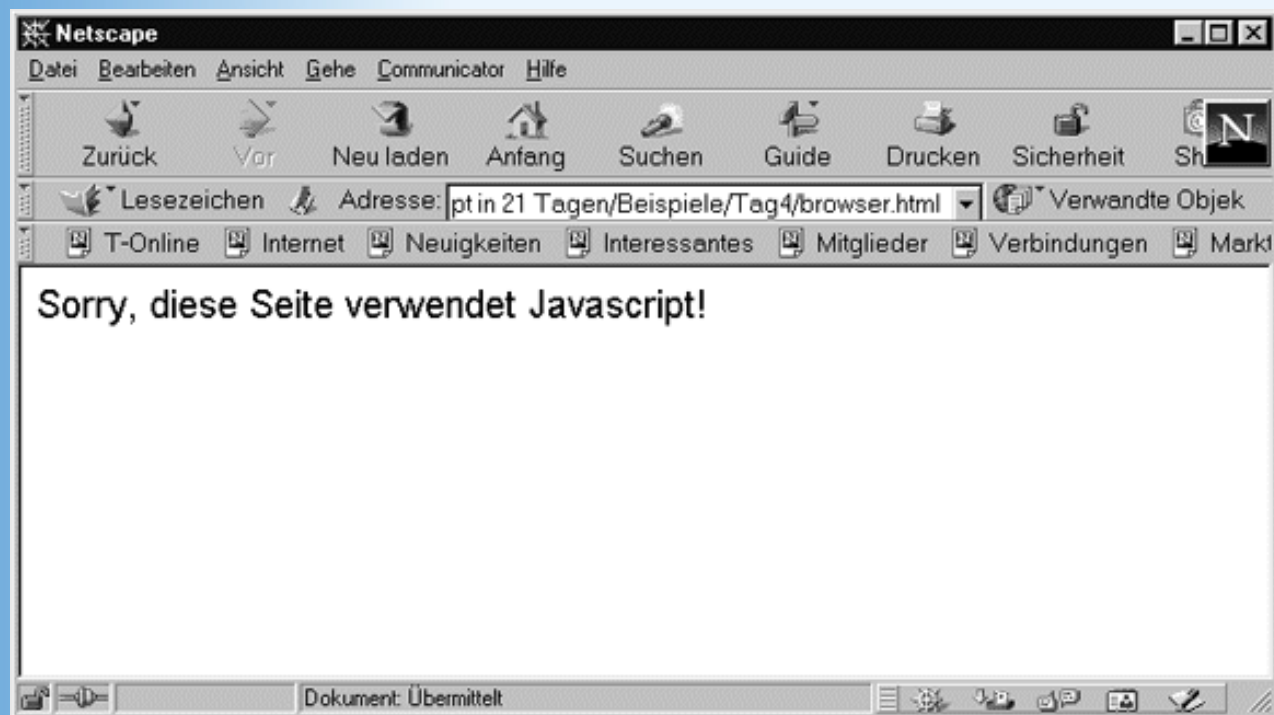


Abbildung 4.2: Die Anzeige im Navigator bei deaktiviertem JavaScript

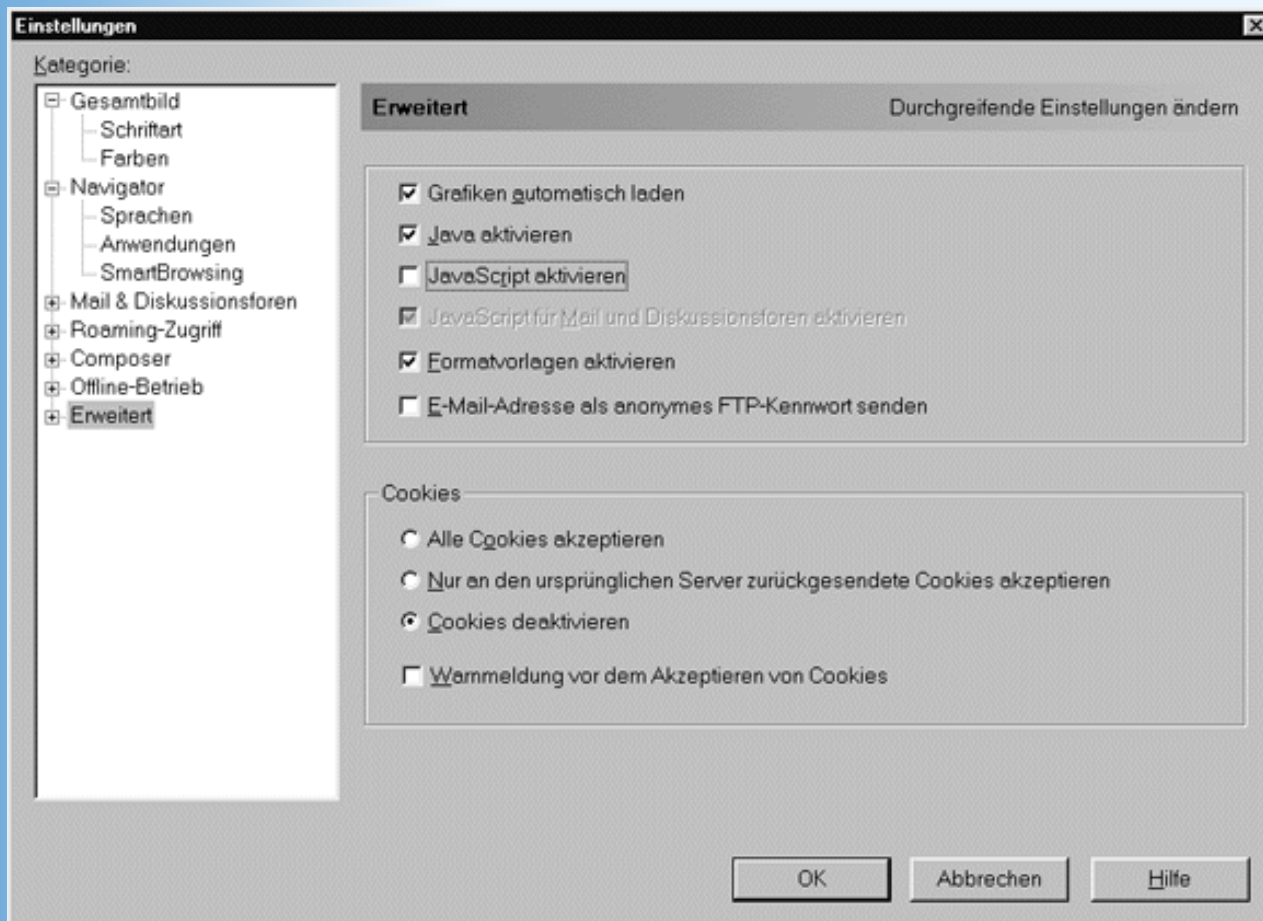


Abbildung 4.3: Deaktivieren von JavaScript im Navigator

Speichern Sie den Text bitte unter dem Dateinamen Browser2.html ab und schauen Sie sich die Datei im Browser an. Klicken Sie auf den Button und experimentieren Sie mit verschiedenen Browsern.

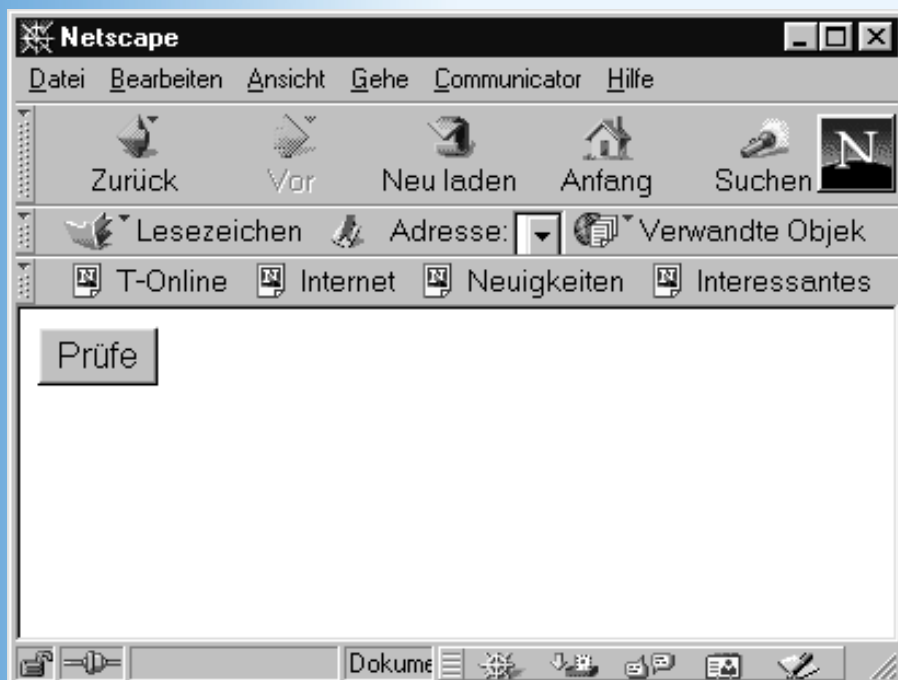


Abbildung 4.4: Klick auf den Button führt die JavaScript-Funktion aus



Die Anzahl der Scriptcontainer in einer Webseite ist nicht beschränkt. Es lassen sich mehrere <SCRIPT>-Container in einem Dokument platzieren. Es gibt ein paar sinnvolle Anwendungen, warum nicht alle Scripts in einem Container an einer Stelle notiert werden. Wichtigster Grund ist die Verwendung von

verschiedenen Scriptsprachen oder -versionen innerhalb einer Webseite. Oder man möchte Funktionalitäten aus verschiedenen Quellen in einer Webseite nutzen. Eine weitere Anwendung für verschiedene <SCRIPT>-Container ist die damit mögliche logische Strukturierung von Inhalten. Richtig sinnvoll sind aber mehrere <SCRIPT>-Container vor allem in Verbindung mit der externen Notation von Scripten, auf die wir jetzt eingehen wollen.

Externe Notation von Scripten

Neben der Notation von Scripten in den HTML- Quelltext selbst lassen sich JavaScripte unter HTML 4.0 und seit JavaScript in der Version 1.1 in einer oder mehreren externen Dateien ablegen. In dem Fall enthält nicht die Webseite den konkreten JavaScript-Code, sondern die externe(n) Datei(en). In der Webseite wird nur der Aufruf einer Funktion durchgeführt. Um die Verbindung zwischen der Webseite und einer externen Datei zu schaffen, muss in der Webseite der Weg zu dieser Datei angegeben werden. Dies erfolgt mit einer Referenz, welche dann die Möglichkeit zum Laden des JavaScript-Codes eröffnet, wenn eine Funktion aufgerufen wird.

Die Referenz sieht für Scripte allgemein wie folgt aus:

```
<SCRIPT language="[Sprache]" src="[URL der separaten Script-Datei]">
</SCRIPT>
```

Im Fall von JavaScript gilt also das:

```
<SCRIPT language="JavaScript" src="[URL der separaten JavaScript-Datei]">
</SCRIPT>
```

Mit dem Attribut src wird der URL der separaten Datei angegeben. Dabei gelten beim Referenzieren von separaten JavaScript-Dateien die üblichen Regeln für URLs. Die separate JavaScript-Datei mit dem Quellcode muss wie HTML-Dateien auch eine reine ASCII-Datei sein und ausschließlich JavaScript-Code enthalten. Es gibt kein Grundgerüst, wie bei einer HTML-Seite. Üblich ist die Dateierweiterung .js, aber das ist nicht zwingend. Optional kann das Attribut type="text/JavaScript" angegeben werden, was einen so genannten Mime-Typ spezifiziert.



MIME steht für Multipurpose Internet Mail Extensions und bedeutet einen Internet-Standard zur Spezifizierung von Dateitypen bei der Kommunikation zwischen Servern und Browser im World Wide Web. Sowohl der Server als auch der Browser kennt bestimmte Dateitypen. Beim Übertragen vom Server zum Browser wird über das HTTP-Protokoll der MIME-Typ mit übertragen. Aufgrund seiner Liste mit MIME-Typen kann der Browser eine Datei eines bekannten Typs korrekt behandeln. Werden unbekannte Dateitypen geladen, kann das Programm nicht direkt damit umgehen, sondern muss die Datei speichern oder über zusätzliche Informationsquellen (etwa einen Benutzerdialog) nach der Behandlungsweise forschen.

Die Verwendung von MIME-Typen umgeht zugleich das Problem, dass viele Dateierweiterungen von mehreren Programmen bearbeitet werden können (etwa .bmp, was unter Windows von zahlreichen Grafikprogrammen bearbeitet werden kann). MIME-Typen geben eindeutig das zu verwendende Programm an. MIME-Typen werden nach folgendem Schema angegeben:

Hauptkategorie/Unterkategorie

Hauptkategorien sind etwa text, image oder audio. Unterkategorien von text sind beispielsweise plain (eine reine Textdatei) oder html (eine HTML-Datei). Unterkategorien von image sind beispielsweise gif oder jpeg. Im Anhang finden Sie eine Tabelle mit üblichen MIME-Typen.



Der <SCRIPT>-Container muss für den Fall einer Einbindung von externen Dateien leer sein. Alle JavaScript-Anweisungen befinden sich in der referenzierten Datei oder in mehreren, einzelnen referenzierten externen Dateien!

Externe Scriptdateien bedeuten zwar etwas mehr Verwaltungsaufwand, sind jedoch oft sinnvoll. Folgende Argumente sprechen dafür:

- Gleiche JavaScript-Funktionalität soll in verschiedenen HTML-Dateien verwendet werden. Statt gleichen Code mehrfach in verschiedenen Dateien zu notieren, muss nur jeweils eine Referenz auf eine zentrale Datei angegeben

werden.

- Nur eine zentrale Stelle ist bei Wartungen des Codes erheblich sinnvoller. Wenn die Funktionalität mehrfach verwendet wird, muss dort keine Veränderung vorgenommen werden.
- Die Orientierung ist in einer reinen JavaScript-Datei einfacher.
- Die über das Netz zu übertragende Datenmenge wird gegenüber dem Fall, wo sich gleicher Code in mehreren HTML-Dateien befindet, reduziert.
- Analysen der Scripte durch einfaches Ansehen des Quellcodes der Webseite sind nicht mehr so einfach möglich. Nicht immer ist gewünscht, dass man selbst lange an Scripten herumfeilt und dann jeder den Quellcode für sich verwendet.
- Eine logische Trennung von Dokumentenbestandteilen ist gewährleistet.
- Der Container wird von JavaScript-inkompatiblen Browsern ignoriert, und da im Inneren keine Anweisungen stehen, kann es nicht zu Problemen kommen.

Führen wir nun ein kleines Beispiel durch, das eine externe Scriptdatei verwendet. Beachten Sie, dass hier zwei Funktionen verwendet werden, die beide in der Scriptdatei abgelegt werden.

Beispiel 3:

Erstellen Sie zuerst eine Datei, in der die JavaScript- Funktionen definiert sind. Geben Sie im Editor den nachfolgenden Quelltext ein:

```
function willkommen()  
{  
  alert("Herzlich Willkommen");  
}  
function bey()  
{  
  alert("Und tschüss");  
}
```

Speichern Sie den Text bitte unter dem Dateinamen extern.js.

Erstellen Sie nun die Datei, in der die Datei extern.js verwendet wird. Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<html>  
<head>  
<SCRIPT language="JavaScript" src="extern.js">  
</SCRIPT>  
</head>  
<body onLoad="willkommen()" onUnload="bey()">  
</body>  
</html>
```

Laden Sie die Datei in einen Browser, nachdem Sie sich zuvor eine andere Datei angesehen haben. Verlassen Sie dann den Browser über die **Zurück**-Schaltfläche. Sowohl beim Laden als auch dem Verlassen der Webseite wird eine JavaScript-Funktion aufgerufen, die jeweils eine unterschiedliche Meldung auf dem Bildschirm anzeigt.



Die Verwendung eines externen JavaScripts darf nicht so verstanden werden, dass alle dort definierten Funktionen auch verwendet werden müssen. Eine externe Datei kann beliebig viele Funktionen enthalten (und auch Variablen, welche wir noch nicht benutzen), von denen bei Bedarf die angegebene Funktion geladen und verwendet wird. Die nicht benötigten Funktionen beeinträchtigen die Webseite in keiner Weise. Es sollte aber beachtet werden, dass eine externe JavaScript-Datei nicht unnötig groß werden sollte. Eine Aufteilung auf mehrere externe Dateien ist u.U. sinnvoll und wird einfach dadurch möglich, dass die einzelnen, leeren Container mit den verschiedenen Referenzen gemeinsam in der Seite notiert werden.



Die Verwendung von externen JavaScript-Dateien kann ein Problem bedeuten, wenn auf dem Server der Mime-Type text/JavaScript für Dateien mit der Endung der externen Datei (etwa .js) in der Konfiguration des

Web-Servers nicht angegeben ist. Dann muss der JavaScript-Text immer direkt in eine HTML-Datei geschrieben werden. Auf jeden Fall sollte die JavaScript-Funktionalität nach dem Laden auf den Server getestet werden.

Der Inline-Aufruf von Script-Anweisungen

Scriptaufrufe lassen sich auch direkt als Attribut in einen HTML-Tag hineingeschrieben. Das nennt man Inline- Aufruf oder auch Inline-Referenz, die in der Regel in Verbindung mit einer Steueranweisung verwendet wird, die auf einen Klick reagieren kann (etwa ein Hyperlink). Letzteres muss so vorsichtig beschrieben werden, weil insbesondere Netscape-Browser diese Variante nur sehr eingeschränkt unterstützen und auch andere Browser damit ihre Probleme haben können. Der Internet Explorer kann diese Technik recht zuverlässig nutzen. Grundsätzlich sieht der Inline-Aufruf eines Scripts syntaktisch so aus:

```
<[verweissensitives Steuerelement] = "[Scriptsprache]:[Scriptanweisung]">  
...  
<[Ende-Tag]>
```

Eine Webseite kommt sogar ganz ohne den <SCRIPT>- Container aus, wenn bei einem Inline-Aufruf nur einzelne vorgefertigte JavaScript-Anweisungen aufgerufen werden.

Führen wir ein Beispiel mit einem Inline-Aufruf durch.

Beispiel 4:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>  
<SCRIPT language="JavaScript">  
function zufall()  
{  
  alert(Math.random());  
}  
</SCRIPT>  
<BODY>  
<A href="JavaScript:zufall()">Klick</A>  
</BODY>  
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Inline.html ab und schauen Sie sich die Datei im Browser an. Klicken Sie mehrfach auf den Hyperlink. Das Beispiel verwendet einen Zufallsgenerator, der ständig neue Zufallszahlen zwischen 0 und 1 generiert (siehe Abbildungen 4.5 und 4.6).

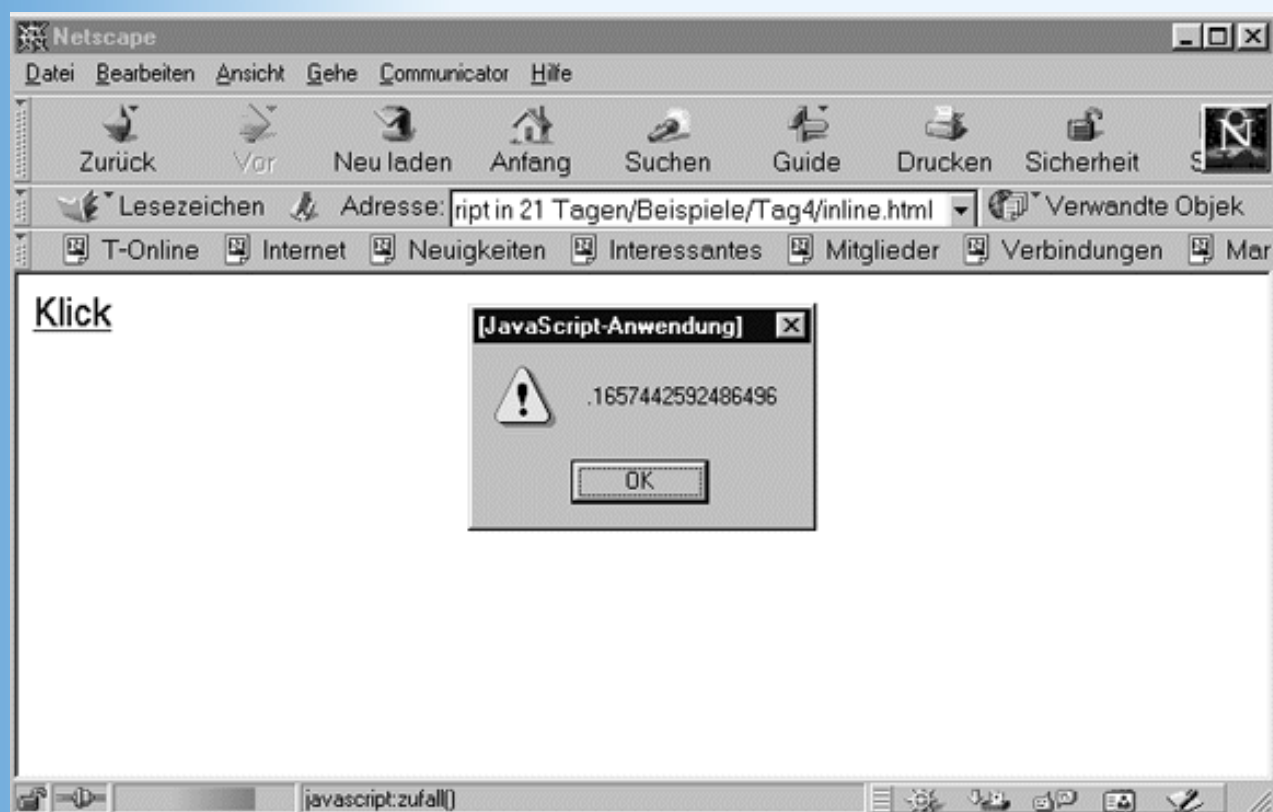


Abbildung 4.5: Ein neuer Klick auf den Link erzeugt eine neue Zufallszahl

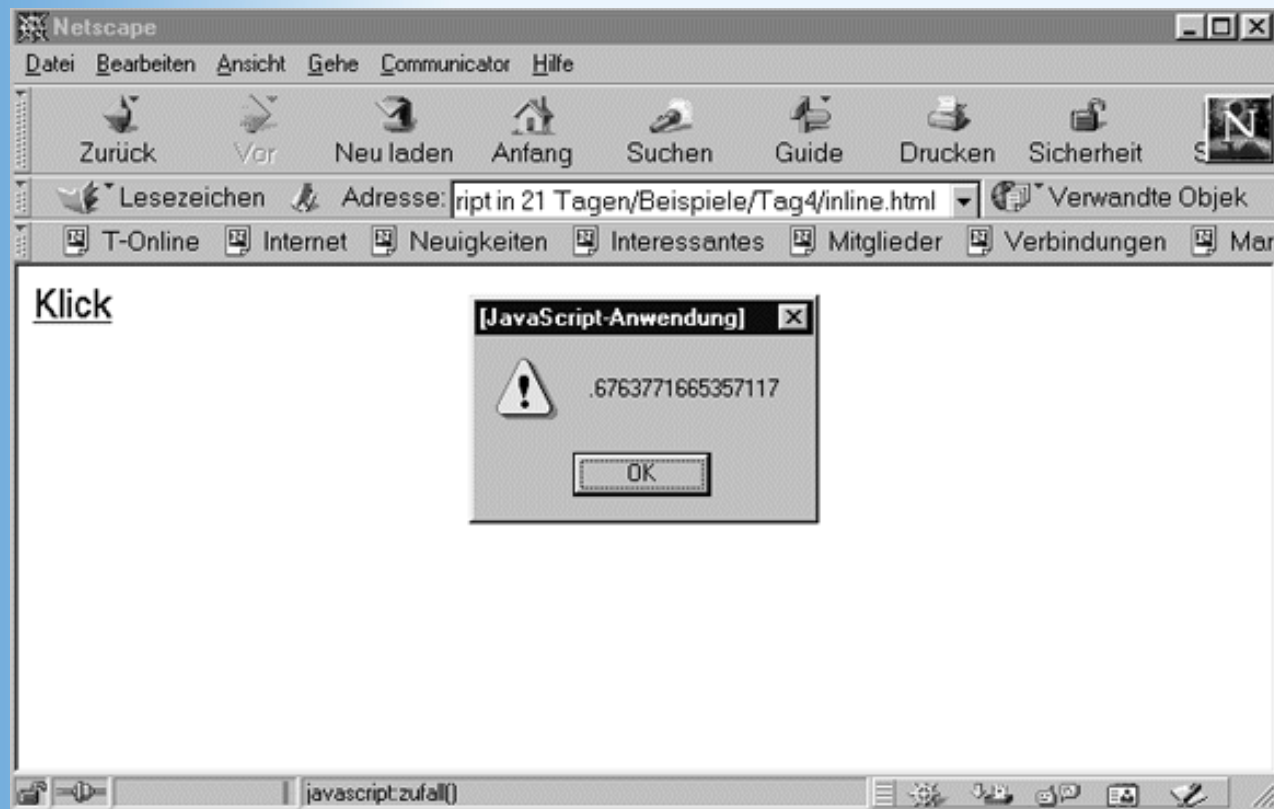


Abbildung 4.6: Ein Klick auf den Link führt die JavaScript-Funktion aus

Spezielle Microsoft-Notation von Scripten für Events in einer HTML-Steueranweisung

Microsoft stellt für den Internet Explorer ein Event-Modell bereit, das auf Ereignisse auf nahezu jedem Element in einer Webseite reagieren kann. Dazu wird der `<SCRIPT>`-Tag dann in Verbindung mit einem `for`- und einem `event`-Attribut eingesetzt. Syntax:

```
<SCRIPT for="[HTML-Steuerelement]"
event="[Eventhandler]" language="[Scriptsprache]">
... irgendwelche Scriptanweisungen
</SCRIPT>
```

Diese Syntax wird vom Netscape Navigator nicht verstanden und bietet sich deshalb hauptsächlich für die Webseiten an, welche nur im Internet Explorer angezeigt werden sollen. Allerdings kann man damit auch einen Trick aufbauen, um Browser-Welten zu trennen - das werden wir in einem späteren Teil des Buchs zeigen.

Die Einbindung von Style Sheets in eine Webseite

Style Sheets oder Stilvereinbarungen bzw. Stilinformationen gehören zu den interessantesten Gestaltungsmitteln im WWW. Vergleichbar sind sie mit den Formatvorlagen, wie sie im Rahmen einer gewöhnlichen Textverarbeitung verwendet werden. Wie auch Scripte gehören sie nicht zu HTML, sondern sind eine eigenständige Technik, die seit HTML 4.0 in Webseiten verankert werden kann. Und zwar auf sehr ähnliche Art wie Scripte. Die Idee von Style Sheets entstand aus mehreren Intentionen:

- HTML entstand aus der Idee, mit möglichst wenig absoluten Regeln Informationen darstellen zu können. Style Sheets sind ein Ansatz, die teils sehr unterschiedlichen Interpretationen von rein aus HTML aufgebauten Webseiten einheitlicher zu gestalten. Entgegen der HTML-Philosophie mit ihren ziemlich großen Interpretationsspielräumen sind Style Sheets in Bezug auf das Layout einer Webseite recht genau festgelegt. Genaue Schriftgröße, exakte Farbangaben, pixelgenaue Position von Elementen usw. sind anzugeben und funktionieren zumindest von der Theorie her. Sofern Browser diese Angaben dann unterstützen (das gigantische Problem), werden damit die Gestaltungsmöglichkeiten gegenüber HTML-Anweisungen erheblich erweitert.
- Durch die immer mehr zunehmenden optischen Gestaltungsanweisungen in einer Webseite verfilzt Information

zunehmend mit Layout. Ein wenig wünschenswerter Zustand, weder in Bezug auf Wartung noch auf Effektivität oder Logik. Style Sheets sollen die Möglichkeit der strikten Trennung von Gestaltungsbefehlen und Informationsträgern bieten und damit diesen Filz wieder entflechten. Damit kann eine klare Aufteilung von Struktur und Layout erreicht werden.

- Die Auslagerung von Layout in eigene Dateistrukturen erlaubt die Verteilung von Aufgaben. Etwa in größeren Projekten.

Style Sheets sind nicht unumstritten und haben sich beileibe noch nicht WWW-weit durchgesetzt. Sie setzen bei der Zielplattform eine erhebliche Kompatibilität zu einem vorgegebenen Standard voraus, den so gut wie immer nur der jeweilige Hersteller eines Browsers garantiert. Zudem müssen für die absolute Kontrolle über bestimmte Elemente wie Schriften auch diese Elemente auf der jeweiligen Zielplattform installiert sein (die auch über Standardnamen angesprochen werden können), die Grafikkarte muss relativ leistungsfähig sein, es muss eine Mindestanzahl von Farben dargestellt werden können, die Auflösung darf nicht zu gering sein. Wer Style Sheets einsetzt, sollte - mehr noch als bei HTML - testen, testen, testen. Und unbedingt eine Seite so aufbauen, dass sie auch ohne die Wirkung von Style Sheets noch Sinn ergibt.

Wir beschäftigen uns im Rahmen dieses Buchs mit Style Sheets, weil der Einsatz dieser Technik im Rahmen einer Webseite hervorragend mit JavaScript gesteuert werden kann und umgekehrt JavaScript erheblich in Bezug auf Layout und Dynamik in der Webseite erweitert wird.

Style Sheets sind keine eigene Sprache, sondern es gibt diverse Sprachen, mit denen Style Sheets erstellt werden können. Die genauen Regeln für die Arbeit mit Style Sheets werden je nach verwendeter Style-Sheet-Sprache etwas differieren, sind aber alle ähnlich. Im Web hat sich hauptsächlich der CSS-Standard (Cascading Style Sheets) durchgesetzt, der vom W3C standardisiert wird. Auf dessen Feinheiten gehen wir an späterer Stelle ein - hier soll nur die Einbindung von Style Sheets in eine Webseite behandelt werden. Dies kann darüber geschehen, dass man Style Sheets direkt in eine Seite einbettet (Linken) oder aus einer externen Datei lädt (Importieren).

Eine externe Lösung hat wieder die gleichen Vorteile wie beim Fall von externen JavaScript-Dateien:

- Gleiche Funktionalität kann in verschiedenen HTML-Dateien verwendet werden. Statt gleichen Code mehrfach in verschiedene Webseiten zu notieren, muss nur jeweils eine Referenz auf eine zentrale Datei angegeben werden.
- Nur eine zentrale Stelle ist bei Wartungen des Codes erheblich sinnvoller. Wenn die Funktionalität mehrfach verwendet wird, muss dort keine Veränderung vorgenommen werden. Das gilt bei der Verwendung von Style Sheets noch mehr als bei Scripten. Denken Sie nur an das Beispiel, wenn die Hintergrundfarbe in einem Projekt global geändert werden muss. Statt unter Umständen in Dutzenden von Webseiten den <BODY>-Tag anpacken zu müssen, kann eine einzige Änderung in der Style-Sheet-Datei das Problem lösen.
- Die Orientierung ist in einer reinen Style-Sheet-Datei einfacher. Das Argument ist bei Style Sheets noch stärker zu berücksichtigen als bei Scripten, da Style Sheets vom Aufbau HTML in einigen Bereichen ähneln. Zudem verschmelzen Stilinformationen über Attribute stark mit HTML-Tags, weshalb man die Techniken auf den ersten Blick schwer auseinander dividieren kann.
- Die über das Netz zu übertragende Datenmenge wird gegenüber dem Fall, wo sich gleicher Code in mehreren HTML-Dateien befindet, reduziert.
- Analysen der Strukturen der gesamten Webseite wird erschwert (Erschwerung von Spionage).
- Insbesondere wird eine der Hauptintentionen für Style Sheets - die logische Entkoppelung von Information und Layout auch auf Dateiebene - konsequent umgesetzt. Wenn das noch in Verbindung mit JavaScript gesehen wird, beinhaltet bei konsequenter Umsetzung die HTML-Datei die Information, die JavaScript-Datei die Logik, Steuerung und Dynamik der Webseite und die Style-Sheet-Datei das Layout (soweit es vollständig entkoppelt werden kann). In der HTML-Datei werden die Bereiche dann zusammengefügt.
- Die externe Style-Sheet-Referenz ist nur ein einzelner Tag und wird von inkompatiblen Browsern ignoriert. Und da es also keinen Container mit unverständlichen Anweisungen im Inneren gibt, kann es nicht zu Problemen kommen.

Eine externe Style-Sheet-Klartextdatei wird bei Stilformatvorlagen nach dem CSS-Standard meist die Erweiterung .css bekommen. Das ist jedoch wie die Erweiterung .js für JavaScript optional.

Eine direkt in eine Webseite geschriebene Stilinformation hat natürlich auch einiges für sich, sonst gäbe es diese Variante nicht. Eine interne Lösung über eingebettete Style Sheets hält alle Informationen über eine Webseite zusammen und erlaubt schneller Änderungen. Außerdem hat man bei kleineren Seiten alles Notwendige direkt im Blick.



Die Verwendung von Style Sheets kann im Header einer HTML-Datei als Metainformation dokumentiert

werden. Dazu wird der `<META>`-Tag mit dem `http-equiv`-Parameter verwendet. Um beispielsweise die Verwendung von Cascading Style Sheets zu dokumentieren, wird im Header die folgende Deklaration notiert:

```
<META http-equiv="Content-Style-Type" content="text/css">
```

Dies ist aber wie gesagt nur eine Dokumentation eines MIME-Typs für die verwendete Style-Sheet-Sprache, keine explizite Referenz.

Direkte Notation von Style Sheets in der Webseite mit dem `<STYLE>`-Container

Die direkte Einbindung von Style Sheets in eine Webseite erfolgt über die Notation eines `<STYLE>`-Containers direkt in den HTML-Code (ein Analogon zu dem `<SCRIPT>`-Container bei Scripten). In dem Inneren des `<STYLE>`-Containers werden - meist in HTML-Kommentare eingeschlossen, damit für ältere Browser die Anweisungen verborgen werden - alle Stilinformationen definiert. Man nennt diese Form der Verknüpfung »eingebettete Formatvorlagen«. Wo das Style Sheet konkret in der HTML-Datei notiert wird, ist wie bei Scripten nicht festgelegt. Jedoch gelten ebenso die gleichen Argumente dafür, Style Sheets möglichst noch vor dem Laden des Body-Abschnitts bereitzustellen. Also im Header oder zwischen Header und Body. Die konkrete Referenz erfolgt mit dem Tag `<STYLE>` wie folgt (CSS):

```
<STYLE type="text/css">
<!--... irgendwelche Stilvereinbarungen ... -->
</STYLE>
```

Im Inneren werden die konkreten Stilinformationen notiert. Fehlt die Typ-Angabe des MIME-Typs, kann die Stilvereinbarung dennoch korrekt umgesetzt werden, wenn CSS die Default-Style-Sheets des Browsers sind. Darauf sollte man sich allerdings nicht verlassen. Die Referenz auf ein internes Style Sheet erfolgt allgemein so:

```
<STYLE type="[Style Sheet-Typ]" >
<!--... irgendwelche Stilvereinbarungen ... -->
</STYLE>
```



Der HTML-Kommentar-Endebefehl kann hier unmaskiert verwendet werden. Die Probleme wie bei dem JavaScript-Interpreter von Netscape tauchen dabei nicht auf.

Das nachfolgende Beispiel verwendet eingebettete Formatvorlagen. Beachten Sie bei diesem und auch den restlichen Beispielen zur Einbindung von Style Sheets, dass wir die Erklärungen der eigentlichen Stilformatsyntax auf die Tage verschieben wollen, wo sie intensiv behandelt werden.

Beispiel 5:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Style-Type" CONTENT="text/css">
</HEAD>
<STYLE TYPE="text/css">
<!--
.meinStil1 {
  font-family: Arial;
  font-size: x-large;
  text-align: center;
}
.meinStil2 {
  font-weight: bold;
  font-style: italic;
  color: #FF0000;
  background: #9BAAE6;
}
-->
</STYLE>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#FF0000" ALINK="#FF0000">
```

```
VLINK="#FF0000">
<H1 class=meinStil2>Herzlich Willkommen</H1>
Diese Seite verwendet Style Sheets
<BR>
<SPAN class="meinStil1">
Etwa hier
</SPAN>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Stileinbind1.html ab und schauen Sie sich die Datei in verschiedenen Browsern an.



Beachten Sie die Punkte vor meinStil1 und meinStil2. Diese sind zwingend notwendig!



Abbildung 4.7: Die Überschrift und die dritte Zeile sind über Style Sheets formatiert

Die Link-Notation auf eine externe Datei

Über die nachfolgende Syntax kann ein Verweis auf ein externes Style Sheet erfolgen. Das nennt man dann ein verknüpftes Style Sheet bzw. eine Link-Notation. Allgemeine Form:

```
<LINK type="[Style Sheet-Typ]" rel=stylesheet
href="[URL einer Datei mit Stilinformationen]"
title="[Name des Style-Sheets]">
```



Beachten Sie, dass im Gegensatz zu dem <STYLE>-Container hier kein Container definiert wird. Es gibt also keinen Ende-Tag.

Eine CSS-Datei wird schematisch so zu einer Webseite gelinkt:

```
<LINK type="text/css" rel=stylesheet
href="[URL einer Datei mit CSS-Informationen]"
```



```
title="[Name des Style-Sheets]">
```

Der Titel ist optional, die Typ-Angabe unter Umständen auch, aber das kann Probleme geben (s.o.). Die restlichen Attribute sind notwendig. Diese Zeile wird am sinnvollsten in den Header der HTML-Seite geschrieben und kennzeichnet eine Style-Sheet-Datei, von wo die in der HTML-Datei verwendeten Stilinformationen bei Bedarf hinzugebunden werden.

Beispiel 6:

Erstellen Sie als erstes die Style-Sheet-Datei:

```
.meinStil3
{
  font-family: Arial;
  font-size: xx-large;
  text-align: right;
}
.meinStil4
{
  font-weight: bold;
  font-style: italic;
  color: #FFFF00;
  background: #911A16;
}
```

Speichern Sie die Datei unter externStil.css. Erstellen Sie dann im Editor die nachfolgende Datei:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Style-Type" CONTENT="text/css">
<LINK type="text/css" rel=stylesheet href="externStil.css">
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#FF0000" ALINK="#FF0000"
VLINK="#FF0000">
<H1 class=meinStil3>
Herzlich Willkommen
</H1>
Diese Seite verwendet Style Sheets
<BR>
<SPAN class="meinStil4">
Etwa hier
</SPAN>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Stileinbind2.html ab und schauen Sie sich die Datei in verschiedenen Browsern an.



Abbildung 4.8: Die Überschrift und die dritte Zeile sind über Style Sheets formatiert

Die Inline-Referenz

Wie auch bei JavaScript gibt es bei Style Sheets die Möglichkeit einer Inline-Referenz. Dies bedeutet die Erweiterung eines gewöhnlichen HTML-Elementes um nur dort gültige Stilinformationen. Über ein zusätzliches Attribut - style (nicht mit dem Tag <STYLE> zu verwechseln) - wird innerhalb des gewünschten HTML- Tags ein Stilattributwert gesetzt. Die Stilinformation gilt ausschließlich innerhalb des mit dem HTML-Befehl definierten Containers (wobei Container im Sinn von Wirkung des Tags zu verstehen ist - das kann auch ohne Ende-Tag gelten). Eine Inline-Deklaration ist dann sinnvoll, wenn ein Element nur ein- oder zweimal verwendet werden soll und nur wenige Stilformatierungen gesetzt werden sollen. Die abstrakte Syntax:

```
<[HTML-Tag] style="Stilinformationen">
```

Optional kann wie bisher auch über den type-Parameter noch der Typ der Style Sheets definiert werden. Die Angabe type="text/css" spezifiziert, wie oben bereits verwendet, beispielsweise CSS.

Beispiel 7:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Style-Type" CONTENT="text/css">
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#FF0000" ALINK="#FF0000"
VLINK="#FF0000">
```

Normaler Text ohne Stilinformationen.

```
<P style="color : gray">
```

Ein Inline-Style Sheet, in dem die Schriftfarbe in dem Absatz auf grau gesetzt wird.

```
</P>
<P type="text/css" style="font-size: 22pt; color: red">
```

Ein Inline-Style Sheet, in dem die Schriftfarbe in dem Absatz auf rot und die Schriftgröße auf 22 gesetzt sowie über den type-Parameter der Typ der Style-Sheets definiert wird.

```
</P>
```

```
</BODY>
```

```
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen `Stileinbind3.html` ab und schauen Sie sich die Datei in verschiedenen Browsern an.

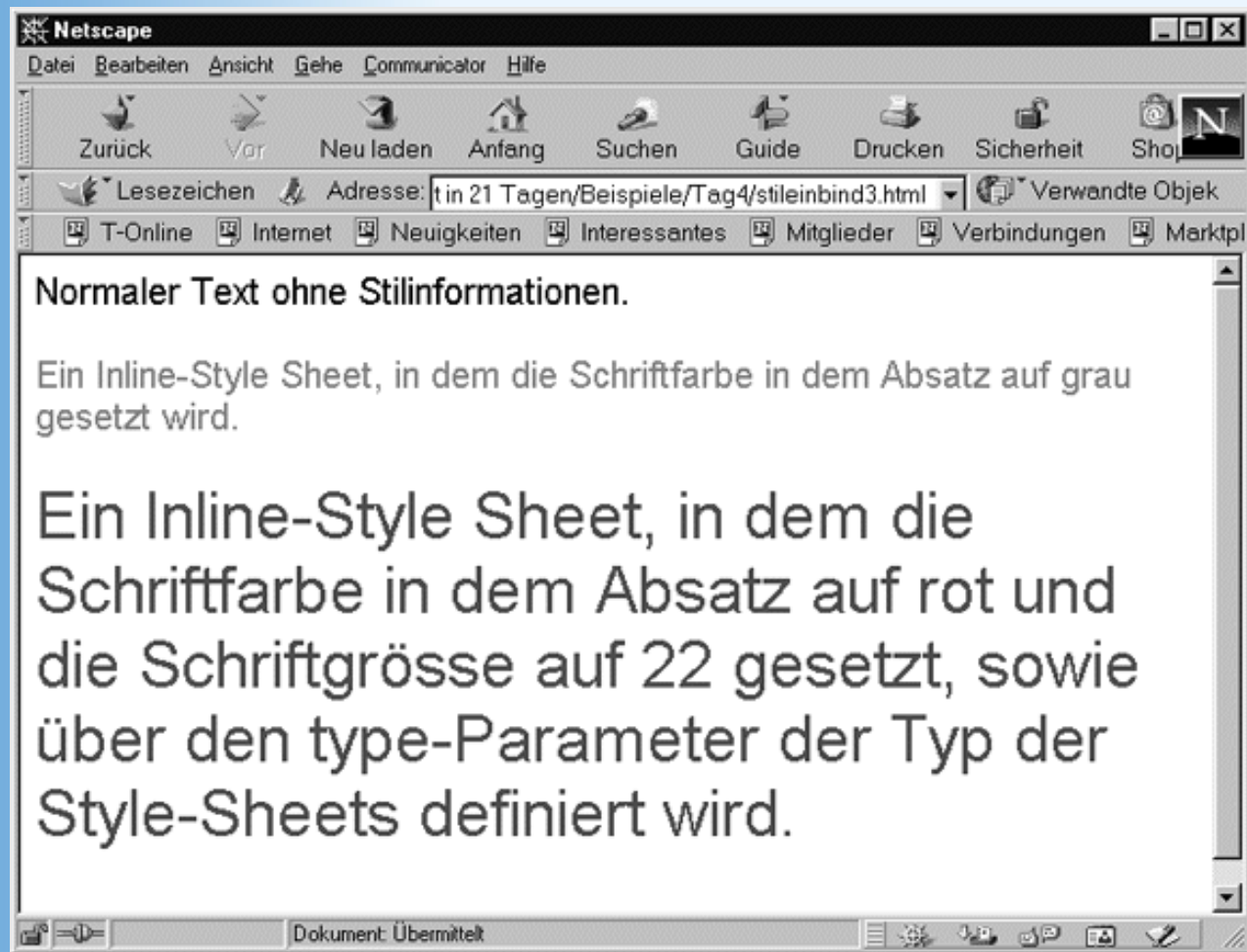


Abbildung 4.9: Die Überschrift und die dritte Zeile sind über Style Sheets formatiert

Prioritäten

Wenn mehrere Style Sheets in einer Seite vorhanden sind, kann es zu Konflikten kommen. Sofern sich zwei Stilinformationen der gleichen Art auf dasselbe Element beziehen, wird die jeweils zuletzt notierte Stileinstellung für ein bestimmtes Element verwendet. Für spezielle Fälle gibt es die Möglichkeit, die Priorität einer Stilinformation über einen Modifier (`important`) zu erhöhen.

Wenn verschiedene Typen von Style-Sheet- Einbindungen in einer Webseite kombiniert werden und es zu Widersprüchen kommt, hat die Inline-Verknüpfung die höchste Priorität, gefolgt von der eingebetteten Formatvorlage. Die verknüpfte Formatvorlage hat die niedrigste Priorität. Das kann auch dazu verwendet werden, über Style Sheets global definierte Elemente mit einer Inline-Deklaration wieder zu überschreiben.



Es gibt auch die Möglichkeit eines Imports externer Stilinformationen über den `<STYLE>`-Container. Der CSS-Entwurf definiert für die Referenz auf die Datei die Anweisung `@import`. Die Syntax würde so aussehen:

```
<STYLE type="text/css">  
  @import url([URL einer Datei mit CSS-Informationen]);  
</STYLE>
```

Diese Variante wird jedoch von mehreren Browsern nicht unterstützt. Man sollte Sie nur dann einsetzen, wenn man alle in Frage kommenden Browser testen kann (etwa in einem Intranet). Aber streng genommen gibt es kaum einen Grund, diese Version der Einbindung überhaupt zu verwenden.

Zusammenfassung

Sie kennen nun die Möglichkeiten zur Einbindung von JavaScripten (genau genommen jeglicher Scripte, die im Rahmen eines Web-Browsers unterstützt werden) und Style Sheets in Webseiten. Beide Techniken werden von der Logik her auf ähnliche Weise mit HTML bzw. einer Webseite verknüpft. Entweder werden

- die Quelltexte direkt in einen Container in einer Webseite notiert,
- eine externe Datei wird mit der Webseite verbunden oder
- eine Inline-Referenz verwendet.

Die konkrete Syntax für einen allgemeinen Script- Container sieht also schematisch immer so aus:

```
<SCRIPT language="[Sprache]">
<!--
... irgendwelche Script-Anweisungen
//-->
</SCRIPT>
```

Im Fall von JavaScript folgendermaßen:

```
<SCRIPT language="JavaScript">
<!--
... irgendwelche Script-Anweisungen
//-->
</SCRIPT>
```

Bei der externen Notation sieht die Referenz für Scripte allgemein wie folgt aus:

```
<SCRIPT language="[Sprache]" src="[URL der separaten Script-Datei]">
</SCRIPT>
```

Im Fall von JavaScript gilt also:

```
<SCRIPT language="JavaScript" src="[URL der separaten JavaScript-Datei]">
</SCRIPT>
```

Mit dem Attribut src wird der URL der separaten Datei angegeben. Der <SCRIPT>-Container muss für den Fall einer Einbindung von externen Dateien leer sein.

Der Inline-Aufruf eines Scripts sieht grundsätzlich syntaktisch so aus:

```
<[verweissensitives Steuerelement] = "[Scriptsprache]:[Scriptanweisung]">
...
<[Ende-Tag]>
```

Für JavaScript also so:

```
<[verweissensitives Steuerelement] = "JavaScript:[Scriptanweisung]">
...
<[Ende-Tag]>
```

Die Einbindung von Style Sheets erfolgt grundsätzlich sehr ähnlich. Die direkte Einbindung von Style Sheets in eine Webseite erfolgt über die folgende Notation (CSS):

```
<STYLE type="text/css">
<!--
... irgendwelche Stilvereinbarungen ...
-->
</STYLE>
```

Die Referenz auf ein externes Style Sheet erfolgt allgemein so:

```
<STYLE type="[Style Sheet-Typ]" >
<!--
... irgendwelche Stilvereinbarungen ...
-->
</STYLE>
```

Im Inneren werden die konkreten Stilinformationen notiert.

Die Link-Referenz auf ein externes Style Sheet erfolgt allgemein so:

```
<LINK type="[Style Sheet-Typ]" rel=stylesheet  
href="[URL einer Datei mit Stilinformationen]"  
title="[Name des Style-Sheets]">
```

Eine CSS-Datei wird schematisch so zu einer Webseite gelinkt:

```
<LINK type="text/css" rel=stylesheet  
href="[URL einer Datei mit CSS-Informationen]"  
title="[Name des Style-Sheets]">
```

Der Titel ist optional, die Typ-Angabe unter Umständen auch, die restlichen Attribute sind notwendig. Im Gegensatz zu dem <STYLE>-Container wird hier kein Container definiert. Es gibt also keinen Ende-Tag.

Eine Inline-Deklaration sieht so aus:

```
<[HTML-Tag] style="Stilinformationen">
```

Fragen und Antworten

Frage:

Wenn man ein Projekt beginnt, in dem Style Sheets und JavaScript verwendet werden, wird ja deutlich im Buch empfohlen, ab einer gewissen Komplexität externe Dateien zu verwenden. Ist es sinnvoll, jeden Dateityp in einem eigenen Verzeichnis abzulegen?

Antwort:

Ja, aber natürlich nicht zwingend. Dann sollte man aber auch so konsequent sein, für Grafiken und andere verwendete Ergänzungstechniken wie Java-Applets eigene Verzeichnisse zu führen. Ab einer gewissen Komplexität eines Web-Projektes zahlt sich eine solche Organisation in Verzeichnissen aus.

Frage:

Welche Style-Sheet-Sprachen sind im Web noch wichtig?

Antwort:

Es gibt verschiedene Versuche, Style Sheets zu standardisieren. Als wichtigster Standard muss man neben CSS JavaScript-Assisted Style Sheets (JSSS) nennen, was ein von Netscape entwickeltes und im Navigator unterstütztes Format ist sowie Extensible Stylesheet Language (XSL - siehe Tag 2). Aber selbst diese wichtigen Formate konnten sich bislang nicht wirklich überzeugend im WWW durchsetzen. Um Cascading Style Sheets führt derzeit kaum ein Weg herum.

Workshop

Experimentieren Sie mit einfachen HTML-Seiten, indem Sie unsere externen JavaScript- und Style-Sheet-Dateien dort einbinden und verwenden.

Gehen Sie online und analysieren Sie Webseiten mit JavaScripten und Style Sheets auf deren Weg der Einbindung. Den Quelltext einer Webseite können Sie sich im Navigator als auch dem Internet Explorer über das **Ansicht**-Menü und dort den jeweiligen Menüpunkt **Seitenquelltext** bzw. **Quelltext anzeigen** ansehen. Bei Frames - auf die gehen wir noch ein - sehen Sie dann unter Umständen nicht viel. Klicken Sie dann dort einfach mit der rechten Maustaste auf den Sie interessierenden Bereich und wählen aus dem Kontextmenü den Befehl zum Anzeigen des Teilbereichs aus.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

Finden Sie den Fehler:

1. <SCRIPT language ="JavaScript 1.2">
2. <SCRIPT langauge="JavaScript" src="extern.js">

```
</SCRIPT>
3. <A href="zufall()">Klick</A>
4. <STYLE TYPE="text/css">
  <!--
  .meinStil1 {
  font-family: Arial
  }
  -->
</STYLE>
5. <LINK type="text/css" href="externStil.css">
6. <P> <style="color : gray">
7. </SCRIPT> "> <SCRIPT language="JavaScript" src="extern.js">
  alert("Hallo");
</SCRIPT>
```

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 1

Tag 5

Wichtige HTML- Anweisungen

Vorgestern haben Sie wichtige Grundlagen zum Erstellen einer Webseite im Allgemeinen kennen gelernt. Dazu zählen der Aufbau von Steueranweisungen in HTML und das Grundgerüst einer Webseite. Gestern kamen die Verbindung von HTML mit JavaScript und Style Sheets hinzu. Heute wollen wir einige weitergehende HTML-Anweisungen einführen. Dazu verlassen wir wieder für einen Tag JavaScript.

Sie müssen - wie schon mehrfach angedeutet - als JavaScript- Programmierer nicht alle HTML-Anweisungen kennen. Deshalb und weil dies ein JavaScript-Buch ist, werden wir hier nur eine Auswahl von wichtigen HTML-Anweisungen behandeln. Einige dieser HTML- Anweisungen müssen allerdings bereitstehen, bevor wir sinnvolle JavaScript-Aktionen durchführen können. Immerhin verankert sich JavaScript in einer HTML-Seite und interagiert mit deren Elementen. Dementsprechend wollen wir Webseiten verwenden, wie sie in der Praxis sinnvoll sind und Sie sollten deren HTML-Aufbau nachvollziehen können. Die einfacheren HTML-Strukturen werden heute bereits behandelt, komplexere HTML-Strukturen wie Frames, Tabellen und Formulare sowie die Referenzierung von Java-Applets etc. besprechen wir an späteren Tagen und insbesondere in direkter Verbindung zu JavaScript-Aktionen. Eine vollständige Auflistung aller HTML-Befehle und -Attribute finden Sie im Anhang.



Die Anweisungen, die wir hier behandeln, werden allesamt im Body-Bereich einer Webseite notiert und gehören zu den allgemeinen Gestaltungsmöglichkeiten unter HTML.

Zeilen- und Absatzschaltungen

Zeilen- und Absatzschaltungen dienen einmal der optischen Gliederung eines Textes. Unter HTML muss jeder explizit gewünschte Zeilenumbruch mit einem Tag definiert werden. Ansonsten kümmert sich der Browser darum, Text am rechten Browserrand automatisch umzubrechen (Fließtextverhalten). Das automatische Umbrechen am rechten Rand kann man auch mit einem eigenen Container verhindern.

Ein Absatz definiert darüber hinaus noch einen Bereich, der bezüglich Ausrichtung als Einheit behandelt werden kann (außerdem ist der Vorschub etwas größer). Die Ausrichtung erfolgt über das optionale Attribut align, welches bei allen HTML-Anweisungen verwendet werden kann, wo es Sinn macht (also Anweisungen, die absatzähnliche Bereiche definieren).

Aktion	Tag	Ende-Tag
Expliziter Zeilenumbruch	 	-
Unterbindung eines automatischen Zeilenumbruchs	<NOBR>	</NOBR>
Absatzschaltung	<P>	</P>
Absatzschaltung mit Ausrichtung	<P align={left, right, center, justify}>	</P>

Tabelle 5.1: Absatz und Zeilenumbruch

Die bei der Absatzschaltung mit Ausrichtung angegebene geschweifte Klammer soll ein Auswahlmenge definieren, aus der genau ein Wert verwendet werden muss. Die Ausprägungen left, right und center entsprechen links, rechts und zentriert, justify entspricht Blocksatz, wobei diese Anweisung nicht von allen Browsern unterstützt wird (dann wird links ausgerichtet).



Die Anweisungen können direkt in den fließenden Text geschrieben werden, aber auch der Übersichtlichkeit halber in eine neue Zeile. Auf das Layout des HTML-Dokumentes hat dies keinerlei Auswirkung. Dies wird auch für alle folgenden Anweisungen gelten.



Der Ende-Tag eines Absatzes erzeugt keinen neuen Absatz. Erst das explizite Setzen eines neuen `<P>` erzeugt einen solchen neuen Absatz (und beendet damit automatisch den bisherigen Absatz). Der Ende-Tag beendet jedoch lokale Vereinbarungen innerhalb eines Absatzes (etwa spezielle Ausrichtung in diesem Absatz) und hinter dem Ende-Tag kommen wieder die globalen Vereinbarungen der umgebenden Struktur zum Tragen. Dennoch kann man oft darauf verzichten, wenn man ausdrücklich einen neuen Absatz setzt.

Testen Sie die Wirkungen der Tags in einem Beispiel.

Beispiel 1:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
Das ist der erste Text.
<BR>
Eine neue Zeile.
<BR>
<NOBR>
Da wird nix umbrochen -
komme ein Browserrand oder nicht.
</NOBR>
<P>
Ein erster Absatz.
</P>
<P align=right>
Der Text steht rechts.
<P align=justify>
Der neue Absatz ist als Blocksatz ausgerichtet. Die neue Absatzausrichtung gilt,
obwohl die vorherige Ausrichtung nicht durch einen Ende-Tag ausgeschaltet wurde.
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen ZeilenUndAbsaeetze.htm ab und schauen Sie sich die Datei im Browser an. Experimentieren Sie mit verschiedenen Fenstergrößen.

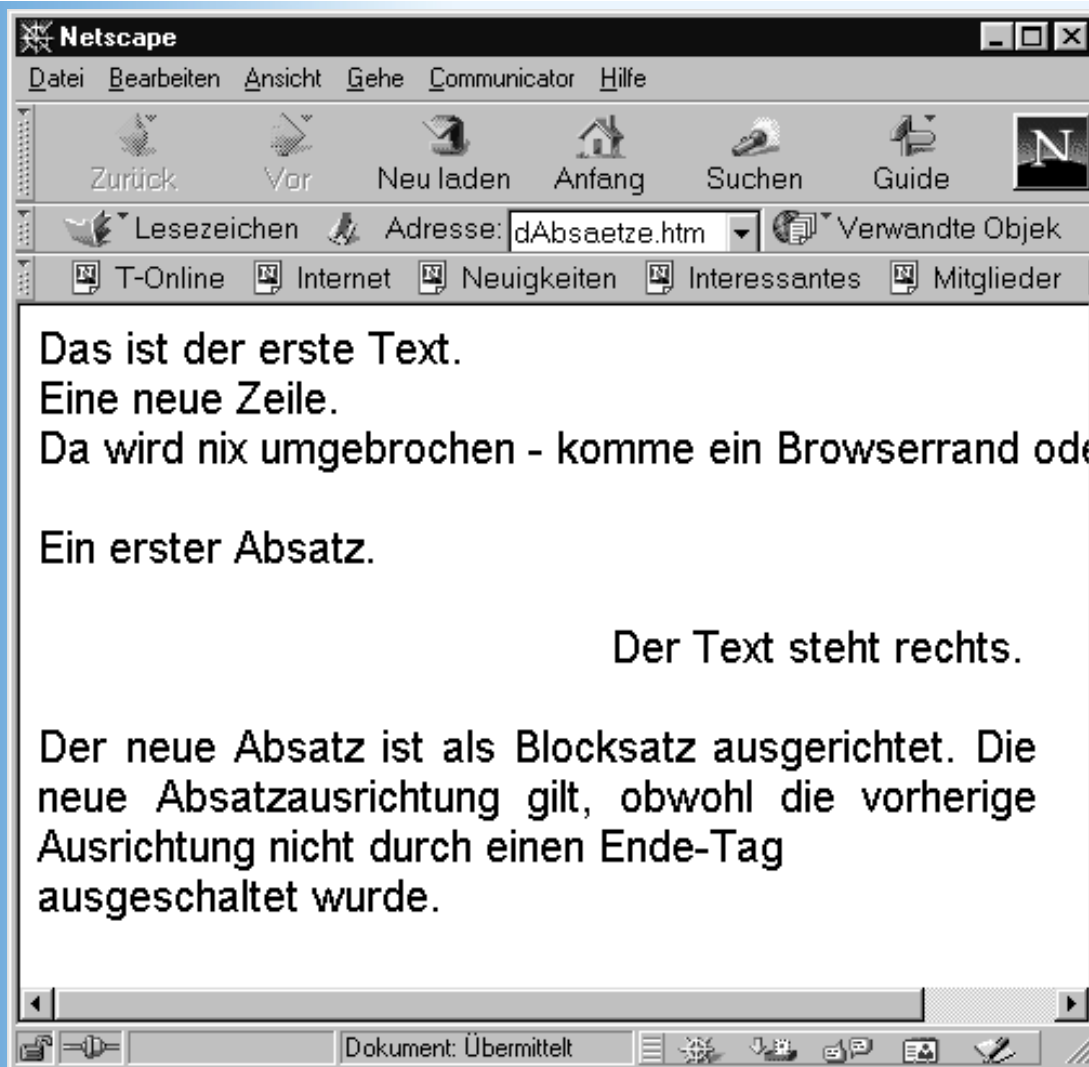


Abbildung 5.1: Die Zeilenschaltungen und Absatzformatierungen wirken

Ausrichtungsanweisungen

Ein allgemeiner Container zum Ausrichten und Zusammenfassen von Objekten ist der <DIV>-Container. Er erzeugt in den meisten Browsern einen Vorschub, gruppiert Elemente und definiert damit einen begrenzten Bereich als eigenständige Unterstruktur innerhalb einer Webseite, die gemeinsam ausgerichtet werden kann. Dazu dient wieder die Parameterangabe align, welche die gleichen Werte annehmen kann wie bei Absätzen. Er ist auch sehr wichtig für die Interaktion zwischen JavaScript, Style Sheets und Elementen einer HTML-Seite. Für diesen Zweck kann er um einige Attribute erweitert werden, die wir aber erst in diesem Zusammenhang besprechen.

Neben der Möglichkeit, in absatzorientierten HTML-Anweisungen über das align-Attribut Ausrichtungen vorzunehmen, gibt es noch einen oft verwendeten und ziemlich alten Befehl, um im Container eingeschlossene Elemente zu zentrieren - <CENTER>.

Diese Anweisung ist ein Paradebeispiel für einen als deprecated (veraltet) gekennzeichneten Befehl (siehe dazu den Abschnitt »Fragen & Antworten« am Tag 3), der immer noch weiter verwendet wird. Nicht nur das: Es zählt zu Standardoptimierungsmaßnahmen einer Webseite, im Fall von Zentrierungen alle align-Attribute aus Anweisungen soweit wie möglich durch diese Anweisung zu ersetzen. (Dies gehört zu einer der Maßnahmen, welche etwa der Optimierungsassistent von dem HTML-Tool 1st Page 2000 durchführt.)

Testen wir die Tags mit einem weiteren Beispiel.

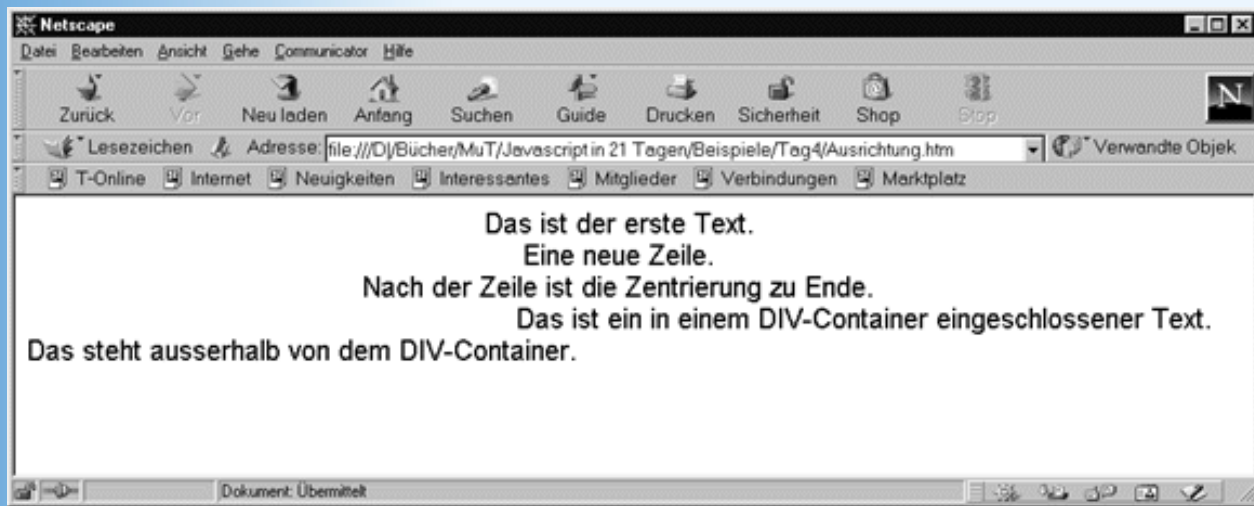


Abbildung 5.2: Die Ausrichtungen wirken

Beispiel 2:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
<CENTER>
Das ist der erste Text.
<BR>
Eine neue Zeile.
<BR>
Nach der Zeile ist die Zentrierung zu Ende.
</CENTER>
<DIV align=right>
Das ist ein in einem DIV-Container eingeschlossener Text.
</DIV>
Das steht außerhalb von dem DIV-Container.
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Ausrichtung.htm ab und schauen Sie sich die Datei im Browser an. Experimentieren Sie mit verschiedenen Fenstergrößen.

Allgemeine Gruppierungsanweisung

In ähnlichem Zusammenhang wie der <DIV>-Container ist der - Container zu sehen. Er unterscheidet sich davon, dass er keinen Vorschub erzeugt. Seine Hauptbedeutung erhält er im Zusammenspiel von JavaScript, Style Sheets und HTML. Dieser Container kann, da er wie gesagt keinen Absatz oder Vorschub erzeugt, im fließenden Text ohne optische Auswirkung in der Webseite verwendet werden. Gestern haben wir ihn bereits bei den Beispielen zu Style Sheets eingesetzt, wir kommen darauf aber zusätzlich nochmals bei der Behandlung von Style Sheets und JavaScript zurück (siehe Abb. Seite 163).

Trennlinien

Trennlinien sind ein einfaches Hilfsmittel zur optischen Abgrenzung von Textbereichen. Sie beginnen immer in einer neuen Zeile und erzeugen neben einer sichtbaren Linie einen weiteren Zeilenvorschub. Trennlinien werden mit dem <HR>-Tag in ein Dokument eingefügt und erstrecken sich ohne weitere Angabe über die gesamte Breite des Anzeigefensters eines Browsers. Der Trennlinien-Tag lässt sich um diverse Parameter zur Spezifizierung erweitern, die in beliebiger Reihenfolge kombiniert werden können (so lange sinnvoll - einige Attribute machen nur in Kombination mit anderen Attributen Sinn).

Attribut	Wirkung
width=[Breite]	Definiert die Breite der Trennlinie. Sofern der Wert in Prozent angegeben wird, beträgt die Trennlinie diesen Prozentsatz von der Breite des Anzeigefensters. Ansonsten wird die Breite der Trennlinie in Pixeln festgelegt.

<code>align=[Ausrichtung]</code>	Die Defaultausrichtung einer Trennlinie ist zentriert, aber mit diesem Attribut und den Werten <code>left</code> , <code>right</code> und <code>center</code> lässt sich die Ausrichtung explizit festlegen. Blocksatz macht hier natürlich keinen Sinn und das Ausrichten von Trennlinien ist nur in Verbindung mit der Angabe <code>width</code> zur Einschränkung der Linienlänge sinnvoll, da die Trennlinie ansonsten stets über die gesamte Breite des Anzeigefensters geht.
<code>size=[Liniendicke]</code>	Definiert die Dicke der Trennlinie. Diese Angabe ist numerisch und immer als relativ zur Standarddicke einer Trennlinie zu sehen. Die Voreinstellung beträgt 2. Mit dem Wert 1 erzwingen Sie eine dünnere Trennlinie, mit Werten größer 2 dickere Trennlinien.
<code>noshade</code>	Die meisten Browser zeigen Trennlinien schattiert an. Sie können den Schatteneffekt mit diesem Attribut unterbinden.

Tabelle 5.2: Trennlinienspezifizierungen

Experimentieren wir auch mit der Definition von Trennlinien.

Beispiel 3:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
<CENTER>
Das ist der erste Text. Danach folgt eine Trennlinie mit Prozentangabe der Größe.
<HR width=45%>
Eine neue Zeile. Danach folgt eine Trennlinie mit Pixelangabe der Größe.
<HR width=300>
Verschiedene Trennlinien mit der Kombination der Ausrichtung
<HR width=50% align=left>
<HR width=25% align=right>
Trennlinien in verschiedenen Größen.
<HR size=1>
<HR size=5>
<HR size=10>
Trennlinie mit dem noshade-Effekt.
<HR width=95% align=left noshade>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen `Trennlinien.htm` ab und schauen Sie sich die Datei im Browser an. Experimentieren Sie mit verschiedenen Fenstergrößen.

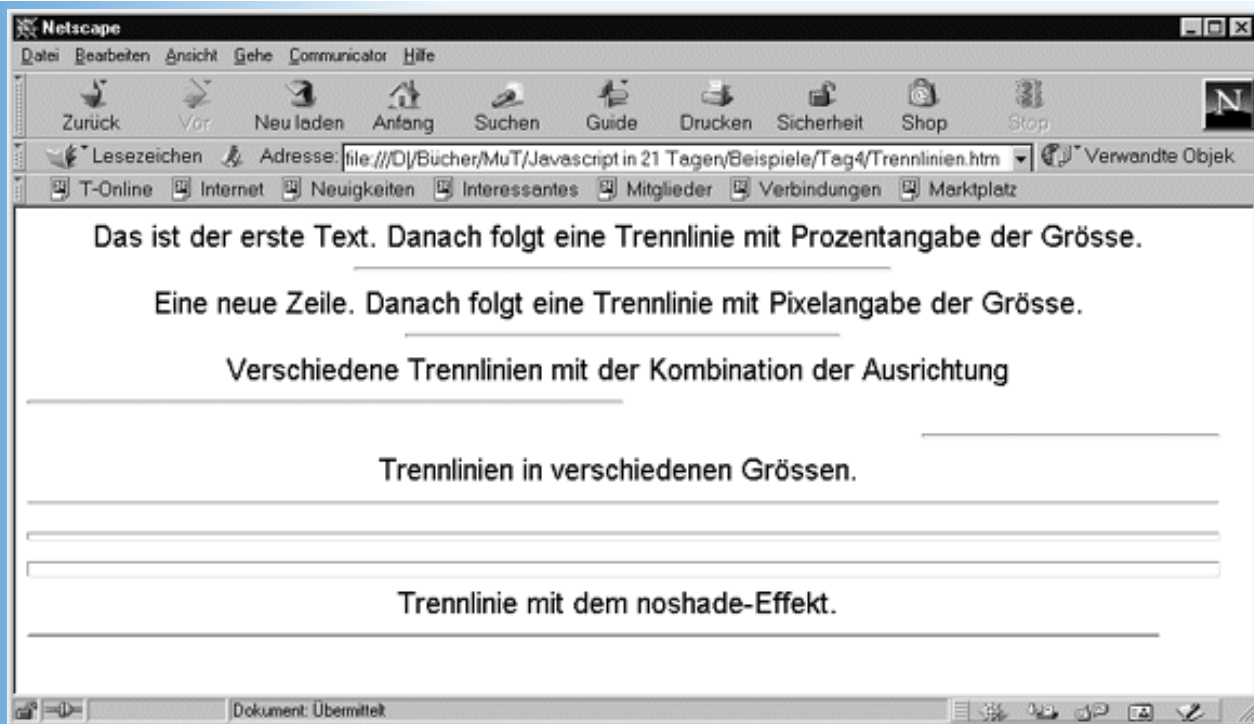


Abbildung 5.3: Verschiedene Trennlinien

Überschriften

Überschriften werden über einen Container festgelegt, der verschiedene Schriftgrößen für den eingeschlossenen Text spezifiziert und eine Absatzschaltung auslöst. HTML kennt sechs Überschriften, die sich normalerweise nur in der Schriftgröße unterscheiden:

Tag	Beschreibung
<H1>	Überschrift 1. Ordnung (größter Text)
<H2>	Überschrift 2. Ordnung
<H3>	Überschrift 3. Ordnung
<H4>	Überschrift 4. Ordnung
<H5>	Überschrift 5. Ordnung
<H6>	Überschrift 6. Ordnung (kleinster Text)

Tabelle 5.3: Überschriften



Die Größenangaben der Überschriften sind immer als relative Angaben zu verstehen, die von der Größe der Standardschrift des Browsers abhängen.

Da Überschriften absatzorientiert sind, können sie über das bekannte Attribut align und den Werten left, right, justify und center ausgerichtet werden (die Wirkung ist auf den Überschriften-Container beschränkt). Standardausrichtung ist linksbündig.

Beispiel 4:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
<H1>
H1 ohne Ausrichtung
</H1>
<H2 align=justify>
```


H2 mit Ausrichtung Blocksatz. Da muss dann viel Text stehen, dass der Effekt sichtbar wird.

</H2>

<H3 align=center>

H3 mit Ausrichtung zentriert

</H3>

<H4 align=right>

H4 mit Ausrichtung rechts

</H4>

<H5 align=left>

H5 mit Ausrichtung Links - entspricht Standardausrichtung.

</H5>

<H6>

H6 ohne Ausrichtung

</H6>

</BODY>

</HTML>

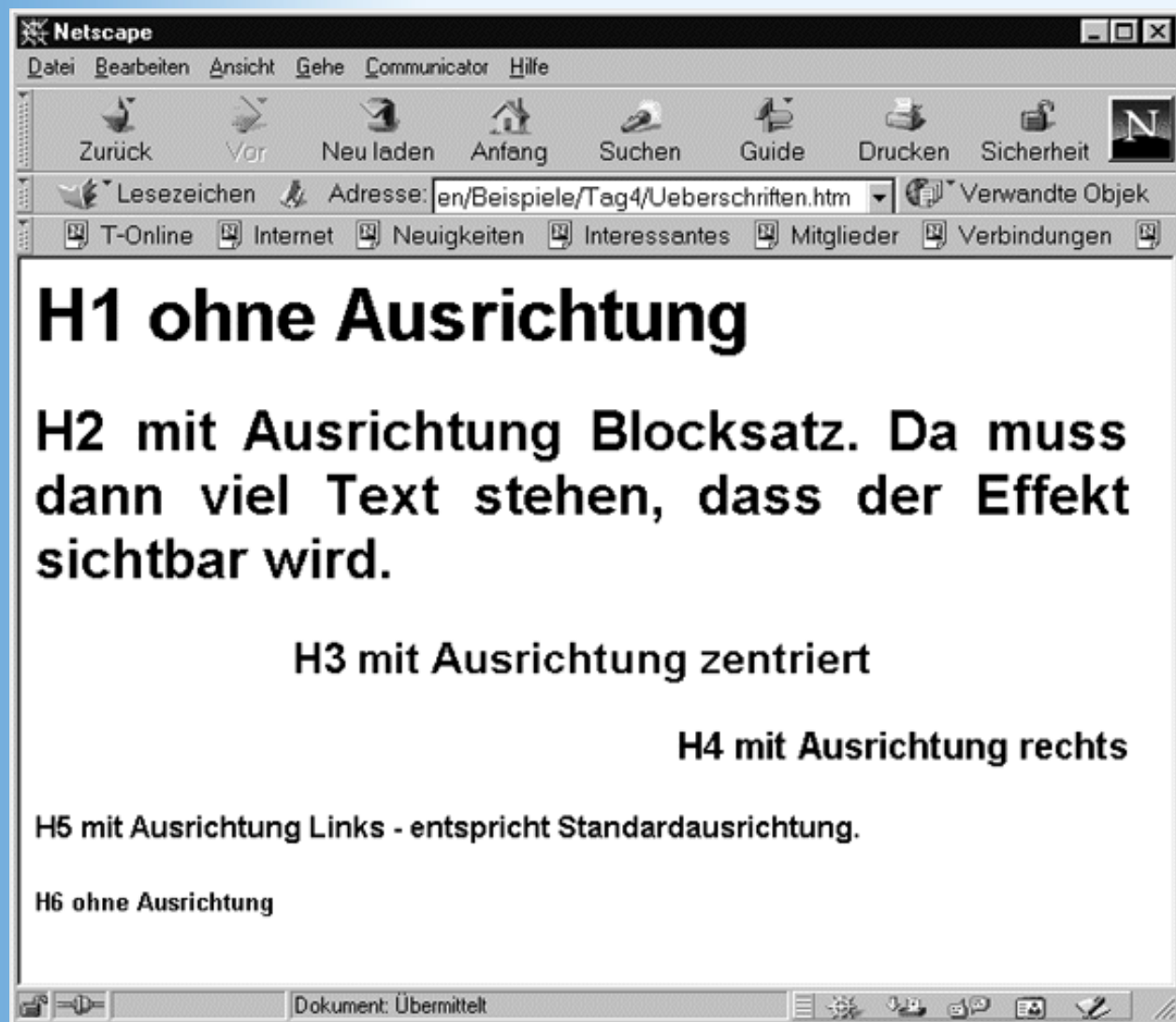


Abbildung 5.4: Verschiedene Überschriften

Speichern Sie den Text bitte unter dem Dateinamen Ueberschriften.htm ab und schauen Sie sich die Datei im Browser an. Experimentieren Sie mit verschiedenen Schriftgrößen. Das erreichen Sie im Netscape Navigator, indem Sie im Menü **Bearbeiten** den Menüpunkt **Einstellungen** auswählen und dann die Schriftarten verändern. Ebenso können Sie unter **Ansicht** die Schriftgrößen verändern. Im Internet Explorer finden Sie die entsprechenden Möglichkeiten unter **Extras** **Internetoptionen...**, Registerblatt **Allgemein** bzw. im Menü **Ansicht** und dort **Schriftgrad**. Die relative Wirkung der Überschriften-Tags wird noch deutlicher, wenn dazwischen Nichttext-Elemente wie Grafiken eingebaut sind. Wir werden das im nächsten Beispiel zeigen.

Grafiken

Mit der Einbindung von Grafiken in eine Webseite referenzieren wir die erste Nichttext-Technik in einer Webseite. Schon im ersten HTML-Standard war diese Möglichkeit vorgesehen und sie zählt zu den elementaren Gestaltungsmitteln einer Webseite. Am dritten Tag hatten wir schon den Parameter background des <BODY>-Tags verwendet, über den eine Grafik als gekacheltes Hintergrundbild in eine Webseite eingebunden werden konnte.

Es ist aber auch selbstverständlich möglich, gezielt an verschiedenen Stellen in einer Seite Grafiken einzubinden. Dabei wird wie beim <BODY>-Tag verfahren, nur nutzen wir an der Stelle, wo die Grafik auftauchen soll, einen individuellen Tag. Ansonsten gilt an dieser Stelle das, was beim Einbinden von Grafiken im <BODY>-Tag ausgeführt wurde: Es lassen sich ohne Risiko im WWW nur zwei Bildformate voraussetzen - das GIF und das JPEG-Format. Sie können auch beliebig animierte GIFs verwenden.

Die konkrete Einbindung von Bildern erfolgt über den Tag an der Stelle, wo das Bild sichtbar sein soll. Dieser Tag legt über das ergänzende, nichtoptionale Attribut src das referenzierte Bild fest. Die gesamte Steueranweisung hätte dann folgende Struktur:

```
<IMG src="[URL]">
```

Ein weiteres Attribut - alt - ist eine sehr sinnvolle Erweiterung, die eigentlich bei keiner Grafikreferenz fehlen sollte. Damit kann in Anführungszeichen eingeschlossen ein Alternativtext angeboten werden, der immer dann erscheint, wenn die Grafik nicht angezeigt werden kann. Aber auch wenn die Grafik angezeigt wird, hat das Attribut eine zusätzliche Wirkung. Es erscheint eine Quickinfo, wenn der Mauszeiger über die Grafik bewegt wird.



Nichtgrafikfähige Browser sind mittlerweile die absolute Ausnahme. Dennoch kann man sich auf die Anzeige von Grafiken nicht verlassen. Man sollte bei der Verwendung von Grafiken (und erst recht weitergehenden Multimedia-Elementen) beachten, dass viele Surfer (vor allem Experten und kostenbewusste Anwender) die automatische Anzeige von Grafiken im Browser deaktiviert haben¹. Dies erfolgt zum einen sicher wegen der oft katastrophal niedrigen Übertragungsraten im Internet, aber noch mehr wegen der zunehmenden Werbebanner und sonstiger nichtssagender Grafiken, welche viele Webseiten überfluten und Geld und Zeit des Besuchers kosten, ohne einen Gegenwert zu bieten.

*Sie können das automatische Laden von Grafiken im Navigator beispielsweise unter dem Menüpunkt **BEARBEITEN - EINSTELLUNGEN...** und dort **ERWEITERT** deaktivieren.*

Eine weitere Angabe, die bei Grafiken normalerweise notwendig ist, ist eine Größenangabe. Über zwei Attribute (height für die Höhe und width für die Breite) kann die Größe einer Grafik skaliert werden. Fehlen diese Angaben, werden Grafiken in ihrer »natürlichen« Größe angezeigt. Die Angaben zur Größenbestimmung können einzeln oder zusammen in beliebiger Reihenfolge verwendet werden. Die Angabe von beiden Werten zusammen ist eigentlich nur bei Verzerrungen notwendig. Wenn solche Verzerrungen nicht gewünscht werden, muss man bei Verwendung beider Werte aufpassen - es kann auch zu ungewollten Verzerrungen kommen. Wird nur eine der beiden Angaben gemacht, wird vom Browser der korrespondierende Werte errechnet.



Selbst wenn die Grafik ohne Größenangabe in der richtigen Größe angezeigt wird, sollten Sie dennoch eine der beiden Angaben setzen. Die Anzeige der Grafik wird damit beschleunigt.

Eine Grafik kann bewusst mit einem Rahmen versehen werden bzw. wird automatisch damit gekennzeichnet (etwa bei Grafiken als Hyperlinks). Ein weiterer Parameter innerhalb der Grafikreferenz dient zur Festlegung eines Rahmens in Pixeln - der Parameter border. Diese Angabe kann man ebenfalls dafür nutzen, explizit einen Rahmen zu unterdrücken (border=0).

Beispiel 5:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
Eine Grafik ohne Attribute
<IMG src="smile.pcx">
```

```
<BR>
Eine Grafik mit Border
<IMG src="tk.pcx" border=5>
<BR>
Eine Grafik mit Breitenangabe
<IMG src="kürbis.pcx" width=80>
<BR>
Eine Grafik mit Breiten- und Höhenangabe zur Verzerrung
<IMG src="smile2.pcx" width=75 height=15>
<BR>
Eine Grafik mit Alternativtext
<IMG src="inside.pcx" alt="Was so alles inside ist">
<BR>
Eine animierte Grafik
<IMG src="alien.pcx">
<BR>
</BODY>
</HTML>
```

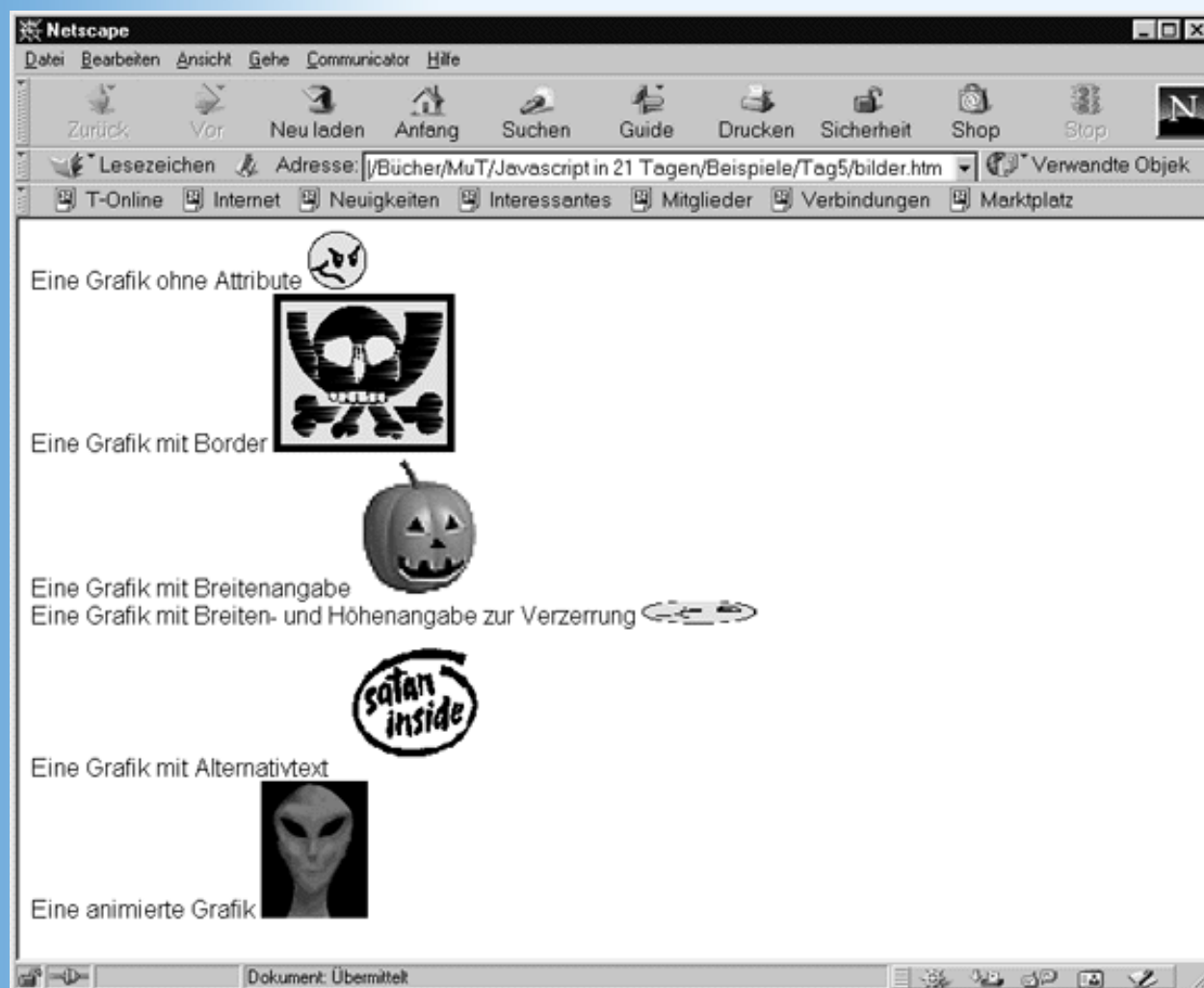


Abbildung 5.5: Verschiedene Grafikreferenzen

Speichern Sie den Text bitte unter dem Dateinamen Bilder.htm ab und schauen Sie sich die Datei im Browser an. Experimentieren Sie wie im Beispiel 4 mit verschiedenen Schriftgrößen und beobachten Sie das Verhalten von dem Text in Relation zu den Grafiken. Bewegen Sie den Mauszeiger auf die Grafik, welche mit dem alt-Attribut versehen ist (siehe Abbildung 5.5).

Deaktivieren Sie das automatische Laden von Grafiken und laden Sie die Datei neu (im Navigator unter **Bearbeiten, Einstellungen..., Erweitert** zu finden.). Um die Wirkung zu sehen, müssen Sie den Browser schließen und neu starten (siehe Abbildung 5.6).

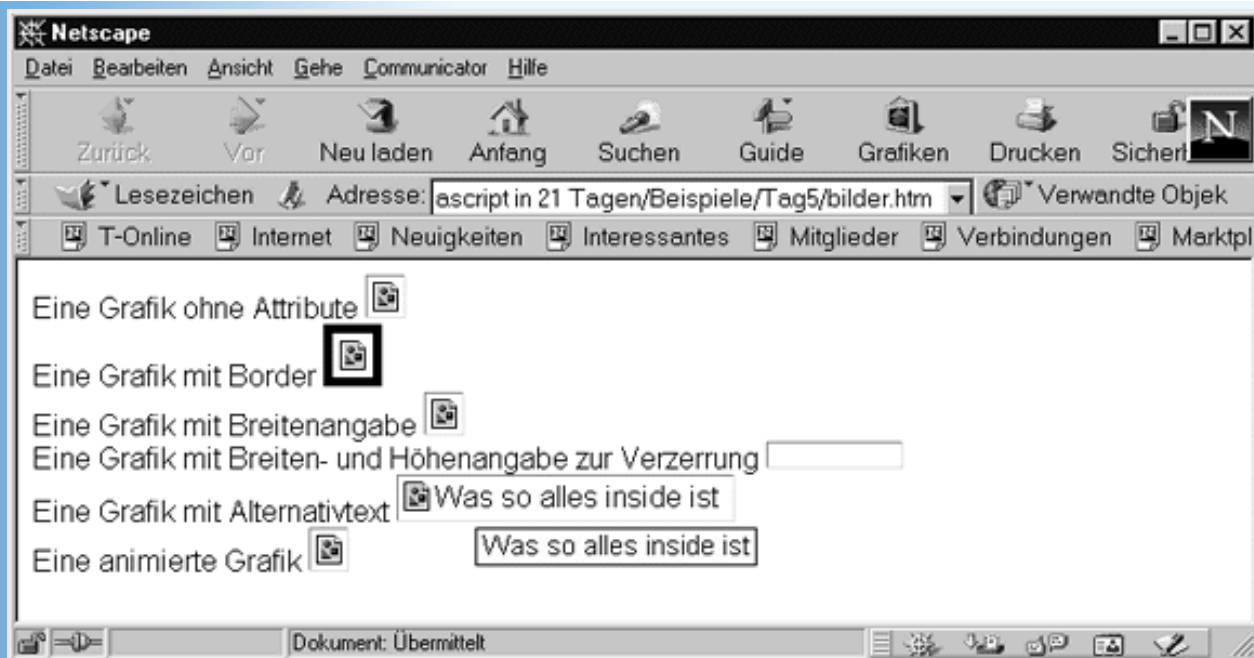


Abbildung 5.6: So sieht es ohne Grafiken aus.

Textformatierungen

Zu den wichtigsten Gestaltungsmitteln einer Webseite zählt die Formatierung der Textdarstellung. Allerdings muss man darauf hinweisen, dass Textformatierung unter HTML nur wenig mit Textformatierung unter einer Textverarbeitung auf einem PC gemeinsam hat. Textformatierung unter HTML ist als eine Anregung an den Browser zu verstehen, eine in einen Container eingeschlossene Zeichenkette entsprechend sehr weiten Spielräumen zu gestalten. Der Ersteller einer Webseite hat nur begrenzten Einfluss auf die tatsächliche Erscheinung der Seite beim Anwender. Dies ist - um auch das nochmals deutlich zu machen - keine Schwäche von HTML (das sehen nur Leute so, die Webseitendesign einfach nicht verstehen wollen), sondern eine gewaltige Stärke.

Textformatierungen können in HTML sowohl die Schriftgröße, die Schriftart und den Schriftstil beeinflussen. Prinzipiell unterscheidet man in HTML zwischen so genannten logischen und physischen Befehlen zur Hervorhebung von Text.

- Logische Anweisungen überlassen weitgehend dem Browser des Betrachters, wie eine konkrete Darstellung aussehen wird. Dementsprechend können logische Anweisungen auf unterschiedlichen Plattformen sehr verschieden dargestellt werden. Logische Anweisungen funktionieren auch unter nicht-grafischen Plattformen und setzen mit der Überlassung der konkreten Darstellung eines Dokumentes an das Darstellungsmedium die eigentliche Idee einer Dokumentbeschreibungssprache konsequent um. Sie verlieren aber immer mehr an Bedeutung.
- Physische Anweisungen legen genauer fest, wie ein Text aussehen soll. Ein so definierter Text wird in den meisten Browsern gleich aussehen, aber auch da können Browser-spezifische Unterschiede zum Tragen kommen. Physikalische Anweisungen bieten viel mehr Gestaltungsmöglichkeiten und setzen sich immer mehr durch.

Logische Anweisungen

Anweisung	Beschreibung
<cite>	Zitat (meist kursive Darstellung)
<code>	Darstellung von Quellcode (meist nichtproportionale Darstellung)
<dfn>	Definitionen
	Betonung (meist kursive Darstellung)
<kbd>	Darstellung von Benutzereingaben (meist nichtproportionale Darstellung)
<samp>	Darstellung von Beispielen (meist nichtproportionale Darstellung)
	Allgemeine kräftige Hervorhebung (meist fette Darstellung)
<tt>	Darstellung von TeleType-Schrift (meist nichtproportionale Darstellung)
<var>	Darstellung von Variablennamen (meist kursive Darstellung)

Tabelle 5.4: Logische Anweisungen

Beide Formen der Textformatierung kann man mischen und diverse Formatierungsanweisungen haben ähnliche Effekte.

Beispiel 6:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
<cite>Zitat
</cite>
<code>Quellcode
</code>
<dfn>Definition
</dfn>
<em>Betonung
</em>
<kbd>Anwendereingaben
</kbd>
<samp>Beispiel
</samp>
<strong>Fett
</strong>
<tt>TeleType
</tt>
<var>Variablen
</var>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen TextFormatLogisch.htm ab und schauen Sie sich die Datei im Browser an.

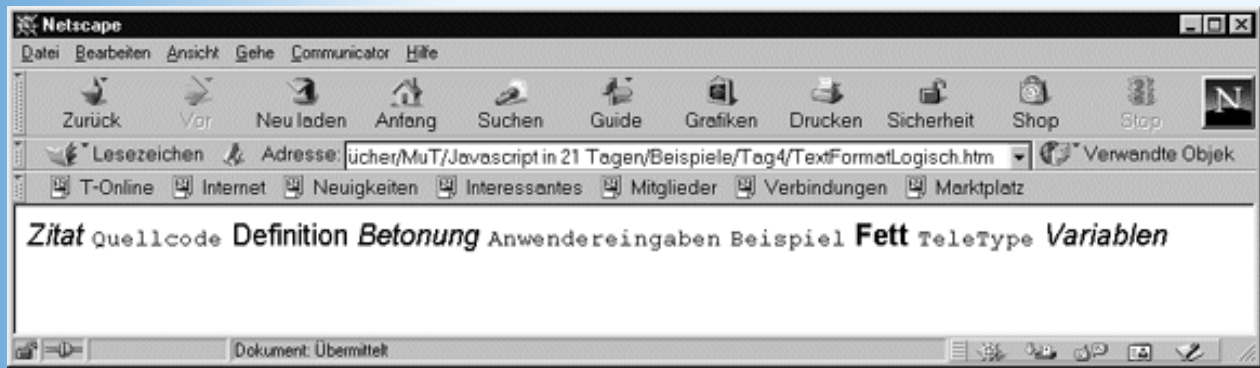


Abbildung 5.7: Verschiedene logische Formatierungen

Physische Anweisungen

Anweisung	Beschreibung
	Fett
<big>	Relativ zur Standardeinstellung größere Schrift
<blockquote>	Zitat, das über mehrere Zeilen gehen kann. Nach einem im Container erhaltenen Zeilenumbruch erfolgt eine Einrückung (Tabulator). Sofern Sie dort mit Absätzen arbeiten, werden die Ebenen der Ausrichtung des Absatzes angepasst. Wenn <blockquote>-Anweisungen ineinander geschachtelt werden, werden die Ebenen weiter eingerückt
<i>	Kursiv
<italic>	Alternative zu dem <i>-Tag
<small>	Relativ zur Standardeinstellung kleinere Schriftgröße
<strike>	Durchgestrichen

<sub>	Tiefgestellt
<sup>	Hochgestellt
<u>	Unterstreichen

Tabelle 5.5: Physische Anweisungen

Beispiel 7:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
<b> Fett,
</b>
<big> groß,
</big>
<small> klein,
</small>
<i> kursiv,
</i>
<sub> tiefgestellt,
</sub>
<strike> durchgestrichen,
</strike>
<u> unterstrichen
</u>
und
<sup> hochgestellt
</sup>.
<br>

<BLOCKQUOTE>
Blockwas?
<br>
Blockquote
<br>
Ach so.
</BLOCKQUOTE>

<P>
<BLOCKQUOTE>
Blockwas?
<BLOCKQUOTE>
Blockquote
<BLOCKQUOTE>
Ach so.
</BLOCKQUOTE>
</BLOCKQUOTE>
</BLOCKQUOTE>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen TextFormatPhysisch.htm ab und schauen Sie sich die Datei im Browser an (siehe Abbildung 5.8).

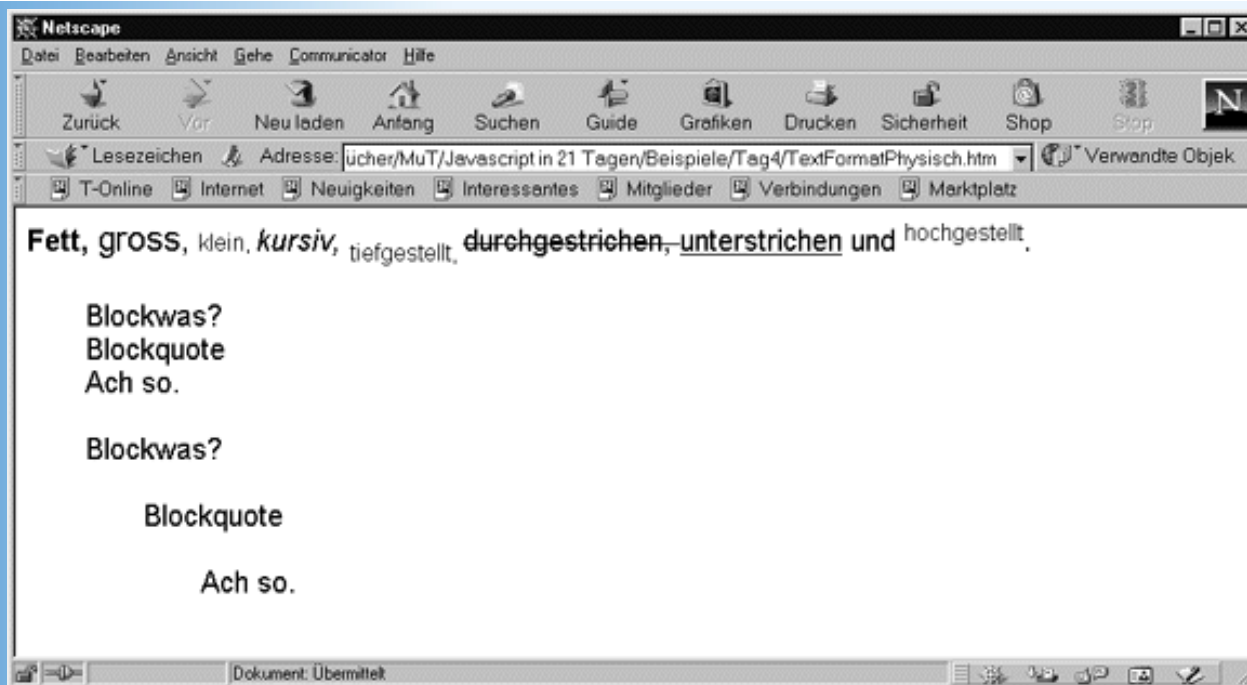


Abbildung 5.8: Verschiedene physische Formatierungen

Sonderzeichen

HTML kann normalerweise nicht mit Sonderzeichen direkt umgehen. D.h. sie in einer Webseite direkt anzeigen. Dies sind beispielsweise alle deutschen Umlaute wie ö, ü, ä oder das ß. Ebenso Zeichen, die eine besondere Bedeutung haben wie die Zeichen < und >. Dies liegt daran, dass durch den heterogenen Standard der Rechner und Betriebssysteme im Internet bereits in einer sehr frühen Phase des Internets ein international genormter Zeichensatz zum Austausch von Zeichen vereinbart wurde. Dieser basierte natürlich auf dem amerikanischen Zeichensatz, denn in der Anfangsphase handelte es sich beim Internet um ein rein amerikanisches Netzwerk. Der bis zu der Version HTML 3.2 gültige Zeichensatz wird ISO- 8859-1-Norm genannt. Diese Norm besteht aus 256 Zeichen, wovon die ersten 128 identisch mit dem klassischen ASCII-Zeichensatz sind. Die zweite Hälfte der Zeichen weicht dagegen davon ab und besteht aus etlichen Sonderzeichen und Buchstaben für verschiedene - insbesondere europäische - Sprachen. In HTML 4.0 wurde im Rahmen der so genannten Internationalisierung auf den 16-Bit-großen Unicode umgestellt, der im unteren Byte mit dem bisherigen Code identisch ist. Für den Anwender hat das keine Konsequenzen, wenn nur Zeichen der ersten Hälfte (unter anderem die normalen Buchstaben des Alphabets und die Ziffern von 0 bis 9) wie gewohnt über die Tastatur eingegeben werden. Auch kann man wie bisher mit einem normalen ASCII-Editor arbeiten und wird nicht gezwungen, einen Unicode-Editor zu verwenden. Wenn Sie allerdings in HTML-Dateien Zeichen aus der zweiten Hälfte der bisherigen ISO 8859-1-Tabelle verwenden wollen (dazu gehören die deutschen Umlaute, das Zeichen ß und Zeichen, die Sie auf der Tastatur nicht finden wie das Copyrightzeichen © oder Sonderzeichen wie < >) oder vor allem die weiteren Zeichen der erweiterten ISO/IEC:10646-Norm, müssen Sie diese verschlüsselt eingeben.



Das Verschlüsseln von Sonderzeichen unter HTML nennt man auch Maskieren.

Die Maskierung von Sonderzeichen und landesspezifischen Zeichen erfolgt entweder über eine Folge von Zahlen oder Zeichen, die einem Steuerzeichenpaar - dem &# -folgen und von einem Semikolon abgeschlossen werden oder einem &, gefolgt von einer Abkürzung, die das maskierte Zeichen beschreibt und beendet mit einem Semikolon.

Bei der numerischen Maskierung teilt das beginnende &# dem Browser mit, dass der nachfolgende Zahlencode in ein konkretes Zeichen umgesetzt werden muss. Das Semikolon beendet die Anweisung. Der eingeschlossene Buchstabencode entspricht der Position des Zeichens in der ISO 8859-1- Tabelle. Er kann dezimal, aber auch hexadezimal angegeben werden. Dabei müssen Groß- und Kleinbuchstaben unterschiedlich maskiert werden.

Die etwas komfortablere Form der Maskierung über Abkürzungen ist recht sprechend. Für deutsche Umlaute werden Sie den Grundbuchstaben finden, gefolgt von den Zeichen uml. Etwa Uuml für Ü oder auml für ä. Andere Abkürzungen sind sprechend genau das, was sie kodieren sollen. Etwa sz gefolgt von lig, was für ß steht. Bei anderen Zeichen sind es die ersten Buchstaben der ausgeschriebenen englischen Form, etwa lt, was für less then (kleiner als) steht und das Zeichen < erzeugt. Im Anhang finden Sie eine Tabelle mit den wichtigsten Maskierungen.

Wenn Sie also Text unter HTML international genormt ausdrücken wollen, müssen Sie sämtliche vorkommenden Sonderzeichen maskieren. Wir werden im Rahmen der Beispiele in dem Buch aber darauf verzichten, damit der Quelltext lesbarer bleibt.



Auch wenn man immer wieder Seiten findet, die solche Sonderzeichen nicht maskieren und diese Sonderzeichen dennoch korrekt angezeigt werden, hängt es nur daran, dass zufällig alle Konfigurationen Ihrer Plattform so eingestellt sind, dass es funktioniert. Auf vielen anderen Plattformen im Internet kann es aber schiefgehen. Auch die von einigen Tools (z.B. Frontpage) statt einer Maskierung verwendete Metainformation `<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">` reicht definitiv nicht aus. Deshalb maskieren bessere HTML-Tools diese Zeichen auf jeden Fall, oft automatisch oder per expliziten Befehl.

Schrift-Anweisung

Neben den Textformatierungsanweisungen gibt es in HTML die `<FONT>`-Anweisung, mit der Schrift noch weitergehend beeinflusst werden kann. Dies betrifft die Schriftgröße, die Schriftart und die Schriftfarbe. Der Tag macht nur mit einem oder mehreren der nachfolgenden Attribute Sinn.

Attribut	Beschreibung
<code>size=[Größe]</code>	Festlegen der relativen Schriftgröße mit einem Wert zwischen 1 und 7. Die Werte sind als relative Angabe zu der Default-Schriftgröße zu verstehen. Die normale Schriftgröße entspricht 3. Die Werte größer als 3 (4 bis 7) stehen für größere Schriftgrade, die Werte 1 und 2 für kleinere Schriften. Daneben geht eine Veränderung der Textgröße um einen gewissen Wert über <code>+(Zahl)</code> beziehungsweise <code>-(Zahl)</code> .
<code>color=[Farbe]</code>	Angabe der Schriftfarbe. Dabei kann für Farbe eine der üblichen Farbangaben gesetzt werden, die wir beim <code><BODY></code> -Tag am dritten Tag behandelt haben.
<code>face=[Schriftart]</code>	Explizite Angabe der verwendeten Schriftart. Dabei wird der jeweilige Name der Schriftart verwendet. Die Verwendung dieses Attributs ist ziemlich gefährlich, denn einmal sind die Namen der Schriftarten nicht immer identisch zu den unter Windows verwendeten. Zum anderen kann man nie voraussetzen, dass eine geforderte Schriftart auf allen Besucherplattformen vorhanden ist.

Tabelle 5.6: Schriftspezifizierungen



Gerade bei der `<FONT>`-Anweisung zeigen die WYSIWYG-Editoren eine ihrer größten Schwächen. Kinderleicht mit einem Menü die Schriftgröße auswählen simuliert eine absolute Kontrolle im Erstellungsmodus, die einfach nicht zu erreichen ist. Unerfahrene Anwender werden dies nicht erkennen und schlechte Webseiten erstellen.

Beispiel 8:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
<P>Text in Standardschrift
</P>
<P>
<FONT size="6">Text größer
</FONT>
</P>
<P>
<FONT size="1">Text kleiner
</FONT>
</P>
<P>
<FONT face="Courier">Text in anderer Schrift
```



```
</FONT>
</P>
<P>
<FONT face="Arial">Text in einer noch anderen Schrift
</FONT>
</P>
<P>
<FONT color="#FF0000" face="Lucida Console">Text in einer noch anderen Schrift und
rot
</FONT>
</P>
<P>
<FONT color="#0000FF" face="Bookman Old Style">Text in einer noch anderen Schrift,
blau und groß
</FONT>
</P>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Font.htm ab und schauen Sie sich die Datei im Browser an.

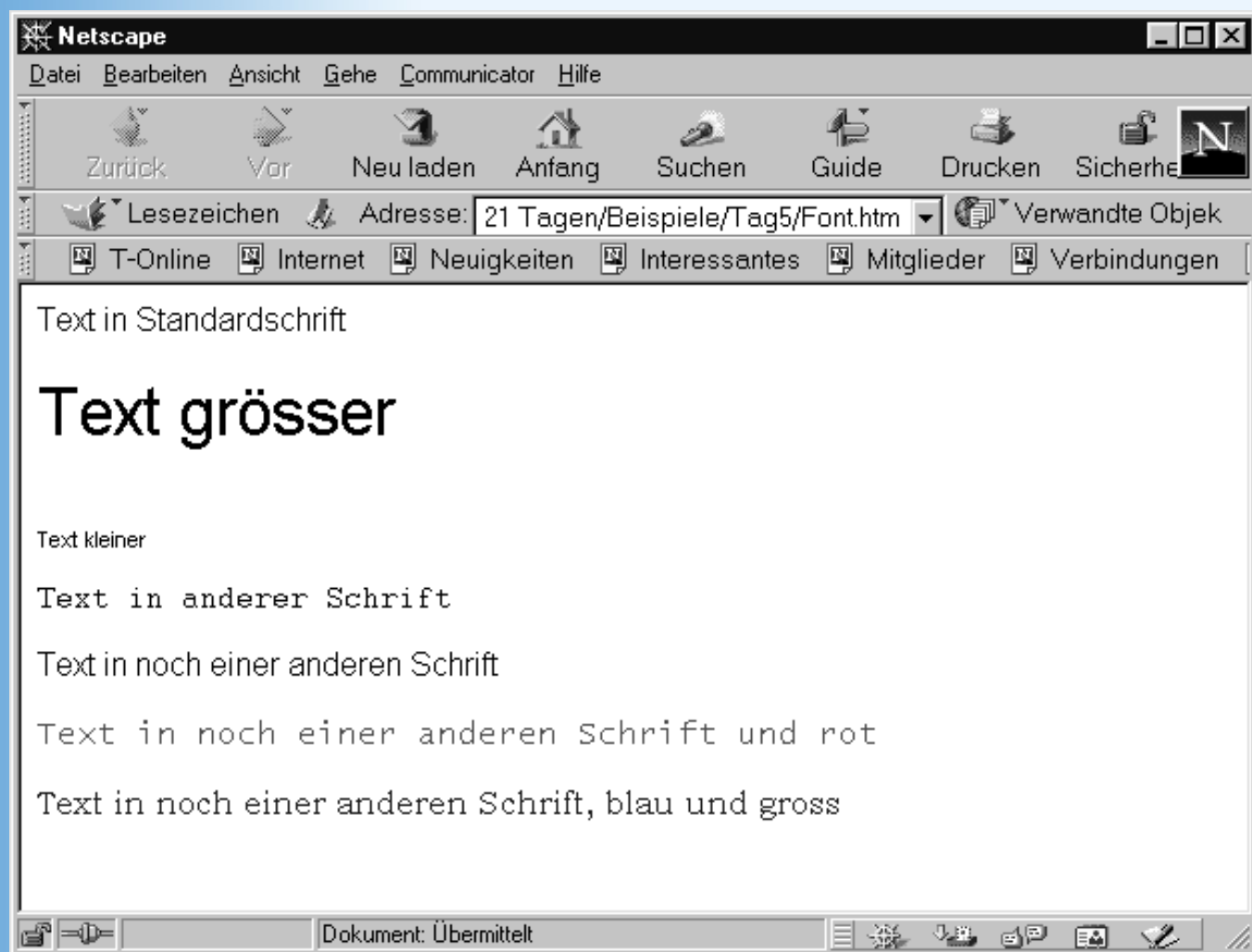


Abbildung 5.9: Verschiedene FONT-Varianten

Hyperlinks

Hyperlinks (auch Verweise oder einfach nur Links genannt) sind das zentrale Element des WWW. Nur darüber eröffnen Sie Besuchern einer Seite die Möglichkeit, mit einem einfachen Mausklick zu einem dahinter liegenden URL zu verzweigen. Ein Verweis kann aber auch dazu genutzt werden, das Laden eines Programms auf den lokalen Rechner auszulösen (das haben wir gestern beim Inline-Aufruf von JavaScripten gesehen), einen FTP-Download oder den Aufruf eines Email-Programms auszulösen, in dessen Adressfeld dann eine Adresse vorbelegt ist. Auch Verweise innerhalb einer - meist größeren - Seite sind

ein sinnvoller Einsatzzweck.

Optisch wird ein Verweis auf einer Webseite meist als unterstrichener und/ oder farbig von normalem Text abgegrenzter Text dargestellt. Ein Link kann auch so gestaltet werden, dass die optische Unterscheidung gegenüber normalem Text unterdrückt wird (eine in letzter Zeit oft zu findende Unart, denn damit wird aus fraglichen ästhetischen Aspekten die Navigation in einer Seite unnötig verkompliziert). Aber auch Grafiken und andere Elemente können als Links fungieren. Allen Links ist jedoch gemein, dass der Anwender einen Link an der Veränderung des Mauszeigers erkennt. Wenn der Mauszeiger auf einen Verweis bewegt wird, verändert er sich zu einer Hand mit ausgestrecktem Zeigefinger. Falls die Statusanzeige des Browsers aktiviert ist und dort keine anderen Informationen angezeigt werden (etwa über ein JavaScript), wird dort in der Regel das Ziel eines Links angezeigt.

Jeder Link basiert auf dem Tag `<A>`, der einen Container einleitet. Es ist aber offensichtlich, dass der Tag um die Angabe, wohin der Link führen soll, erweitert werden muss.

Allgemeine Verweise

Die Grundstruktur eines Verweises sieht so aus:

```
<A href="[URL]">[Anzeige in Webseite]</A>
```

Das, was im Container eingeschlossen ist, wird der Besucher der Webseite sehen. Also einen beschreibenden Text, eine Grafik usw.

Der URL kann im Prinzip jedes passende Protokoll angeben. Damit kann über einen Hyperlink im Rahmen des WWW Kontakt zu einem beliebigen Internet-Dienst aufgebaut werden. Die wichtigsten Fälle sind jedoch folgende:

- Mit `http` adressieren Sie beliebige andere HTML-Dateien im WWW. Aber auch der Download von Dateien, welche nicht im Browser angezeigt werden, kann über `http` realisiert werden (etwa ein `exe`-File oder eine MP3-Datei).
- Fehlt das Protokoll beim URL gänzlich, wird ein Link als relativer `http`- Link von der angezeigten Webseite aus gesehen. Dies nutzt man insbesondere für Verweise innerhalb des eigenen Projekts. Großer Vorteil dieses Verfahrens ist, dass komplette Verzeichnisstrukturen gemeinsam verlagert werden können, ohne dass eine Änderung der Links notwendig wird. Außerdem sind die Links kürzer.
Die Pfadangaben erfolgen analog dem gewohnten relativen Adressieren im Rahmen der Arbeit auf dem lokalen Rechner. Beachten muss man nur, dass bei einem URL der Slash `/` als Trennzeichen der Verzeichnisebenen verwendet wird.
- Mit `file` kann eine beliebige Datei adressiert werden, auch auf dem lokalen Rechner. Ist es eine HTML-Datei, wird sie im Browser angezeigt. Ist es ein anderer Dateityp, geschieht das, was in den Optionen des Browsers für diesen Fall eingestellt ist. Es kann etwa ein Programm gestartet werden, das diesen Dateityp verarbeitet, oder die Datei kann einfach auf den lokalen Rechner herunter geladen werden. Sehr interessant ist das Protokoll in Verbindung mit der Angabe von Festplattenamen und Verzeichnissen auf dem lokalen Rechner. Bei einem solchen URL fungiert der Browser als lokaler Dateimananger. Beim Internet Explorer wird sogar direkt der Browser zu einem Arbeitsplatz-ähnlichen Tool umgeschaltet. Der Navigator äußert sich optisch wie bei einem FTP-Zugriff auf einen Datenträger.
- Mit `ftp` wird ein FTP-Download einer Datei auf den lokalen Rechner des Besuchers gestartet. Aus einer Webseite heraus können Sie damit einem Besucher die Möglichkeit geben, statt einen Download über das langsame `http`-Protokoll durchführen zu müssen, die Aktion per FTP schneller zu bewerkstelligen. Dies setzt allerdings voraus, dass auch ein FTP-Server bereitsteht, der diese Anfrage bearbeiten kann. Es funktioniert nicht, wenn man eine Homepage auf einem WWW-Server hat, dort eine Datei zum Download bereitstellen möchte und einfach das Protokoll ersetzt. Das muss der Provider auch unterstützen.
- Mit `mailto` kann das dem Browser assoziierte Email-Programm per Klick gestartet werden. Im Adressfeld ist die vorgegebene Adresse vorbelegt.
- Die weiteren Protokolle wie `gopher`, `telnet` oder `news` starten die jeweiligen Dienste - eventuell mit Hilfe eines zusätzlichen, dem Browser zugeordneten, Programms.

Wir wollen nun die verschiedenen Typen von Links in einem Beispiel durchspielen. Beachten Sie, dass sie für einige Links online sein sollten.

Beispiel 9:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
```

```
<BODY>
```

Zugriff auf lokalen Rechner:

```
<A href="file://C:">
```

Zum Festplattenlaufwerk C:

HTTP-Link mit Grafik:

FTP-Link:

Keine Ahnung, ob es den Link noch gibt

Download mit HTTP:

Down, down, down

Mail-Link:

Mehl mich

Relativer HTTP-Link mit Grafik und unterdrücktem Border:

Relativer HTTP-Link auf eine Datei im Parallelverzeichnis Tag3:

Zurück zum Tag 3

</BODY>

</HTML>

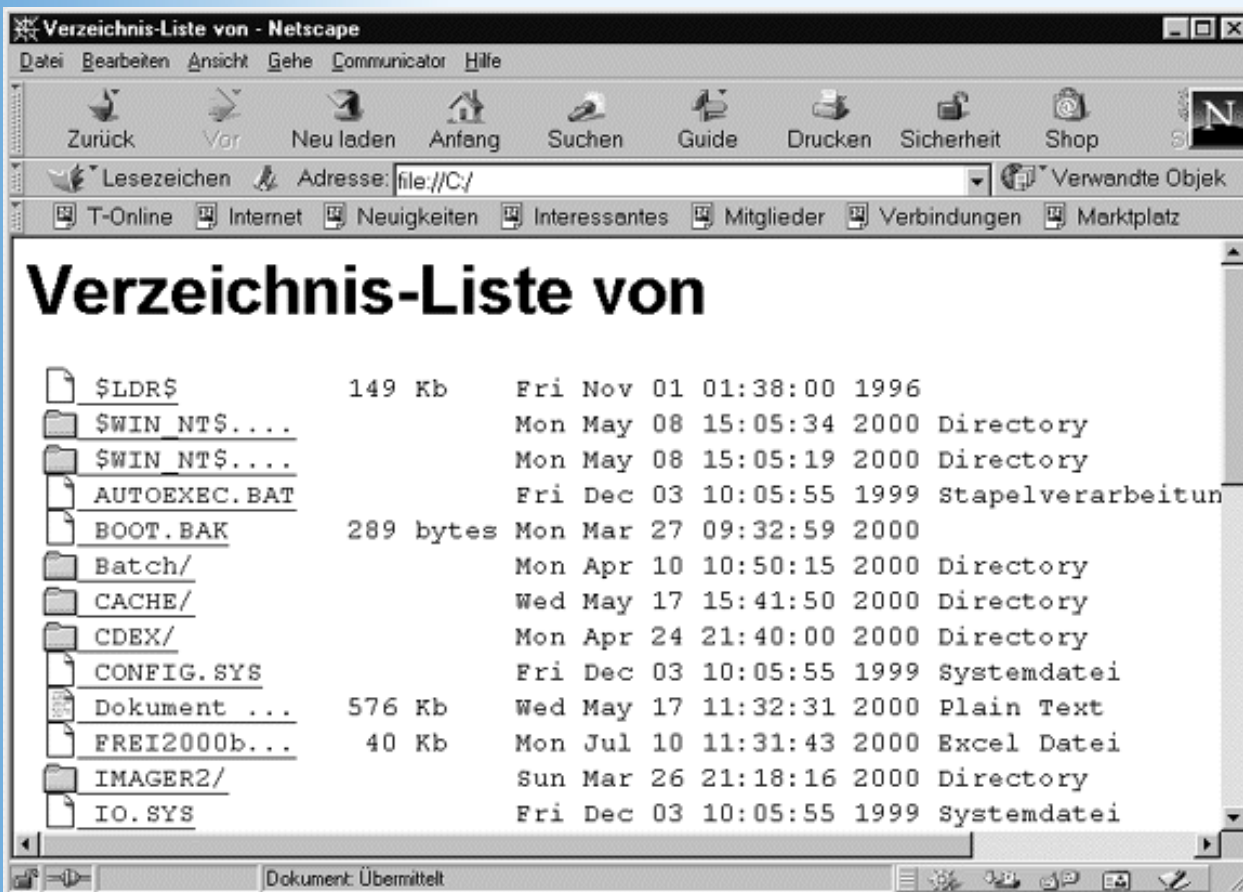


Abbildung 5.10: So sieht der Navigator beim Zugriff über file auf C: aus

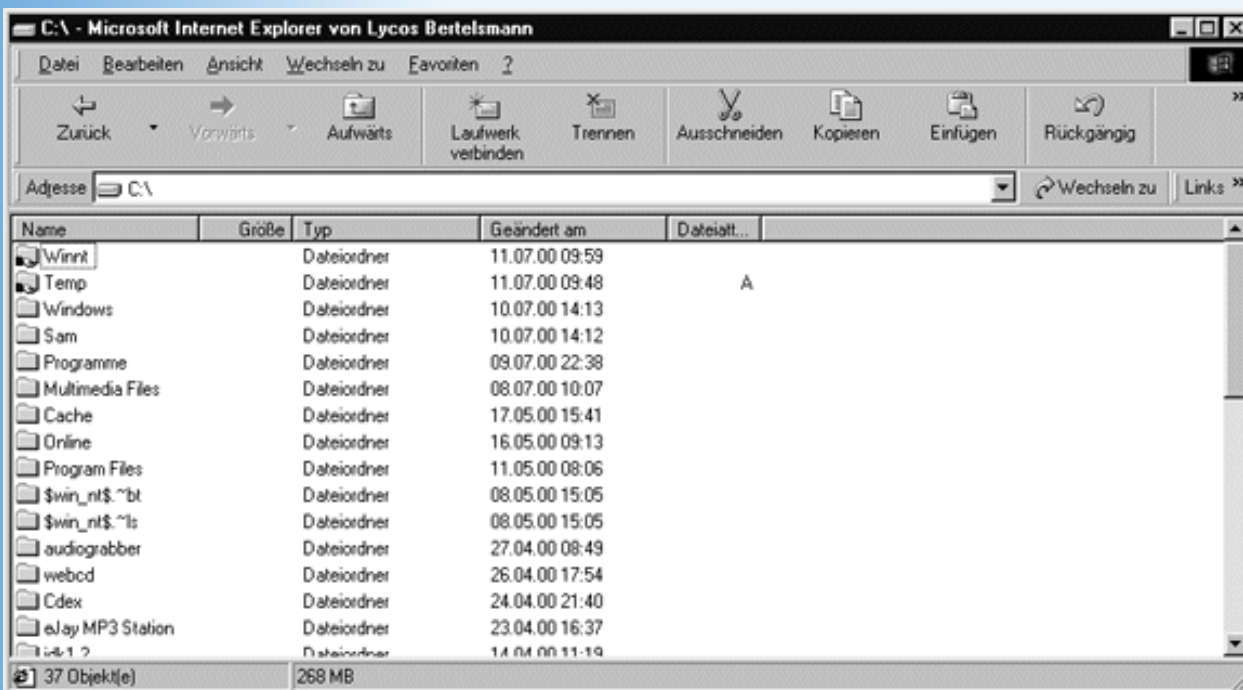


Abbildung 5.11: Das ist in der Tat noch der Internet Explorer - beim Zugriff über file auf C:

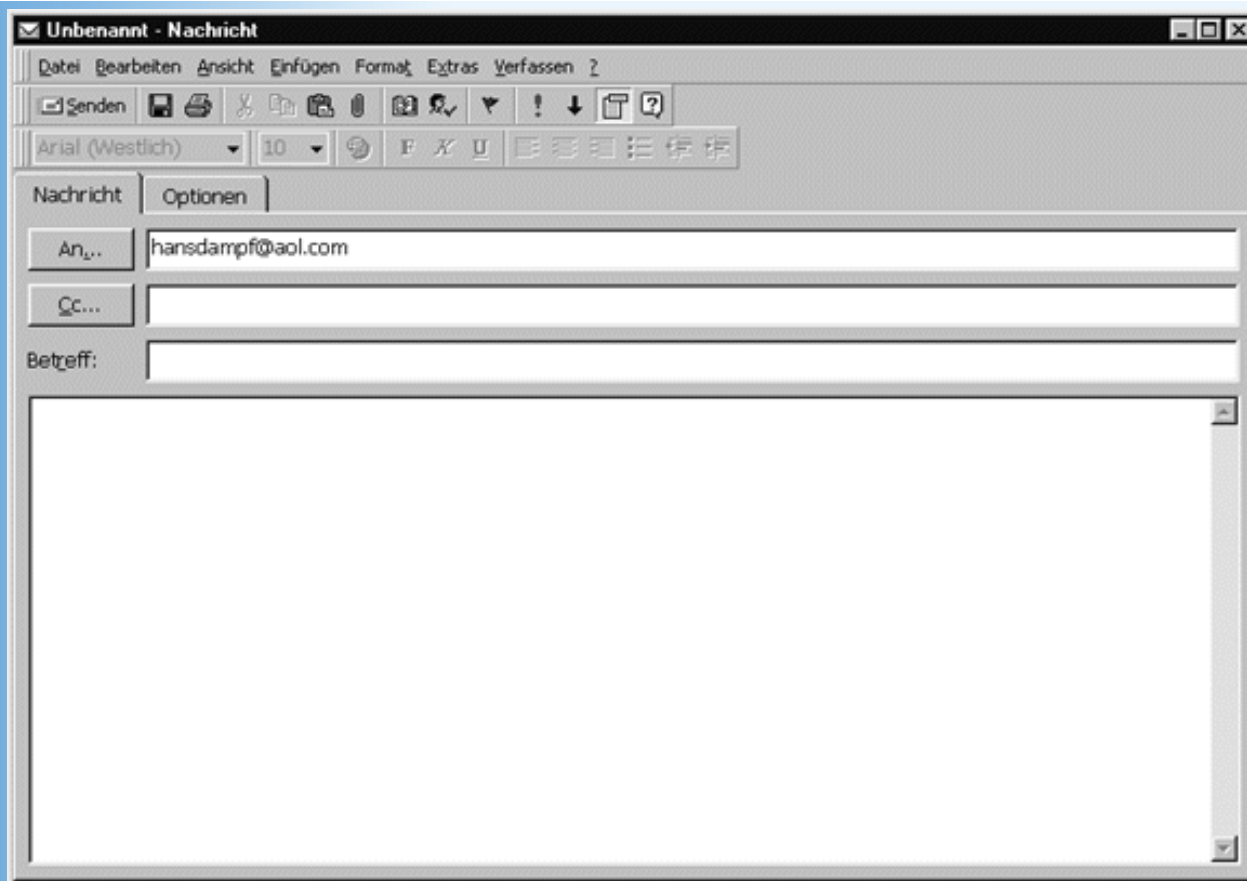


Abbildung 5.12: Der Mail-Client wird mit vorbelegter Adresse gestartet

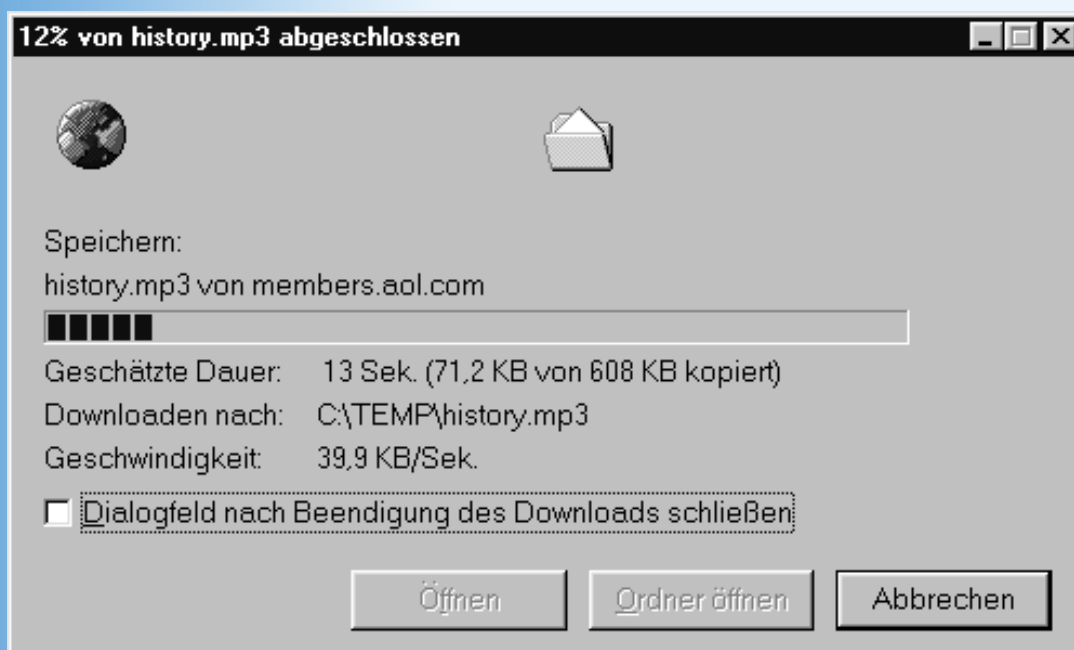


Abbildung 5.13: Der HTTP-Download läuft

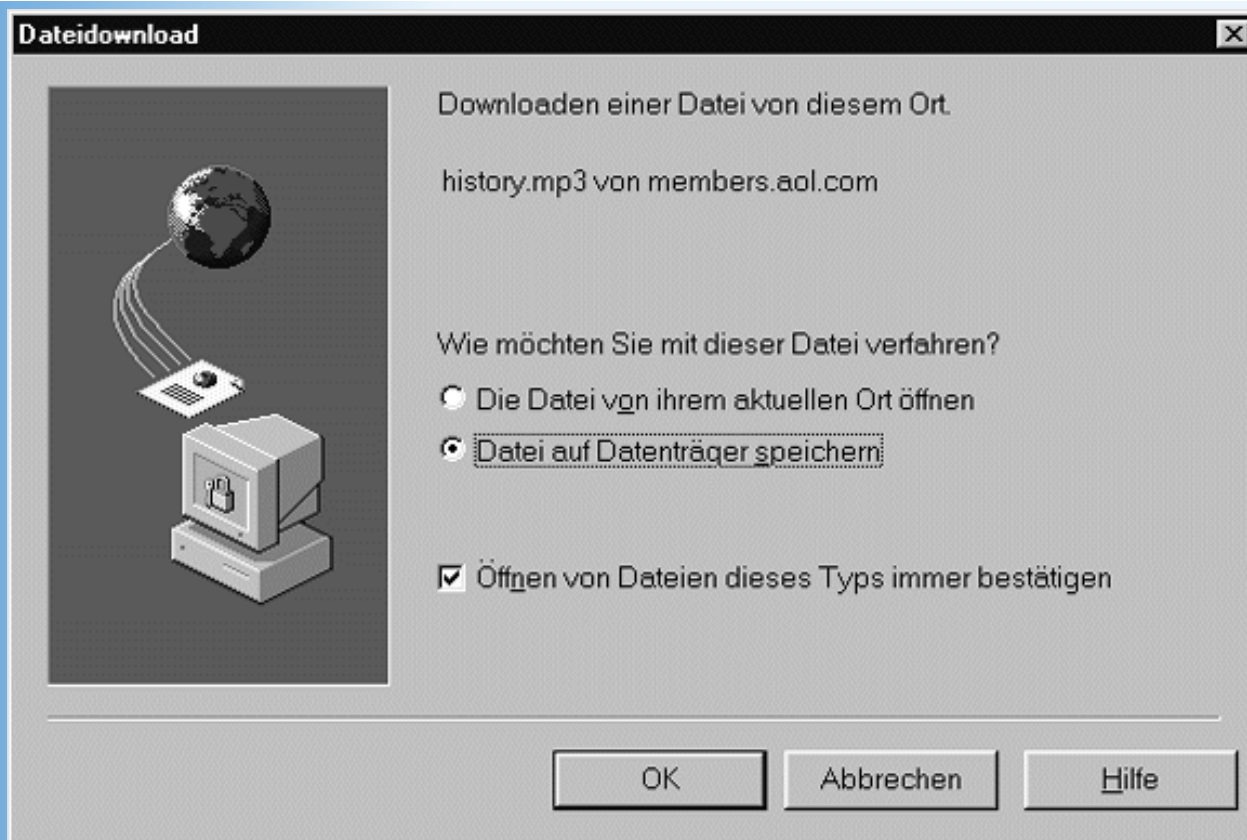


Abbildung 5.14: Auch mit HTTP funktioniert ein Download

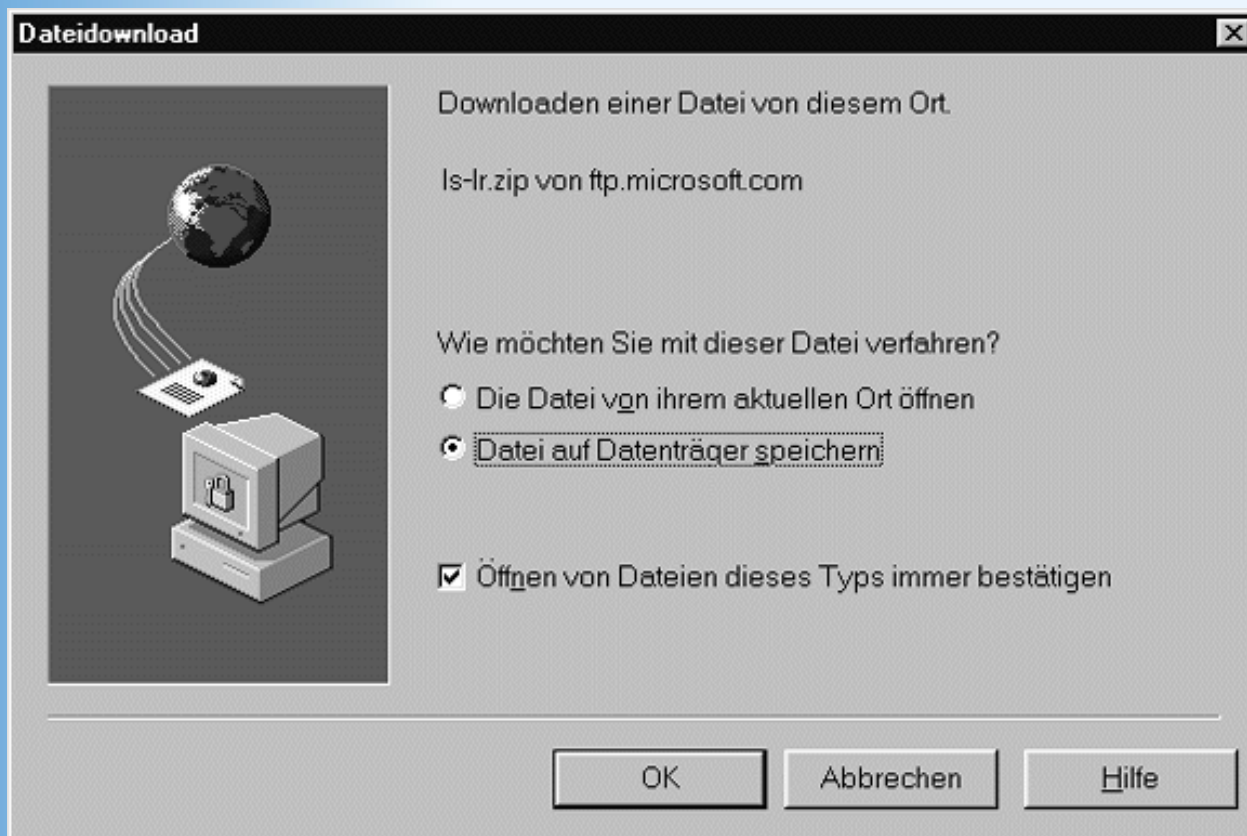


Abbildung 5.15: Der FTP-Download wird gestartet

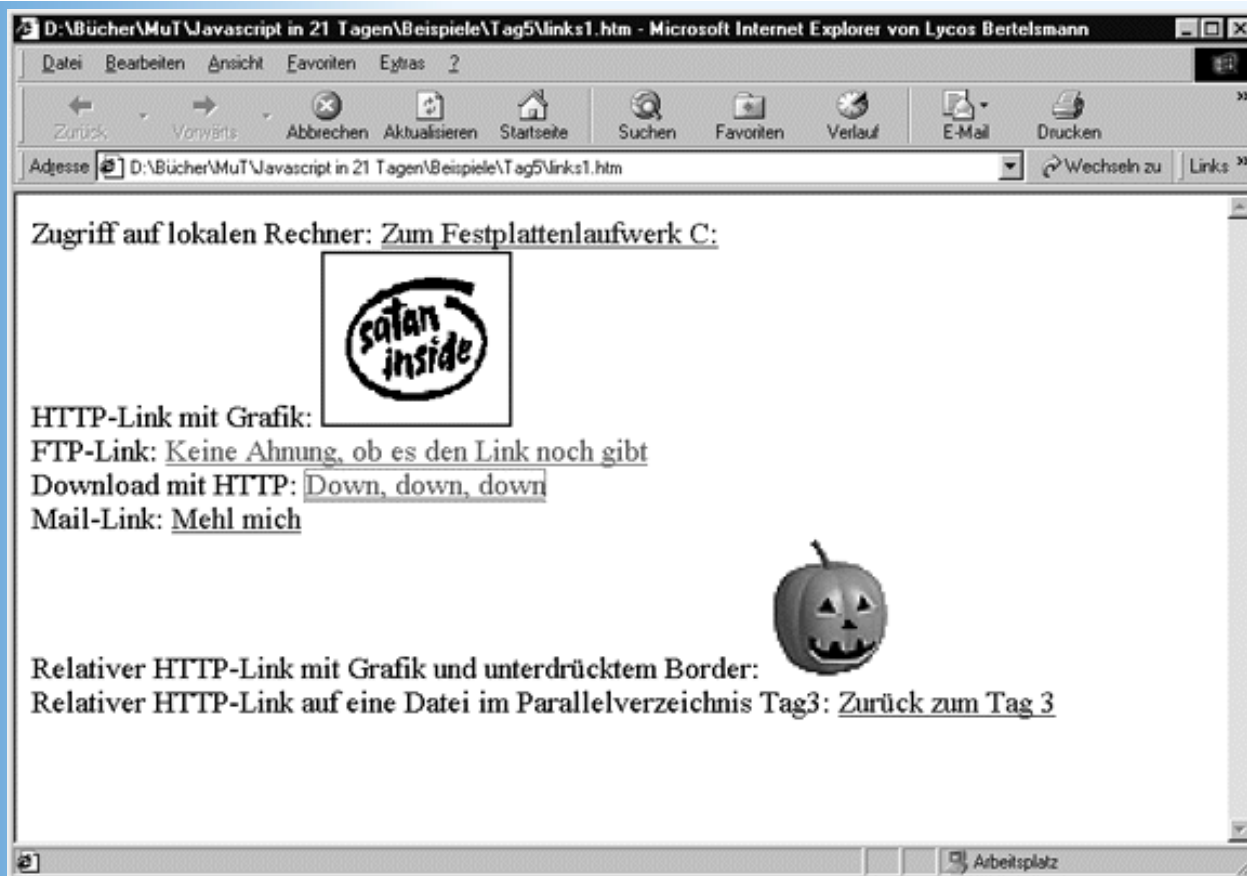


Abbildung 5.16: Die Seite im Internet Explorer

Speichern Sie den Text bitte unter dem Dateinamen Links1.htm ab und schauen Sie sich die Datei im Browser an. Testen Sie die Links. Für die Online-Links benötigen Sie Internet-Kontakt, die relativen und lokalen Links funktionieren offline. Die referenzierten Dateien müssen aber selbstverständlich vorhanden sein. (siehe Abbildungen 5.10-5.16).

Verweise innerhalb einer Seite

Wenn verschiedene Verweise innerhalb einer (meist recht großen) Seite gesetzt werden, erlaubt dies einem Besucher ein schnelles Orientieren und Aufsuchen von für ihn interessanten Themen ohne langwieriges Scrollen. Für solche Verweise innerhalb einer Seite müssen zwei Dinge zur Verfügung stehen:

- Ein Link, der den Verweis in der Seite auslöst.
- Ein Sprungziel für den Verweis. Im Gegensatz zu allgemeinen Links laden Sie ja keine neue Datei. Das Sprungziel eines Verweises muss eine eindeutige Bezeichnung bekommen. Dies ist der so genannte Anker.

Die Anweisung für den Anker baut wie der Link selbst auf dem Tag <A>- auf. Der Tag wird aber mit dem Attribut name="[Name]" erweitert, das den Ankernamen angibt.

Gesetzt wird der Anker an der Stelle, wohin der Sprung erfolgen soll. Ankernamen sollten nicht zu lang sein und dürfen keine Leerzeichen enthalten. Deutsche Umlaute und Sonderzeichen (bis auf den Unterstrich _) sind verboten. Der Name des Ankers wird in Anführungszeichen gesetzt. Ein Anker-Container kann jeden beliebigen Inhalt umschließen oder auch leer sein. Das ist nicht entscheidend, da ein Anker keinerlei optische Auswirkung in der Webseite hat. Insbesondere wird ein Anker normalerweise nicht durch zusätzlichen Text gekennzeichnet, sondern markiert nur bereits vorhandenen Inhalt. Umschließt ein Anker-Container ausdrücklich einen Inhalt, wird bei einem Sprung zum Anker der Anfang des Inhaltes links oben im Browserfenster dargestellt, ansonsten der direkt dem Ende-Tag folgende Inhalt.

Der Link zu dem Anker ist nahezu identisch zu den bisherigen Varianten. Es muss nur noch der Ankernamen - über ein # abgetrennt - dem normalen URL nachgestellt werden. Für den Fall einer Ankerreferenz innerhalb einer Seite besteht die gesamte Referenz sogar nur aus dem Zeichen # und dem Ankernamen. Im Gegensatz zu Ankernamen, die eindeutig sein müssen und nur einmal in einer Seite vorkommen dürfen, können Links zu einem Anker beliebig oft innerhalb einer Seite auftauchen. Beispielsweise der Sprung zum Ende der Webseite von verschiedenen Stellen aus.



Wenn Sie einen Anker auf einem fremden Rechner oder auch nur in einer anderen Datei in Ihrem Projekt

referenzieren wollen, haben Sie natürlich das Problem, dass Sie die Struktur der dortigen Webseite kennen müssen. Insbesondere sind Sie damit abhängig davon, dass in der anderen Seite Anker nicht gelöscht oder umstrukturiert werden. Deshalb findet man selten Ankerverweise auf fremde Seiten. Für den Fall, dass ein Ankernamen bei einem Link nicht vorhanden ist, wird an den Anfang einer Seite verzweigt.

Beispiel 10:

Erstellen Sie zuerst eine Datei, in der ein Anker gesetzt ist und den wir dann aus einer anderen Datei direkt referenzieren wollen. Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
Vor der ersten Grafik:
<P>
<IMG src="bild3.jpg" width=600>
<P>
<A name="danach">Nach der Grafik.</A>
<BR>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen nureinanker.htm ab.

Erstellen Sie nun die Datei, in der sowohl einige Anker gesetzt werden, aber auch die Links zu den Ankern und auch zu dem Anker in der Datei nureinanker.htm. Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<BODY>
<A name="eins"></A>
Vor der ersten Grafik:
<BR>
<A href="#drei">Spring 3</A>
<BR>
<A href="#zwei">Spring 2</A>
<P>
<IMG src="nodos.pcx" width=200>
<P>
<A name="zwei"></A>
Nach der ersten Grafik, aber vor der zweiten Grafik.
<BR>
<A href="#eins">Spring 1</A>
<BR>
<A href="#drei">Spring 3</A>
<BR>
<IMG src="bild1.jpg" width=400>
<BR>
Nach der zweiten Grafik, aber vor der dritten Grafik.
<BR>
<A name="drei"><IMG src="bild2.jpg" width=400></A>
<BR>
<A href="#eins">Spring 1</A>
<BR>
<A href="#zwei">Spring 2</A>
<BR>
<A href="nureinanker.htm#danach">
Ein Ankerverweis auf in eine externe Datei</A>
<BR>
<A href="nureinanker.htm">
Der Verweis auf die gleiche Datei ohne Anker</A>
</BODY>
</HTML>
```




Abbildung 5.17: Die Einsprungsposition in der externen Datei

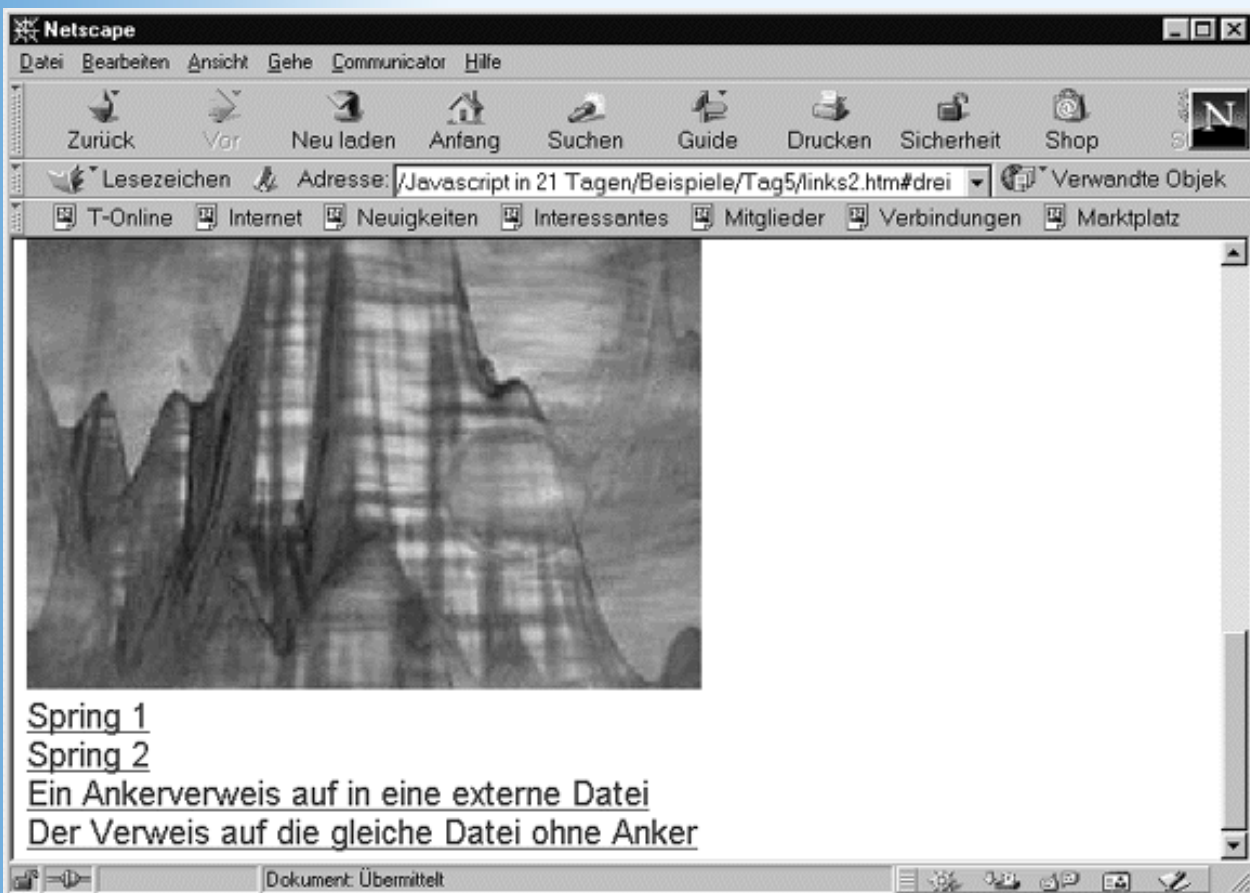


Abbildung 5.18: Ankerposition 3



Abbildung 5.19: Ankerposition 2



Abbildung 5.20: Der Anker am Anfang der Datei

Speichern Sie die Datei unter Links2.htm und schauen Sie sich die Datei im Browser an. Testen Sie die Links. Das Browserfenster sollte ausreichend verkleinert werden, damit die Sprünge auch sichtbar werden. Beachten Sie am Ende die Referenzen auf die Datei nureinanker.htm.

Zusammenfassung

Sie haben am heutigen Tag grundlegende Layoutanweisungen von HTML kennen gelernt. Insbesondere wurde erneut darauf verwiesen, dass unter HTML der Ersteller einer Webseite das Layout beim späteren Betrachter nicht bis in das letzte Detail kontrollieren kann und auch nicht muss. Wichtige Anweisungen zur Gestaltung, die Sie heute kennen gelernt haben, waren

- die Zeilenschaltung über
 und <NOBR> zur Unterdrückung automatischer Umbrüche,
- die Absatzschaltung über <P> und deren Ausrichtungsmöglichkeiten,
- die Ausrichtungen von Objekten über <DIV> und <CENTER>,
- Gruppierungen über ,
- Trennlinien über den <HR>-Tag,
- Überschriften,
- Probleme bei der Darstellung von Sonderzeichen und deren Maskierung,
- Textdarstellungen über logische und physische Befehle und
- den -Tag zur Schriftbeeinflussung.

Neben diesen integrierten HTML-Gestaltungsmöglichkeiten wurden externe Referenzen in Form von Grafikeinbindung über den -Tag und das absolut grundlegende Thema der Hyperlinks besprochen.

Fragen und Antworten

Frage:

Was ist mit der Verwendung von Java-Applets, allgemeinen Multimedia-Objekten, Formularen, Frames, Tabellen und anderen komplexeren Elementen unter HTML?

Antwort:

Darauf kommen wir später zurück. Insbesondere in direkter Verbindung mit JavaScript.

Frage:

Es wird im Buch immer wieder darauf hingewiesen, dass man unter HTML bei vielen Vorgängen keine absolute Kontrolle hat. Wie kommt es, dass es dennoch viele Seiten gibt, die auf fast allen Plattformen gleich aussehen?

Antwort:

Entweder sind diese Seiten nur mit einfachen HTML-Befehlen aufgebaut, die wenig oder gar keinen Interpretationsspielraum lassen und zudem so ideal gestaltet, dass weder zu viele Farben noch eine besondere Auflösung vorausgesetzt werden. Oder es wird auf ergänzende Techniken wie JavaScript und Style Sheets zurückgegriffen. Dann aber muss meist ein erheblicher Aufwand getrieben werden, um jede Plattform individuell zu bedienen. Grundsätzlich ist es aber fraglich, das Designempfinden von Medien wie Papier oder einer wenig heterogenen Bildschirmausgabe unter Windows auf das Internet zwingen zu wollen. Gute Designer spielen mit den optischen Unwägbarkeiten des Webs und freuen sich, dass man nicht alles im Web festlegen muss.

Frage:

Wieso suggerieren viele HTML-Editoren, ein Ersteller einer Webseite könne Schrift absolut festlegen?

Antwort:

Die Erstellung von Webseiten soll so einfach wie möglich werden. Viele Tools und auch Webspeicherplatzanbieter locken mit dem Versprechen, ein Anwender brauche kein HTML zu können und könne dennoch eine Homepage erstellen. Sowohl hinsichtlich der Bedienung des Programms als auch des Verständnisses des Webs soll vom Anwender nicht viel gefordert werden. Das macht Mühe und reduziert die Kundenanzahl.

Workshop

Wir haben an dem heutigen Tag viel Praxis in Form von Beispielen gehabt. Der Workshop baut auf diesen Beispielen auf. Vergegenwärtigen Sie sich nochmals die entsprechenden Beispiele und experimentieren Sie damit. Zusätzlich sollten Sie ins Internet gehen und sich dort verschiedene Webseiten ansehen. Analysieren Sie deren Quelltext nach Elementen, die Sie jetzt bereits kennen. Wie Sie den Quelltext ansehen können, haben wir im Workshop an Tag 4 besprochen.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was bedeutet der Tag <H3>?
2. Wie erzeugt man eine Trennlinie?
3. Mit welchem Tag wird eine Grafik referenziert?
4. Wie verhindert man einen Rahmen bei einer als Hyperlink verwendeten Grafik?
5. Was bedeutet der Tag <I>?
6. Was bedeutet der Tag ?
7. Wie erzeugt man eine Absatzschaltung?
8. Was ist ein Anker?
9. Wie kann man einen Absatz rechts ausrichten?

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

1

Erinnern Sie sich an die Vorstellung der Browser? Opera führt diese komfortable Deaktivierungsmöglichkeit ja als Vorteil des Browsers an.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 1

Tag 6

JavaScript-Praxis

An unserem sechsten Tag soll nun JavaScript-Praxis ganz im Blickfeld stehen. Dies bedeutet, es wird eine ganze Reihe von ausgesuchten Beispielen durchgesprochen, welche diverse Aspekte des Einsatzes von JavaScript zeigen.

Wir zeigen zuerst, wie mit JavaScript ein dynamischer Aufbau von Webseiten erfolgen kann. Dies wird insbesondere dann interessant, wenn in eine Webseite Informationen eingearbeitet werden sollen, die beim Erstellen einer Webseite nicht zur Verfügung stehen oder sich bei jedem Anwender unterscheiden. Anschließend kümmern wir uns um erste Versuche mit Kontrollanweisungen in JavaScript sowie die Programmierung des Browsers aus einer Webseite heraus. Der vierte Abschnitt wird erste Experimente mit dem Zugriff auf die Anwender- Plattform - auch jenseits des Browsers - durchführen.

Dynamischer Aufbau von Webseiten

In vielen Situationen können bei der Erstellung von Webseiten noch nicht alle Tatsachen bekannt sein, die zum vollständigen Anzeigen aller Fakten notwendig wären. Etwa bei einer Datenbankabfrage, wie sie beispielsweise im Rahmen einer Suchmaschine vorkommt. Oder die Integration eines aktuellen Tagesdatums in eine Webseite. Oder die Integration einer Benutzereingabe in die Webseite selbst. JavaScript bietet dazu die Lösung. Über die Anweisung

```
document.write()
```

kann eine Webseite dynamisch geschrieben werden. Damit kann man beispielsweise statt einer überall gleich aussehenden Webseite eine für jeden Besucher individuell angepasste Seite generieren. Wir werden auf die Hintergründe noch eingehen. Nur soweit vorab - die Webseite wird beim Aufruf dieser Technik vollständig neu geschrieben. Deshalb steht üblicherweise nichts im Body-Bereich einer solchen Seite, sondern dessen Inhalt wird inklusive aller notwendigen Informationen (auch sämtlicher HTML-Befehle) dynamisch geschrieben. Wenn Sie die nachfolgenden Beispiele durchspielen, wird das sicher klarer. Dabei sollten alle Beispiele der Reihe nach behandelt werden, da die Erklärungen teils aufeinander aufbauen.

Beispiel 1:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var a;
a = prompt("Bitte geben Sie Ihren Namen ein","");
```

```
document.write("<HTML>");
document.write("<BODY bgcolor=green text=yellow>");
document.write("<H1 align=center>Willkommen</H1>");
document.write("<P align=center>");
document.write(a);
document.write("</P>");
document.write("</BODY>");
document.write("</HTML>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen NamenInWebSeite.htm ab und schauen Sie sich die Datei im Browser an. Über prompt() wird beim Laden der Seite ein Benutzerdialog angezeigt, in dem der Besucher eine Eingabe vornehmen kann. Diese Eingabe wird in einer Variablen zwischengespeichert und dann bei der dynamischen Erzeugung der Webseite verwendet. Beachten Sie, dass das gesamte Grundgerüst der Seite wieder vollständig neu geschrieben wird. Sogar mit globalen Farbangaben. Dies bedeutet, nachdem die Eingabe durch den Anwender durchgeführt wurde, werden dynamisch die angegebenen Farben für den Hintergrund und den Text der Webseite umgestellt.



Abbildung 6.1: Die dynamisch hinzugekommene Information wird in der Webseite verwendet. Außerdem sind die Farben umgestellt

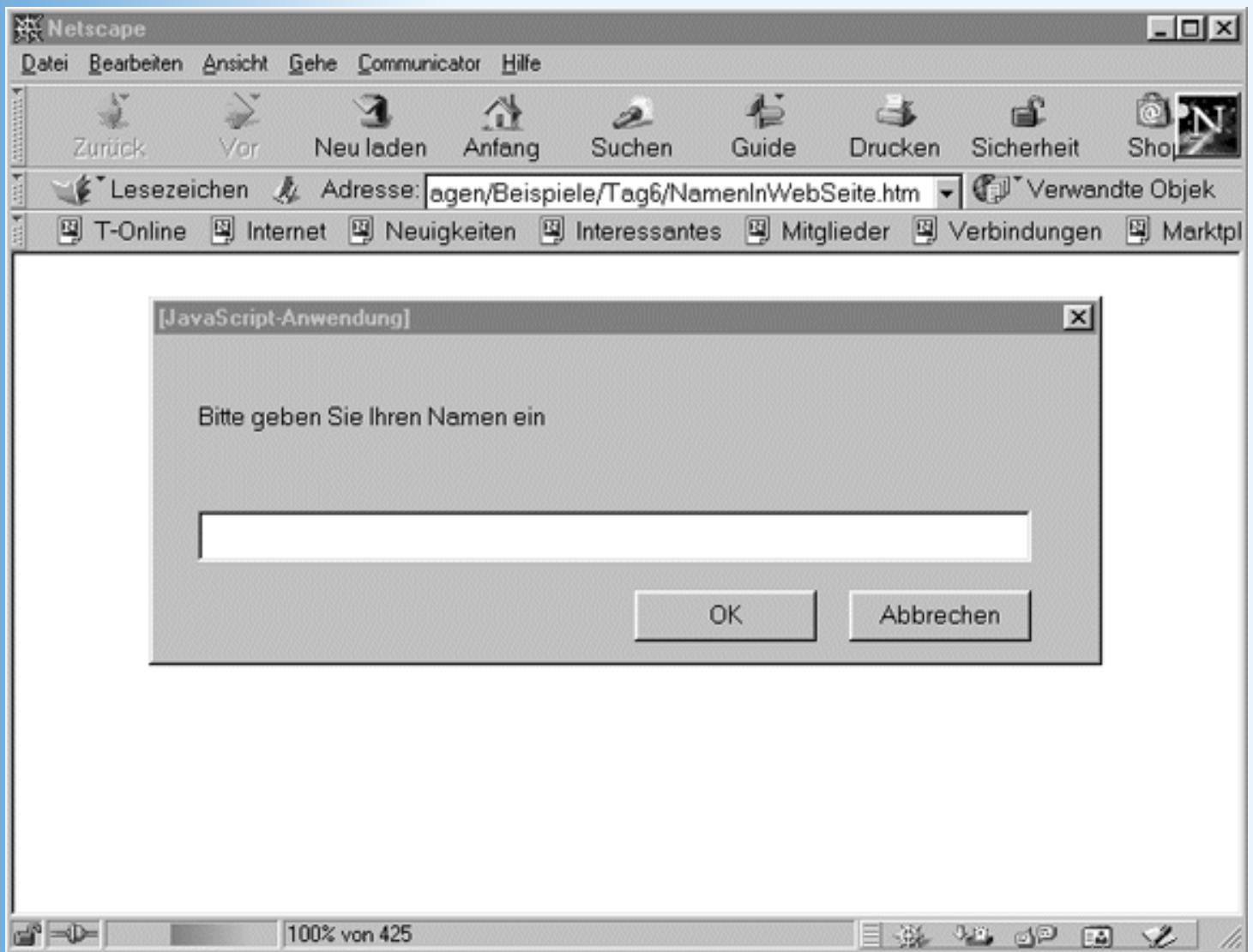


Abbildung 6.2: Die Anzeige im Navigator beim Laden der Webseite - der Anwender gibt seinen Namen ein



Der Netscape Navigator hat in einigen Versionen ein Problem mit der auf die obige Weise deklarierten Variablen. Und zwar dann, wenn er der Standardbrowser ist, er noch nicht gestartet und die Datei mit Doppelklick geladen wird. Ihm ist beim ersten Laden der Seite die Variable unter Umständen nicht bekannt. Beim erneuten Laden der Seite funktioniert das Script einwandfrei (wie im Internet Explorer bereits beim ersten Laden). Aber auch der Navigator hat kein Problem, wenn er bereits gestartet ist und die Datei dann geladen wird (etwa mit der Maus in das geöffnete Browserfenster gezogen). Bei Verzicht auf das Neuschreiben des Grundgerüsts kommt auch der Navigator mit dem Script in den sonst für ihn kritischen Situationen zurecht. Falls dieses Verhalten auftritt, ist es ein Fehler des Browsers, nicht des Scripts, weshalb nicht auf die Programmierung dieses Beispiels auf diese Art und Weise verzichtet werden soll. Das Problem ist freilich nicht gravierend, da so ohne umgebende Funktion notierte Script-Anweisungen in der Praxis so gut wie gar nicht mehr vorkommen. Überdies tritt die kritische Konstellation beim Laden aus dem Internet sowieso nicht auf, da hier ja kein Doppelklick auf die Webseite den Browser startet. Wir werden nachfolgend dennoch einen einfachen Workaround (eine selbstgestrickte Fehlerbereinigung) für den Netscape-Fehler sehen, der auf der Verwendung von einer JavaScript-Funktion beruht. Damit kommt der Navigator dann auch in der gerade beschriebenen Situation zurecht.

Beispiel 2:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function lade()
{
var a = prompt("Bitte geben Sie Ihren Namen ein","");

document.write("<HTML>");
document.write("<BODY bgcolor=green text=yellow>");
document.write("<H1 align=center>Willkommen</H1>");
document.write("<P align=center>");
document.write(a);
document.write("</P>");
document.write("</BODY>");
document.write("</HTML>");
}
</SCRIPT>
</HEAD>
<BODY onLoad=lade(>
</BODY>
</HTML>
```

Beispiel 3:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var a = 42;
var b = 5;
var c;
c=a+b;
document.write(c);
document.write("<br>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Addition.htm ab und schauen Sie sich die Datei im Browser an. Das Beispiel kommt ganz ohne Interaktion mit dem Anwender aus, addiert einfach zwei Variablen und gibt das Ergebnis dann aus. Das Beispiel zeigt, wie Grundrechenarten mit JavaScript verwendet werden können.

Beispiel 4:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
```



```
var a;  
a = prompt("Geben Sie bitte eine Zahl ein","");  
document.write("Das Quadrat von ", a , " ist ", a*a);  
document.write("<br>");  
</SCRIPT>  
</HEAD>  
<BODY>  
</BODY>  
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Quadrat.htm ab und schauen Sie sich die Datei im Browser an. Über prompt() wird beim Laden der Seite wieder ein Benutzerdialog angezeigt, in dem Sie eine Eingabe vornehmen können. Diese Eingabe wird wie im ersten Beispiel in einer Variablen zwischengespeichert und dann bei der dynamischen Erzeugung der Webseite verwendet. Beachten Sie, dass wir diesmal auf die Generierung des Grundgerüsts der Seite verzichtet haben. Dennoch funktioniert der dynamische Neuaufbau der Seite. Es greift das Prinzip der Fehlertoleranz, sprich der Browser »denkt sich die fehlenden Anweisungen hinzu«. Insbesondere treten hier auch die Probleme mit dem Netscape Navigator nicht auf.

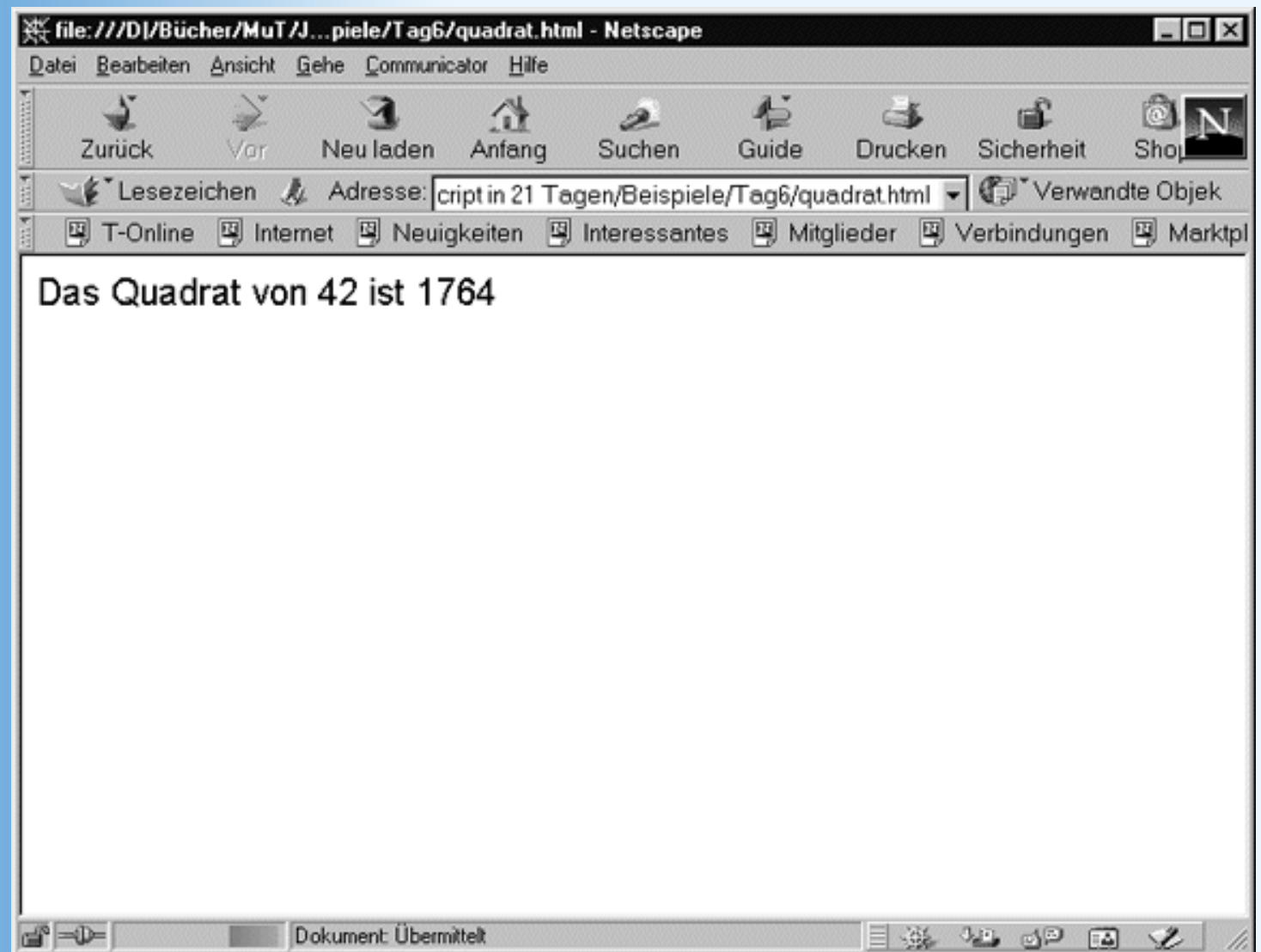


Abbildung 6.3: Die dynamisch hinzugekommene Information wird in der Webseite verwendet

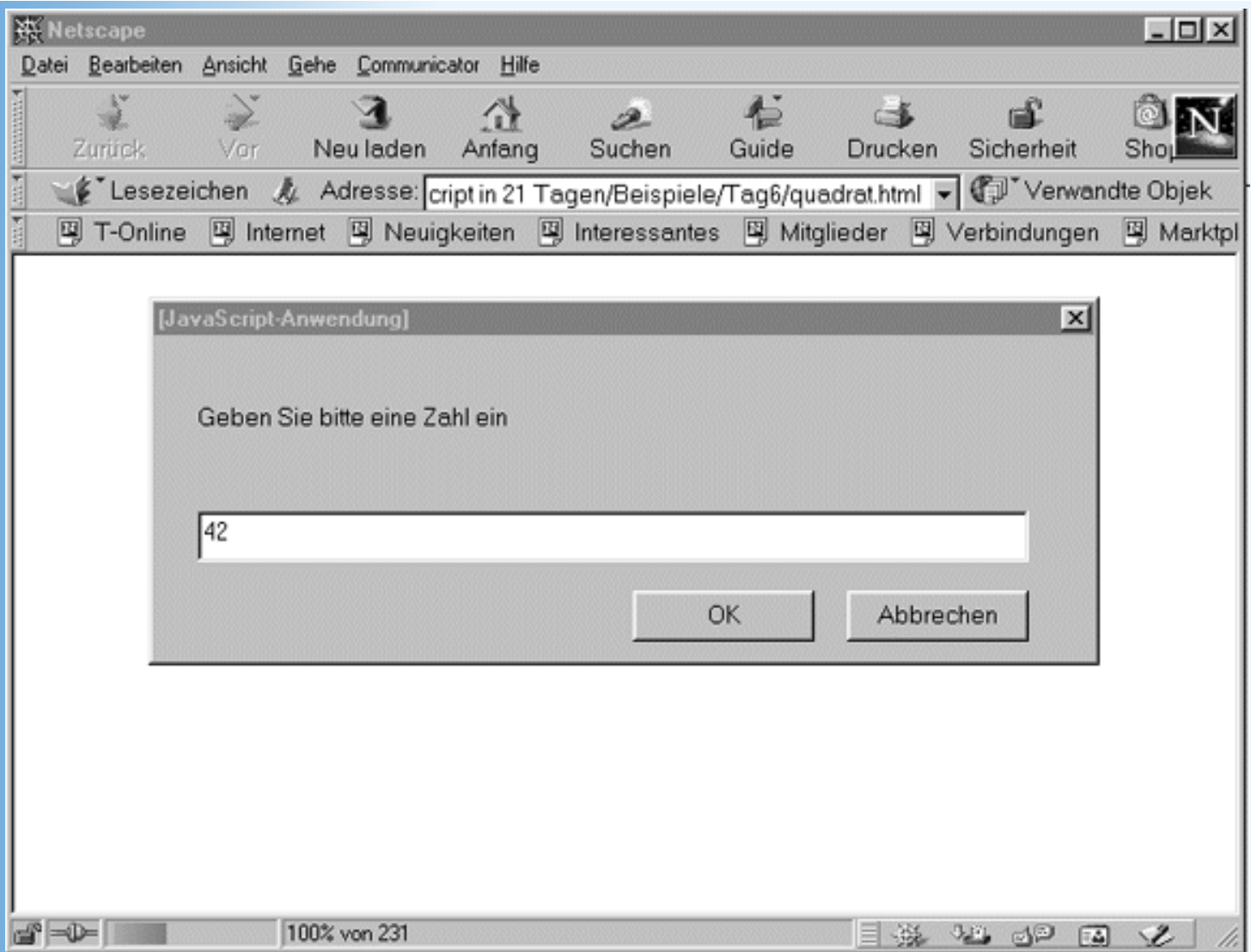


Abbildung 6.4: Die Anzeige im Navigator beim Laden der Webseite. Der Anwender gibt eine Zahl ein

Das Beispiel zeigt, wie man mit JavaScript rechnen kann. JavaScript hat weitgehend nur die Grundrechenarten direkt integriert (von vorgefertigten Funktionen abgesehen), aber man kann jede gewünschte mathematische Funktionalität nachprogrammieren. Hier ist es die Quadratberechnung.



Beachten Sie, dass das Beispiel nicht gegen Falscheingaben - etwa Text - geschützt ist.

Kontrollstrukturen

Ein wesentliches Kriterium, weshalb HTML keine Programmiersprache sein kann, ist das Fehlen von Kontrollstrukturen, um den Programmfluss zu steuern. Schauen wir uns in der Praxis mit JavaScript an, was das bedeutet.

Beispiel 5:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var temp=prompt("Eingabe der Temperatur","");
```

```
if(temp > 15)
{
document.write("Es ist warm genug.");
}
else
{
document.write("Es ist zu kalt.");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Entscheidung.htm ab und schauen Sie sich die Datei im Browser an. Über prompt() wird beim Laden der Seite wieder ein Benutzerdialog angezeigt, in dem Sie eine Eingabe vornehmen können (möglichst eine Zahl - das Beispiel reagiert bei Text nicht sinnvoll). Diese Eingabe wird wie im ersten und dritten Beispiel in einer Variablen zwischengespeichert und dann bei der dynamischen Erzeugung der Webseite verwendet. Das Beispiel verwendet das erste Mal eine Entscheidungsstruktur (die if-Schleife), mit der man in einem Script unterschiedlich auf verschiedene Situationen reagieren kann.



Beachten Sie, dass auch dieses Beispiel nicht gegen Falscheingaben - etwa Text - geschützt ist.

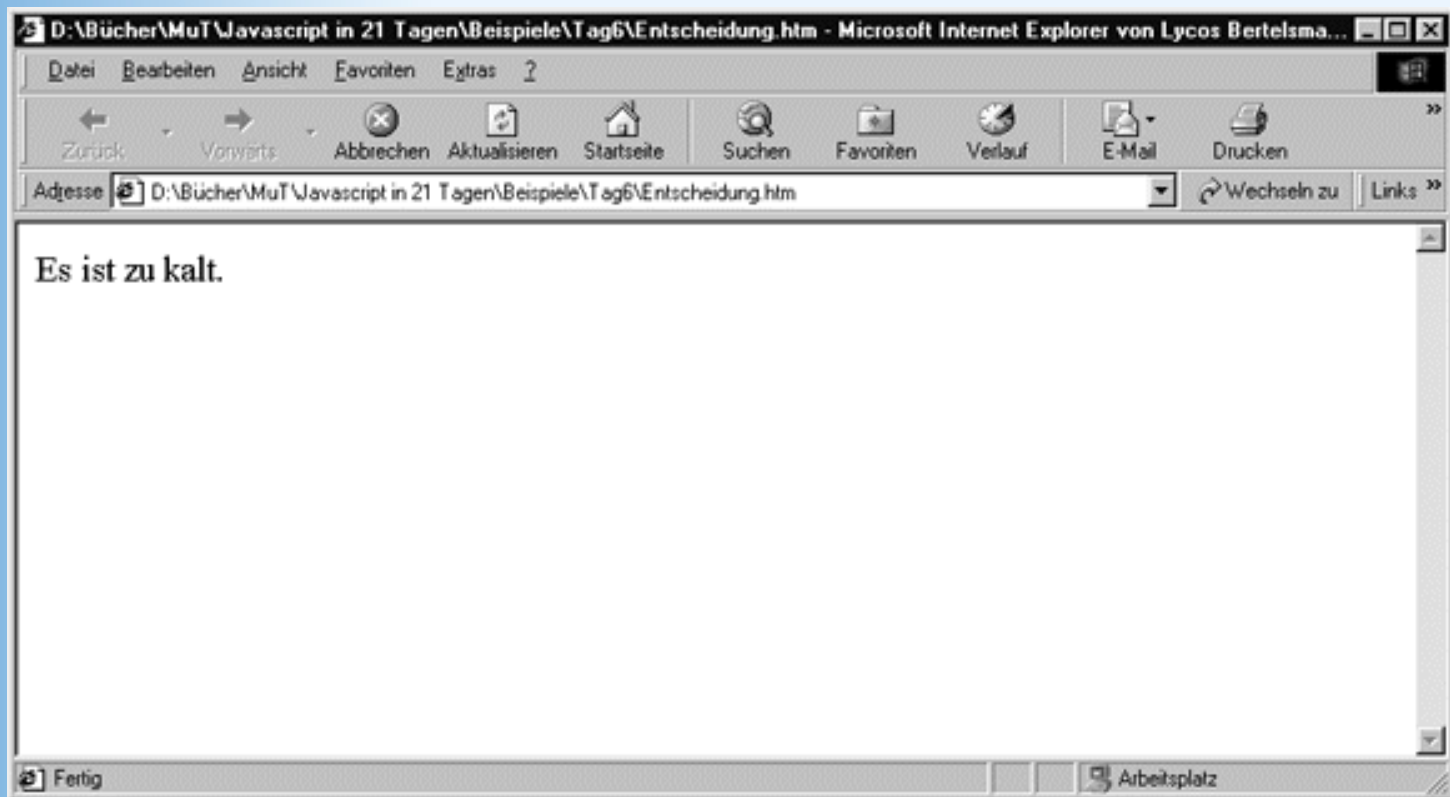


Abbildung 6.5: Entscheidung für die andere Variante (hier am Beispiel der Eingabe »10«).

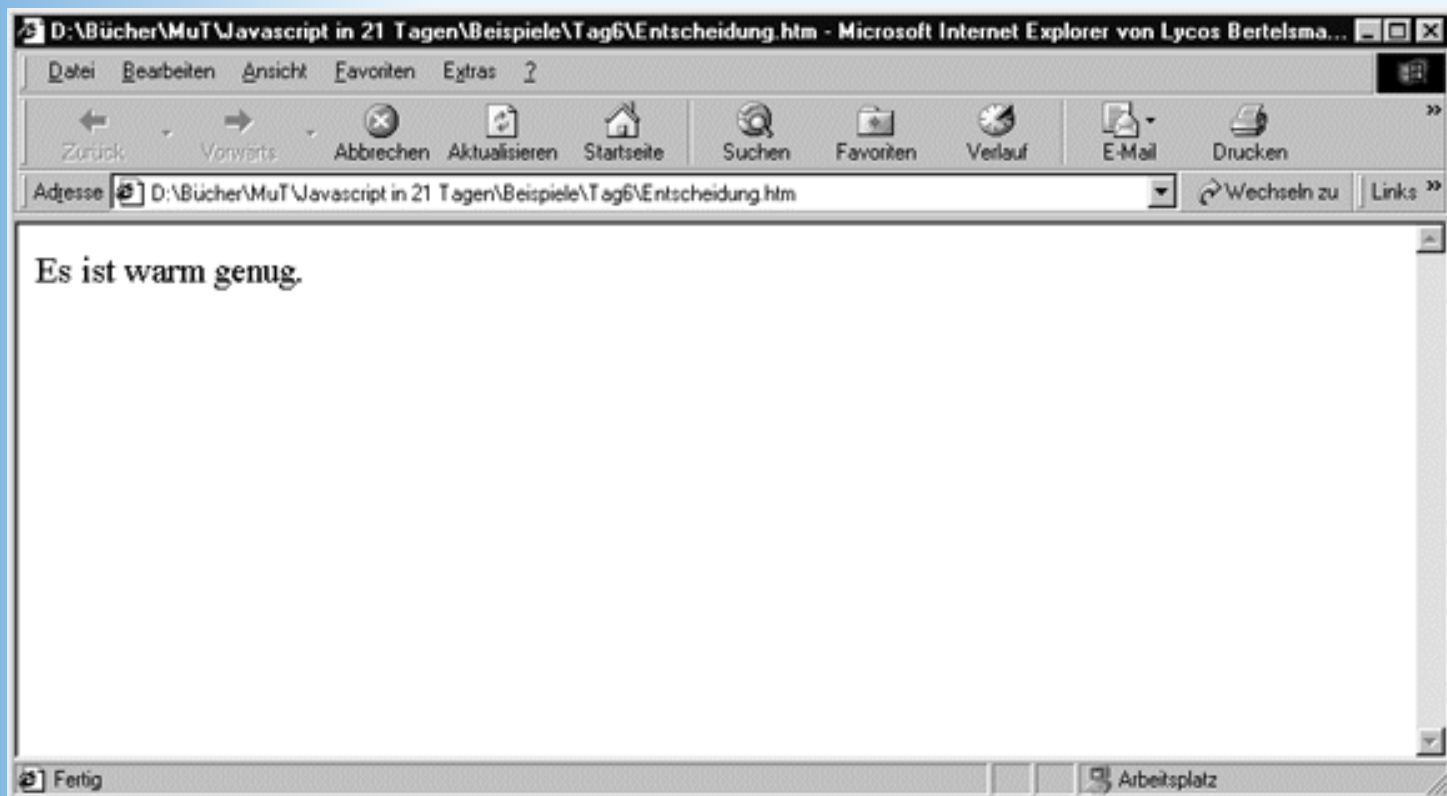


Abbildung 6.6: Entscheidung für die eine Variante

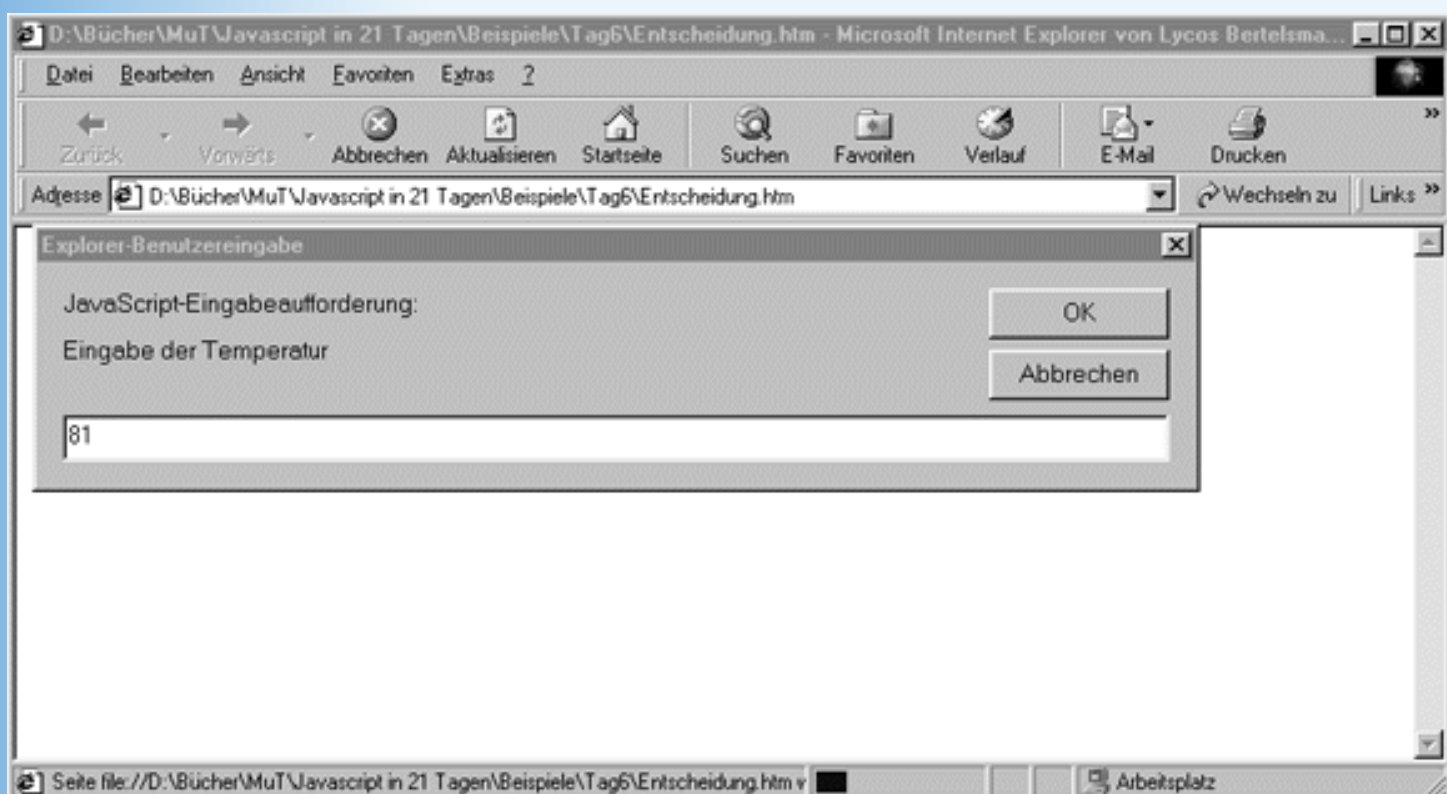


Abbildung 6.7: Die Anzeige diesmal im Internet Explorer beim Laden der Webseite. Der Anwender gibt eine Zahl ein

Beispiel 6:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
```



```
function lade()  
{  
var zaehler=0;  
var anzahl = prompt("Anzahl Leerzeilen", "");  
document.write("<H1>Willkommen</H1>");  
while(zaehler<anzahl)  
{  
document.write("<br>");  
zaehler++;  
}  
document.write("<H3>Nächster Text</H3>");  
}  
</SCRIPT>  
</HEAD>  
<BODY onLoad="lade()">  
</BODY>  
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen DynamischeSeite.html ab und schauen Sie sich die Datei im Browser an. Das Beispiel demonstriert, wie ein Anwender vor dem Aufbau der Seite darauf Einfluss nehmen kann, wie die Seite überhaupt aussieht. Dabei ist für uns von Bedeutung, dass die von dem Anwender eingegebene Zahl in einer Kontrollstruktur (die while-Schleife) verwendet wird. So oft wie dort angegeben, wird ein Zeilenvorschub in die Webseite geschrieben. Neu ist, dass hier mit zwei Variablen gearbeitet wird. Eine nimmt die Anwendereingabe entgegen, die andere zählt mit, wie viele Zeilenvorschübe bereits durchgeführt wurden. Wenn die Anwendereingabe erreicht wird, bricht die Schleife ab.

Besondere Beachtung sollte dem Ausdruck

zaehler++;

gelten. Die Zeichenkette ++ ist ein Operator, der den Wert einer Variablen um eins erhöht. Aber dieser Operator ist so noch nicht vollständig beschrieben. Wir kommen darauf noch detaillierter zurück.

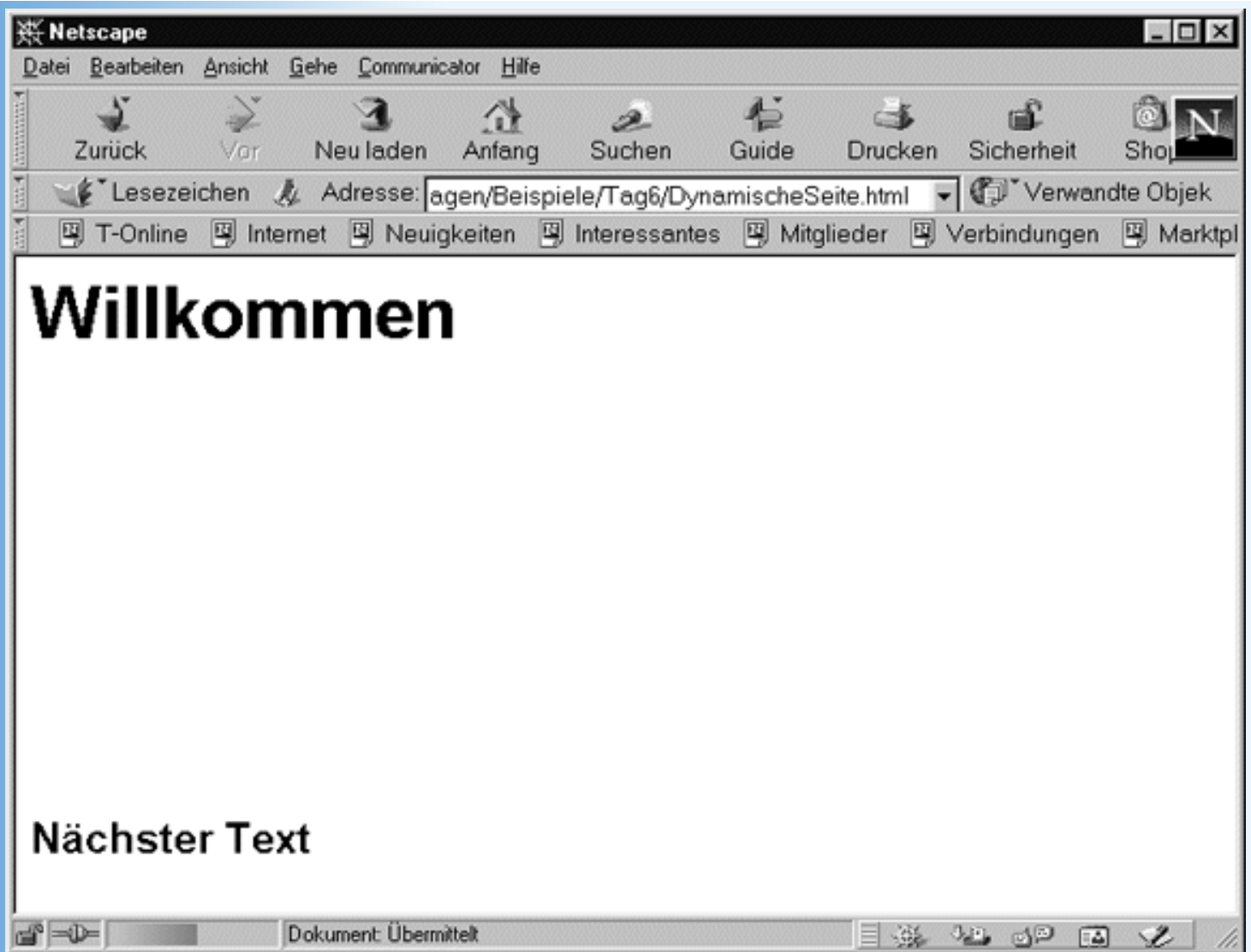


Abbildung 6.8: Der Abstand ist bei der Eingabe »8« groß

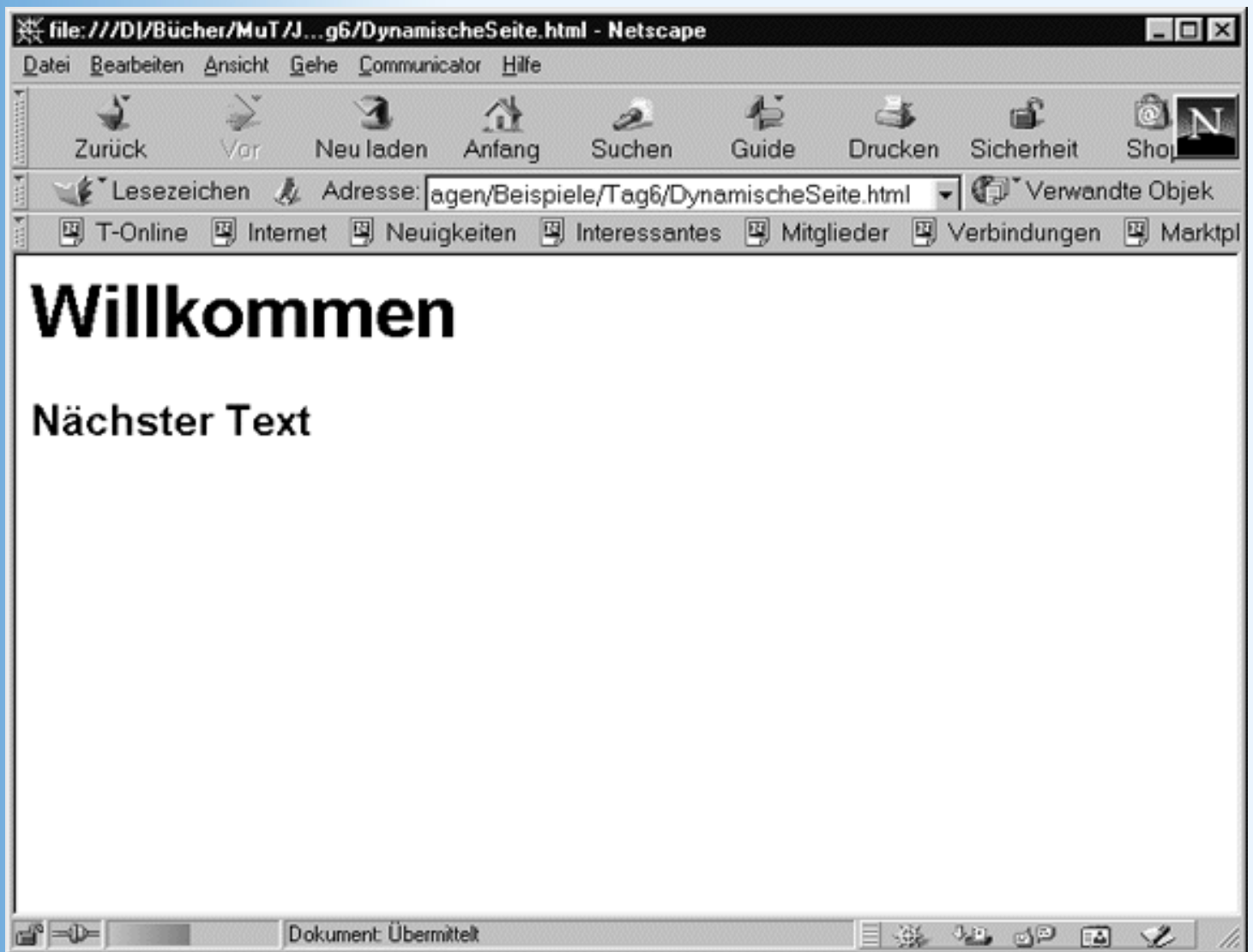


Abbildung 6.9: Der Abstand ist bei der Eingabe »1« klein



Beachten Sie, dass auch dieses Beispiel nicht gegen Falscheingaben - etwa Text - geschützt ist.

Beispiel 7:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var zahl=prompt("Geben Sie eine Zahl ein","");
var zaehler=1;
var ergebnis=1;
while(zaehler <= zahl)
{
    ergebnis=ergebnis * zaehler;
    zaehler++;
}
document.write("Der Wert ist ",ergebnis);
document.write("<br>");
</SCRIPT>
```

```
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Fakultat.html ab und schauen Sie sich die Datei im Browser an. Das Beispiel demonstriert eine weitere Verwendung der Kontrollstruktur while. Der Anwender gibt wieder eine Zahl ein und mittels der while-Schleife wird deren Fakultät errechnet (die Fakultät ist das Produkt aller Zahlen bis zur eingegebenen Zahl - z.B. $6! = 1 * 2 * 3 * 4 * 5 * 6 = 720$). Wenn die Anwendereingabe erreicht wird, bricht die Schleife ab und das Ergebnis wird angezeigt.



Beachten Sie, dass auch das Beispiel nicht gegen Falscheingaben - etwa Text - geschützt ist.

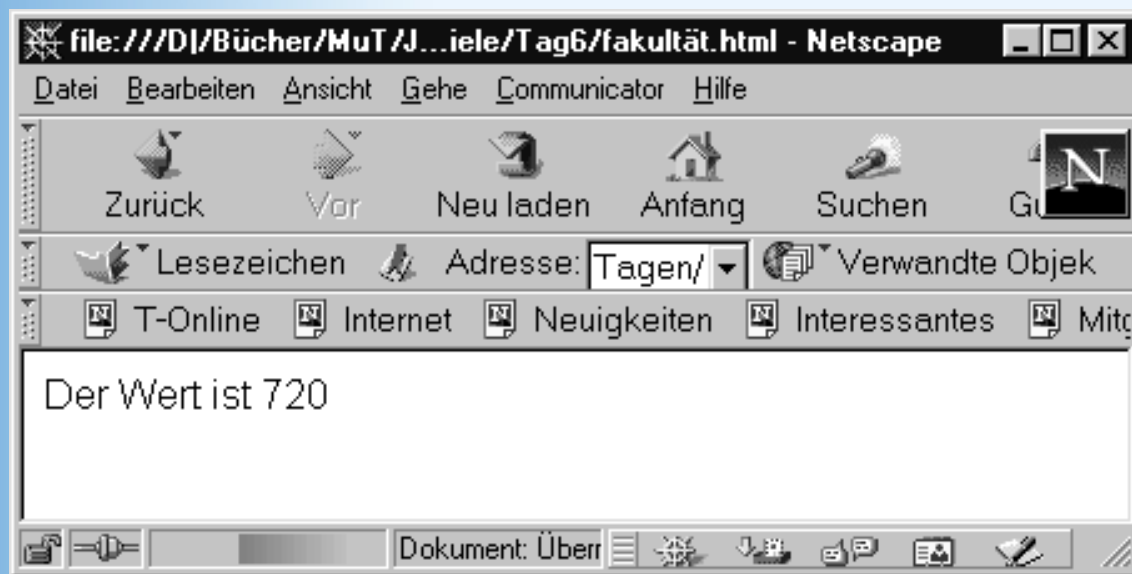


Abbildung 6.10: Ausgabe des Ergebnisses (Eingabe »6«)

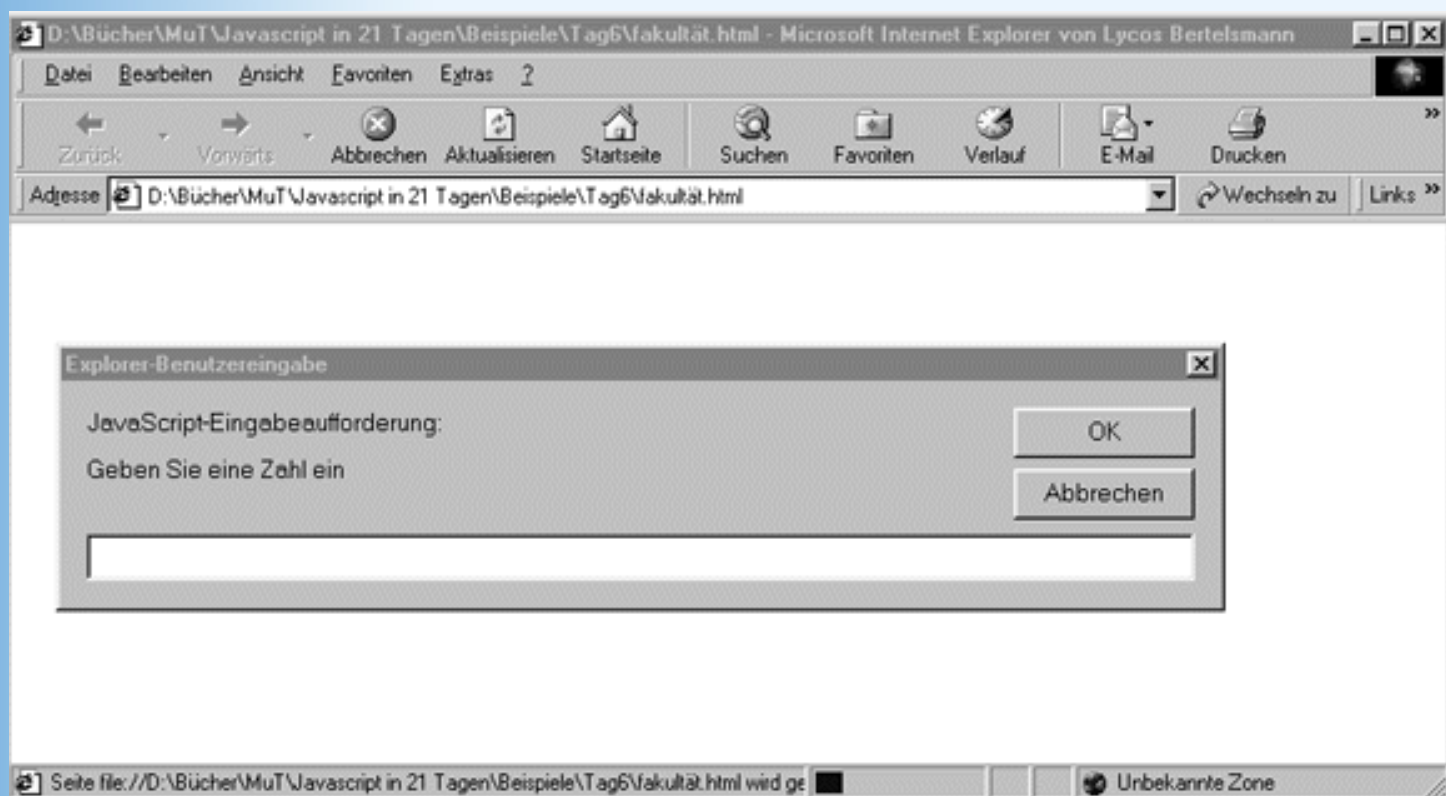


Abbildung 6.11: Die Eingabe über prompt()

Zugriff auf den Browser

Mit JavaScript kann man recht einfach auf den Browser des Anwenders zugreifen und ihn beeinflussen. Die nachfolgenden zwei Beispiele werden einmal eine Weiterleitung auf eine Folgeseite und den Zugriff auf die Statuszeile des Browsers zeigen.

Beispiel 8:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function fenster_neu()
{
location.href='Addition.HTM'
}
window.setTimeout('fenster_neu()', 3000);
</SCRIPT>
</HEAD>
<BODY>
<H1 align=center>Herzlich Willkommen</H1>
<P align=center>
Sie werden in wenigen Sekunden auf eine Folgeseite weitergeleitet.
</P>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Weiterleitung.html ab und schauen Sie sich die Datei im Browser an. Das Beispiel demonstriert eine Weiterleitung von einer Seite auf eine Folgeseite per JavaScript (in diesem Fall auf die Seite Addition.htm, die Sie ja als vorheriges Beispiel noch zur Verfügung haben sollten). Die Weiterleitung erfolgt nach ca. drei Sekunden. Diese Weiterleitung ist eine Alternative zu der Weiterleitung per <META>-Tag, wie wir es an Tag 3 gesehen haben.

Beispiel 9:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function zeigeStatus()
{
window.status="Da gibt es ganz viel zu sehen!"
}
</SCRIPT>
</HEAD>
<BODY onLoad="zeigeStatus()">
<H1 align=center>
Herzlich Willkommen
</H1>
<H2 align=center>
```

Schauen Sie mal in die Statuszeile.

</H2>

</BODY>

</HTML>

Speichern Sie den Text bitte unter dem Dateinamen status.htm ab und schauen Sie sich die Datei im Browser an. Das Beispiel demonstriert, wie auf Teile des Browsers selbst (in diesem Fall die Statuszeile) zugegriffen werden kann.

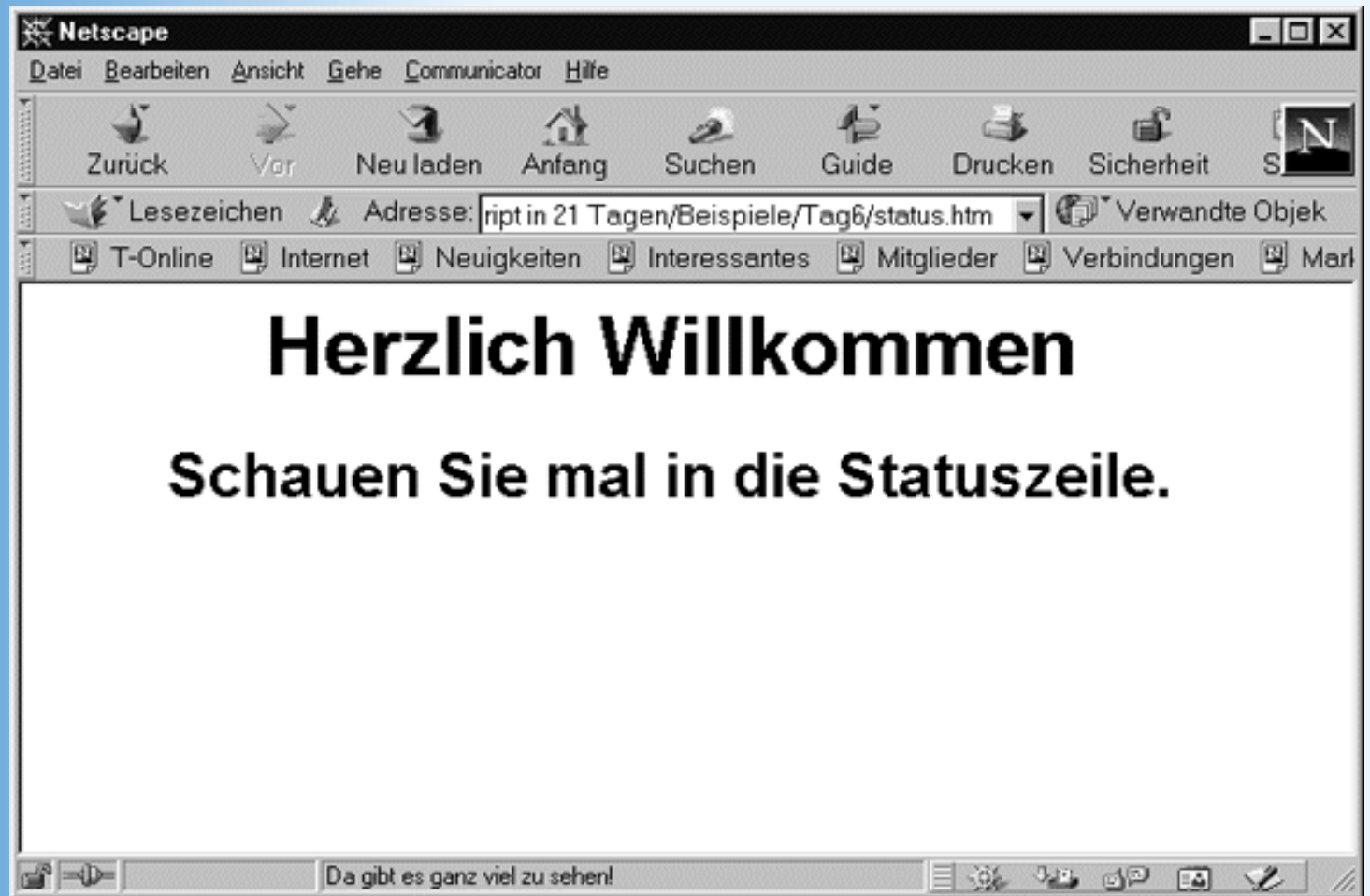


Abbildung 6.12: Beachten Sie die Statuszeile

Zugriff auf die Besucherplattform

Mit JavaScript kann man innerhalb eng gefasster Grenzen auch jenseits des Browsers auf den Rechner des Anwenders zugreifen. Die engen Grenzen bedeuten, es lassen sich nur wenige ausgewählte Dinge tun. Insbesondere kann man - bis auf wenige Ausnahmen, etwa Cookies - nur Informationen abfragen, nicht Werte setzen. Diese Begrenzung erscheint vielleicht auf den ersten Blick als Einschränkung, ist aber genau das Gegenteil.

Wie sollte sich eine Technik durchsetzen, die auf dem Rechner des Anwenders weitergehende Manipulationsmöglichkeiten hat? Der Anwender muss ja eine Aktivität auf seinem Rechner akzeptieren, die jemand Fremdes programmiert und auf seinen Rechner geschickt hat. Nur wenn die verwendete Technik bereits so sicher ist, dass keine schädliche Aktion möglich ist, wird diese Technik von der breiten Masse auch akzeptiert. Das Beispiel VBScript in Verbindung mit ActiveX-Controls zeigt ja, was eine zu weit getriebene Leistungsfähigkeit bewirken kann. Kaum ein Anwender akzeptiert diese Technik und wer es dennoch wagt, ist selbst schuld ;-). Die nachfolgenden zwei Beispiele werden einmal das Systemdatum der Plattform abfragen und in die dynamische Generierung der Webseite einarbeiten (ein kleiner Trick, um dem Anwender ein Topaktualität einer Homepage zu suggerieren - auch wenn sich die Seite seit Ewigkeiten nicht verändert

hat ;-)). Das zweite Beispiel fragt die Auflösung des Bildschirms beim Anwender ab. Dies kann man für weitergehende Aktionen verwenden (das machen wir an späterer Stelle) - wir wollen hier diese Informationen nur anzeigen.

Beispiel 10:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function schreibeSeite()
{
var datum = new Date();
var tag = datum.getDate();
if(tag<10)
{
tag="0" + tag;
}
var monat = datum.getMonth() + 1;
if(monat<10)
{
monat="0" + monat;
}
var jahr = 2000 + (datum.getYear()%100);
document.write("<HTML>");
document.write("<BODY bgcolor=blue text=white>");
document.write(
"<H1 align=center>Herzlich Willkommen</H1>");
document.write("<H3 align=center>bei</H3>");
document.write("<H1 align=center>Strom & Schlag</H1>");
document.write("<H4 align=center>
Ihr Elektriker um die Ecke.</H4>");
document.write("<FONT size=4>");
document.write("<P align=right>");
document.write("Aktueller Stand: ");
document.write(tag);
document.write('.');
document.write(monat);
document.write('.');
document.write(jahr);
document.write("</P>");
document.write("</FONT>");
document.write("</BODY>");
document.write("</HTML>");
}
</SCRIPT>
</HEAD>
<BODY onLoad="schreibeSeite()">
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Tagesdatum.htm ab und schauen Sie sich die Datei im

Browser an. Setzen Sie das Tagesdatum auf einen neuen Wert und laden Sie die Datei neu. Das Beispiel verwendet ein Datums-Objekt (Date), in welchem u.a. das Tagesdatum gespeichert ist. Die Information liegt jedoch noch nicht in einer solchen Form vor, dass man sie direkt verwenden kann (zumindest nicht, wenn man das in Westeuropa übliche Format darstellen will). Deshalb wird das Datum noch über verschiedene Schritte aufbereitet.



Abbildung 6.13: Die Illusion von Tagesaktualität

Beispiel 11:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function frageBildschirmAb()
{
alert("Ihre Bildschirmauflösung ist " +
  screen.width +
  " x " +
  screen.height +
  ".");
}
</SCRIPT>
</HEAD>
<BODY onLoad="frageBildschirmAb()">
</BODY>
</HTML>
```

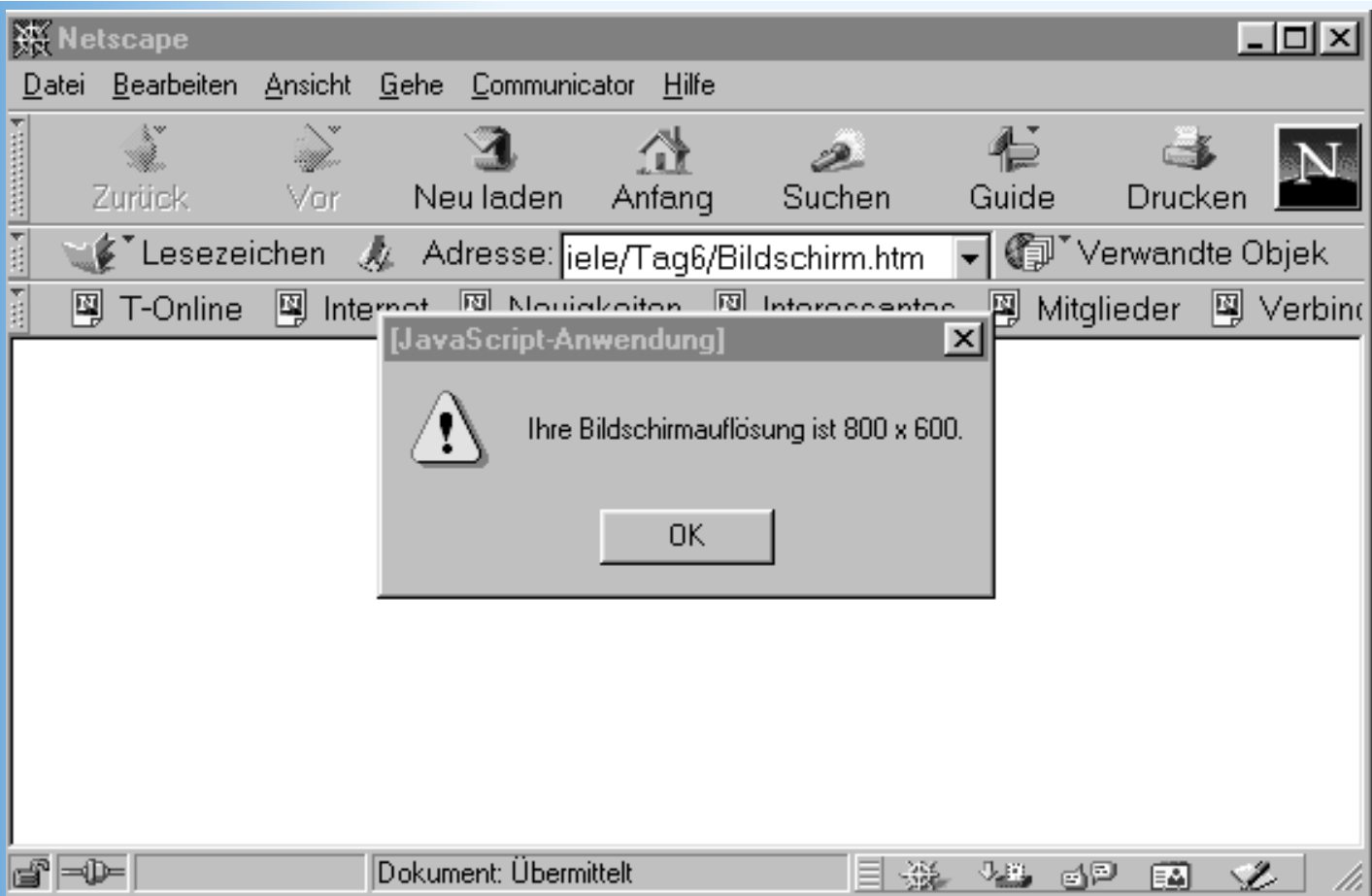



Abbildung 6.14: Auflösung 2

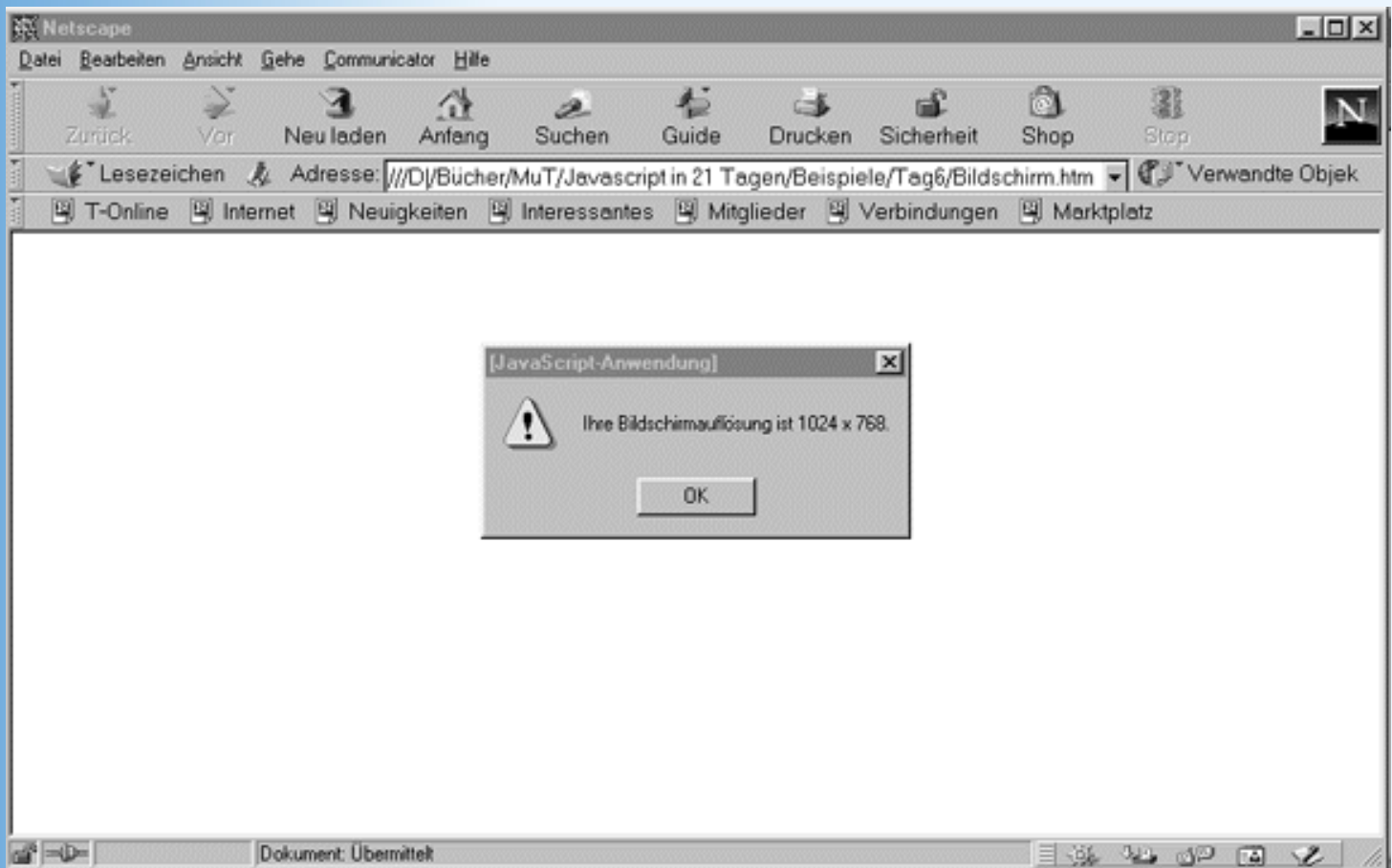


Abbildung 6.15: Auflösung 1

Speichern Sie den Text bitte unter dem Dateinamen Bildschirm.htm ab und schauen Sie sich die Datei im Browser an. Verändern Sie die Bildschirmauflösung und laden Sie die Datei neu. Das Beispiel verwendet ein

Bildschirm-Objekt (screen), in welchem u.a. die Bildschirmauflösung gespeichert ist.

Zusammenfassung

Der heutige Tagesabschnitt sollte hauptsächlich ein Gefühl für JavaScript vermitteln. Die Beispiele gaben Ihnen die Möglichkeit, an Hand praktischer Situationen die Fähigkeiten von JavaScript auszuloten und vor allem Praxis zu erwerben.

Fragen und Antworten

Frage:

Warum ist in den Beispielen der Body-Bereich leer, wenn eine Seite mit document.write() neu geschrieben wird? Kann da nicht schon etwas drin stehen?

Antwort:

Es können durchaus schon Dinge notiert werden. Aber mit document.write() wird die Webseite vollkommen neu geschrieben. Egal ob da vorher etwas drin steht oder nicht. Am besten stellen Sie sich den Vorgang wie das Schreiben auf einer Tafel vor, die vorher gewischt wird. Sinnvoll ist ein vorheriges Füllen, wenn die Seite nicht beim Laden automatisch geschrieben wird, sondern erst auf Grund einer in der Webseite ausgelösten Aktion.

Frage:

Die Beispiele mit der Fakultät und der Quadratberechnung haben aus Grundrechenarten komplexere Berechnungen nachprogrammiert. Stellt JavaScript für komplexere mathematische Berechnungen keine eigenen Anweisungen bereit?

Antwort:

Doch. JavaScript beinhaltet einige leistungsfähige mathematische Funktionen und vor allem das Objekt Math, das diverse mathematische Methoden bereitstellt.

Frage:

Über prompt() wird beim Laden der Seite ja ein Benutzerdialog angezeigt, in dem ein Anwender eine Eingabe vornehmen kann. Gibt es noch weitere Möglichkeiten, etwa mit mehreren Schaltflächen?

Antwort:

Ja, aber nicht so viele, wie sie beispielsweise unter Windows möglich sind. Das ist ein Preis der Plattformneutralität.

Frage:

Es wurden in den Beispielen zwei Kontrollstrukturen verwendet. Sind das alle unter JavaScript vorhandenen Möglichkeiten?

Antwort:

Nein. Wir werden im Laufe des Buches noch andere kennen lernen.

Frage:

Kann man mit HTML dynamische Informationen in eine Webseite integrieren?

Antwort:

Nein. Dazu braucht man aktive Anweisungen, Kontrollstrukturen und Variablen. Mit all dem kann HTML nicht dienen. Gerade diese fehlenden Elemente waren entscheidende Argumente zur Entwicklung von JavaScript als Ergänzung von HTML.

Workshop

Dieser Tagesabschnitt bestand aus viel Praxis, weshalb wir den Workshop heute im Wesentlichen auf die Modifikation der Beispiele ausrichten wollen. Dafür ist natürlich Voraussetzung, dass Sie alle Beispiele wirklich programmiert haben. Dann sollten Sie noch einmal folgende weitere Dinge ausprobieren:

- Beispiel 3: Ersetzen Sie das +- Zeichen jeweils durch - für die Subtraktion, * für die Multiplikation und / für die Division und laden Sie die Datei jeweils neu.
- Beispiel 4: Bilden Sie die dritte Potenz einer Zahl (d.h. eine weitere Multiplikation mit a).
- Beispiel 5: Verändern Sie das Zeichen < in der Prüfung der if- Struktur durch > und testen Sie das Verhalten.
- Beispiel 6: Ersetzen Sie
`document.write("<br>");`
durch
`document.write("<HR>");`
- Beispiel 8: Verändern Sie die Zeit der Weiterleitung und die Zielfile. Experimentieren Sie auch mit URLs, die auf Adressen direkt im Internet verweisen.
- Beispiel 9: Verändern Sie die Meldungen in der Statuszeile.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was macht ++?
2. Was macht prompt()?
3. Was heißt if-else übersetzt?
4. Warum schreibt man überhaupt über JavaScript Webseiten dynamisch?
5. Was macht alert()?
6. Kann man über JavaScript Dinge wie die Bildschirmauflösung auf der Plattform des Anwenders abfragen?
7. Kann man über JavaScript Dinge wie die Farbauflösung auf der Plattform des Anwenders verändern?

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 1

Tag 7

Fehlerbeseitigung und Optimierung in JavaScript

Am Ende der ersten Woche sollten Sie schon einiges über Programmierung im Allgemeinen und JavaScript im Speziellen sowie die allgemeine Webseitenerstellung mit HTML wissen. Insbesondere können Sie mittlerweile so viel Stoff, dass es leicht möglich ist, eigenständig zu experimentieren. Dabei werden Sie aber unweigerlich auf einige Probleme stoßen. Deshalb wollen wir uns mit Fragestellungen befassen, die solche Probleme umschreiben und Ihnen Lösungsansätze anbieten. Wir befassen uns mit den folgenden Themen:

- Der Fehlervermeidung.
- Der Fehlerbehandlung und Debugging und Techniken zum Lokalisieren von Fehlern.
- Den Hilfsmitteln in den Browsern zur Fehlerlokalisierung.
- HTML-Editoren zur Fehlervermeidung und -lokalisierung einsetzen.
- Sicherungsmassnahmen.
- Der Optimierung von Webseiten, Scripten und Programmen im Allgemeinen.

Fehler

Dieser Abschnitt soll mit einer harten Behauptung begonnen werden:

Es gibt kein etwas komplexeres Programm/Script ohne Fehler!

Zumindest nicht, bevor es nicht gründlichst durchgetestet und bereinigt wurde (und auch dann sind absolut fehlerfreie - etwas aufwendigere - Programme die absolute Ausnahmen, wie die unseligen Windows- Anwendungen mit ihren Schreibschutzverletzungen beweisen). Wenn Sie also Ihre Scripte oder Webseiten erstellen, werden diese während der Erstellungsphase immer wieder fehlerhaft sein. Programmieren bedeutet, Fehler zu machen. Diese müssen dann aber gefunden und beseitigt werden, bis das Script oder die Webseite fehlerfrei ist (soweit man es überblicken kann). Dann erst ist ein Script oder eine HTML-Seite fertig und kann veröffentlicht werden.

Bei HTML sind Fehler beileibe nicht so schlimm wie bei Programmiersprachen. Es gibt ja das Prinzip der Fehlertoleranz in HTML, das besagt, dass falsche Anweisungen einfach ignoriert werden. Wenn Sie unter HTML einen Fehler machen, kann nicht viel passieren. Meist schreiben Sie eine falsche Steueranweisung, und der Browser ignoriert diese oder zumindest die Attribute, welche er nicht versteht. Im schlimmsten Fall werden Informationen unterdrückt oder Teile einer Steueranweisung als Klartext im Browser ausgegeben. Nicht schön, aber nicht schlimm. Eine fehlerhafte HTML-Struktur kann durchaus noch sinnvoll genutzt werden.

JavaScript ist da schon gefährlicher. JavaScript ist nicht so fehlertolerant wie HTML und kann es auch nicht sein. Falls Ihr Script also einen Fehler hat, müssen Sie ihn finden, bevor die Seite ausgeliefert werden kann. Grundsätzlich muss man aber beachten, dass jeder Korrekturversuch eines Fehlers in einem Programm/Script die Gefahr neuer - meist noch schlimmerer - Fehler nach sich zieht. Die gleiche Gefahr droht, wenn man ein funktionierendes Programm oder Script optimieren will. Es ist gängige Erfahrung, dass jedes Programm/Script soweit optimiert wird, bis es unbrauchbar ist! Dabei folgt in der Regel als Krönung, dass jedes zur Unbrauchbarkeit optimierte oder bezüglich Fehler »verschlimmbesserte« Programm/Script sich unmöglich in den Zustand zurückversetzen lässt, wo es noch lauffähig war

oder zumindest nicht so extrem verhunzt!

Bei solchen Fehlern muss man verschiedene Situationen unterscheiden. Es gibt einmal typographische Fehler beim Schreiben des Scripts. Wenn Sie sich beim Schreiben des Quelltexts einfach vertippt und ein Schlüsselwort, eine Variable oder sonst etwas Entscheidendes falsch geschrieben haben, können zwei Fälle auftreten.

Sofern Sie ein Wort absolut falsch geschrieben haben, ist die Situation oft leicht in den Griff zu bekommen. In dem Fall haben Sie nicht zufällig einen solchen Schreibfehler gemacht, sodass der falsche Begriff wieder einen sinnvollen Begriff und keinen syntaktischen Widerspruch ergibt (was der zweite und viel schlimmere Fall ist). Nehmen wir ein Beispiel aus der Textverarbeitung. Sie wollten beispielsweise *Kirsche* schreiben und haben *Kirche* getippt - ein ebenfalls sinnvolles Wort. Das Rechtschreibprogramm der Textverarbeitung hätte keine Chance, diesen typographischen Fehler zu finden. Auch Sie haben in JavaScript dann nur die Möglichkeit, den Fehler im Quelltext durch aufmerksames Lesen oder Analyse des Resultates von dem Script zu lokalisieren. Etwa wird die Syntax

```
summe = a - b;
```

durchaus funktionieren. Der Programmcode wird nur nicht das tun, was er tun soll.

Wenn Sie jedoch einen Schreibfehler haben, der keinen vernünftigen Ausdruck ergibt, sind Sie (obwohl es erst einmal schlimmer aussieht) meist in der besseren Situation. Der Ausdruck wird in der Regel einen syntaktischen Widerspruch innerhalb der Programmiersprache erzeugen, und der Interpreter wird bei seiner Arbeit den Fehler entdecken. Danach reagiert der Interpreter mit Abbruch der Verarbeitung und gibt eine Information zurück, wie Sie den Fehler finden können (gleich mehr dazu).



Einer der heimtückischsten typographischen Fehler unter JavaScript ist die Nichtbeachtung von Groß- und Kleinschreibung.

Ein weiterer eng verwandter Fehlertyp ist der syntaktische Fehler. Dies sind Fehler bei der Verwendung der Programmiersprachensyntax. Viele syntaktische Fehler beruhen auf typographischen Fehlern, aber beileibe nicht alle. Syntaktische Fehler haben den Vorteil, dass der Interpreter (oder der Compiler im Compilerfall) diese Regelverletzung recht gut erkennen kann und meist gute Hilfestellung zum Lokalisieren anbietet. Viele syntaktische Fehler beruhen darauf, dass

- ein Semikolon oder ein Komma vergessen wird oder
- Klammern vergessen oder nicht richtig geschlossen werden.

An dieser Stelle hilft es auch gewaltig, wenn ein Editor Syntaxhighlighting beherrscht (siehe dazu Tag 1).

Gefährliche Programmfehler sind diese, welche erst zur Laufzeit auftreten und die auf logische Fehler im Programmaufbau zurückzuführen sind. Hierbei handelt es sich meist um ein syntaktisch vollkommen korrektes Script, das aber Fehler zur Laufzeit zulässt. Dies geschieht am häufigsten, wenn eine Variable zur Laufzeit einen falschen Wert zugeordnet bekommt. In eine ähnliche Kategorie fallen Programmfehler zur Laufzeit, welche auf äußeren Umständen (Situationen, die erst zur Laufzeit entstehen, die Umgebung des Programms oder den Anwender) beruhen. Diese Fehlerkonstellation kann durch verschiedene Dinge entstehen. Etwa wird in einer Webseite eine bestimmte Schriftart vorausgesetzt, die auf der Plattform des Anwenders nicht vorhanden ist. Unter HTML nur ärgerlich, in einem Script oder Programm unter Umständen bereits ein ernstes Problem. Oder ein Script lässt Anwendungsfehler durch den User zu. Dies sind also Fehler, die auf konzeptioneller Seite zu suchen sind. Während der Planung eines Programms oder Scripts muss dies einkalkuliert werden¹.



Das Auffinden und die Beseitigung der typographischen und syntaktischen Fehler sowie der logischen Laufzeitfehler ist das, was man unter Debugging versteht. Der Begriff Debugging geht auf das englische Wort Bug zurück, was übersetzt Wanze bzw. Käfer bedeutet. Es gibt einige Anekdoten, woher der Name kommt. Die bekannteste Anekdote hat zum Inhalt, dass in frühen EDV-Zeiten in einem Röhrengroßrechner ein Programmfehler die damaligen Entwickler zur Verzweiflung getrieben haben soll. Der Fehler zur Laufzeit war absolut unlogisch, da im Quelltext kein Fehler zu finden war. Die Lösung fand sich erst, als man den Rechner selbst auseinander schraubte und zwischen den Röhren eine tote Wanze fand, die mit ihrem

Körper Schaltkreise störte. Wenn man heute von einem Hilfsprogramm redet, das zum Auffinden von Fehlern in Programmen genutzt wird, redet man von einem Debugger.

Fehler im Vorfeld vermeiden

Um Fehler bei der Erstellung einer HTML-Seite, eines Scripts oder eines Programms bereits im Vorfeld zu minimieren, hilft eine sorgfältige Vorbereitung. Dies umfasst beispielsweise folgende Punkte:

- Programmplanung und Konzeption.
- Nachschlagewerke und Infomaterial bereithalten.
- Bei Bedarf vernünftige Entwicklungswerkzeuge einsetzen.

Insbesondere machen Anfänger und Hobbyprogrammierer den Fehler, die Programmplanung und Konzeption zu vergessen. Dabei ist gerade dieser Punkt im Zyklus der Programmerstellung der entscheidende (insbesondere, um logische Fehler zu vermeiden und spätere - sehr fehlerträchtige - Umstrukturierungen zu vermeiden). In professionellen Projekten nimmt diese Phase mindestens 30% der geplanten Zeit ein². Gerade Leute, die nicht genügend Disziplin aufbringen, um in der Anfangsphase eines Webauftritts den Computer einen guten Mann sein zu lassen und sich Gedanken um das geplante Konzept zu machen, sollten in diesem Stadium den Computer gar nicht erst anschalten. Ein paar Blatt Papier und Ruhe sind in dieser Phase dann bessere Werkzeuge als der Computer, der zum einfachen Losprogrammieren und Spielen verleitet. In der konkreten Programmierphase hilft dann natürlich vor allem die sorgfältige und konzentrierte Eingabe.

Sicherungsmaßnahmen

Trotz der gerade beschriebenen Maßnahmen lassen sich Fehler nie ganz vermeiden. Dabei muss man sich verdeutlichen, dass man ja immer von einem Stand aus weiterentwickelt, wo ein Script läuft (außer direkt bei Beginn). Oft kommt man auch in die Situation, dass ein fast fertiges und lauffähiges Script noch mal schnell erweitert oder verändert wird. Meist werden dann mehrere Änderungen vor dem nächsten Test durchgeführt, und prompt hat man einen Fehler. Leider ist es so, dass man sich normalerweise nicht jede Änderung merken kann (obwohl dies - oder noch besser eine Dokumentation - sinnvoll wäre). Oft kann diese Veränderung seit dem letzten lauffähigen Stand dann nicht mehr rückgängig gemacht werden. Hat man dann den letzten lauffähigen Stand nicht gesichert, bleibt oft nur ein aufwendiges Try-and-Error-Verfahren, um etwas wiederherzustellen, was man schon einmal hatte. Es gibt deshalb die unbedingte Regel, jede lauffähige und bereits halbwegs richtige Scriptversion zu sichern und die Erweiterung in einem anderen Verzeichnis oder unter einem neuen Namen zu speichern. Im Abschnitt über Optimierung werden wir dies durchspielen. Dazu kommen natürlich die - hoffentlich - selbstverständlichen regelmäßigen Sicherungen aller wichtiger Daten außerhalb des Arbeitsrechners (auf CDs, ZIP-Disketten, Disketten, Bändern, Servern usw.).

Konkrete Fehlerbehandlung

Wenn nun ein Fehler aufgetreten ist und sich der fehlerfreie Zustand nicht reproduzieren lässt oder sonst nicht gewünscht ist, bleibt nur die Fehlerbeseitigung. Dabei stellt sich zuerst das Problem, den Fehler überhaupt zu lokalisieren. Dies kann mit einem Debugger geschehen, aber auch ohne ein solches Tool. Gerade unter JavaScript werden wir ohne Debugger arbeiten. Dazu gibt es gewisse Standardtechniken.

Hilfe durch den JavaScript-Interpreter

Da gibt es einmal die Hilfe durch den JavaScript-Interpreter im Browser, der bei syntaktischen Fehlern in einem Script die Abarbeitung unterbrechen wird. Dabei ist es jedem Browser überlassen, wie er dann weiter verfährt. Wenn ein JavaScript bei der Ausführung einen Fehler aufweist, wird der Internet Explorer bei entsprechender Einstellung automatisch ein Fehlerfenster öffnen und die Zeile mit dem Fehler sowie einen Grund für den Fehler angeben. Wenn das Fehlerfenster nicht automatisch aufgeblendet wird, wird links unten in der Statuszeile ein Symbol erscheinen, auf das Sie doppelklicken können, um das Fehlerfenster zu aktivieren.

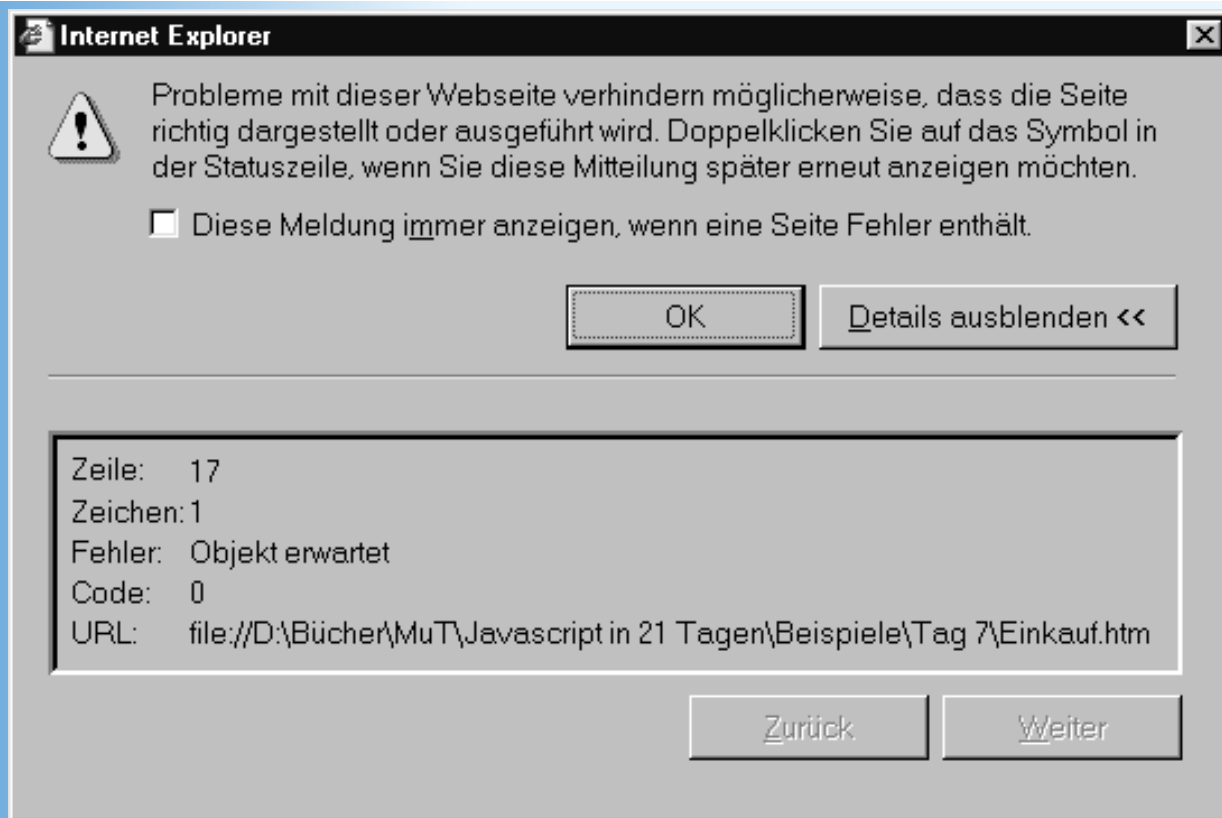


Abbildung 7.1: Das Fehlerfenster des Internet Explorer

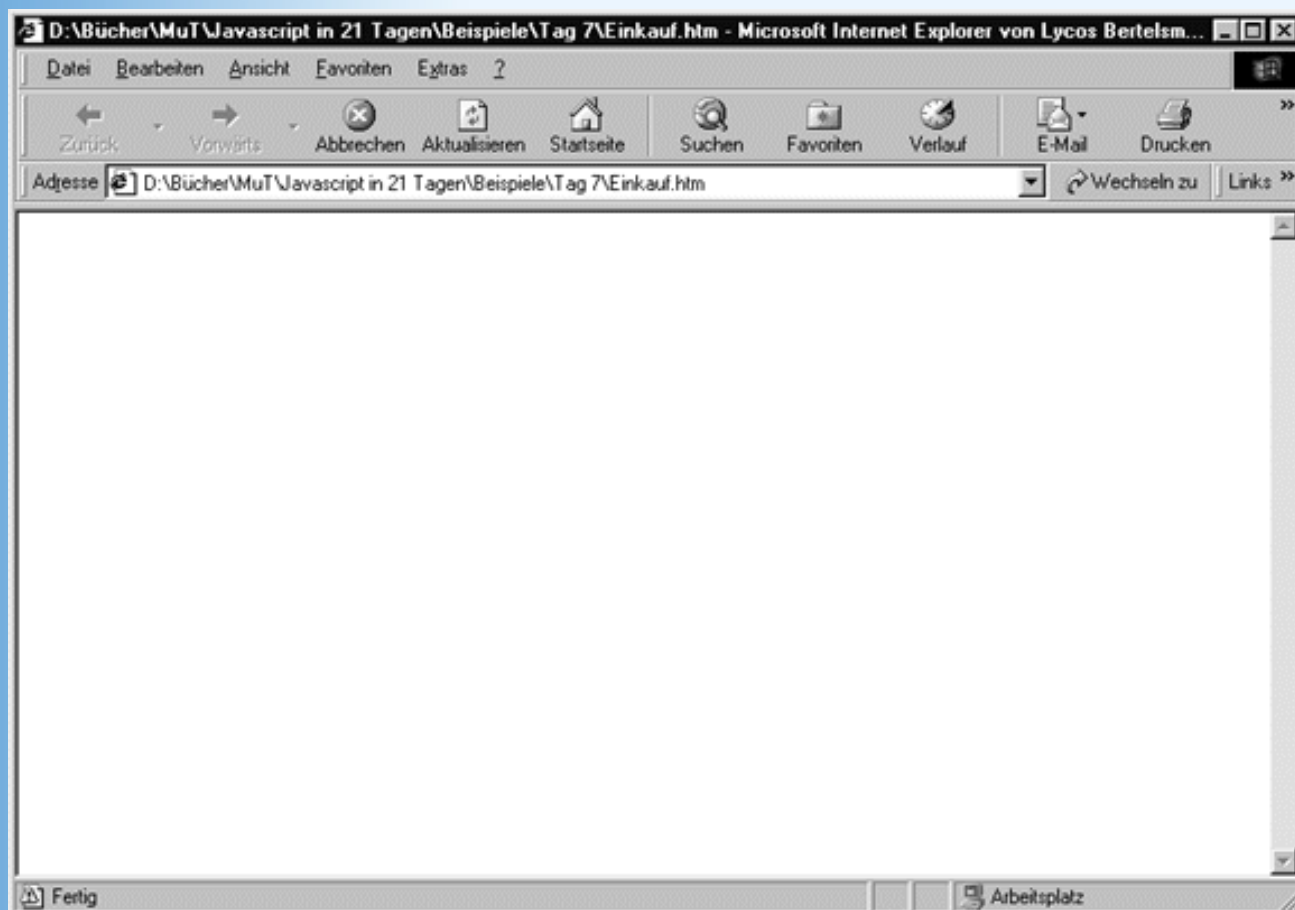


Abbildung 7.2: In der Statuszeile kann über das Symbol links per Doppelklick das Fehlerfenster aktiviert werden

Der Netscape Navigator hält bei einem Fehler in einem Script die Ausführung an und zeigt erst einmal keine weiteren Informationen an.

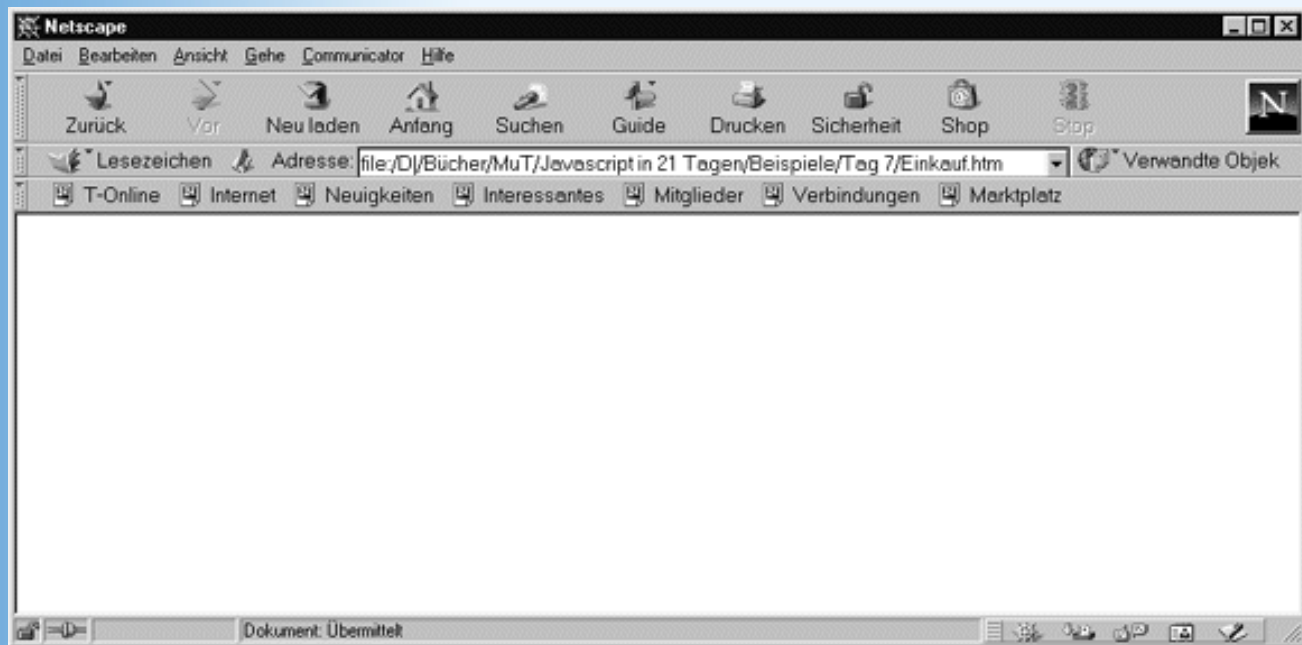


Abbildung 7.3: Keine Fehlermeldung im Navigator. Das Script wird einfach nicht ausgeführt

Um diese zu bekommen, müssen Sie in der Adresszeile des Browsers javascript: eingeben (der Doppelpunkt gehört unbedingt dazu).

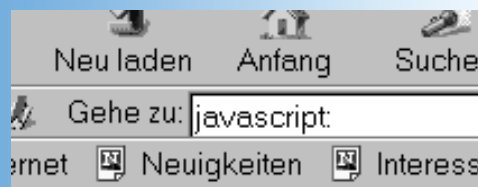


Abbildung 7.4: Fehlerinformationen anzeigen im Navigator

Daraufhin öffnet sich ein zusätzliches Communicator-Fenster, in dem Sie die Zeile mit dem Fehler plus einer Beschreibung für den Fehler finden.

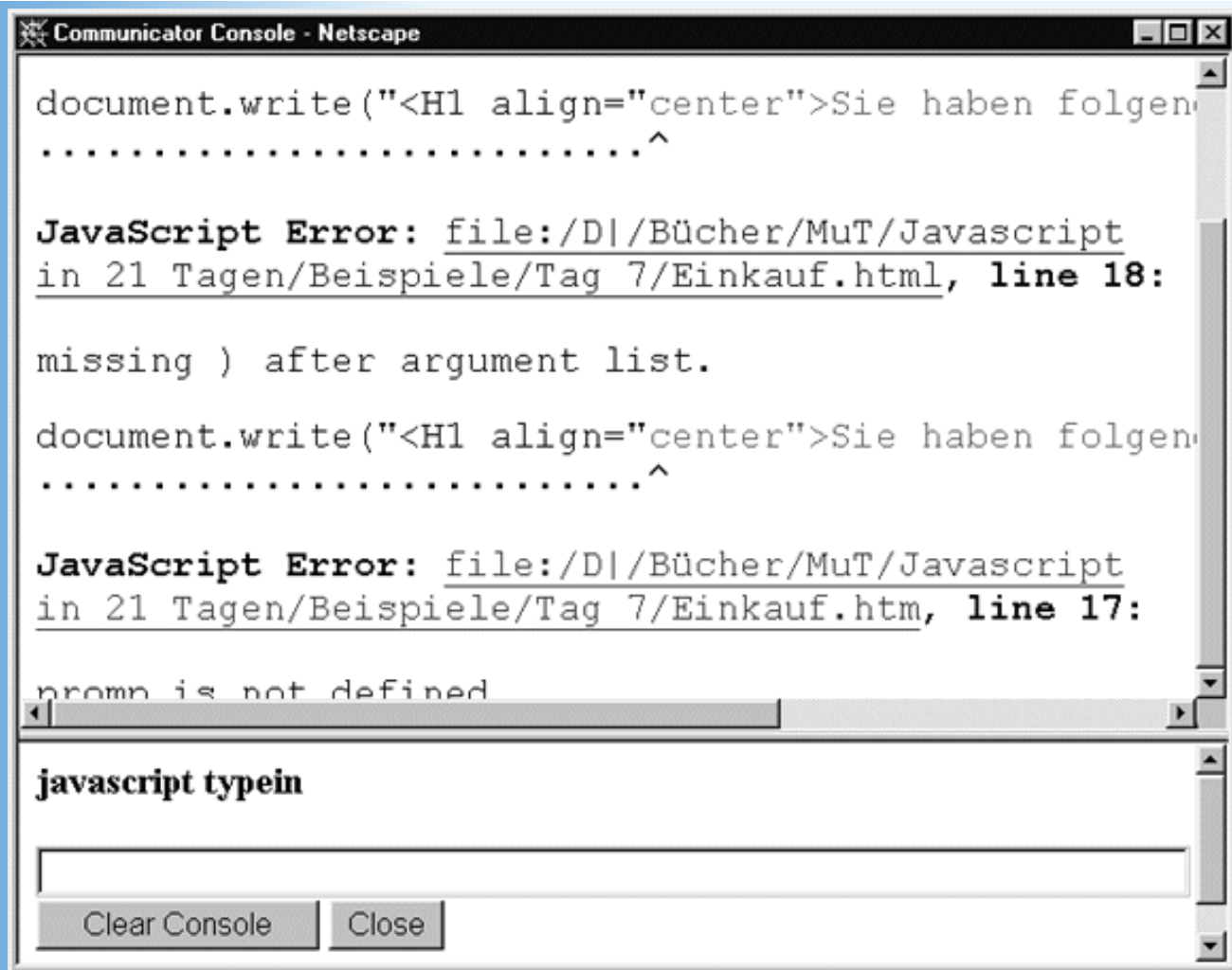


Abbildung 7.5: Das Fehlerfenster des Navigator

Leider gilt bei allen Adress- und Fehlerangaben, dass weder die Adresse stimmen muss noch der angemerkte Fehlertyp wirklich richtig ist. Falls Sie direkt an der angegebenen Adressangabe den Fehler finden, haben Sie einfach Glück, denn diese Adressangabe ist nur die Stelle, wo sich ein Fehler auswirkt und nicht unbedingt der Entstehungsort. Dies muss und wird nicht immer identisch sein, aber zumindest ist der Fehler schon einmal eingegrenzt. Definitiv kann man aber dann sagen, dass der Fehler irgendwo davor im Quelltext zu finden ist. Der Fehlertyp kann deshalb falsch sein, weil ein Fehler sich erst an einer späteren Stelle auswirkt und dann eine andere Situation bewirkt. Sie können dies mit dem nachfolgenden kleinen Beispiel testen, in dem ein winziger, aber extrem tückischer Fehler eingebaut ist.

Starten Sie einen Editor und geben sie dort den nachfolgenden Quelltext ein (beachten Sie bitte unbedingt Groß- und Kleinschreibung):

```
<html>
<body>
<SCRIPT LANGUAGE="JavaScript">
var pw = prompt("Bitte geben Sie Ihre Spende ein","");
If(pw < 5000)
{
document.write("<H1 align=center>Geizhals</H1>");
document.write("<br>");
}
else if(pw < 10000)
{
document.write("<H1 align=center>Vielen Dank</H1>");
document.write("<br>");
}
else
{
document.write(
"<H1 align=center>Vielen, vielen Dank</H1>");
```

```
document.write("<br>");  
}  
</SCRIPT>  
</body>  
</html>
```

Starten Sie die Datei in dem Netscape Navigator. Der Navigator wird nur eine leere Seite anzeigen. Wenn Sie dann mit *JavaScript*: das Fehlerfenster aufrufen, wird er die Zeile 10 und dort das Schlüsselwort `else` »anmosern«.

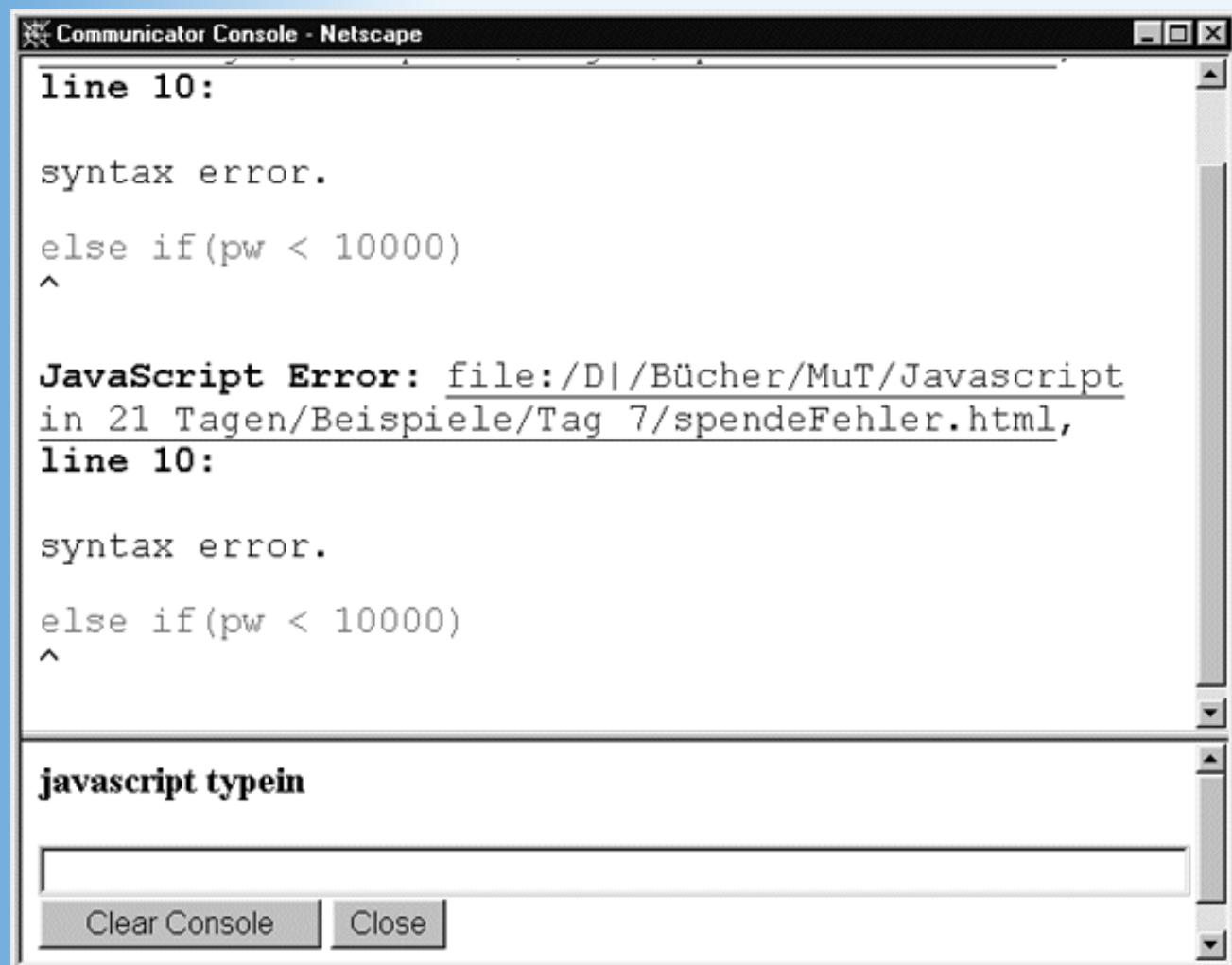


Abbildung 7.6: Das `else` in Zeile 10 wird vom Navigator moniert

Das ist aber nicht der Fehler. Haben Sie ihn entdeckt? Öffnen Sie auf jeden Fall erneut die Datei im Internet Explorer. Auch dort erscheint nur eine leere Seite und das Fehlerfenster enthält analog die Zeile 10 als Fehlerposition.

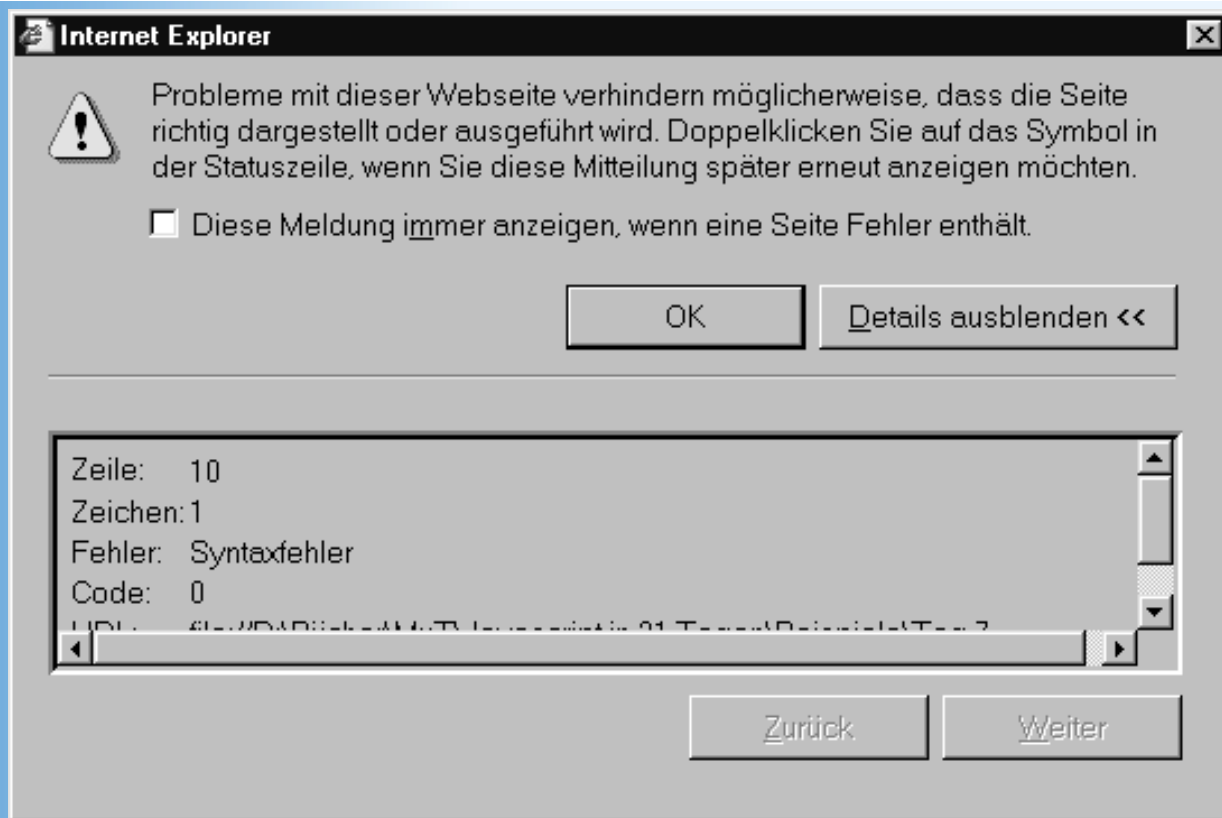


Abbildung 7.7: Auch der Internet Explorer gibt Zeile 10 als Fehlerposition an

Wo ist aber wirklich der Fehler? Die Sache ist wirklich heimtückisch - das if in Zeile 6 ist im Beispiel mit großem I geschrieben. Damit gibt es für den Interpreter ein else ohne vorangestelltes if und das ist eine falsche Syntax. Die ungewöhnliche Mischung von Groß- und Kleinschreibung bei den restlichen Worten im Quelltext sollte nur die Fehlerquelle verschleiern und die Sache für Sie etwas schwieriger machen.

Fehlerlokalisierung mit Kontrollausgaben

Der klassische Weg, um Fehler zu finden, sind Kontrollausgaben, welche im endgültigen Source dann beseitigt werden. Gängige Praxis ist, vor einem vermuteten Fehler eine Bildschirmausgabe der »Wanze« (also der Variablen, in der man die Fehlerursache vermutet) zu erzeugen. Dies könnte so aussehen (die Variable meinWert ist die vermutete Wanze):

```
// Debuggausgabe
document.write("Kontrollpunkt 1: ", meinWert);
// vermutete Fehlerstelle
meinWert=funktionMitRueckgabewert();
```

Vor der Entstehung des Laufzeitfehlers sehen Sie auf dem Bildschirm den Wert der Variablen. Sie können selbstverständlich an den unterschiedlichsten Stellen im Programm solche Debug-Ausgaben einfügen, um eine Variable über mehrere Schritte hinweg zu verfolgen.

Einsatz von Editoren mit farblicher Kennung von Syntaxelementen

Insbesondere im Fehlerfall macht der Einsatz von Editoren Sinn, die verschiedenartige Syntaxelemente farblich unterschiedlich kennzeichnen. Insbesondere kann eine solche farbliche Darstellung von Bereichen den Anfang des kritischen Bereichs deutlich machen. Dazu muss aber selbstverständlich der Editor die Schlüsselworte der verwendeten Sprache kennen. Wir haben dazu ja am ersten Tag zwei schöne Programme kurz vorgestellt. Phase 5 kann beispielsweise auf Syntaxhighlighting für JavaScript eingestellt werden.

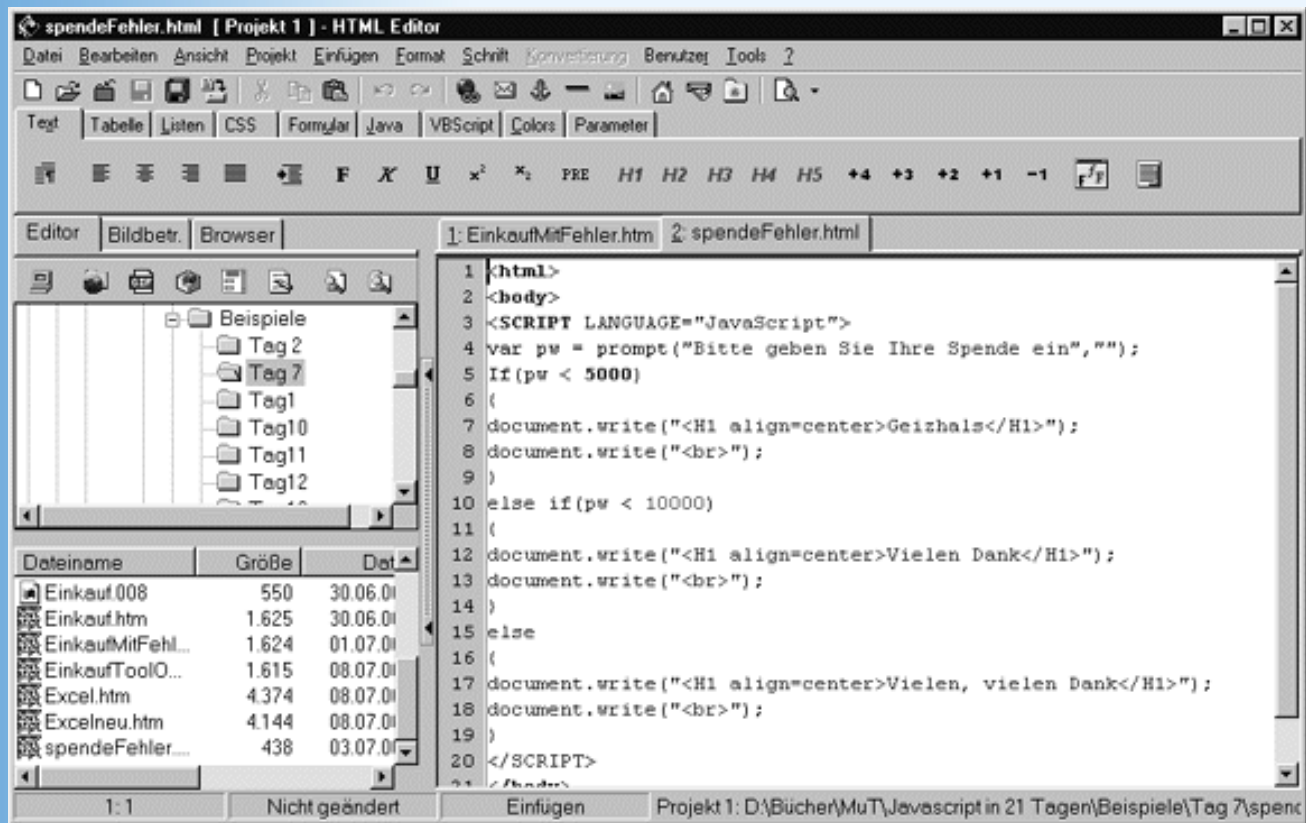


Abbildung 7.8: Deutlich wird If in der falschen Schreibweise nicht als Schlüsselwort dargestellt.

Sinnvolle Schritte zu Lokalisierung von Fehlern

Grundsätzlich sollten Sie im Fehlerfall die Informationen nutzen, die Ihnen der Interpreter liefert. Falls dies aber nicht ausreicht, die Macken zu lokalisieren, gibt es einige Tricks, Fehler dennoch zu finden. Die Erfahrung zeigt, dass es einige genau zu benennende Situationen gibt, auf die besonders geachtet werden sollte:

- Verwenden Sie bei einem Verdacht auf jeden Fall wie oben beschrieben Kontrollausgaben.
- Überprüfen Sie die Bezeichner von Variablen und Funktionen überall wo sie im Script vorkommen. Besonders bei Variablen entsteht ein Problem durch das lose Typisieren von diesen in JavaScript (wobei das Problem beispielsweise bei VBScript noch extremer ist). Sie werden von JavaScript nicht gezwungen, eine neue Variable explizit zu deklarieren. Alleine die Wertzuweisung zu einem neuen Bezeichner lässt eine Variable existent werden. Etwa wie in dem nachfolgenden skizzierten Beispiel:

```

var meinText;
...
meinText = "Das soll erstmal drin stehen";
...
meinTest = "Das soll nun zugewiesen werden";
...
alert(meinText);

```

Die deklarierte Variable `meinText` wird bei der Ausgabe noch den zuerst zugewiesenen Wert haben und zusätzlich hat die Zeile

`meinTest = "Das soll nun zugewiesen werden";`

eine neue Variable `meinTest` erzeugt. In einem komplexeren Script kann so was zu finden recht diffizil werden.

- Überprüfen Sie alle notwendigen Trennzeichen wie Kommata, Semikola, Klammern, Punkte (gerade beim Zugriff auf Objekte) usw. Gerne genommen ;-)) wird bei Funktionen und Methoden der Fehler, dass die Klammern nach dem Bezeichner vergessen werden.
- Versuchen Sie von der Stelle, wo sich der Fehler auswirkt, den Quelltext nach oben durchzuarbeiten. Kontrollieren Sie insbesondere bei Funktionsaufrufen die jeweilig aufgerufene Funktion.
- Wenn das Script wie im Navigator überhaupt nicht abgearbeitet wird oder es sonst erst einmal soweit wie möglich laufen soll, kommentieren Sie die verdächtigen Stellen aus. Bei Bedarf kommentieren Sie zuerst sogar große Bereiche aus und sorgen dafür, dass das Script überhaupt läuft. Die Bereiche können dann schrittweise verkleinert

werden. Diejenige Verkleinerung des auskommentierten Bereichs, nach der der Fehler wieder auftritt, gibt recht gute Ansätze zum Auffinden der Fehlerstelle.

Optimierung von Webseiten und Scripten

Wenn man eine HTML-Seite oder ein Script erstellt oder allgemein programmiert, kommt man irgendwann an die Stelle, wo eine Webseite so aussieht wie gewünscht bzw. ein Script oder Programm - soweit man es erkennen kann - fehlerfrei läuft, aber noch verschiedene Wünsche offen bleiben. Dann führt man - wenn Zeit bleibt - eine Aktion durch, die Optimierung genannt wird. Dies kann in mehreren Richtungen verstanden werden.

Man kann ein HTML-Dokument, ein Script oder ein Programm bezüglich seiner Funktionalität, seiner optischen Gestaltung oder seiner Bedienerfreundlichkeit optimieren. Dies sind dann Verbesserungen, die offensichtlich sind, wenn man die Vorgängerversion betrachtet oder sie im Vergleich zur verbesserten Variante bedient. Das wollen wir hier nicht (!) intensiv verfolgen. Da gibt es zu viele unterschiedliche Ansätze, als dass dies so allgemein besprochen werden kann. Am besten verfolgt man solche Optimierungen, indem man bisher unbeteiligte Testpersonen (die der potentiellen Zielgruppe zuzuordnen sind) hinzuzieht und deren Eindrücke und Vorschläge als Ansatz nimmt. Aber auch der eigenen Phantasie sind da keine Grenzen gesetzt.

Wir wollen hier eine Optimierung ansprechen, welche eine Webseite effektiver macht bezüglich

- der Übertragungszeit und Größe,
- der Wartbarkeit und
- der Verschleierung von sensiblen Informationen.

Letzteres bedeutet, dass es nicht jedem so einfach gemacht werden soll, Ihren ausgefeilten Quelltext zu analysieren und dann für sich weiter zu verwenden.

Diese mehr im Hintergrund laufenden Optimierungen werden vom Anwender normalerweise eher unterbewusst wahrgenommen³, sind aber dennoch sehr wichtig. Dabei muss beachtet werden, dass die angesprochenen Punkte sich teilweise ausschließen und man meist einen Kompromiss finden muss. Insbesondere schließen sich Wartbarkeit und Reduzierung der Größe ab einer gewissen Optimierung in Hinsicht der Dateigröße aus.



Erste und effektivste Maßnahme zum Optimieren eines Webprojekts ist natürlich, Grafiken und andere große Dateitypen zu verkleinern. Das beinhaltet, das geeignetste Dateiformat zu testen, die Farbtiefe und Bildgröße zu perfektionieren, bei Animationen die Framerate so niedrig wie möglich einzustellen, die Tonqualität bei Sounds soweit wie möglich zu reduzieren etc. Diese Maßnahmen sollten natürlich auf jeden Fall und mit den entsprechenden Programmen erfolgen - wir wollen hier nachfolgend aber die Maßnahmen vorantreiben, die auf Textseite möglich sind⁴.

Um was geht es bei der Optimierung der Übertragungszeit und Größe bzw. der Verbesserung der Wartbarkeit? Nun, wir wollen den gesamten Ablauf mit einem Beispiel verbinden, an dem wir einen typischen Optimierungsvorgang sehen werden. Erstellen Sie als Erstes die nachfolgende einfache Webseite mit einem kleinen JavaScript (Dateiname Einkauf.html):

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
/* Variablendeklaration */
// Die Preise der Produkte - fest vorbelegte DM-Beträge
var preisProduktA = 31.82;
var preisProduktB = 2.94;
var preisProduktC = 4.22;
// Deklaration der Variablen für die Menge des
// jeweiligen Produktes
var anzahlProduktA;
var anzahlProduktB;
var anzahlProduktC;
```

```
// Summe der Gesamtbestellung
var gesamtSumme;
/* Eingabe der Anzahl für jedes Produkt */
anzahlProduktA = prompt("Anzahl Hosen","");
anzahlProduktB = prompt("Anzahl Socken","");
anzahlProduktC = prompt("Anzahl T-Shirts","");
/* Berechnung der Gesamtsumme */
gesamtSumme =
    anzahlProduktA * preisProduktA +
    anzahlProduktB * preisProduktB +
    anzahlProduktC * preisProduktC;
/* Ausgabe */
document.write("<H1 align=center>Sie haben folgende Produkte bestellt:</H1>");
// Hosen
document.write(
    "Hosen: ",
    anzahlProduktA ,
    " Stücke a einem Einzelpreis von DM ",
    preisProduktA ,
    " = DM ",
    anzahlProduktA * preisProduktA,
    " gesamt.");
document.write("<br>");
// Socken
document.write(
    "Socken: ",
    anzahlProduktB ,
    " Stücke a einem Einzelpreis von DM ",
    preisProduktB ,
    " = DM ",
    anzahlProduktB * preisProduktB,
    " gesamt.");
document.write("<br>");
// T-Shirts
document.write(
    "T-Shirts: ",
    anzahlProduktC ,
    " Stücke a einem Einzelpreis von DM ",
    preisProduktC ,
    " = DM ",
    anzahlProduktC * preisProduktC,
    " gesamt.");
document.write("<P>");
// Gesamtsumme
document.write(
    "<H3>Gesamtsumme DM ",
    gesamtSumme ,
    ".</H3>");
document.write("<br>");
</SCRIPT>
</BODY>
</HTML>
```

Wir erstellen hier eine Art Warenkorbsystem (ganz ohne Formulare als Eingabemöglichkeit und einfachster Bauart - es geht nicht um eine Praxisfähigkeit). Das Beispiel wird beim Laden der Webseite ausgeführt und fragt mit drei aufeinander folgenden Eingabemasken die Anzahl der Produkte ab, die ein Anwender von jedem dieser Produkte kaufen möchte.

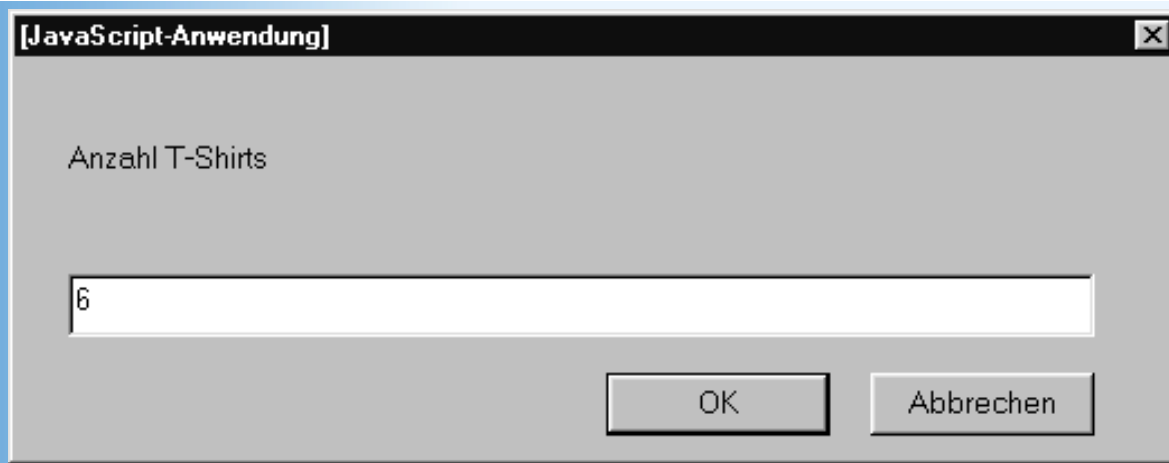


Abbildung 7.9: Letzte Eingabe ist die Anzahl des dritten Produktes

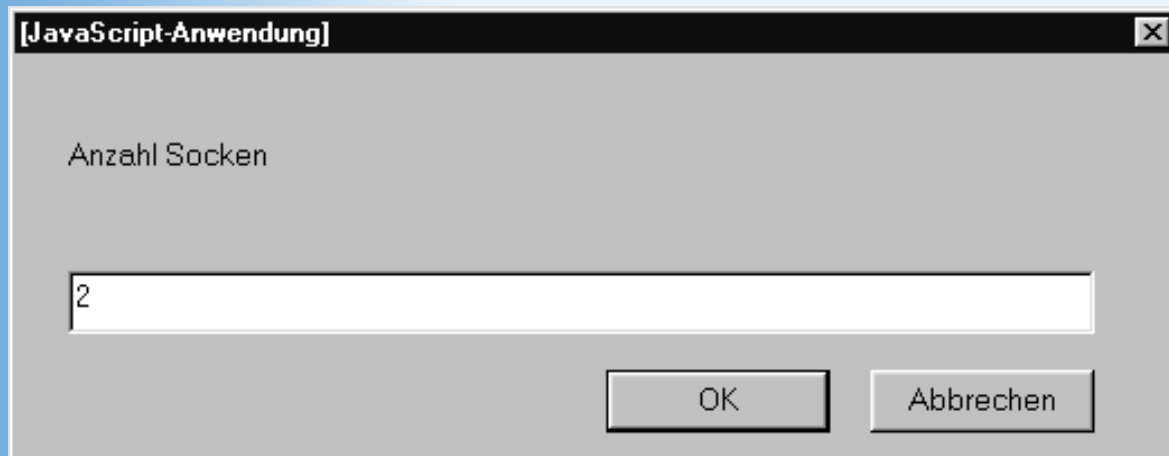


Abbildung 7.10: Die Anzahl des zweiten Produktes

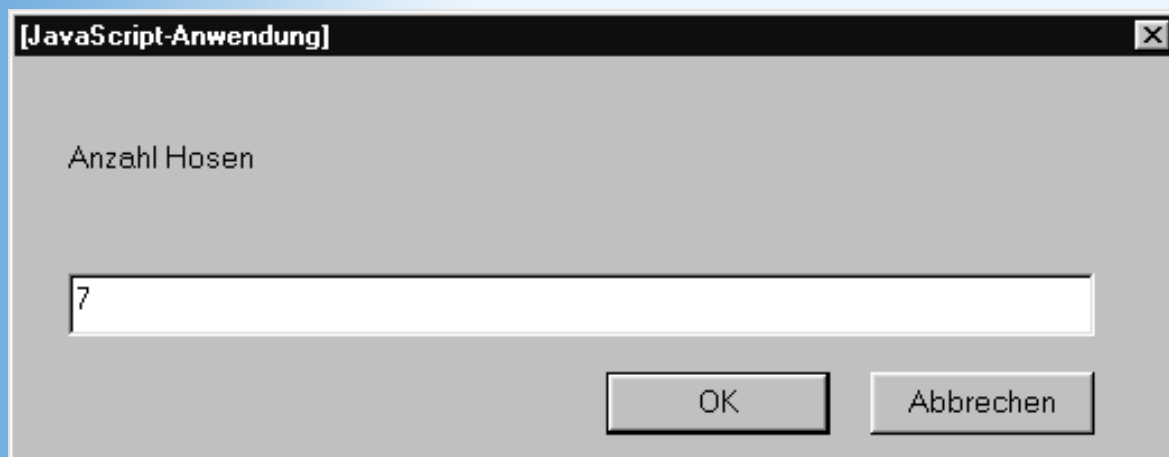


Abbildung 7.11: Die Anzahl des ersten Produktes

Jedes der Produkte hat einen vordefinierten Preis (jeweils in einer Variablen im Quelltext gespeichert - `preisProduktA`, `preisProduktB`, `preisProduktC`), der dann mit der jeweiligen Anzahl (ebenfalls jeweils in einer weiteren Variablen gespeichert - `anzahlProduktA`, `anzahlProduktB`, `anzahlProduktC`) zu multiplizieren ist.

Daraus wird dann eine Gesamtsumme errechnet, die ebenso in einer Variablen gespeichert wird (`gesamtSumme`). Die Webseite selbst wird mit diesen Informationen dynamisch geschrieben (Ausgabe des jeweiligen Artikels und der Einzelpreise, der jeweiligen Einzelpreise mal der individuellen Anzahl und die Gesamtsumme der Bestellung). Soweit funktioniert das Beispiel in der notierten ersten Version. Das werden Sie sehen, wenn Sie es in einen Browser laden und vernünftige Zahlenwerte eingeben (beachten Sie bitte, dass Rundungsungenauigkeiten und Falscheingaben nicht abgefangen werden - das soll hier keine Rolle spielen). Beachten Sie, dass das Script im Quelltext ausführlich dokumentiert ist.

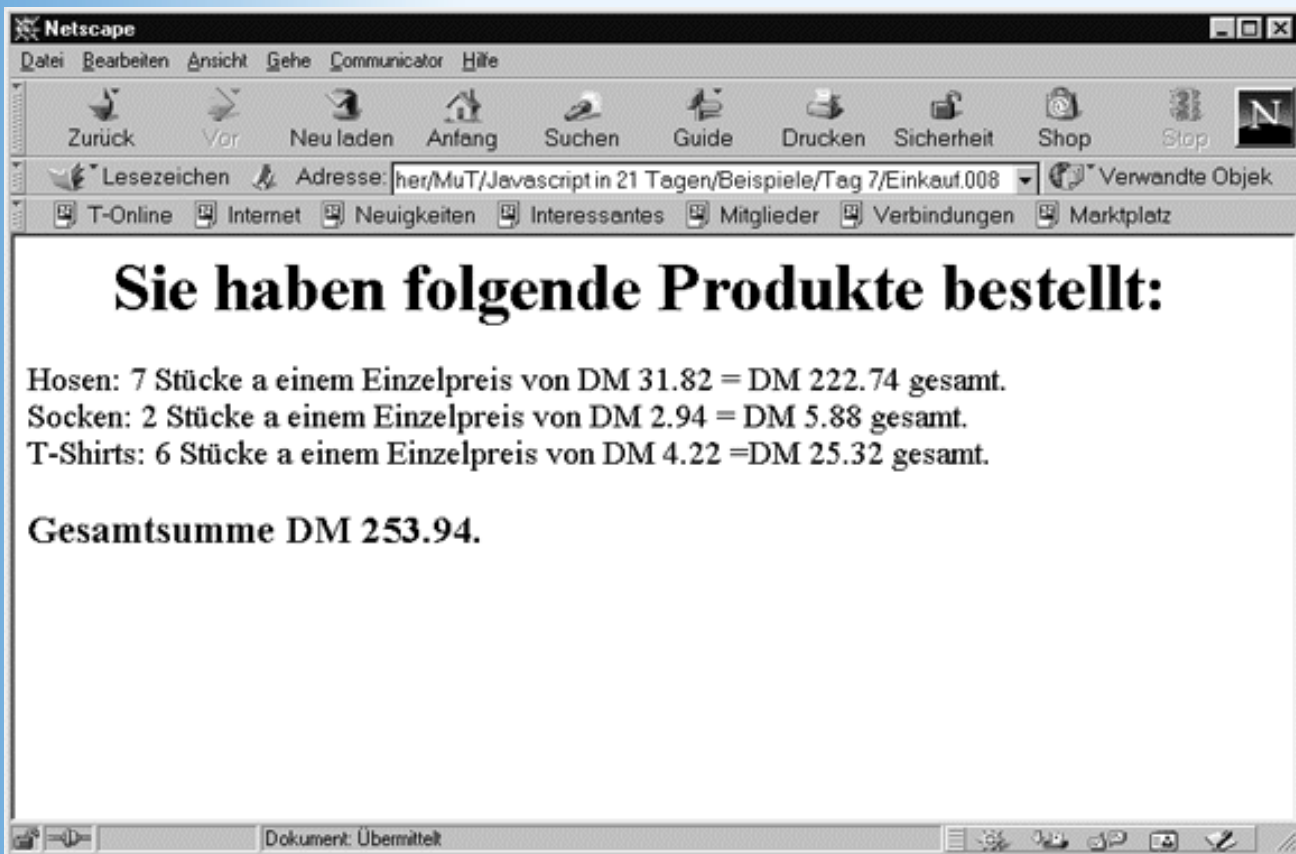


Abbildung 7.12: Die Ausgabe mit den berechneten Werten

Der Quelltext ist jedoch wirklich nicht optimal. Es bieten sich zahlreiche Stellen für eine Verbesserung an. Zur Optimierung kommen beispielsweise folgende Details in Betracht:

1. Die Variablenbezeichner. Diese können entweder mit sprechenderen Bezeichnern gewählt werden oder aber verkürzt, um die Webseite zu verkleinern.
2. Sich wiederholende Passagen im Script könnten in Variablen gespeichert werden, um damit das Script kleiner und besser wartbar zu machen. Insbesondere lassen sich dann Änderungen und Wartungen leichter durchführen, weil nur an einer zentralen Stelle geändert werden muss. Die Verwendung von Textbausteinen für sich wiederholende Textpassagen, die Variablen zugeordnet werden, verkleinert das Script merklich.
3. Zusammenfassen von Codestrukturen, welche dadurch kürzer werden. Dies kann sowohl Deklarationen betreffen, aber auch Aufrufe von Methoden und ähnliche Situationen.
4. Entfernen sämtlicher Kommentare.



Warum macht man sich eigentlich diese Mühe mit der Optimierung des Quelltextes? Nun, jedes Zeichen, das im Quelltext vorkommt, muss auch in den Speicher des Anwenderrechners geladen werden. Bei Netzwerken und insbesondere dem Spezialfall des Internets muss jedes Zeichen zudem noch über die (langsamen Internet-) Leitungen übertragen werden. Wenn sich mit den genannten Optimierungen das Script erheblich verkleinern lässt, hat dies die Folge, dass eine Webseite auch schneller beim Anwender in den Browser geladen wird. Zwar werden Buchstaben im Verhältnis zu Grafiken oder anderen Multimediaformaten nicht so stark ins Gewicht fallen, aber eine Rolle kann dies - gerade bei großen Seiten - schon spielen.

Die nachfolgenden Schritte zur Optimierung des Quelltexts haben ab einer gewissen Grenze zugleich den oft nicht unerwünschten Nebeneffekt, dass ihr Script immer schlechter zu lesen ist (das werden Sie gleich sehen). Dabei ist (was das »nicht unerwünscht« angeht) an die Leute gedacht, die eine interessante Seite im Web sehen und einfach kopieren oder geringfügig an ihre Bedürfnisse anpassen. Man sollte sich nichts vormachen - die Beachtung des geistigen Eigentums und des Urheberrechts ist im Internet nicht sonderlich ausgeprägt - gelinde ausgedrückt⁵. Die Optimierung verändert dabei in keinsten Weise die Funktionalität des Scripts - wenn Sie keine Fehler einbauen. Sie macht nur den Raben das Stibitzen etwas unbequemer.

Für Sie selbst ist es jedoch unbedingt notwendig, dass Sie irgendwie dokumentieren, was Sie beim

Optimieren alles machen und wie Sie gegebenenfalls die Datei warten oder verändern können. Das sollte am besten in einer zusätzlichen Dokumentation geschehen.

Wie gehen Sie nun am besten vor? Das Beispiel ist bisher so aufgebaut, dass es fehlerfrei läuft und all das tut, was das Script auch endgültig tun soll. Auch ist es gut lesbar und ausreichend im Quelltext dokumentiert. Auf diese Weise ein Script erst einmal zu erstellen und lauffähig zu machen, ist sehr sinnvoll. Als erstes sollte dieser Stand eingefroren werden. Das bedeutet, diese Version ihres Scripts wird nicht mehr verändert, bevor nicht irgendwo eine Kopie gespeichert ist. Dann erst wird mit der Datei weitergearbeitet und perfektioniert. So eine permanente Sicherung eines bis dahin fehlerfreien Standes sollten Sie auf jeden Fall anlegen (siehe die Bemerkungen zu Fehler und Debuggen am Anfang des Kapitels). Wir erstellen nun eine Kopie und wollen die Datei einfach umbenennen in `Einkauf.001`. Danach vervollkommen wir die Variablenbezeichner - sowohl in Bezug auf die Größe als auch die Lesbarkeit.



Die Webseite ist in ihrer nicht optimierten Form genau 1625 Byte groß. Dies sollte als Startwert festgehalten werden. Sie können die genaue Größe einer Datei überprüfen, indem Sie diese mit der rechten Maustaste anklicken und Eigenschaften aus dem Kontextmenü auswählen.

Bei den Variablenbezeichnern muss man einen Kompromiss zwischen Eindeutigkeit, möglichst wenig Zeichen und halbwegs vernünftiger Lesbarkeit eingehen. Ein einzelner Buchstabe ist kaum sinnvoll, denn das ist dann auch für Sie kaum noch wartbar (selbst wenn man im Web den Kiebitzen den Spass verderben möchte) - außerdem kann es vorkommen, dass die Buchstaben des Alphabets nicht reichen. Variablen mit einer Länge von einem Buchstaben werden meist nur für Zählvariablen in Schleifen verwendet.

Mit einer Verkürzung der Variablenbezeichner auf zwei bis drei Buchstaben für die in dem Beispiel gewählten Variablen hat man wahrscheinlich den besten Kompromiss gefunden. Wir wollen folgende Ersetzungen vornehmen (so oder so ähnlich könnten Sie das dann auch in der Praxis dokumentieren):

preisProduktA - pHo (steht für preisHosen)
preisProduktB - pSo (steht für preisSocken)
preisProduktC - pTs (steht für preisT-Shirts)
anzahlProduktA - aHo (steht für anzahlHosen)
anzahlProduktB - aSo (steht für anzahlSocken)
anzahlProduktC - aTs (steht für anzahlT-Shirts)
gesamtSumme - gS (steht für gesamtSumme)

Wenn dies so in einer separaten Dokumentation festgehalten wird, sollte die Aktion sowohl die Dateigröße reduzieren als auch die Wartbarkeit und sogar die Lesbarkeit verbessern.



Ersetzen Sie Bezeichner - wenn es geht - nicht von Hand. Nutzen Sie die »Replace«-Funktionalität Ihres Editor, sofern Sie eine entsprechende Funktion dort haben. Ansonsten gehen Ihnen unter Umständen Bezeichner durch die Lappen und dann entstehen Fehler, die vollkommen überflüssig sind. Beachten Sie aber, dass Sie bei »Replace-all« unter Umständen auch Gefahren ausgesetzt sind. Am besten aktivieren Sie, dass nur ganze Wörter ausgetauscht werden und dabei Groß- und Kleinschreibung beachtet wird.

Nach dieser Optimierung sieht das Script so aus:

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
/* Variablendeklaration */
// Die Preise der Produkte - fest vorgelegte DM-Beträge
var pHo = 31.82;
var pSo = 2.94;
```

```
var pTs = 4.22;
// Deklaration der Variablen für die Menge des
// jeweiligen Produktes
var aHo;
var aSo;
var aTs;
// Summe der Gesamtbestellung
var gS;
/* Eingabe der Anzahl für jedes Produkt */
aHo = prompt("Anzahl Hosen","");
aSo = prompt("Anzahl Socken","");
aTs = prompt("Anzahl T-Shirts","");
/* Berechnung der Gesamtsumme */
gS =
    aHo *   pHo +
    aSo *   pSo +
    aTs *   pTs;
/* Ausgabe */
document.write("<H1 align=center>Sie haben folgende Produkte bestellt:</H1>");
// Hosen
document.write(
    "Hosen: ",
    aHo ,
    " Stücke a einem Einzelpreis von DM ",
    pHo ,
    " = DM ",
    aHo *   pHo,
    " gesamt.");
document.write("<br>");
// Socken
document.write(
    "Socken: ",
    aSo ,
    " Stücke a einem Einzelpreis von DM ",
    pSo ,
    " = DM ",
    aSo *   pSo,
    " gesamt.");
document.write("<br>");
// T-Shirts
document.write(
    "T-Shirts: ",
    aTs ,
    " Stücke a einem Einzelpreis von DM ",
    pTs ,
    " = DM ",
    aTs *   pTs,
    " gesamt.");
document.write("<P>");
// Gesamtsumme
document.write(
    "<H3>Gesamtsumme DM ",
    gS ,
    ".</H3>");
document.write("<br>");
</SCRIPT>
</BODY>
</HTML>
```

Die Datei ist jetzt nur noch 1313 Byte groß - wir haben bereits ein Viertel der Zeichen »wegoptimiert«. Speichern Sie diesen Stand unter `Einkauf.002`, nachdem Sie die Funktionalität der Webseite überprüft haben. Die Vorgängerdatei sollte dennoch beibehalten werden - vielleicht muss man nochmals darauf zurückgreifen. Sie ist auf jeden Fall eine gute Dokumentation. Wir entfernen nur die Kommentare - das reduziert die Geschichte bereits auf 980 Byte (Stand anschließend einfrieren unter `Einkauf.003`). Allerdings geht dieser Schritt bereits zu Lasten einer späteren Wartung.

Der nächste Schritt fasst Code zusammen, der ohne Funktionalitätsverlust verkürzt werden kann. Die Zeilen

```
var aHo;  
var aSo;  
var aTs;  
var gS;
```

können zu

```
var aHo,aSo,aTs,gS;
```

zusammengefasst werden. Das bringt 18 Byte. Wir wollen aber in diesem Schritt noch weiter gehen. Beachten Sie, dass auch Leerzeichen, die nur zur Übersichtlichkeit eingefügt werden, in Klartextdateien jeweils ein Byte benötigen. Aber auch Zeilenvorschübe und Tabulatoren, die Quelltext nur übersichtlicher machen, sind effektive Bytes in der Webseite, die mit übertragen werden müssen (das gilt auch für reine HTML-Dateien).

An der Stelle müssen wir uns entscheiden, ob wir die Wartbarkeit oder die Optimierung in Hinsicht der Größe priorisieren wollen. Im Web ist Dateigröße sehr wichtig und wir wollen - auch auf Kosten der Wartbarkeit - diese perfektionieren. Wenn der Code sowieso nicht gut lesbar sein muss, kann auch auf diese Trennzeichen verzichtet werden. Die Eliminierung von überflüssigen Leerzeichen, Tabulatoren und Zeilenumbrüchen bringt (wenn einige wenige Zeilenumbrüche vorerst erhalten bleiben) eine Reduktion auf 799 Byte. Abermals zur Betonung - ohne die geringste Beeinflussung der Funktionalität. Das Script (Dateiname `Einkauf.005`) sieht jetzt so aus (wobei beim Lesen im Buch satztechnische Zeilenumbrüche und Leerzeichen berücksichtigt werden müssen):

```
<HTML><BODY><SCRIPT LANGUAGE="JavaScript">  
var pHo = 31.82;var pSo=2.94;var pTs=4.22;var aHo,aSo,aTs,gS;aHo = prompt("Anzahl  
Hosen","");aSo = prompt("Anzahl Socken","");aTs = prompt("Anzahl T-Shirts","");  
gS=aHo*pHo+aSo*pSo+aTs*pTs;  
document.write("<H1 align=center>Sie haben folgende Produkte bestellt:</H1>");  
document.write("Hosen: ",aHo," Stücke a einem Einzelpreis von DM ",pHo," = DM  
",aHo*pHo," gesamt.");  
document.write("<br>");  
document.write("Socken: ",aSo," Stücke a einem Einzelpreis von DM ",pSo," = DM  
",aSo*pSo," gesamt.");  
document.write("<br>");  
document.write("T-Shirts: ",aTs," Stücke a einem Einzelpreis von DM ",pTs," =DM  
",aTs*pTs," gesamt.");  
document.write("<P>");  
document.write("<H3>Gesamtsumme DM ",gS,".</H3>");  
document.write("<br>");  
</SCRIPT></BODY></HTML>
```

Nun sollen alle mehrfach verwendeten Textpassagen durch Textbausteine in Variablen abgelegt werden. Folgende Ersetzungen sind sinnvoll:

```
" Stücke a einem Einzelpreis von DM "          -   SDM  
" gesamt."          -   gsa
```

Dies sinnvoll - d.h. ohne Zeilenvorschub - in den Quelltext notiert, reduziert die Datei auf 741 Byte (Stand `Einkauf.006`). Wenn wir nun noch JavaScript bezüglich der losen Typisierung ausnutzen und auf das Schlüsselwort `var` bei der Variablendeklaration verzichten und die Variablen `aHo`, `aSo` und `aTs` erst bei der Wertzuweisung der `prompt()`-Funktion erzeugen, bekommen wir die Größe der Datei (Stand `Einkauf.007`) auf 702 Byte.

Unsere letzten - nochmals sehr effektiven - Optimierungsmaßnahmen bestehen darin, auf die mehrfache Verwendung von `document.write()` zu verzichten und per Kommata getrennt nur noch eine solche Anweisung zu verwenden. Dazu eliminieren wir die letzten Zeilenumbrüche. Dies beides bringt erneut eine Menge und reduziert unsere Abschlussdatei auf nunmehr nur 550 Byte. Obwohl immer noch geringfügig mehr Luft drin ist, wollen wir es dabei bewenden lassen und uns die Datei `Einkauf.008` abschließend anschauen (Zeilenumbrüche und scheinbare Leerzeichen außerhalb der Strings

sind nur satztechnisch im Buch notwendig):

```
<HTML><BODY><SCRIPT LANGUAGE="JavaScript">pHo=31.82;pSo=2.94;pTs=4.22;aHo =
prompt("Anzahl Hosen","");aSo = prompt("Anzahl Socken","");aTs = prompt("Anzahl
T-Shirts","");sDM=" Stücke a einem Einzelpreis von DM ";gsa="
gesamt.";gS=aHo*pHo+aSo*pSo+aTs*pTs;document.write("<H1 align=center>Sie haben
folgende Produkte bestellt:</H1>", "Hosen: ",aHo,sDM,pHo," = DM
",aHo*pHo,gsa,"<br>", "Socken: ",aSo,sDM,pSo," = DM ",aSo*pSo,gsa,"<br>", "T-Shirts:
",aTs,sDM,pTs," =DM ",aTs*pTs,gsa,"<P>", "<H3>Gesamtsumme DM
",gS,".</H3>", "<br>");</SCRIPT></BODY></HTML>
```

Die Webseite wurde ohne irgendeine Beeinträchtigung der Funktionalität von 1625 Byte auf ein Drittel der Ursprungsgröße verkleinert. Dies bedeutet auch, dass die Seite bei einer konstanten Übertragungsgeschwindigkeit im Netz in einem Drittel der Zeit da ist. Was dies bei größeren Seiten für Optimierungsmöglichkeiten eröffnet, wird sicher klar sein.

Lesbar ist so ein Quelltext dann nicht mehr sonderlich gut, was aber wie gesagt oft auch gewünscht ist. Ob Sie eine Webseite aber wirklich so stark optimieren müssen, das sei dahin gestellt. Wir wollten hier nur die Machbarkeit zeigen. Sie müssen entscheiden, was für Sie noch sinnvoll ist und welche der Optimierungsmaßnahmen Sie bei Bedarf auswählen.

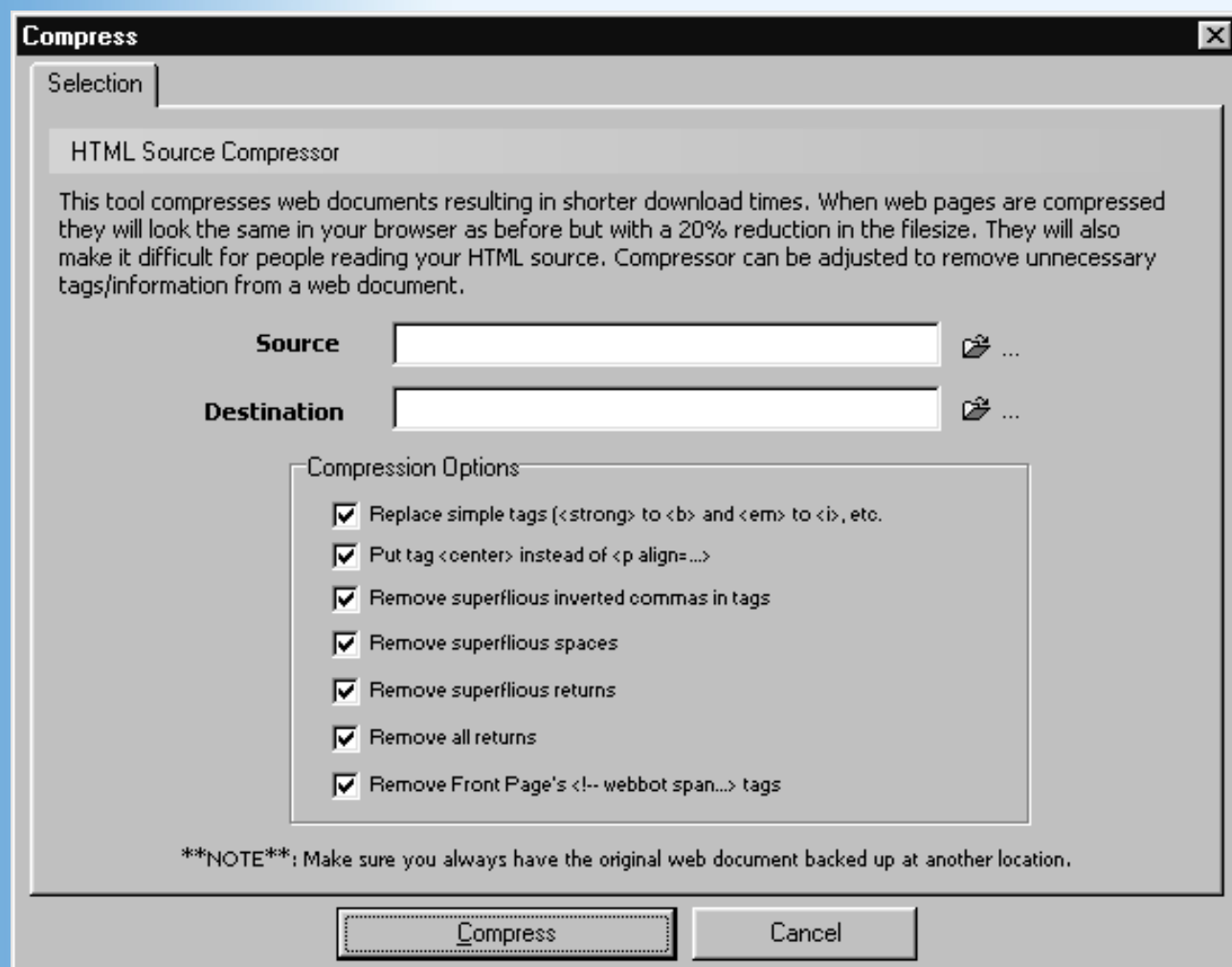


Abbildung 7.13: Das Kompressionstool von 1 Page 2000

Optimierungen müssen auch nicht unbedingt mit der Hand erfolgen. Insbesondere wenn es um die Optimierung von den reinen HTML- Passagen innerhalb der Webseite geht. Einige HTML-Tools beinhalten Kompressions-Features. So auch der Editor 1st Page 2000, den wir am ersten Tag kurz vorgestellt haben. Eine Optimierung von den HTML- Passagen in einer Webseite ist besonders dann angeraten, wenn solche HTML-Tools zum Erstellen der Seite erstellt wurden, wo der Anwender wenig oder keinen direkten Einfluss auf den erzeugten Quelltext hat.

Wenn wir den ursprünglichen Quelltext durch den Komprimierungsalgorithmus von 1st Page 2000 laufen lassen, kommt ein 1615 Byte großer Source heraus. Keine allzu starke Optimierung, aber man muss der Fairness halber zugestehen, dass die Optimierungen von dem Tool schlechte HTML-Anweisungen verbessern, die im Original schon gar nicht verwendet wurden. Das Tool optimiert Seiten, indem statt langer HTML-Anweisungen identisch wirkende, aber kürzer zu

schreibende Anweisungen genommen werden - etwa statt oder der <CENTER>-Tag statt <P align=center>. Dazu wird speziell die Frontpage- Eigenwerbung eliminiert. Die anderen Schritte wie das Löschen von überflüssigen Kommata, Leerzeichen und Returns entspricht unseren manuellen Maßnahmen, nur lässt das Kompressionstool alle Script- Bereiche unberührt, wo wir ja unsere Hauptersparnis erreicht haben.

Wir wollen noch eine weitere Optimierungsmaßnahme durchführen (diesmal ohne Scripte). In diesem Fall lassen wir zwei Programme gegeneinander »kämpfen«. Eine Tabelle, welche mit Excel 2000 erstellt wird, soll als Webseite gespeichert werden. Danach kontrollieren wir die Dateigröße, betrachten den Quelltext und versuchen, mit dem HTML- Editor 1st Page 2000 diese Seite zu optimieren. Das »Kampf« ist deshalb brutal, weil Office-Programme fürchterlichen HTML-Code erzeugen (in Bezug auf Kompatibilität und vor allem Dateigröße).



The screenshot shows the Microsoft Excel 2000 interface. The title bar reads 'Microsoft Excel - Mappe1'. The menu bar includes 'Datei', 'Bearbeiten', 'Ansicht', 'Einfügen', 'Format', 'Extras', 'Daten', and 'Fenster'. The toolbar contains various icons for file operations, editing, and formatting. The status bar at the bottom shows 'Bereit' and 'NF'. The spreadsheet has a single sheet named 'Tabelle1'. The data is as follows:

	A	B	C	D	E	F
1	Anwesenheit in Stunden					
2		Müller	Meier	Schulze		
3	Montag	8	8	8		
4	Dienstag	7	8	8		
5	Mittwoch	8	8	5		
6	Donnerstag	8	4	6		
7	Freitag	8	5	7		
8	Samstag	5	5	8		
9						
10						
11						
12						

Abbildung 7.14: Die zu speichernde Exceldatei

Die Datei in unserem Beispiel wird im HTML-Format 4.374 Byte groß sein. Der Quelltext enthält neben Eigenwerbung eine gigantische Anzahl von Style Sheets, obwohl es dafür keinerlei Notwendigkeit gibt. Daneben gibt es einige überflüssige HTML-Tags.

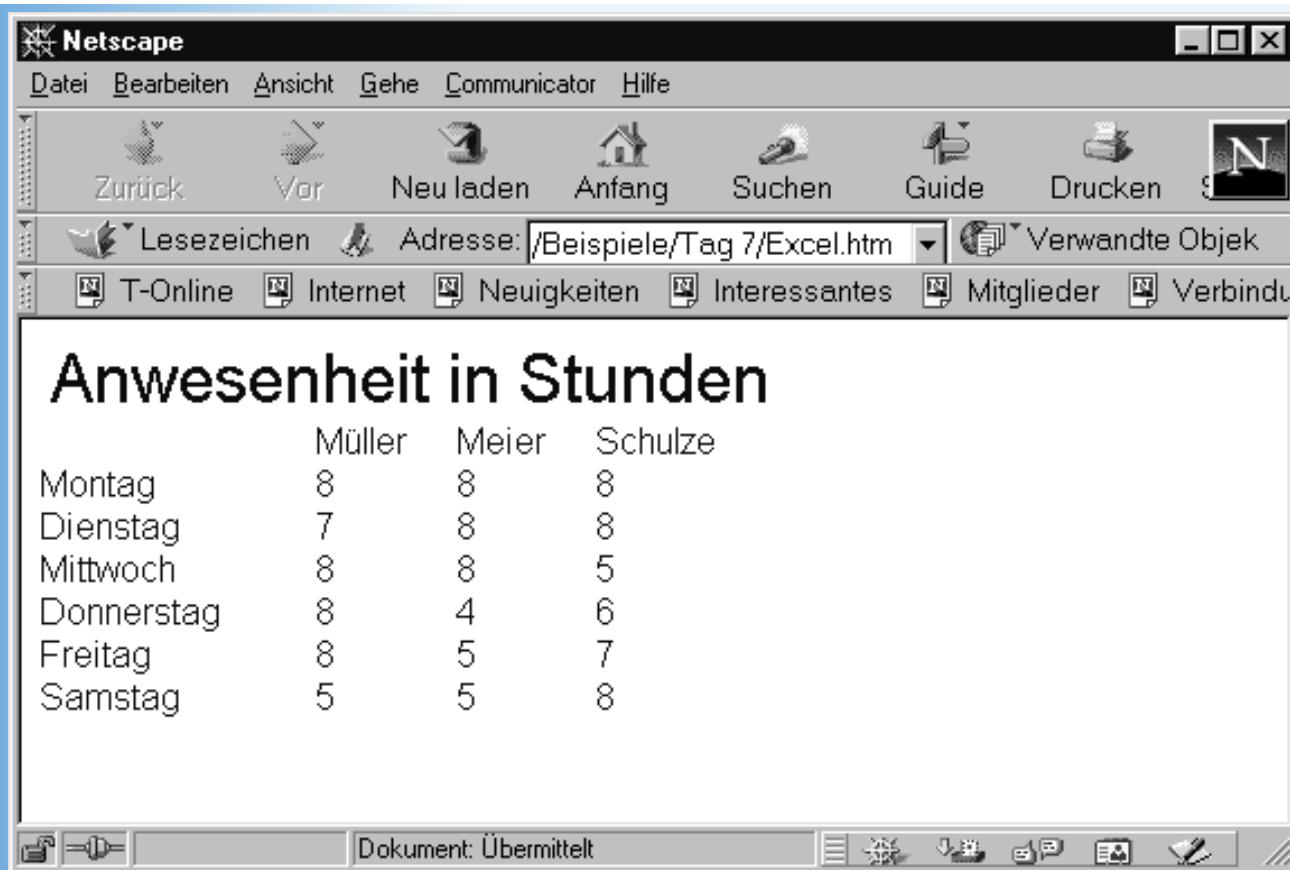


Abbildung 7.15: So sieht die Originaldatei in einem neuen Browser aus

Abgesehen davon, dass die Datei in älteren Browsern Schwierigkeiten machen kann, wird sogar ein Tool mit Kompressionsverfahren an den Style Sheets scheitern, denn diese werden nicht optimiert. Ohne manuelles Eingreifen bekommt das Kompressionstool von 1st Page 2000 die Datei nur auf 4.144 Bytes. Realistisch könnte man die Datei manuell auf vielleicht 800 Byte oder weniger bekommen, indem man die Style Sheets durch HTML-Anweisungen ersetzt und schlechtes HTML optimiert (ohne wesentlichen Verlust der Darstellungsqualität). Aber durch die ausufernde Verwendung von Style Sheets wird auch die manuelle Optimierung von solchen Webseiten ein ungemütliches und aufwändiges Verfahren.



Die Optimierung von Quellcode auf diese Art und Weise macht übrigens bei compilierten Sprachen keinen Sinn. Dort sorgt der Compiler (zumindest bei neueren Techniken) dafür, dass der resultierende Programmcode in dieser Art sowieso optimiert wird. Weder die Länge der Bezeichner noch die reduzierte Verwendung von Syntaxstrukturen hat bei optimierten Compilern einen Effekt auf den endgültigen Programmcode. Da hat Lesbarkeit und Wartbarkeit des Quelltextes Priorität. Zumal dort zudem die Verschleierung von Quelltextinformationen irrelevant wird.

Strukturieren und aufteilen

Ein in Bezug auf Wartbarkeit sehr wichtiges Thema ist die Strukturierung von Inhalten. Auch wenn es am Anfang kaum sinnvoll erscheint, sollte jedes Webprojekt soweit wie möglich strukturiert werden. Das bedeutet, soweit es realisierbar ist, sollten unterscheidbare Elemente einer Webseite auseinander dividiert werden. Also Scripts, Stilvereinbarungen und HTML-Text sollten erst einmal möglichst in getrennte Dateien gegliedert werden.

Dazu sollte für jeden referenzierten Dateityp ein eigenes Verzeichnis angelegt werden. Das bedeutet, ein Verzeichnis für Grafiken (eventuell sogar noch unterteilt in verschiedene Unterverzeichnisse oder parallele Ordner für animierte Grafiken, Buttons usw.), eines für Java-Applets, eines für externe Scripts, eines für externe Stildateien usw. Dazu sollte man auf jeden Fall prägnant gewählte Dateinamen wählen.



Es macht übrigens wenig Sinn, Dateinamen durchzunummerieren. Spätestens, wenn eine Datei gelöscht wird oder in der Logik eine Datei zwischengeschoben werden muss, wird das Konzept hinfällig.

Das Verfahren scheint am Anfang recht aufwändig und überzogen, aber selbst bei kleinen Webprojekten verliert man im Laufe der Zeit schnell den Überblick, wenn man alle Dateien in einem Ordner vorhält. Die Wartbarkeit wird extrem erhöht, wenn man die Aufteilung vornimmt.



Weder Stilvereinbarungen noch Scripte müssen in jeweils einer einzelnen externen Datei notiert werden. Sofern man mit getrennten Verzeichnissen arbeitet und sinnvolle Dateinamen wählt (plus geeigneter Dokumentation), spricht viel dafür, jeweils mehrere Dateien zu nehmen. Sie sollten bedenken, dass auch diese Dateien natürlich über das langsame Netz übertragen werden müssen. Wenn eine HTML-Datei beispielsweise nur eine Funktion aus einer externen JavaScript-Datei benötigt und diese in einer riesigen Datei - möglichst noch am Ende - notiert ist, kann das natürlich auch auf Kosten der Ladezeit gehen. Lieber unterteilt man in verschiedene kleinere Dateien und referenziert bei Bedarf mehrere Dateien in einer Webseite gemeinsam.

Zusammenfassung

Es gibt zwar keine Programmierung, ohne dass Fehler gemacht werden. Es gibt aber zahlreiche Möglichkeiten, diese Fehlerquellen zu reduzieren und - falls doch Fehler auftreten - diese mit sinnvollen Konzepten zu suchen. Die Techniken zum Auffinden und Beseitigen von Fehlern gelten nicht nur in JavaScript, sondern in nahezu jeder Programmier- und Scriptsprache gleich oder zumindest ähnlich. Auch in HTML kann man viele der Überlegungen sinnvoll anwenden. Dies gilt in gleicher Weise für den Vorgang der Optimierung, der bei HTML und Scriptsprachen eine erhebliche Beschleunigung des Ladevorgangs einer Webseite bewirken kann.

Fragen und Antworten

Frage:

Warum ist JavaScript nicht so fehlertolerant wie HTML?

Antwort:

In Scripten bauen Schritte massiv aufeinander auf. Außerdem gibt es gewisse Syntaxstrukturen, die geschützt werden müssen. Wenn vorher ein Problem durchgelassen würde, könnte das ungeahnte Folgen haben. Aber auch so kann ein falsches Script sogar den Browser abschießen.

Frage:

Lässt sich jede Webseite nach der funktionellen Fertigstellung noch optimieren?

Antwort:

Im Prinzip ja, denn meist steht ja bei der Erstellung erst die Funktion im Vordergrund. Erst wenn man das gesamte Werk beurteilen kann, zeigen sich die Stellen, welche sich für eine Optimierung anbieten. Besonders von Tools erstellte Webseiten bieten gigantische Optimierungsmöglichkeiten.

Frage:

Wann sollte man eine Webseite optimieren?

Antwort:

Möglichst erst, wenn die Webseite sonst vollkommen fertig ist. Auch hier gilt, dass erst die Betrachtung der Gesamtseite die richtigen Ansätze offenbart.

Frage:

Sollte man jede Webseite optimieren?

Antwort:

Nein, bei kleinen Seiten mit wenig Text und im Verhältnis vielen Multimedia-Elementen macht der Aufwand kaum Sinn. Die Gefahr, Fehler zu machen, ist auch nicht unerheblich. Eine schnell mal durchgeführte Optimierung führt oft dazu, dass man ein Fass ohne Boden vorfindet.

Workshop

- Suchen Sie sich einige der bisher erstellten Webseiten und Scripte aus und optimieren Sie diese in Hinsicht auf die Dateigröße.
- Wenn Sie Excel zur Verfügung haben, erstellen Sie dort eine Tabelle und speichern Sie diese als Webseite. Ansonsten finden Sie auf der Buch-CD eine Musterdatei (Excel.htm). Kontrollieren Sie die Dateigröße, analysieren Sie den Quelltext in einem Editor und versuchen Sie, die Datei zu verkleinern. Löschen Sie Werbung und sonstige überflüssige Tags.
- Erstellen Sie Webseiten mit den in den Browser-Paketen integrierten Tools. Kontrollieren Sie die Dateigröße, analysieren Sie den Quelltext in einem Editor und versuchen Sie, die Datei zu verkleinern. Löschen Sie Werbung und sonstige überflüssige Tags.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was ist ein typographischer Fehler?
2. Was stimmt nicht an der Variablendeklaration?

```
var 5i;
```

3. Was stimmt nicht an der Syntax?

```
function test();
{
document.write('<HTML>');
}
```

4. Was stimmt nicht an der Syntax?

```
var i;j;k;l;
```

5. Was stimmt nicht an der Syntax?

```
var i,j,k,l
```

6. Was stimmt nicht an der Syntax?

```
var i j k l;
```

7. Was lässt sich an der nachfolgenden Syntax ohne großen Aufwand optimieren?

```
function schreibewebseite()
{
document.write('<html>');
document.write('<head>');
document.write('</head>');
document.write('<body link="#9F9F9F" bgcolor="#FFFFFF">');
document.write('</body>');
document.write('</html>');
}
```


2 In kleinen Projekten und in Hobby-Webseiten muss man das natürlich nicht so eng sehen.

3 Es sei denn, der Anwender schaut sich den Quelltext der Seite an.

4 Wir kümmern uns in dem Buch ja hauptsächlich um textbasierende Details der Webseite wie HTML
und vor allem Scripte.

5 Oder verschärft - im Internet wird geklaut wie bei den Raben ;-)

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 1 Rückblick

Sie haben die erste Woche nun geschafft. Fassen wir noch einmal knapp zusammen, was an Stoff behandelt wurde. In dieser Woche wurde besprochen,

- wie Client-Server-Netze, das WWW und das Internet in Grundzügen funktionieren,
- was die Bedeutung von HTML und Style Sheets ist und wie insbesondere die Grundlagen von HTML aussehen,
- was Browser tun und was man dabei beachten muss,
- welche Programme Sie benötigen, um Webseiten erstellen und JavaScript programmieren zu können,
- wie man Scripte und Style Sheets in Webseiten integriert,
- was JavaScript darstellt und wie man erste Scripte erstellen kann,
- wie sich JavaScript im Web im Vergleich zu weiteren Techniken einordnet und
- wie man diese geniale Scriptsprache als Erweiterung von HTML einsetzen kann,
- wie man Fehler reduzieren bzw. behandeln und lokalisieren kann und
- wie die Optimierung von Webseiten, Scripten und Programmen im Allgemeinen abläuft.

Didaktisch wurde die Vermittlung des Einmaleins einer Webseite - insbesondere HTML-Grundlagen - dem systematischen Aufbau von JavaScript vorgezogen. Dies ist aber unabdingbar, denn um JavaScript einsetzen zu können, muss dieses HTML-Elementarwissen vorhanden sein. Sie haben nun einen Punkt erreicht, wo sie bereits den Quelltext von Webseiten lesen und erste Scripte selbständig erstellen können.

Woche 2

In der nächsten Woche werden wir den systematischen Aufbau von JavaScript in Angriff nehmen. Wir wollen die Syntax von JavaScript untersuchen und dabei wichtige Themen wie

- Variablen,
- Datentypen,
- Literale,
- Ausdrücke,
- Funktionen,
- Prozeduren,
- Objekte,
- Methoden,
- Eigenschaften,
- Wertzuweisungen,
- Operatoren und Operanden,
- Schlüsselworte,
- Anweisungen,
- JavaScript-Aufrufe,
- Listen, Aufzählungen und Tabellen und
- die JavaScript-Verbindung zu Java sowie allgemeinen Multimedia-Techniken

behandeln.

Woche 2

Tag 8

Variablen, Datentypen, Funktionen und Methoden

Effektives Programmieren in JavaScript setzt - wie bei jeder Programmier- oder Scriptsprache - voraus, dass man sich mit der Syntax der Sprache vertraut macht. Dies wiederum beinhaltet, dass elementare Bestandteile der Syntax wie

- Variablen und Datentypen,
- Literale,
- Ausdrücke,
- Objekte mit ihren Methoden und Eigenschaften,
- Funktionen und Prozeduren,
- Wertzuweisungen,
- Operatoren und
- Schlüsselworte

und noch einige andere Begriffe vertraut sind. Dieser Tagesabschnitt soll Ihnen darüber einen grundsätzlichen Überblick über Variablen und Datentypen sowie Funktionen bzw. Prozeduren und Methoden bieten.

Funktionen, Prozeduren und Methoden

Uns sind an einigen Stellen bereits Funktionen im Rahmen von diversen Beispielen begegnet. Bisher definieren wir Funktionen als Zusammenfassungen von einem¹ oder mehreren Befehlen zu einem Block, der mit einem Bezeichner benannt wird. Wesentliche Gründe für das Verwenden von Funktionen sind folgende:

- Immer wieder stellt sich heraus, dass in einem Script bestimmte Befehlsfolgen öfter benötigt werden. Um sie nicht jedes Mal neu schreiben zu müssen, werden sie zusammengefasst, mit einem Namen versehen und dieser wird nur hingeschrieben, wenn die Folge von Befehlsschritten benötigt wird. Das bedeutet kürzeren Quellcode, ist schneller geschrieben und viel weniger fehlerträchtig.
- Oft sollen bestimmte JavaScript-Schritte nicht gleich beim Laden eines Scripts ausgeführt werden. Eine Definition als Funktion schützt sie gegenüber der direkten Abarbeitung beim Laden. Erst wenn die Funktion über ihren Namen aufgerufen wird, wird die Befehlsfolge ausgeführt.
- Quellcode kann viel stärker strukturiert werden, was die Les- und Wartbarkeit erhöht.
- Eine Aufgabenverteilung bei komplexeren Scripten ist möglich.

Die Definition von Funktionen erfolgt, indem das dafür in JavaScript vorgesehene Schlüsselwort `function` die Definition einleitet, dann der Bezeichner der Funktion folgt und abschließend ein Klammernpaar notiert wird, in dem eventuell benötigte Parameter an die Funktion weitergegeben werden können. Solche Parameter sind Werte, die dann innerhalb der Funktion verwendet werden können. Dies sind dann Variablen, die in der Funktion lokal verwendet werden können (siehe dazu etwas weiter unten bei lokalen Variablen). Anschließend folgt der Block mit den enthaltenen Anweisungen - ohne (!) dass vorher ein Semikolon steht. Dies sieht schematisch also so aus:


```
function [Funktionsbezeichner]([optionale Parameter])
{
    ...irgendwelche Anweisungen
}
```

Der Aufruf erfolgt an einer beliebigen Stelle im Script (auch innerhalb einer anderen Funktion) oder über einen Eventhandler in HTML (das ist so was wie onLoad oder onClick, was wir schon benutzt haben). Wenn im Script selbst eine Funktion aufgerufen wird, muss sie mit dem Bezeichner, den Klammern, eventuell notwendigen Übergabewerten und einem abschließenden Semikolon notiert werden.



Der am häufigsten gemachte Fehler im Umgang mit Funktionen ist das Vergessen von Klammern. Einer Funktion (und einer Methode - s.u.) muss so gut wie immer (!!) ein Klammerpaar hinter dem Bezeichner stehen haben (die wenigen Sondersituationen, wo dies unterbleiben kann, treten so selten auf, dass man deswegen nicht auf die nachdrücklichste Betonung von »immer« verzichten sollte). Ob da etwas drin steht oder nicht, ist egal. Dabei muss die öffnende Klammer ohne Leerzeichen dem Bezeichner folgen. Es gibt zwar Sprachen wie VBScript oder auch Visual Basic, die das Prozedere mit den Klammern nicht so streng handhaben. Obwohl es auf den ersten Blick als eine Erleichterung erscheint, erhöht es die Fehlergefahr exorbitant, erschwert die Fehlersuche und reduziert die Wartbarkeit. Seien Sie froh, dass JavaScript da ziemlich pingelig ist.

Der Name einer Funktion ist relativ frei wählbar. Nur folgende Regeln dürfen nicht verletzt werden:

1. Das erste Zeichen eines Bezeichners muss ein Buchstabe oder der Unterstrich _ sein.
2. Es darf keines der in JavaScript gültigen Schlüsselworte als Bezeichner verwendet werden.
3. Ein Bezeichner darf keine Leerzeichen enthalten.

Ob nun ein Bezeichner aus Groß- oder Kleinbuchstaben besteht oder man diese mischt, das ist erst mal egal (es sei dann, Sie werden von Projektrichtlinien zu einer bestimmten Notation gezwungen). Wenn Sie allerdings bei der Deklaration eine bestimmte Schreibweise gewählt haben, muss diese konsequent bei jedem Aufruf so geschrieben werden. Man kann es nicht oft genug betonen - JavaScript unterscheidet Groß- und Kleinbuchstaben. Es haben sich bezüglich der Schreibweise von Bezeichnern in der Praxis einige sinnvolle Regeln etabliert. Diese sind nicht verbindlich, aber wie gesagt sinnvoll (insbesondere in Bezug auf die Übersichtlichkeit und spätere Wartung):

- Wenn eine Schreibweise für Bezeichner von Funktionen einmal gewählt wurde, sollten alle Bezeichner so geschrieben werden. Das bedeutet, wenn der eine Bezeichner einer Funktion vollkommen klein geschrieben wird, sollten auch alle Bezeichner von anderen Funktionen vollkommen klein geschrieben werden. Eine inkonsequente Schreibweise in einer Webseite oder einem Projekt - also mal klein, mal gemischt, mal groß - ist nicht verboten, aber extrem unglücklich. Das gilt vollkommen analog für Bezeichner von Variablen und anderen Syntaxelementen, die frei wählbare Bezeichner bekommen können.
- Funktionsbezeichner sollten nicht zu lang, aber dennoch einigermaßen sprechend sein. Wenn es notwendig ist, werden mehrere Worte zu einem Bezeichner zusammengesetzt.
- Oft nutzt man für Bezeichner von Funktionen folgende Regel: Funktionsbezeichner fangen mit einem Kleinbuchstaben an, werden dann so lange klein weitergeschrieben, bis - bei sinnmässig zusammengesetzten Begriffen - ein neues Wort im Bezeichner beginnt. Dieses beginnt dann mit einem Großbuchstaben. Etwa so:

```
meineFunktion(),
ladeSeite(),
holeCookiewert().
```

Beispiel 1:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function eins()
```

```
{
alert("Eins");
}
function zwei()
{
alert("Zwei");
}
function drei()
{
alert("Vier?");
}
function lade()
{
eins();
zwei();
drei();
}
</SCRIPT>
</HEAD>
<BODY onLoad="lade()">
</BODY>
</HTML>
```

Das Beispiel (der Dateiname soll funktion1.htm sein) definiert vier Funktionen ohne Übergabeparameter. Nur die Funktion lade() wird beim Laden der Seite aufgerufen. Diese wiederum lädt nacheinander die anderen drei Funktionen über die Angabe ihres Bezeichners, der leeren Klammern und eines abschließenden Semikolons.

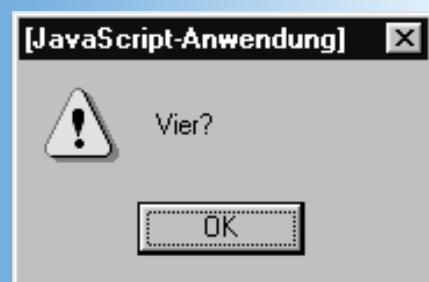


Abbildung 8.1: Der Aufruf der Funktion drei()



Abbildung 8.2: Der Aufruf der Funktion zwei()

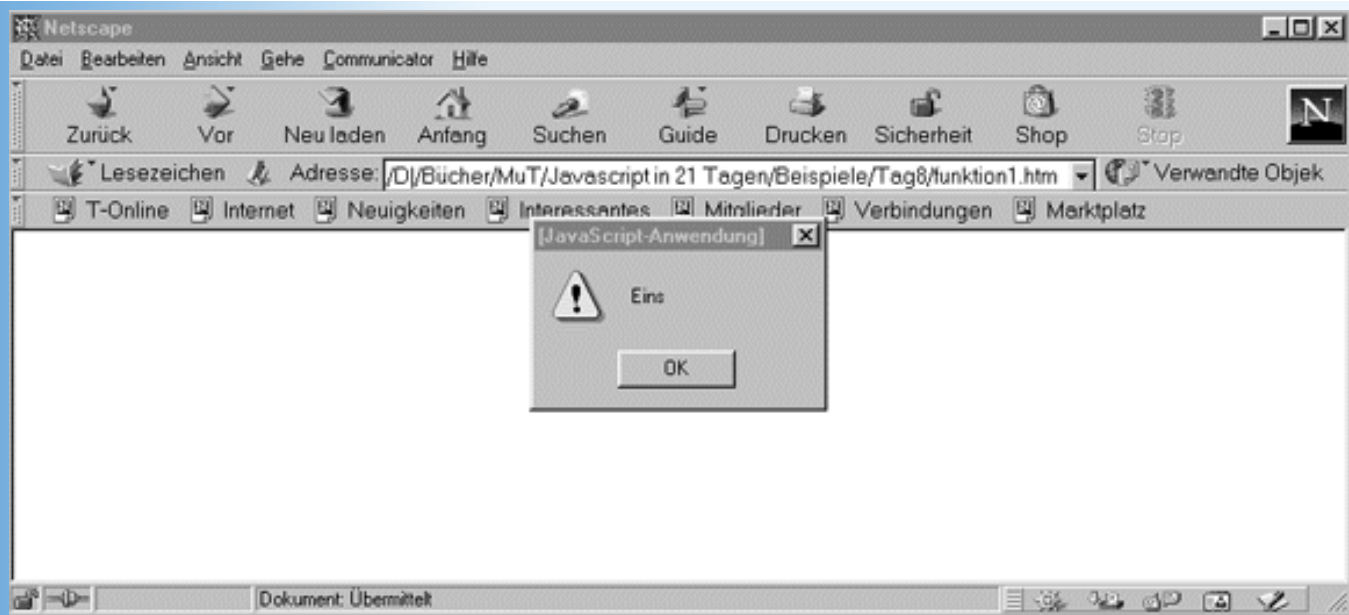


Abbildung 8.3: Der Aufruf der Funktion eins()

Rückgabewerte

Eine Funktion kann nun am Ende ihrer Arbeit ein Ergebnis zurückgeben. Etwa wenn Sie eine Funktion geschrieben haben, die zwei Zahlen addiert und dann die Summe als Ergebnis liefert. Das wird in JavaScript darüber geregelt, dass das Schlüsselwort

`return`

gefolgt von dem Ergebnis notiert wird.

Das hat noch den zusätzlichen Effekt, dass an dieser Stelle die Abarbeitung der Funktion abgebrochen und in die Zeile hinter dem Funktionsaufruf zurückgesprungen wird. Die Anweisung `return` steht entweder am Ende der Funktion oder innerhalb einer Kontrollfluss-Struktur, sodass unter gewissen Umständen die Funktion abgebrochen wird, in einer anderen Situation aber weiter abgearbeitet werden kann. Mit einer solchen Kontrollfluss-Struktur und der Anweisung `return` kann sehr flexibel auf Konstellationen reagiert werden. Eine Funktion mit Rückgabewert sieht schematisch also so aus:

```
function [Funktionsbezeichner]([optionale Parameter])
{
    ...irgendwelche Anweisungen
    return [Rückgabewert];
}
```

Die Verwendung des Rückgabewertes kann innerhalb der Bedingung einer Kontrollstruktur verwendet (das werden wir an späteren Tagen noch sehen) oder einer Variablen zugewiesen werden. Es ist auch möglich, den Aufruf der Funktion dort zu notieren, wo ein dem Rückgabewert entsprechender Wert oder eine adäquate Variable stehen könnte.

Beispiel 2:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function addiert()
{
    return (2 + 2);
}
function lade()
{
```

```
alert(addiert());  
}  
</SCRIPT>  
</HEAD>  
<BODY onLoad="lade()">  
</BODY>  
</HTML>
```

Das Beispiel (der Dateiname soll funktion2.htm sein) definiert zwei Funktionen ohne Übergabeparameter. Wieder wird nur die Funktion lade() beim Laden der Seite aufgerufen. Diese wiederum gibt mit alert() eine Meldung aus. Was aber ausgegeben wird, ist der Rückgabewert der Funktion addiert(), die dort anstelle eines konkreten Wertes steht und ausgeführt wird.



Abbildung 8.4: Die Rückgabe der Funktion lade()

Wenn nun eine Funktion keinen Rückgabewert zurückgibt, wird das in den meisten Programmiersprachen als *Procedure* oder - eingedeutscht - *Prozedur* bezeichnet (in der Regel auch bei der Deklaration von einem anderen Schlüsselwort wie eine Funktion eingeleitet). Von der Logik her bedeutet dies, dass bei Funktionen das Ergebnis einer Operation von Bedeutung sein sollte, während die Operation einer Prozedur Selbstzweck ist. JavaScript unterscheidet die beiden eng verwandten Techniken überhaupt nicht (unser erstes Beispiel wäre der Fall einer Prozedur gewesen, das zweite das einer echten Funktion). Damit verwischen die Unterschiede zwischen Funktionen und Prozeduren. Eine Funktion muss nicht mehr einen Rückgabewert liefern oder zumindest als »ohne Rückgabewert« definiert werden. Überhaupt behandelt JavaScript Funktionen sehr locker. Von der Syntax sehr streng strukturierte Sprachen verlangen, dass der Rückgabewert einer Funktion auch von der aufrufenden Stelle verwendet werden muss (!). Das bedeutet, ein Aufruf einer Funktion ist dort nur dann möglich, wenn der Rückgabewert einer Variablen zugewiesen oder sonstwie verwendet wird (etwa in Vergleichen einer Kontrollstruktur). JavaScript ist da weniger streng. Dies bedeutet, im zweiten Beispiel könnte auch die Funktion addiert() aufgerufen werden, ohne dass das Ergebnis 4 irgendwo in der Folge im Script zur Kenntnis genommen wird (auch wenn man sich dann fragen kann, wozu die Funktion aufgerufen wird - in anderen Fällen kann eine Funktion aber schon etwas Sinnvolles erledigt haben). Ein Verhalten, das den Umgang mit Funktionen erst einmal erleichtert. Aber wenn Sie sich erinnern, leite ich so immer Situationen ein, die dann auch einen Haken haben. So auch hier. Eine lockere Leine bei der Syntax erschwert immer die Suche von Fehlern, wenn etwas schiefgeht. Ebenso können viel leichter logische Probleme in Programmen auftreten. Eine weitere Gefahr ist, dass sich leicht ein unsauberer Programmierstil einschleicht. Aber es hilft nichts - Sie müssen JavaScript halt nehmen, wie es ist. JavaScript ist immer noch nicht so bedenklich wie VBScript, was da noch mehr Freiheiten lässt (und damit die Menge der potentiellen Probleme im Fehlerfall noch steigert). Es liegt auch viel am Programmierer selbst, inwieweit er die Freiheiten einer Sprache ausnutzt (missbraucht?) oder sich einen disziplinierten Programmierstil angewöhnt.



Wir werden bei den Variablen noch einige praktische Beispiele durchspielen, denn diese kann man erst in Verbindung damit sinnvoll programmieren. Dazu zählt etwa die Situation mit Übergabewerten.

Sowohl Funktionen als auch Prozeduren gibt es in streng objektorientierten Sprachen nicht. Dort übernimmt eine Technik deren Funktionalität, die wir schon einige Male angerissen haben - die so genannten *Methoden*. Dabei handelt es sich eigentlich nur um Funktionen oder Prozeduren, die ausschließlich über ihr zugehöriges Objekt verwendet werden können. Wir kennen da schon einige Fälle:

- Die Methode `write()` - über das Objekt `document` verwendet - (`document.write()`).

- Die Methode `random()`, die über das Objekt `Math` verwendet wird (`Math.random()`).
- Die Methode `alert()`. Ja, auch das ist eine Methode. Nur - zu welchem Objekt gehört sie? Wir verwenden sie scheinbar ohne zugehöriges Objekt. Aber die Betonung liegt auf scheinbar. Man kann dann eine Methode ohne vorangestelltes Objekt verwenden, wenn man sich im »Inneren« des Objektes befindet. Das ist so wie bei Angaben auf Verzeichnisebene. Wenn man sich in einem Verzeichnis befindet, kann bei einer relativen Angabe auf die Voranstellung des absoluten Pfades verzichten. Die Methode `alert()` gehört zu einem Objekt, das `window` heißt und alle im Browser befindlichen Elemente enthält. Die Syntax `alert()` ist also gleichbedeutend mit `window.alert()`.
- Auch `prompt()` ist eine Methode von `window`.

Wir trennen in diesem Buch von der Terminologie her nicht ganz so streng zwischen Funktionen und Methoden, obwohl es eigentlich ein bisschen schlampig ist. Nur - JavaScript (wie die meisten Scriptsprachen) stellt sowohl Funktionen (und streng genommen auch Prozeduren) als auch Objekte mit ihren Methoden bereit. Das bewirkt den hybriden (d.h. sowohl in Ansätzen objektorientierten als auch prozeduralen) Charakter der Sprache und erleichtert insbesondere Einsteigern und nicht mit der objektorientierten Denkweise vertrauten Programmierern am Anfang die Arbeit. Und - auch wenn Sie dieses ständige Miesmachen von Erleichterungen für Einsteiger nervt ;-) - macht größere Strukturen schlechter durchschaubar als bei strengerer Verfahrensweise.



JavaScript kennt einige vorgefertigte Funktionen. Wir werden die wichtigsten davon noch kennen lernen.

Variablen, Datentypen und Literale

Eine elementare Grundanforderung an eine Programmier- oder Scriptsprache sind Variablen. Aber was ist das genau? Wir haben Variablen bisher bereits mehrfach in Beispielen verwendet, aber sie sind quasi vom Himmel gefallen.

Variablen bezeichnen technisch gesehen Stellen im Hauptspeicher, in denen irgendwelche Werte temporär gespeichert werden können. Dazu haben sie einen Namen oder Bezeichner, der einigen Regeln in JavaScript genügen muss, und einen so genannten Datentyp.



Im Zusammenhang mit Variablen fällt oft der Begriff Literal. Ein Literal ist eine Darstellung eines eindeutig festgelegten und unveränderlichen Werts, etwa 42 oder 3.14 oder "Die Antwort ist". Ein Literal ist also das, was einer Variablen zugewiesen werden kann.

Die Regeln bezüglich des Namens einer Variablen sind in JavaScript sehr locker und entsprechen denen von Funktionen:

1. Das erste Zeichen eines Variablenbezeichners muss ein Buchstabe oder der Unterstrich `_` sein.
2. Es darf keines der in JavaScript gültigen Schlüsselworte als Bezeichner verwendet werden.
3. Ein Bezeichner darf keine Leerzeichen enthalten.

Grundsätzlich muss eine Variable vor ihrer ersten Verwendung deklariert, d.h. eingeführt oder im Script bekannt gemacht werden. Dies erfolgt in JavaScript allgemein mit folgender Syntax:

```
var [Variablenbezeichner];
```

Der Begriff `var` bezeichnet dabei ein JavaScript-Schlüsselwort, das die Deklaration der Variablen einleitet. Nun gehört JavaScript aber zu einem Typ von Sprachen, der als lose typisiert bezeichnet wird. Dies hat einmal zur Folge, dass eine Variable auch dann entstehen kann, wenn einem neuen Bezeichner einfach nur ein Wert zugewiesen wird (auch ohne das Schlüsselwort `var` voranzustellen). Dies hat extrem weitreichende Konsequenzen - positive, aber auch überaus negative.

Die positive Auswirkung ist, dass sich ein JavaScript-Programmierer wenig Gedanken um Variablen machen muss.

Er kann einfach einen Begriff zu einer Variablen machen, indem dem Begriff ein Wert zugewiesen wird. Das bedeutet, dass Variablen nicht explizit deklariert werden müssen. Der Preis für so eine Bequemlichkeit (denn nicht mehr und nicht weniger ist das) ist aber hoch. Das wollen wir uns mit einem kleinen Beispiel deutlich machen.

Beispiel 3:

Geben Sie im Editor den nachfolgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function lade()
{
var username;
userName = prompt(
"Bitte geben Sie Ihren Namen ein", "");
document.write("<HTML>");
document.write("<BODY bgcolor=green text=yellow>");
document.write("<H1 align=center>Willkommen</H1>");
document.write("<P align=center>");
document.write(username);
document.write("</P>");
document.write("</BODY>");
document.write("</HTML>");
}
</SCRIPT>
</HEAD>
<BODY onLoad=lade(>
</BODY>
</HTML>
```

Erinnern Sie sich an das Beispiel? Wir hatten es am Tag 6 bereits in fast (!) gleicher Form. Speichern Sie den Text bitte unter dem Dateinamen var1.htm ab und schauen Sie sich die Datei im Browser an. Über prompt() wird beim Laden der Seite ein Benutzerdialog angezeigt, in dem der Besucher eine Eingabe vornehmen kann. Diese Eingabe wird in einer Variablen gespeichert und soll dann bei der dynamischen Erzeugung der Webseite verwendet werden. Was aber passiert wirklich?



Abbildung 8.5: Der eingegebene Name wird nicht (!) in der Webseite verwendet

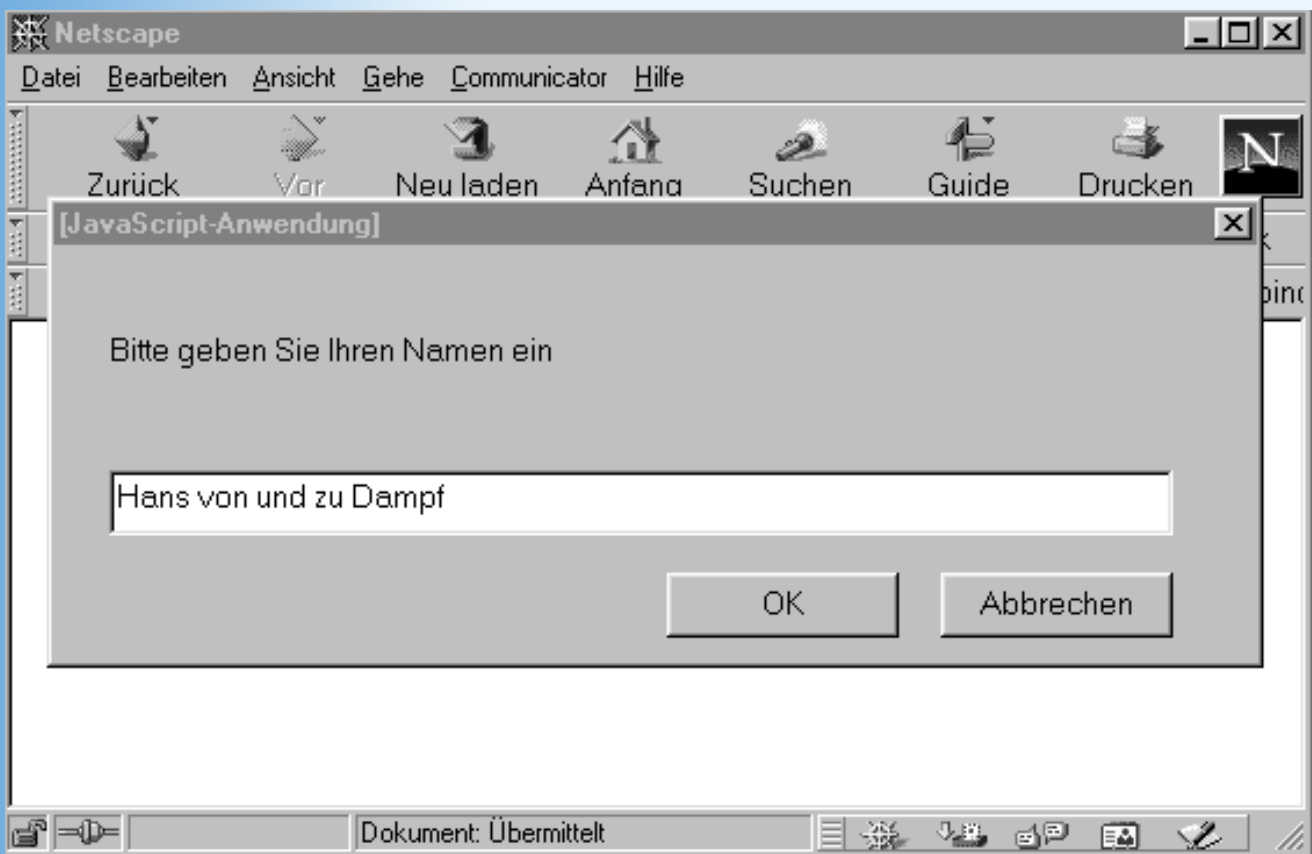


Abbildung 8.6: Der Anwender gibt seinen Namen ein

Der eingegebene Name wird nicht in der Webseite verwendet. Statt dessen steht dort etwas von wegen undefiniert (undefined). Aber wo ist das Problem?

Die Variable username ist nicht (!) identisch mit userName. JavaScript unterscheidet Groß- und Kleinschreibung! Das sollten Sie mittlerweile wissen ;-). Aber es ist ein oft gemachter Leichtsinnsfehler - auch bei Profis. Die Konsequenzen sind aber in Verbindung mit dem Verhalten von JavaScript in Bezug auf Variablendeklarationen

drastisch.

Die Zeile

```
var username;
```

legt eine Variable mit dem Namen an. Die nachfolgende Zeile

```
userName = prompt(
"Bitte geben Sie Ihren Namen ein", "");
```

sollte von der Planung her dieser Variablen einen Wert zuweisen. Da aber userName ein neuer Bezeichner ist (für JavaScript so unterschiedlich, als ob dort Quasimodo stehen würde), erzeugt die Zeile eine neue Variable mit Namen userName. Insbesondere bleibt die Variable username ohne gültigen Wert - sie ist undefiniert. Die nachfolgende Ausgabe über

```
document.write(username);
```

kann also keinen sinnvollen Wert aus der Variablen entnehmen.

Der Preis für die nicht notwendige explizite Deklaration über das Schlüsselwort var ist also der, dass bei Tippfehlern ungewollt neue Variablen entstehen können und dafür Variablen, denen man eigentlich einen neuen Wert zuweisen wollte, diesen Wert nicht annehmen. Solche Fehler sind in komplexeren Scripten extrem tückisch aufzufinden.

Ein wenig wird die Situation durch eine Syntaxregel von JavaScript wieder entschärft. Jede Variable muss vor ihrer ersten Verwendung initialisiert werden, was dann oft im Fehlerfall auch eine sinnvolle Fehlermeldung bewirkt.



Initialisierung nennt man die Vorbelegung von Variablen mit einem Vorgabewert.

Diese Initialisierung einer Variablen unter JavaScript wird einfach mit einer Wertzuweisung durchgeführt, wie es in unserem Beispiel erfolgt ist. Es kann aber gelegentlich helfen zu erkennen, dass irgendwo eine neue Variable versehentlich eingeführt worden sein muss.



Wertzuweisung bedeutet, eine Variable bekommt irgendwie einen Wert zugeordnet. Eine Wertzuweisung erfolgt in der Regel so, dass auf der linken Seite eine Variable steht, dann das Gleichheitszeichen und dann auf der rechten Seite der Wert, eine Berechnung eines Wertes oder eine Funktion, die einen Rückgabewert liefert. Es gibt aber zahlreiche Feinheiten, die noch gesondert behandelt werden.

Grundsätzlich kann sowohl beim Anlegen einer Variablen als auch später im Laufe des Scriptablaufs ein Wert zugewiesen werden. Sehr oft wird einer Variablen bei der Deklaration auch gleich ein Defaultwert zur Vorbelegung zugewiesen. Dies erfolgt allgemein mit folgender Syntax:

```
var [Variablenbezeichner] = [Wert];
```

Der Wert einer Variablen zur Laufzeit wird einfach durch Zuweisung eines neuen Wertes über ein Gleichheitszeichen verändert. Dabei können auf der rechten Seite direkt Werte, aber auch Ausdrücke mit Wertrückgabe (Berechnungen, Funktionen etc.) stehen. Darauf kommen wir bei dem Abschnitt über Wertzuweisungen umfassend zurück.

Ein weiterer Begriff ist bei der Deklaration von Variablen gefallen - Datentypen. Ein Datentyp ist eine Angabe darüber, wie eine Variable (oder ein einfaches Objekt - für nähere Erläuterungen dazu haben wir aber derzeit noch nicht das Rüstzeug) im Rahmen einer Computersprache im Hauptspeicher des Computers dargestellt wird. Im Wesentlichen ist dies die Angabe, wie viele Bit dafür reserviert werden. Eine solche Angabe enthält normalerweise ebenfalls Hinweise darüber, welche Operationen bezüglich des dort gespeicherten Werts erlaubt sind und was bei einer Verknüpfung mit anderen Elementen passieren soll.

In JavaScript wird - wie in den meisten Scriptsprachen - bei der Deklaration einer Variablen nicht explizit ein Datentyp festgelegt. Eine Festlegung des Datentyps erfolgt implizit durch die Art von Wert, die der Variablen

irgendwann zugewiesen wird. Bei einer erneuten Zuweisung mit einem andersartigen Wert verändert sich sogar der Datentyp der Variablen während der Laufzeit des Scripts. Die Datentypen werden also intern von JavaScript verwaltet.

Ein solcher Datentyp mit der Festlegung einer Anzahl von Bit hat die Konsequenz anzugeben, wie viele Zeichen in einer Variablen dieses Typs gespeichert werden können. Nehmen wir eine Variable, die einen Datentyp hat, der ein Byte (8 Bit) groß ist. In einem Byte können genau $2^8 = 256$ verschiedene Zeichen oder Werte gespeichert werden. Dies können beispielsweise die Zahlen von 0 bis 255 sein. Auf jeden Fall ist die Anzahl der darstellbaren Zeichen durch die Länge des Datentyps beschränkt. Wenn größere Zahlen oder mehr Zeichen in der Variablen gespeichert werden sollen, kann dies nicht in einer Variablen dieses Datentyps geschehen - es ist einfach nicht genug Platz im Speicher dafür reserviert.

Jede Programmier- und Scriptsprache muss Variablen mit einem Datentyp versehen. Wie gesagt: es muss ja Hauptspeicher für aufzunehmende Werte reserviert werden. JavaScript besitzt vier Hauptdatentypen und zwei Sonderdatentypen, die aber wie gerade besprochen implizit verwaltet werden. Das explizite Verhalten der Datentypen wird für den Anwender weitgehend verdeckt erfolgen und die Plattform kümmert sich selbst um alle notwendigen Schritte. JavaScript unterstützt folgende vier Grundtypen von Variablen sowie die am Ende der Tabelle notierten Sonderdatentypen:

Typ	Beschreibung
Boolean	Boolean (oder auch Boolsche - nach dem Mathematiker George Boole) Werte sind Wahrheitswerte. Variablen dieses Typs können nur einen der beiden Werte true oder false (wahr oder falsch) annehmen. Solche Variablen werden im Rahmen von Vergleichen genutzt.
Number	Dieser Variablentyp kann eine Ganzzahl oder einen Gleitkommawert enthalten. Diese beiden Varianten sind streng genommen zwei Datentypen, denn offensichtlich braucht es zur Darstellung einer Gleitkommazahl mehr Platz im Hauptspeicher als für eine Ganzzahl (bei einer Gleitkommazahl muss ja sowohl der Ganzzahlanteil als auch der Nachkommanteil gespeichert werden).
	Kommazahlen werden in JavaScript anstelle eines Kommas mit einem Punkt getrennt. Neben der normalen Zahlenschreibweise gibt es die wissenschaftliche Notationsmöglichkeit über die Angaben e oder E. In diesem Fall wird der Potenzwert hinter dieser Exponentialkennzeichnung notiert. In JavaScript können Variablen vom Typ Number nicht nur dezimal, sondern auch oktal (mit einer führenden 0 eingeleitet) oder hexadezimal (eingeleitet durch das Präfix 0x) angegeben werden.
Object	Dieser allgemeine Datentyp kann einen Wert eines beliebigen Typs enthalten. Er wird für das Speichern von Objekten verwendet.
String	Dieser Variablentyp steht für eine Zeichenkettenvariable und kann eine Reihe von alphanummerischen Zeichen enthalten (maximal 256 in einer Zeichenkette - bei Bedarf können aber mehrere Zeichenkette mit + verknüpft werden). Dabei können alphanummerische Zeichen auch maskiert werden. Insbesondere ist so etwas notwendig, wenn Zeichenkettenvariablen Sonderzeichen wie Hochkommata, Tabulatoren oder einen Zeilenumbruch enthalten sollen. Diese kann man nicht direkt eingeben, da sie in der Regel eine Steuerfunktion unter JavaScript oder im Editor haben. Sie müssen maskiert werden, was so ja auch aus HTML bekannt ist. Maskierte Zeichen werden in JavaScript durch das Zeichen \ eingeleitet, gefolgt von einem Buchstaben, der das Steuerzeichen beschreibt. Es gibt folgende Steuerzeichen: Steuerzeichen Funktion \n Zeilenumbruch (nur im Rahmen einer JavaScript-Ausgabe zu verwenden, nicht in einer HTML-Seite.) \f Wagenrücklauf \b Backspace \r DOS-Zeilenumbruch als Ergänzung von \n \t Tabulator

	\ " Anführungszeichen
undefined	Der Datentyp beschreibt eine nichtdefinierte Situation. Eine Variable besitzt so lange diesen Wert, bis ihr nach dem Anlegen explizit ein Wert zugewiesen worden ist.
null	Dieser Sonderdatentyp beschreibt die Situation, wenn ein Objekt noch keinen Wert hat und entspricht »keiner Bedeutung« oder Null. Das ist durchaus ein sinnvoller Wert, der sich von einer leeren Zeichenkette oder der Zahl 0 unterscheidet. Der Datentyp null kann beispielsweise zurückgegeben werden, wenn in einem Dialogfenster mit Abbrechen -Schaltfläche diese betätigt wird.

Tabelle 8.1: Datentypen in JavaScript**Beispiel 4:**

Wir testen nun verschiedene Variablendatentypen. Erstellen Sie die nachfolgende Datei:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function lade()
{
var abc = true;
var def = 123.45;
var hij = 42;
var klm = 5.123E2;
var nop = 5e-2;
var qrs = "Das ist ja doll";
var tuv = new Date();
var wxy = new Array();
var z = document;
document.write("<HTML>");
document.write("<BODY bgcolor=green text=yellow>");
document.write("<H1 align=center>Datentypen</H1>");
document.write("<P align=center>");
document.write("Variable abc hat den Wert ", abc);
document.write("<BR>");
document.write("Variable def hat den Wert ", def);
document.write("<BR>");
document.write("Variable hij hat den Wert ", hij);
document.write("<BR>");
document.write("Variable klm hat den Wert ", klm);
document.write("<BR>");
document.write("Variable nop hat den Wert ", nop);
document.write("<BR>");
document.write("Variable qrs hat den Wert ", qrs);
document.write("<BR>");
document.write("Variable tuv hat den Wert ", tuv);
document.write("<BR>");
document.write("Variable wxy hat den Wert ", wxy[0]);
document.write("<BR>");
document.write("Variable z hat den Wert ", z);
document.write("<BR>");
document.write("</BODY>");
document.write("</HTML>");
}
</SCRIPT>
</HEAD>
```

```
<BODY onLoad=lade()>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Datentypen.htm ab und schauen Sie sich die Datei im Browser an.



Abbildung 8.7: Daten in verschiedener Ausprägung

Da der String-Datentyp für bestimmte Situationen maskierte Zeichen beinhalten kann, werden wir das im nächsten Beispiel testen.

Beispiel 5:

Erstellen Sie die nachfolgende Datei:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function lade()
{
document.write("<HTML>");
document.write("<BODY bgcolor=green text=yellow>");
document.write("<H1 align=center>Datentypen</H1>");
document.write("<P align=center>");
alert(
"Hier ist die erste Zeile der Meldung. Dann folgt ein\nZeilenumbruch.");
alert("

```

```
Hier steht wieder Text. Dann folgt ein\tTabulator.");
document.write(
"\t\"Anführungszeichen kann man nicht direkt verwenden.\"");
document.write("<BR>");
document.write("</BODY>");
document.write("</HTML>");
}
</SCRIPT>
</HEAD>
<BODY onLoad=lade()>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen Datentypen2.htm ab und schauen Sie sich die Datei im Browser an.



Beachten Sie, dass der mit /n gesetzte Zeilenumbruch und auch der Tabulator sich nicht beim direkten Schreiben in eine Webseite auswirkt, sondern nur bei JavaScript-Ausgaben wie der alert()-Methode, die wir im Beispiel verwenden.



Abbildung 8.8: Anführungszeichen können in die Webseite geschrieben werden



Abbildung 8.9: Der Tabulator im Mitteilungsfenster



Abbildung 8.10: Der Zeilenumbruch im Mitteilungsfenster

Casting

Texte sind keine Zahlen und Objekte keine Texte. Das soll erst mal als Aussage so stehen bleiben. Wenn nun in JavaScript die Regel gilt, dass Variablen nicht explizit bei der Deklaration einen Datentyp zugewiesen bekommen, sondern der Datentyp sich implizit durch die Zuweisung eines bestimmten Werts ergibt - was passiert, wenn eine Variable zuerst eine Zahl beinhaltet hat und ihr dann ein Text zugewiesen wird? Dass dies mehrere Schwierigkeiten aufwirft, können Sie bereits erahnen. Beispielsweise folgende:

- Wenn eine Variable eine Zahl zwischen 0 und 255 beinhaltet, kommt sie mit einem Byte im Hauptspeicher aus. Wenn nun dieser Variablen ein Text über - sagen wir mal - 20 Zeichen zugewiesen wird, benötigt die Variable auch plötzlich 20 Byte Platz im Hauptspeicher. Die Umorganisation und Verwaltung des Hauptspeichers muss von der Plattform gemanagt werden.
- Wie sollen sich Variablen verhalten, wenn sie miteinander verknüpft werden und nicht vom gleichen Datentyp sind? Etwa wenn ein Text mit einer Zahl verknüpft wird. Welchen Datentyp hat das Ergebnis? Beispielsweise wenn eine Variable mit dem Inhalt »Müller« mit einer numerischen Variablen verknüpft wird, die den Wert 5 hat?
- Was für Operationen sind mit welchen Datentypen erlaubt? Kann man zwei Texte miteinander multiplizieren? Oder voneinander abziehen?

Grundsätzlich ist bei solchen Situationen ein Vorgang notwendig, der Casting oder Typkonvertierung genannt wird. Dies bedeutet die Umwandlung von einem Datentyp in einen anderen. Prinzipiell unterscheidet man in der Programmierung zwei Arten von Konvertierungen:

- Ad-hoc-Konvertierungen
- explizite Konvertierungen

Explizite Konvertierungen werden nur von sehr leistungsfähigen Programmiersprachen unterstützt (etwa Java). Bei expliziten Konvertierungen löst der Programmierer mittels einer dafür von der Programmiersprache bereitgestellten Syntax selbst eine Umwandlung in einen anderen Datentyp aus. JavaScript unterstützt solch explizites Konvertieren nicht, was auch schon deshalb logisch ist, weil ja schon bei der Deklaration gar kein Datentyp explizit angegeben wird. JavaScript verwendet Ad-hoc-Typkonvertierungen.



Es gibt allerdings in JavaScript einige Funktionen und Methoden, die indirekt eine explizite Typkonvertierung innerhalb enger Grenzen erlauben. Dies sind beispielsweise `parseInt()` und `parseFloat()`, `Number()` sowie `eval()`. Wir kommen darauf an Tag 13 zurück.

Dabei werden bei der Auswertung von Ausdrücken, in denen Variablen oder Werte verschiedenen Datentyps miteinander verbunden werden, durch den Interpreter die eventuell notwendigen Typkonvertierungen automatisch durchgeführt.



Ad-hoc-Konvertierungen sind nicht ungefährlich. Und zwar in Bezug auf zwei Situationen. Es sollte nicht vorkommen, dass eine Konvertierung von einem Datentyp mit mehr Informationen in einen mit weniger Informationen automatisch durchgeführt wird. Was geschieht etwa mit dem Nachkommateil, wenn von einem Datentyp mit Nachkommastellen automatisch in einen Datentyp konvertiert wird, der nur Ganzzahlen aufnehmen kann? Oder was geschieht mit führenden Nullen bei einer als String gespeicherten Telefonnummer, wenn daraus eine Zahl wird? Je nach Strenge einer Sprache geschehen automatische Konvertierungen deshalb nur innerhalb sehr enger Grenzen. Java ist beispielsweise eine extrem strenge Sprache, wo eine automatische Konvertierung nur dann stattfindet, wenn keine Informationen verloren gehen können.

Bei JavaScript wird das Verfahren etwas lockerer gehandhabt. Wenn die übrige Syntax darauf schließen lässt, dass die Konvertierung sinnvoll ist, wird sie meist so sinnvoll wie möglich durchgeführt. Dennoch - es können unter Umständen Informationen verloren gehen (etwa bei besagter Konvertierung von einer als String gespeicherten Telefonnummer in eine Zahl - siehe Beispiel).

Das zweite Problem ist, dass automatische Typkonvertierungen auch dann stattfinden können, wenn es eigentlich nicht sinnvoll ist (oder eventuell auch unterbleiben, obwohl sie gewünscht werden). Auch dazu werden wir Beispiele durchspielen.

Die folgenden Beispiele demonstrieren verschiedene Typkonvertierungen und einige potentiellen Probleme damit.

Beispiel 6:

Erstellen Sie die nachfolgende Datei:

```
<html>
<body>
<SCRIPT LANGUAGE="JavaScript">
var a;
var b;
a = prompt("Geben Sie die erste Zahl ein","");
b = prompt("Geben Sie die zweite Zahl","");
document.write(
"Das Ergebnis von ", a, " + ", b, " ist : ");
document.write(a + b);
document.write("<br>");
</SCRIPT>
</body>
</html>
```

Speichern Sie den Text bitte unter dem Dateinamen Add1.htm ab und schauen Sie sich die Datei im Browser an. Sie müssen nacheinander zwei Zahlen eingeben, die dann addiert werden. Das macht die Zeile

```
document.write(a + b);
```

Aber was wird wirklich ausgegeben?

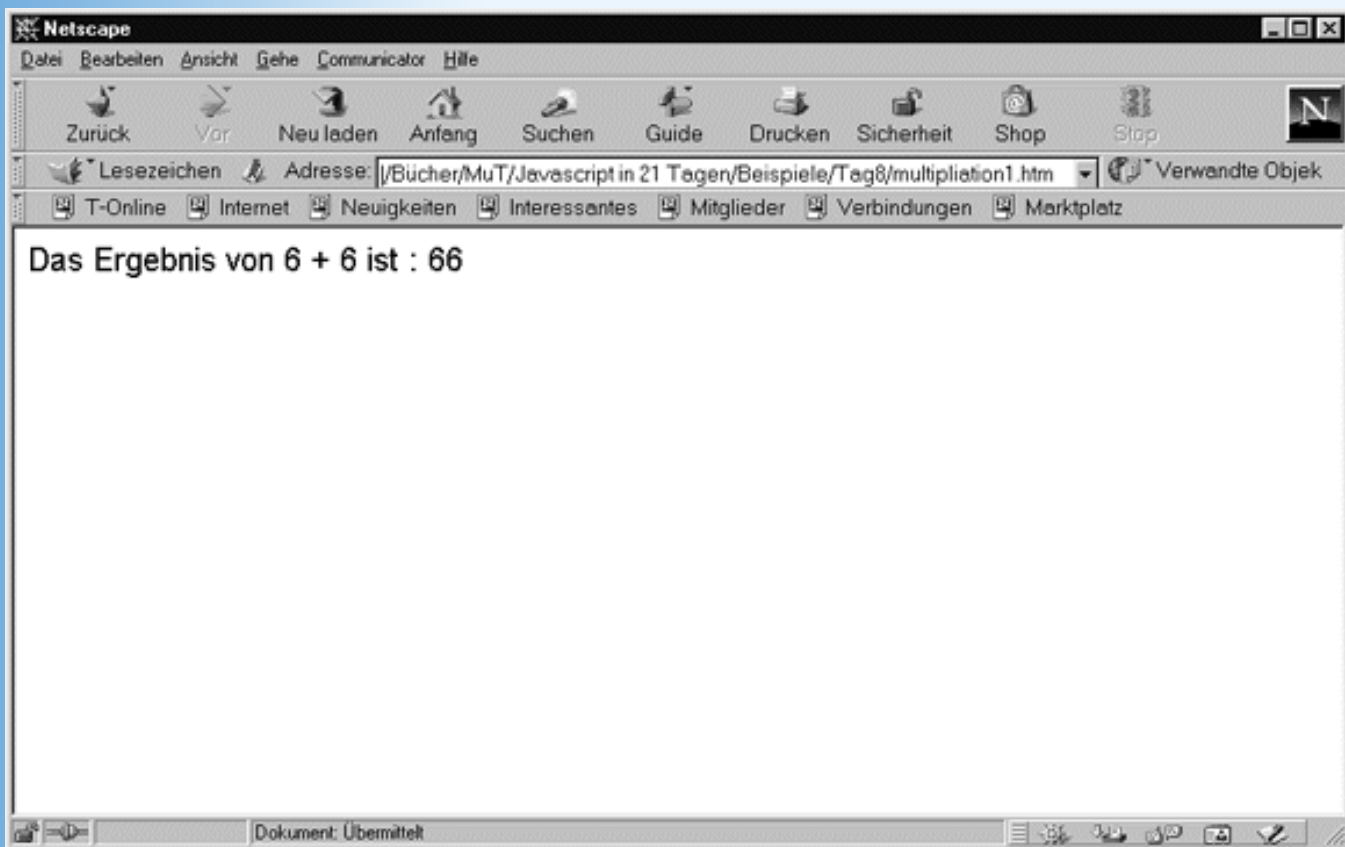


Abbildung 8.11: Seit wann ist $6 + 6 = 66$?

Die beiden Eingaben werden nicht numerisch addiert, sondern als Texte zusammengesetzt. Grund dafür ist, dass die Anwendereingaben von der `prompt()`-Methode als Strings zurückgegeben werden. Strings können aber mit dem `+`-Operator verbunden werden. Es wird einfach Zeichen für Zeichen aneinander gehängt. Das Verfahren haben wir uns schon mehrfach bei Beispielen zunutze gemacht. Hier aber ist uns dieses Verhalten im Weg. Mit dem nachfolgenden Trick können wir JavaScript zwingen, die automatische Konvertierung vor der Verbindung mit dem `+`-Operator auszuführen und die Benutzereingaben in Zahlen umzuwandeln. Die Zeile

```
document.write(a + b);
```

wird dabei durch

```
document.write(a*1 + b*1);
```

ersetzt. Die Multiplikation mit eins verändert nicht den Wert einer numerischen Eingabe, erzwingt aber die Ad-hoc-Typkonvertierung (ein String, der nur Zahlen beinhaltet und mit einer Zahl multipliziert wird, wird zu einer Zahl). Beachten Sie, dass wir Punkt-vor-Strich-Rechnung ausnutzen. Testen wir diese Verhaltensweise nun mit dem bereits angedeuteten Verhalten an einer Zahl mit führenden Nullen, wo durch bestimmte Situationen diese Informationen verschwinden können.

Beispiel 7:

Erstellen Sie die nachfolgende Datei:

```
<html>
<body>
<SCRIPT LANGUAGE="JavaScript">
var a, b;
a = prompt(
"Geben Sie die Telefonnummer mit Vorwahl ein","");
b = a*1;
document.write("Die Telefonnummer ist ", a);
document.write("<br>");
document.write(
"Die Telefonnummer multipliziert mit 1 ist ", b);
```

```
document.write("<br>");  
</SCRIPT>  
</body>  
</html>
```

Speichern Sie den Text bitte unter dem Dateinamen Telefon.htm ab und schauen Sie sich die Datei im Browser an. Wenn Sie eine Telefonnummer mit führenden Nullen eingeben, wird diese Information bei der zweiten Zeile der Ausgabe nicht mehr zur Verfügung stehen.

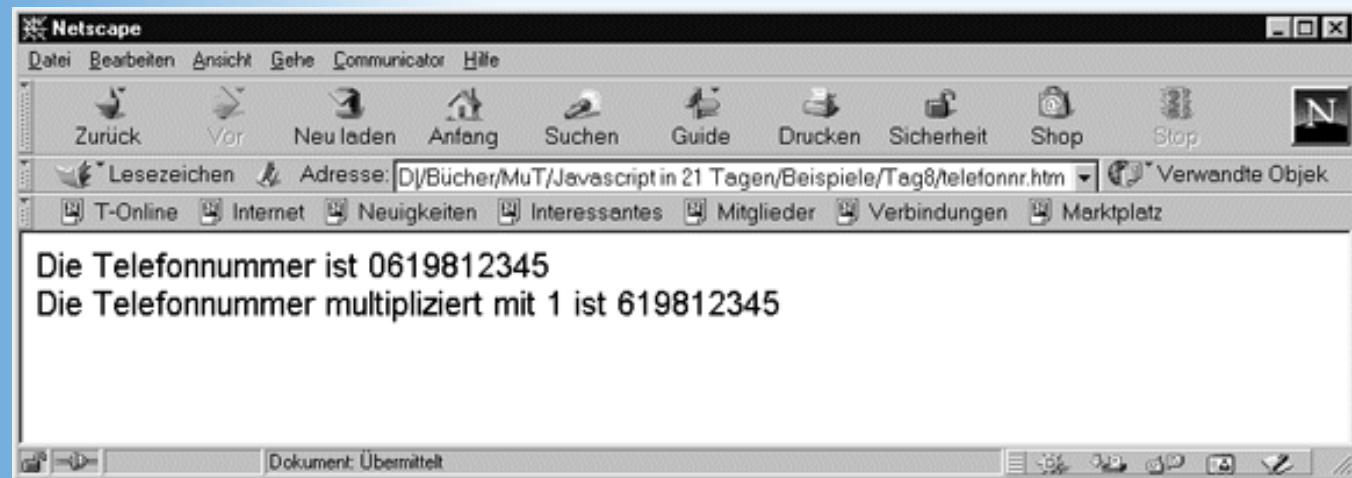


Abbildung 8.12: Informationsverlust



Obwohl das Beispiel sehr konstruiert ist, kann eine solche Situation durchaus in der Praxis auftreten.

Nehmen wir uns nun noch ein Beispiel vor, in dem ein und derselben Variablen nacheinander Werte verschiedener Datentypen zugewiesen werden.

Beispiel 8:

Erstellen Sie die nachfolgende Datei:

```
<html>  
<body>  
<SCRIPT language="JavaScript">  
var a;  
a = 1;  
document.write("Wert von a: ", a);  
document.write("<br>");  
a = 1.2345;  
document.write("Wert von a: ", a);  
document.write("<br>");  
a = "Gude wie"2;  
document.write("Wert von a: ", a);  
document.write("<br>");  
a = new Date();  
document.write("Wert von a: ", a);  
document.write("<br>");  
a = true;  
document.write("Wert von a: ", a);  
document.write("<br>");  
</SCRIPT>  
</body>  
</html>
```

Speichern Sie den Text bitte unter dem Dateinamen `einevar.htm` ab und schauen Sie sich die Datei im Browser an. Die Variable nimmt jegliche Werte auf - gleich welchen Datentyps.



Abbildung 8.13: Eine für alles

Nehmen wir uns nun noch ein Beispiel vor, in dem über bestimmte Operationen Variablen miteinander verbunden werden, und verfolgen wir, was dann passiert.

Beispiel 9:

Erstellen Sie die nachfolgende Datei:

```
<html>
<body>
<SCRIPT language="JavaScript">
var a = 42;
var b = 3.14;
var meinText1 = "Am Brunnen vor "
var meinText2 = "dem Tore."
document.write("Wert von a + b : ", a + b );
document.write("<br>");
document.write("Wert von meinText1 + meinText2 : ",
meinText1 + meinText2);
document.write("<br>");
document.write("Wert von a - b :", a - b );
document.write("<br>");
document.write("Wert von meinText1 - meinText2 : ",
meinText1 - meinText2);
document.write("<br>");
document.write("Wert von a * b :", a * b );
document.write("<br>");
document.write("Wert von meinText1 * meinText2 : ",
meinText1 * meinText2);
document.write("<br>");
document.write("Wert von a / b :", a / b );
document.write("<br>");
document.write("Wert von meinText1 / meinText2 : ",
meinText1 / meinText2);
document.write("<br>");
document.write("Wert von meinText1 + a : ",
meinText1 + a);
document.write("<br>");
```

```
document.write("Wert von meinText1 - a : ",  
meinText1 - a);  
document.write("<br>");  
document.write("Wert von meinText1 * a : ",  
meinText1 * a);  
document.write("<br>");  
document.write("Wert von meinText1 / a : ",  
meinText1 / a);  
document.write("<br>");  
</SCRIPT>  
</body>  
</html>
```

Speichern Sie den Text bitte unter dem Dateinamen Datentypen3.htm ab und schauen Sie sich die Datei im Browser an. Sie haben sicherlich gleich vermutet, dass bei dem Beispiel einige Operationen nicht funktionieren. Das ist immer dann der Fall, wenn der Wert NaN (Not a Number) als Ergebnis herauskommt. Dies ist übrigens ein wohldefinierter Wert, den man sinnvoll nutzen kann, wie wir noch sehen werden.

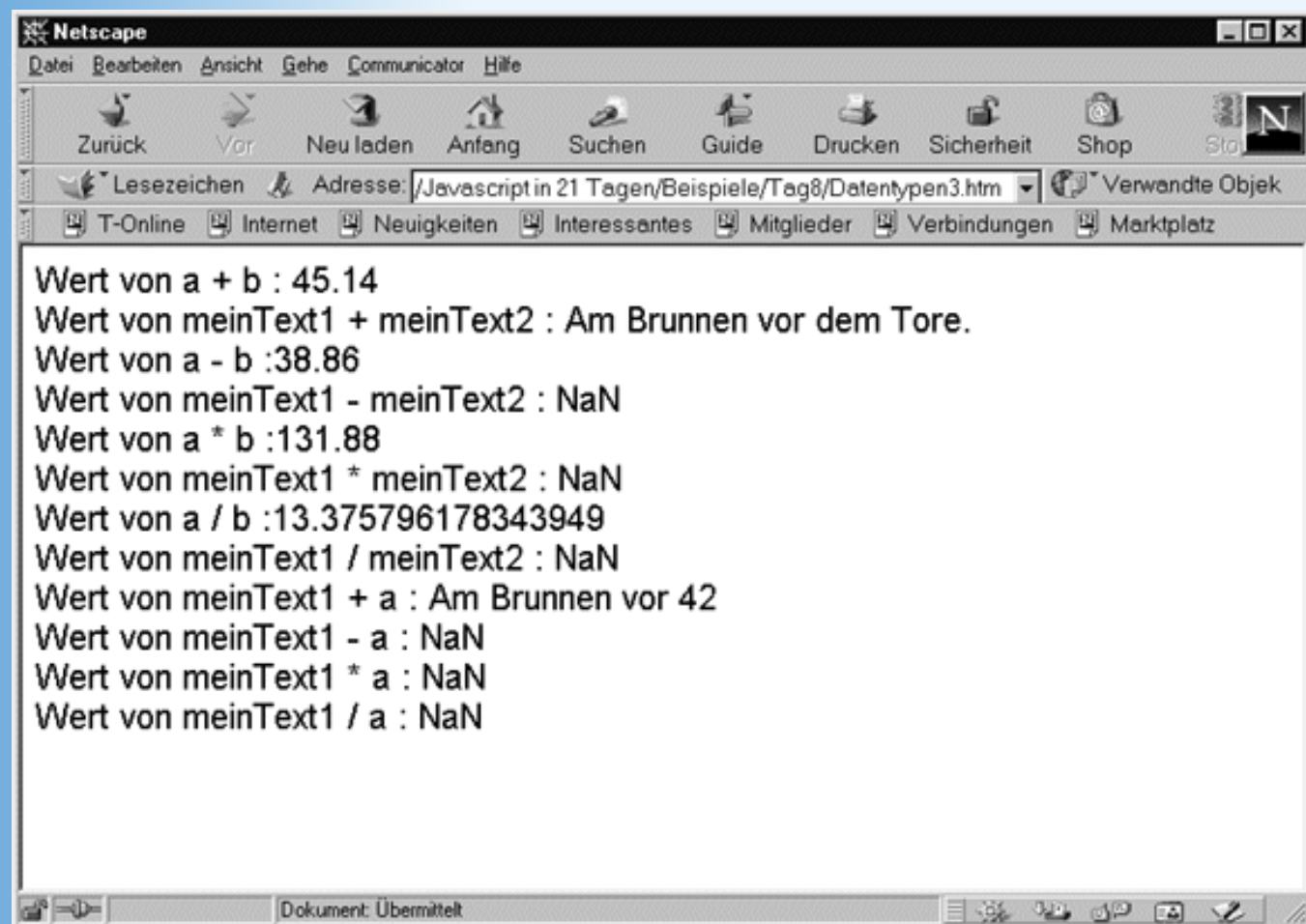


Abbildung 8.14: Manches geht, aber nicht alles

In dem Beispiel werden vier Variablen verknüpft. Es gibt eine Ganzzahl, eine Gleitkommazahl und zwei Strings. Die beiden Zahlen lassen sich ohne wesentliche Probleme miteinander verknüpfen. Es findet jedoch insoweit immer eine Typkonvertierung statt, als dass als Ergebnis immer eine Gleitkommazahl herauskommt (also ein Datentyp, der mehr Platz im Speicher braucht - sowohl für den Vorkommateil als auch den Nachkommateil). Dies ist zwingend, damit keine Information verloren geht. Nur wenn zwei Ganzzahlen miteinander verknüpft würden, würde wieder eine Ganzzahl herauskommen.

Die beiden Texte lassen sich nur so miteinander verbinden, dass die Zeichen mit + einfach aneinander gehängt werden. Alle anderen Verbindungen machen keinen Sinn. Auch Text und Zahl lassen sich nur mit dem +-Operator in dem Sinn miteinander verbinden, dass die Zahl zuerst automatisch in einen String konvertiert und dann einfach angehängt wird.

Variablenvarianten

Bei Variablen unterscheidet man von der Konzeption verschiedene Arten, wobei der Ort, wo sie vorhanden sind, die Variante bestimmt. Grundsätzlich gibt es:

- Instanzvariablen
- Klassenvariablen
- Lokale Variablen
- Globale Variablen



Das Thema Klassenvariablen wollen wir nicht weiter verfolgen. Nur soweit eine kleine Erklärung: Im Rahmen der objektorientierten Programmierung werden ähnliche Objekte zusammengefasst, um eine leichtere Klassifizierung von diesen Objekten zu ermöglichen. Die Eigenschaften der Objekte werden in übergeordneten Gruppierungen - den so genannten Klassen oder Objektdeklarationen - als eine Gruppe zusammenhängender Informationen und Methoden gesammelt und für eine spätere Erzeugung von realen Objekten verwendet. Klassen können allgemein als die Bauvorschriften von Objekten verstanden werden. Klassenvariablen sind solche Variablen, die innerhalb einer Klasse definiert werden und in allen Instanzen der Klasse (den konkreten Objekten) zur Verfügung stehen. Wir werden auf dieses Thema am Tag 15 genauer eingehen.

Instanzvariablen



Eine Instanz einer Klasse bzw. einer Objektdeklaration ist ein anderer Begriff für das aus der Klasse erzeugte Objekt. Man kann beide Begriffe synonym verwenden, nur benutzt man Instanz meist dann, wenn man noch einen deutlichen Hinweis auf die erzeugende Klasse bzw. Objektdeklaration mit einschließen will. Man redet also beispielsweise von dem Array-Objekt test oder von test als Instanz der Klasse Array. Das ist aber nicht zwingend.

Instanzvariablen beziehungsweise Objektvariablen werden zum Definieren von Attributen eines bestimmten Objekts benutzt. Es handelt sich also bei Eigenschaften eines Objekts um nichts anderes als Instanzvariablen. Angesprochen werden sie ausschließlich über die Punktnotation. Dabei steht bei Bedarf auf der linken Seite des Punktes das Objekt, zu dem die Variable gehört, und auf der rechten Seite der Name der Variablen. Instanzvariablen sind nur innerhalb eines Objekts vorhanden. Ein paar Beispiele kennen Sie schon, auch wenn wir bisher nicht von Instanzvariablen gesprochen haben. Etwa status, welche zum Objekt navigator gehört und den Inhalt der Statuszeile beinhaltet. Oder die Eigenschaften des screen-Objekts - width und height. In der dritten Woche werden wir uns genauer mit dem Thema auseinandersetzen.

Globale Variablen

Globale Variablen werden außerhalb von Funktionen/Prozeduren/ Methoden/Objekten deklariert und stehen für alle Bestandteile eines Scripts zur Verfügung. Sie können damit zum Datenaustausch zwischen Funktionen/Prozeduren oder der Initialisierung von Werten genutzt werden.

Die beiden nachfolgenden Beispiele greifen auf Formulare zurück, die wir an späterer Stelle noch intensiver angehen. Hier sollen nur Teilaspekte davon verwendet werden.

Beispiel 10:

Erstellen Sie die nachfolgende Datei:

```
<HTML>
<HEAD>
```



```
<SCRIPT language="JavaScript">
var zaehler=0;
function klick()
{
  zaehler++;
  alert("Sie haben " + zaehler + " mal geklickt");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT type="Button" value="Klick" onClick="klick()">
</FORM>
</BODY>
</HTML>
```

Speichern Sie den Text bitte unter dem Dateinamen global1.htm ab und laden Sie die Datei in den Browser. Zuerst wird die Variable `zaehler` global deklariert und mit dem Wert 0 vorbelegt. Die Funktion `klick()` wird jedes Mal aufgerufen, wenn in dem Formular auf den Button geklickt wird. Sie erzeugt eine Meldung, in der der Wert der Variablen ausgegeben wird. Diese Variable wird bei jedem Aufruf der Funktion um eins hochgezählt. Dies erzwingt aber, dass der vorherige Wert noch bekannt ist, was nur die globale Deklaration der Variablen gewährleistet (siehe Abbildung 8.15).

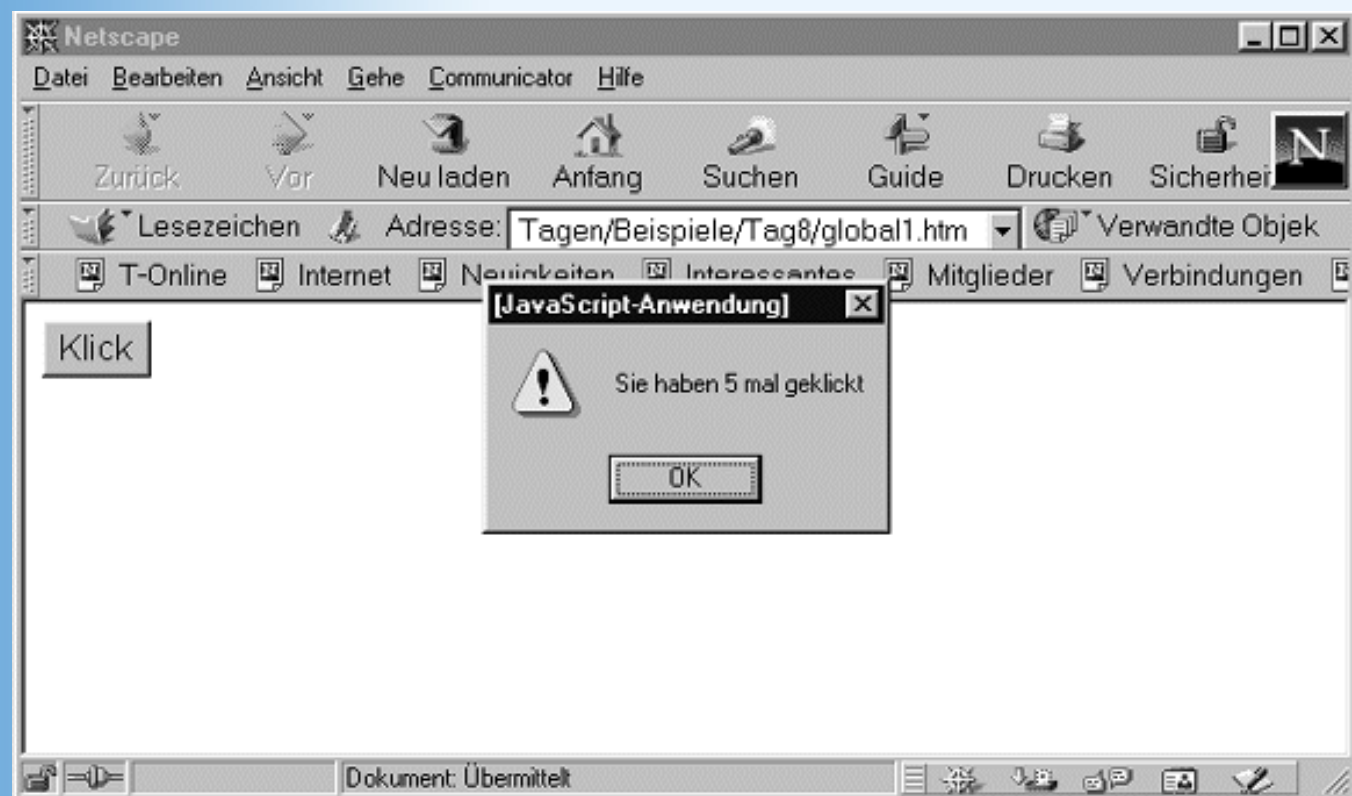


Abbildung 8.15: Die Klicks werden mitgezählt

Beispiel 11:

Ein weiteres Beispiel tauscht über globale Variablen Werte aus, die ein Anwender in verschiedenen Formularen eingegeben hat. Das erste Formular wird direkt aus HTML-Anweisungen beim Laden der Seite aufgebaut. Der Anwender kann seinen Namen, Vornamen, eine Produktnummer und eine Anzahl eingeben.

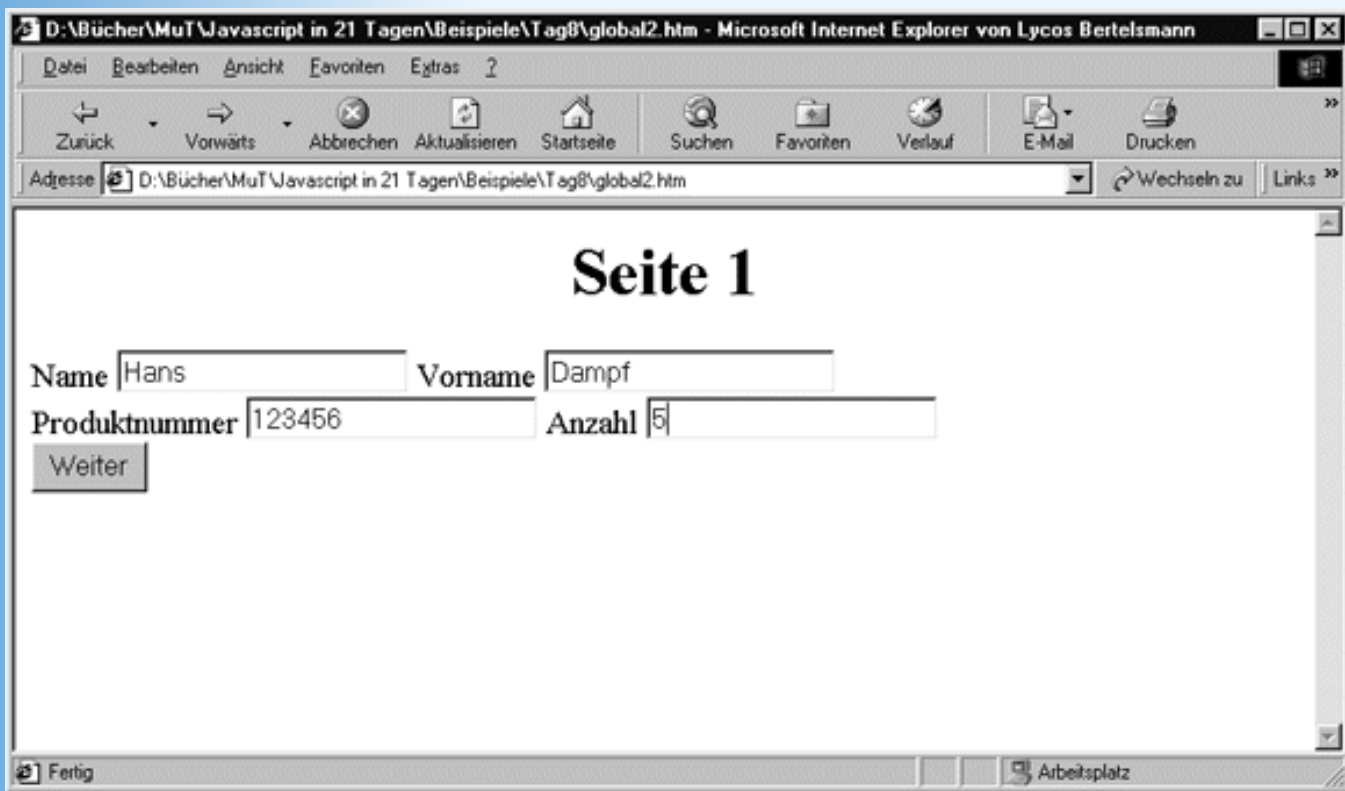


Abbildung 8.16: Die Eingaben im ersten Formular

Diese Eingaben sollen nun in eine neue Webseite samt neuem Formular übernommen werden. Das nachfolgende, etwas umfangreichere Beispiel leistet dies unter Verwendung einer dynamisch neu geschriebenen Seite und der Datenweitergabe mittels globaler Variablen:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var pn=0;
var anz=0;
var name="";
var vorname="";

function schreibeNeuesFormular()
{
document.write("<H1 align=center>Seite 2</H1>");
document.write("<P align=center>");
document.write(
"Produktbestellung für ", vorname, " ", name, ".");
document.write("<BR>Ist die Bestellung so korrekt?");
document.write("<P align=left>");
document.write("<FORM>");
document.write("Produktnummer <INPUT> ");
document.write("Anzahl <INPUT><BR> ");
document.write("Kommentar <INPUT><BR> ");
document.write(
"Formular absenden <INPUT type='Submit' value='OK'>");
document.write("</FORM>");
fuelleForm2();
}

function fuelleForm2()
{
document.forms[0].elements[0].value=pn;
```

```
document.forms[0].elements[1].value=anz;
}

function weiter()
{
name=document.forms[0].elements[0].value;
vorname=document.forms[0].elements[1].value;
pn=document.forms[0].elements[2].value;
anz=document.forms[0].elements[3].value;
schreibeNeuesFormular();
}
</SCRIPT>
</HEAD>
<BODY>
<H1 align=center>Seite 1</H1>
<FORM>
Name <INPUT>
Vorname <INPUT>
<BR>
Produktnummer <INPUT>
Anzahl <INPUT>
<BR>
<INPUT type="Button" value="Weiter" onClick="weiter()">
</FORM>
</BODY>
</HTML>
```

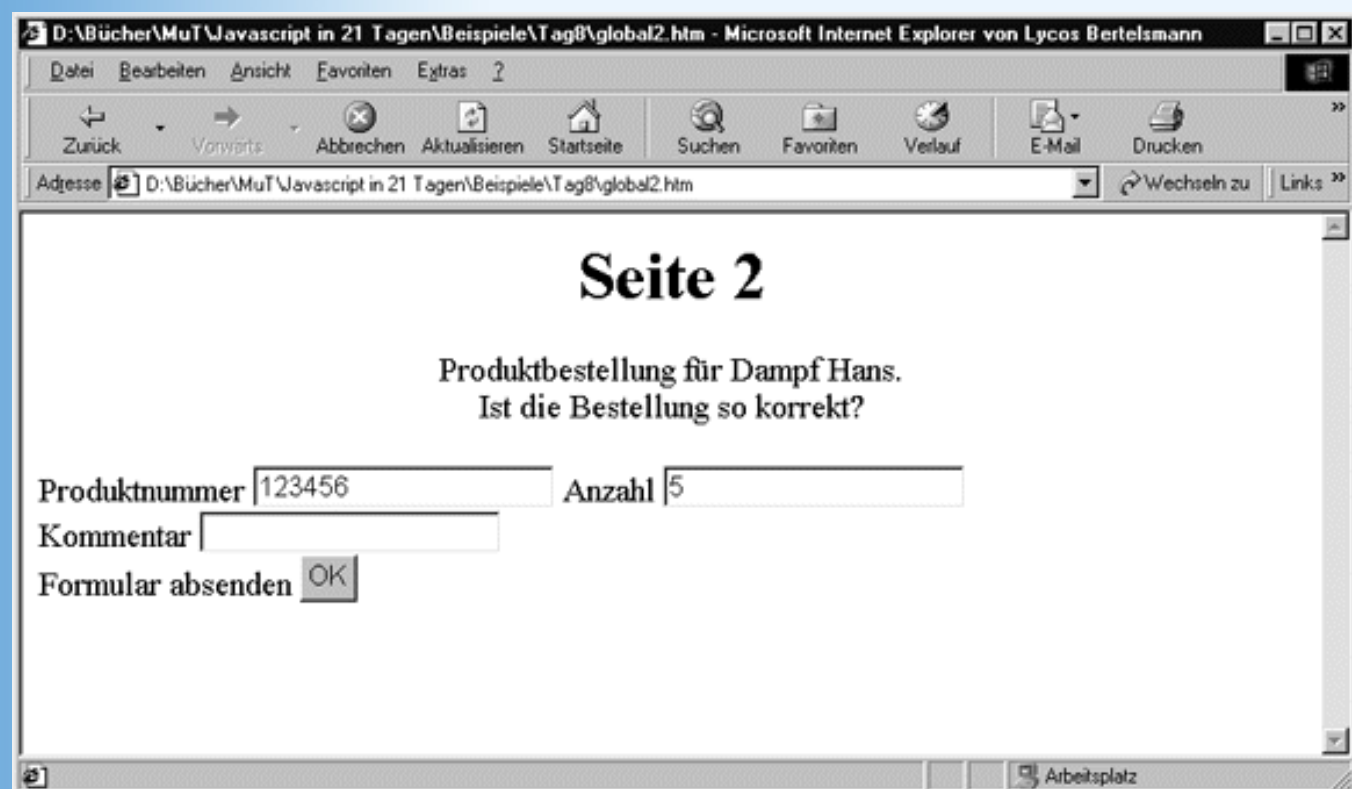


Abbildung 8.17: Die Eingaben im Formular werden in die neue Webseite und das zweite Formular übernommen

Testen Sie das Beispiel bitte im Internet Explorer und beachten Sie die Warnung nach der Erläuterung.

Zuerst wird rein über HTML ein Formular mit vier Eingabefeldern und einem Button generiert. Das JavaScript verwendet vier globale Variablen, die mit Defaultwerten initialisiert werden:

```
var pn=0;
```

```
var anz=0;  
var name="";  
var vorname="";
```

Diese speichern die Eingaben, welche der Anwender macht. Die Zuweisung erfolgt in der Funktion weiter(), die beim Klick auf den Button aufgerufen wird. Die nachfolgenden vier Zeilen belegen die jeweiligen globalen Variablen mit dem Wert, den der Anwender jeweils einträgt (auf die Details gehen wir bei der Behandlung von Formularen ein):

```
name=document.forms[0].elements[0].value;  
vorname=document.forms[0].elements[1].value;  
pn=document.forms[0].elements[2].value;  
anz=document.forms[0].elements[3].value;
```

Die Funktion ruft dann die zweite Funktion schreibeNeuesFormular() auf. Diese generiert dynamisch eine neue Webseite mit normalen HTML- Anweisungen und einem neuen Formular. Interessant ist erst einmal die Zeile

```
document.write(  
"Produktbestellung für ", vorname, " ", name, ".");
```

Dort werden zwei der globalen Variablen verwendet und deren Wert in den Text der Webseite geschrieben. Danach wird das Formular (noch ohne Vorbelegung) geschrieben.

Bevor die Seite angezeigt wird, wird die Funktion fuelleForm2() aufgerufen. Diese belegt mit den beiden Zeilen

```
document.forms[0].elements[0].value=pn;  
document.forms[0].elements[1].value=anz;
```

das erste und das zweite Feld des neuen Formulars vor. Dazu werden die anderen beiden globalen Variablen verwendet.



Der Netscape Navigator hat in einigen Versionen ein Problem mit der auf obige Weise deklarierten globalen Variablen. Dies ist wieder eine ähnliche Situation wie an Tag 6 in Beispiel 1. Der Navigator kommt mit diesem Beispiel zurecht, wenn lokale Variablen in Form von Übergabeparametern verwendet werden. Den Workaround finden Sie als zweites Beispiel bei den jetzt behandelten lokalen Variablen.

Lokale Variablen

Lokale Variablen werden innerhalb von Methoden-, Funktions- oder Prozedurdefinitionen deklariert und können auch nur dort benutzt werden. Dazu gibt es den Fall der schleifenlokalen Variablen, die nur im Rahmen einer Schleife definiert sind. Lokale Variablen existieren nur so lange im Speicher, wie die Methode, die Funktion, die Prozedur oder der Block existiert.

Die Verwendung von lokalen Variablen hat einige Gründe:

- Wenn eine Information nur temporär benötigt wird (etwa so lange eine aufgerufene Funktion abgearbeitet wird), muss man auch nur für diese Zeit Hauptspeicher belegen. Würde man mit ständig vorhandenen Variablen (den nachfolgend besprochenen globalen Variablen) arbeiten, würde der Hauptspeicher während der Laufzeit des gesamten Scripts blockiert, obwohl er in der meisten Zeiten nicht benutzt werden würde.
- Ein Variablenbezeichner kann in einem Script mehrfach verwendet werden. Das findet man oft bei schleifenlokalen Zählvariablen. Für diese hat sich eingebürgert, i oder j als Bezeichner zu verwenden.
- Das Script wird übersichtlicher und strukturierter.

Grundsätzlich ist die Verwendung von lokalen Variablen der Verwendung von globalen Variablen vorzuziehen, wenn es ohne Einschränkung möglich ist. Testen wir die Verwendung einer lokalen Variablen.

Beispiel 12:

Erstellen Sie die nachfolgende Datei:

```

<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function intern()
{
var a = 42;
alert("Intern: " + a);
}
function lade()
{
var a = 21;
alert("Aussen: " + a);
intern();
alert("Aussen 2: " + a);
}
</SCRIPT>
</HEAD>
<BODY onLoad=lade(>
</BODY>
</HTML>

```

Speichern Sie den Text bitte unter dem Dateinamen LokaleVar.htm ab und laden Sie die Datei in den Browser. Zuerst wird die Variable a mit dem Wert 21 gesetzt und dann angezeigt. Anschließend wird eine Funktion intern() aufgerufen, welche eine Variable gleichen Namens lokal definiert und mit dem Wert 42 belegt. Die Ausgabe beweist dies. Nachfolgend wird in der Funktion lade() erneut der Wert von a ausgegeben. Er ist unverändert, d.h. die Variable a in der Funktion intern() ist nur lokal auf die Funktion beschränkt gültig und hat die Variable in der Funktion lade() unberührt gelassen.

Beispiel 13:

Das nachfolgende Listing löst das Dilemma, das manche Varianten von dem Netscape Navigator mit Beispiel 11 haben. Dabei wird auf lokale Variablen zurückgegriffen. Insbesondere ist hier von Bedeutung, dass Parameter an Funktionen übergeben werden.

```

<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function schreibeNeuesFormular(pn, anz, name, vorname)
{
document.write("<H1 align=center>Seite 2</H1>");
document.write("<P align=center>");
document.write(
"Produktbestellung für ", vorname, " ", name, ".");
document.write("<BR>Ist die Bestellung so korrekt?");
document.write("<P align=left>");
document.write("<FORM>");
document.write("Produktnummer <INPUT> ");
document.write("Anzahl <INPUT><BR> ");
document.write("Kommentar <INPUT><BR> ");
document.write(
"Formular absenden <INPUT type='Submit' value='OK'>");
document.write("</FORM>");
document.forms[0].elements[0].value=pn;
document.forms[0].elements[1].value=anz;
}
function weiter()
{
name=document.forms[0].elements[0].value;

```



```
vorname=document.forms[0].elements[1].value;
pn=document.forms[0].elements[2].value;
anz=document.forms[0].elements[3].value;
schreibeNeuesFormular(pn,anz,name,vorname);
}
</SCRIPT>
</HEAD>
<BODY>
<H1 align=center>Seite 1</H1>
<FORM>
Name <INPUT>
Vorname <INPUT>
<BR>
Produktnummer <INPUT>
Anzahl <INPUT>
<BR>
<INPUT type="Button" value="Weiter" onClick="weiter()">
</FORM>
</BODY>
</HTML>
```

Beachten Sie die Zeile

```
function schreibeNeuesFormular(pn, anz, name, vorname)
```

wo in den Klammern vier Übergabewerte angegeben werden. Wenn die Funktion aufgerufen wird, müssen für diese Variablen (die innerhalb der Funktion über diese hier angegebenen Bezeichner verwendet werden können) in den Klammern vier Werte oder Variablen angegeben werden. Das sieht im Beispiel so aus:

```
schreibeNeuesFormular(pn,anz,name,vorname);
```

Die Namen der Übergabewerte im Aufruf der Funktion müssen nicht (!) übereinstimmen, obwohl es in diesem Beispiel der Fall ist.



Selbstverständlich gibt es unter JavaScript Datenfelder (Arrays) jeglichen Datentyps. Ein Array ist eine Sammlung von Variablen, die alle über einen Bezeichner und einen in eckigen Klammern notierten Index (bei 0 beginnend) angesprochen werden können. Etwa forms[0] bis forms[4] oder elements[1]. Auf Details zu diesem Thema werden wir aber erst an Tag 15 bei den Grundlagen zu Objekten eingehen.

Zusammenfassung

Wir haben am heutigen Tag die Technik von Funktionen, Prozeduren und Methoden genauer erläutert. Sowohl was die Gründe dafür sind, wo die Unterschiede zu sehen sind als auch, wie Funktionen zu erstellen und einzusetzen sind.

Der zweite Schwerpunkt waren Variablen und Datentypen, die in JavaScript für den Anwender recht simpel zu handhaben sind. Obwohl auch JavaScript Datentypen kennt und einsetzt, verwaltet JavaScript diese sehr oft im Hintergrund. Nur im Fall von Situationen, wo das Ergebnis nicht dem gewünschten Zustand entspricht, muss ein Programmierer manuell eingreifen.

Fragen und Antworten

Frage:

Es finden sich immer wieder Hinweise im Buch, dass JavaScript in vielen Bereichen sehr lockere Syntaxregeln bevorzugt. Warum ist das so?

Antwort:

Grundsätzlich gilt in der Tat, dass damit Einsteigern die Programmierung erheblich erleichtert wird. Die Sprache könnte nicht ein so großes Zielpublikum erreichen, wenn man die Strenge von Programmiersprachen wie Java anlegen würde.

Frage:

Kann man mehrere Variablen auch in einer Zeile deklarieren?

Antwort:

Ja. Dabei werden die Variablen einfach per Kommata getrennt:

```
var a, b, c, d;
```

Frage:

Kann man mehrere Variablen auch in einer Zeile deklarieren und gleichzeitig jeweils einen Wert zuweisen?

Antwort:

Nein, nicht in dem Sinn wie oben. Da man aber mehrere Anweisungen durch Semikolon getrennt in eine Textzeile schreiben kann, geht es in gewissem Sinn schon.

Workshop

Erstellen Sie eigene Funktionen und experimentieren Sie damit herum. Als Basis können Ihnen die Beispiele in diesem Kapitel dienen. Eine Sache sollten Sie auf jeden Fall ausprobieren. Wenn Sie in Beispiel 10 die Variable `zaehler` nicht außerhalb der Funktion deklarieren, bekommen Sie mehrere Probleme. Zum einen wird jeder gesetzte Wert beim Beenden der Funktion wieder »vergessen«, zum anderen muss aber die Variable vor ihrer ersten Verwendung initialisiert werden. Letzteres bedeutet, es wird immer der gleiche Wert angezeigt. Experimentieren Sie damit herum.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Ist das eine korrekte Variablendeklaration?

```
var i, j, k = 8, z;
```

2. Stimmt die Deklaration der Funktion?

```
function 2mal()  
{  
  alert("Guckmal");  
}
```

3. Stimmt die Deklaration der Funktion?

```
function test();  
{  
  alert("Guckmal");  
}
```

4. Was passt bei dem Funktions-Aufruf nicht?

```
test
```

5. Kann das stimmen?

```
function test()  
{  
  return;
```

}

❖Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

1 Wobei man das Zusammenfassen von einer Anweisung sehr abstrakt sehen sollte.

2 Für Nicht-Hessen: Ein typischer Gruß, mit dem sich diese Landsleute überall erkennen ;).

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 2

Tag 9

Schlüsselworte, Operatoren und Wertzuweisungen

Am Tag 9 werden wir uns weiter mit Syntaxgrundlagen von JavaScript beschäftigen. Nachdem wir gestern

- Variablen und Datentypen,
- Methoden und Objekt-Eigenschaften und
- Funktionen und Prozeduren

besprochen haben, soll heute auf die diese Themen erweiternden Punkte

- Wertzuweisungen,
- Operatoren und
- Schlüsselworte

eingegangen werden.

JavaScript-Schlüsselworte

Jede Programmier- und Script-Sprache besitzt bestimmte Worte, die ein essentieller Teil der Sprachdefinition sind. Das sind Worte, die bereits eine feste Bedeutung haben. Etwa unter JavaScript den Begriff `var`, welcher bei der Variablendeklaration verwendet wird. Oder `function` zur Einleitung einer Funktionsdeklaration. Wenn solche Worte als Bezeichner für Variablen oder Funktionen verwendet werden, kann der Interpreter nicht zwischen dem Bezeichner und dem Schlüsselwort unterscheiden. Wenn also eines der reservierten Schlüsselworte als Bezeichner verwendet wird, führt dies beim ersten Laden des Scripts zu einem Fehler im Interpreter. Testen wir das einfach mal.

Beispiel 1

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var function=8;
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter schluesselwortFehler.htm und laden Sie es in einen Browser.

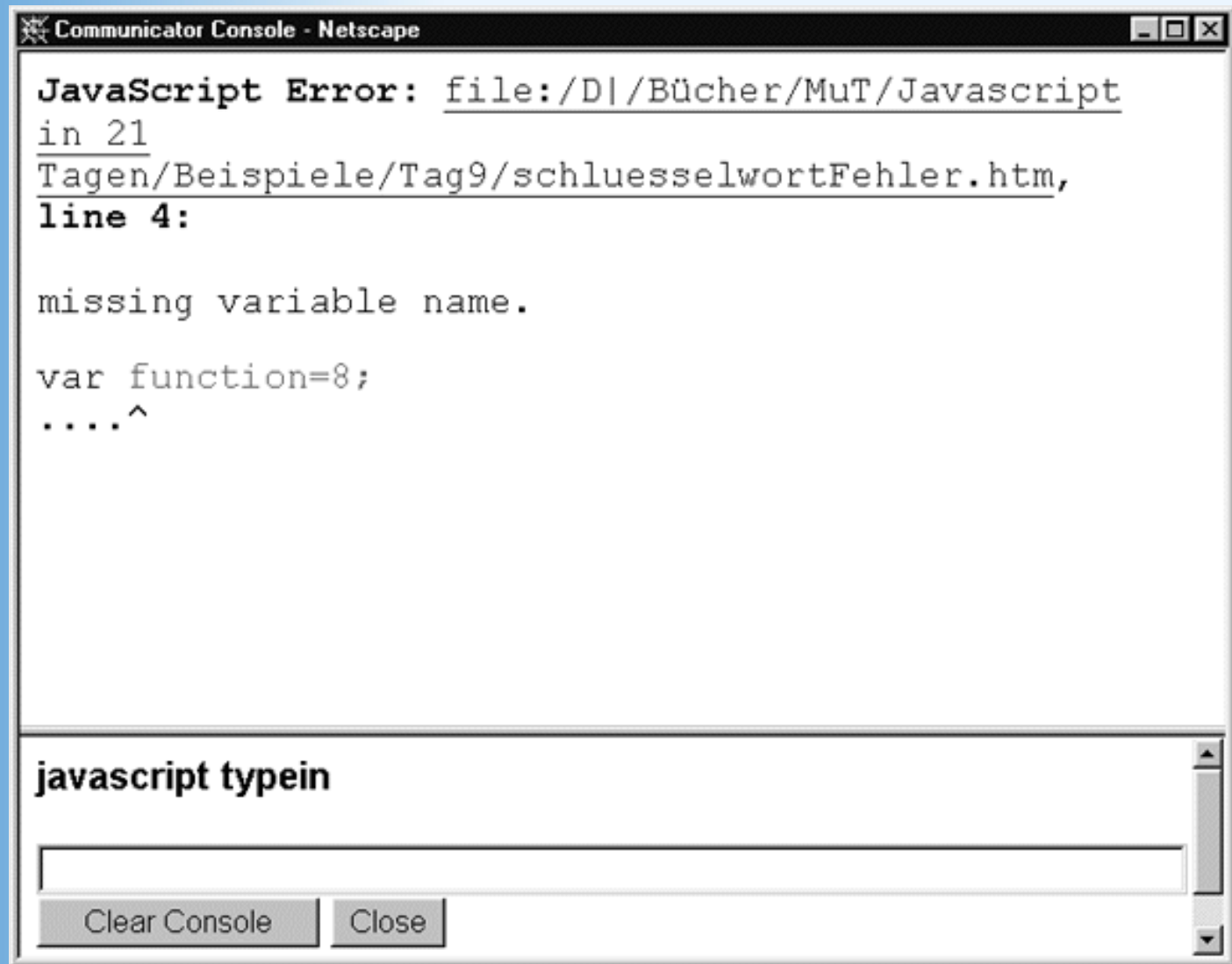


Abbildung 9.1: So was geht nicht

JavaScript hat sowohl aktuell benutzte Schlüsselworte reserviert, aber auch bestimmte Token für zukünftige Sprachvarianten, um potentielle Inkompatibilitäten für die Zukunft zu vermeiden.



Unter einem Token kann man einen Sinnzusammenhang verstehen, der aus im Quelltext vorkommenden Zeichen gebildet wird, wenn der Quelltext in ausführbare Befehle übersetzt werden soll. In der Situation muss zunächst geklärt werden, welche einzelnen Zeichen und Symbole oder zusammengeschrriebene Zeichen- und Symbolgruppen im Code welche Bedeutung haben. Wenn der Quellcode vom interpretierenden System analysiert wird, wird er von dem System dabei in einzelne kleine Bestandteile (Token) zerlegt. Ein Token ist dabei entweder ein bestimmtes einzelnes Zeichen/Symbol oder meistens ein aus mehreren Zeichen zusammengesetzter Begriff. Das interpretierende System versteht ein gültiges Zeichen oder die Sammlung von Zeichen. Erkennt das System das spezifische Zeichen oder die Ansammlung von Zeichen, wird daraus ein Sinnzusammenhang, dem eine bestimmte Aktion durch das System folgt. Sonst bleibt das spezifische Zeichen oder die Ansammlung von Zeichen einfach nur die Summe der Buchstaben oder Zeichen. Die Sprachelemente werden von vielen Systemen bei der Zerlegung in Token bereits auf ihre Richtigkeit geprüft.

Tabelle der JavaScript-Schlüsselworte

Schlüsselwort	Beschreibung
abstract	reserviert
boolean	reserviert
break	Abbruch in Schleifen
byte	reserviert
case	Fallunterscheidungen
catch	reserviert
char	reserviert
class	reserviert
const	reserviert
continue	Fortsetzung in Schleifen
debugger	reserviert
default	Fallunterscheidungen
delete	Löschen eines Array-Elements oder einer selbstdefinierten Objekteigenschaft
do	Beginn einer Erweiterung der while-Schleife (do-while)
double	reserviert
else	Einleitung des alternativen Blocks in einer if-Schleife
enum	reserviert
export	Objekte oder Funktionen für fremde Scripts ausführbar machen
extends	reserviert
false	der Wert »falsch«
final	reserviert
finally	reserviert
float	reserviert
for	Einleitung von for-Schleifen
function	Einleitung von Funktionen
goto	reserviert
if	Einleitung von if-Schleifen
implements	reserviert
import	Objekte oder Funktionen eines fremden Scripts importieren
in	bedingte Anweisungen in if-Schleifen
instanceof	reserviert
int	reserviert
interface	reserviert
long	reserviert
native	reserviert
new	Definition von Objekten
null	reserviert

package	reserviert
private	reserviert
protected	reserviert
public	reserviert
return	Übergabe eines Rückgabewertes in Funktionen
short	reserviert
static	reserviert
super	reserviert
switch	Fallunterscheidung
synchronized	reserviert
this	Bezug auf die aktuelle Instanz eines Objekts
throw	reserviert
throws	reserviert
transient	reserviert
true	der Wert »wahr«
try	reserviert
typeof	Typ eines Elements
var	Definition einer Variablen
void	leerer Funktionstyp
volatile	reserviert
while	Einleitung einer while-Schleife
with	erlaubt, mehrere Anweisungen mit einem Objekt durchzuführen

Tabelle 9.1: JavaScript-Schlüsselworte (JavaScript 1.3)

Wir werden im Laufe des Buches die meisten der hier aufgelisteten und derzeit in Verwendung befindlichen Schlüsselworte in der Praxis sehen.

Operatoren und Operanden

Ein wichtiger Begriff in der Programmierung ist *Ausdruck*. Einziges wichtiges Kennzeichen eines Ausdrucks ist, dass er einen Wert besitzt. Im einfachsten Fall besteht ein Ausdruck nur aus einem Literal oder einer Variablen. In diesem trivialen Fall ist der Wert des Ausdrucks identisch mit dem Literal oder dem Wert der Variablen. Es gibt aber auch zusammengesetzte Ausdrücke. In diesem Fall werden Literale und/oder Variablen miteinander verknüpft. Der Wert des Ausdrucks ist dann das Ergebnis der Verknüpfung. Die Verknüpfung selbst erfolgt über so genannte *Operatoren*. Operatoren sind besondere Zeichen oder Zeichenkombinationen, die eine auszuführende Handlung mit einer oder mehreren Variablen oder konstanten Werten ausführen. Diese werden dabei *Operanden* genannt.

JavaScript kennt drei Formen von Operatoren:

- Unäre Operatoren. Dies sind Operatoren, die mit nur einem Operanden arbeiten. Diese Operatoren können entweder vor dem Operanden stehen (so genannte Präfix-Operatoren) oder hinter dem Operanden (Postfix-Operatoren).
- Binäre Operatoren arbeiten mit zwei Operanden. Dabei steht in JavaScript der Operator immer zwischen den Operanden (die so genannte Infixschreibweise).

- Ternäre Operatoren. Dies ist ein Operator mit drei Operanden. JavaScript kennt einen solchen.

In der Regel werden Operatoren in mehrere Kategorien eingeteilt.

Arithmetische Operatoren

Arithmetische Operatoren benutzen zwei numerische Operanden (Zahlen oder Variablen mit numerischem Inhalt) und verbinden diese so, dass daraus ein neuer Wert wird. Es gibt in JavaScript folgende arithmetischen Operatoren (die wir schon aus Beispielen kennen):

Operator	Beschreibung
+	Additionsoperator
-	Subtraktionsoperator
*	Multiplikationsoperator
/	Divisionsoperator
%	Modulo-Operator

Tabelle 9.2: Arithmetische Operatoren

Testen wir diese Operatoren in einem Beispiel. Dabei wird insbesondere der Modulo-Operator von Interesse sein. Er gibt den Rest einer Division zurück und zählt bei vielen anspruchsvolleren Techniken (Verschlüsselungsverfahren, Grafikberechnungen, technischen Scripten) zu den dringend notwendigen Hilfsmitteln.

Beispiel 2:

Erstellen Sie die Datei ArithOp.htm:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
document.write("1 + 3 = ", 1+3);
document.write("<BR>");
document.write("2 * 3 = ", 2*3);
document.write("<BR>");
document.write("5 / 3 = ", 5/3);
document.write("<BR>");
document.write("3 - 1 = ", 3-1);
document.write("<BR>");
document.write("10 % 3 = ", 10%3);
document.write("<BR>");
document.write("10 % 4 = ", 10%4);
document.write("<BR>");
document.write("11 % 4 = ", 11%4);
document.write("<BR>");
document.write("12 % 3 = ", 12%3);
document.write("<BR>");
document.write("19 % 7 = ", 19%7);
document.write("<BR>");
document.write("5 % 3 = ", 5%3);
document.write("<HR>");
document.write("Beachten Sie, was 5 / 0 gibt: ", 5/0);
</SCRIPT>
</HEAD>
```

<BODY>
</BODY>
</HTML>

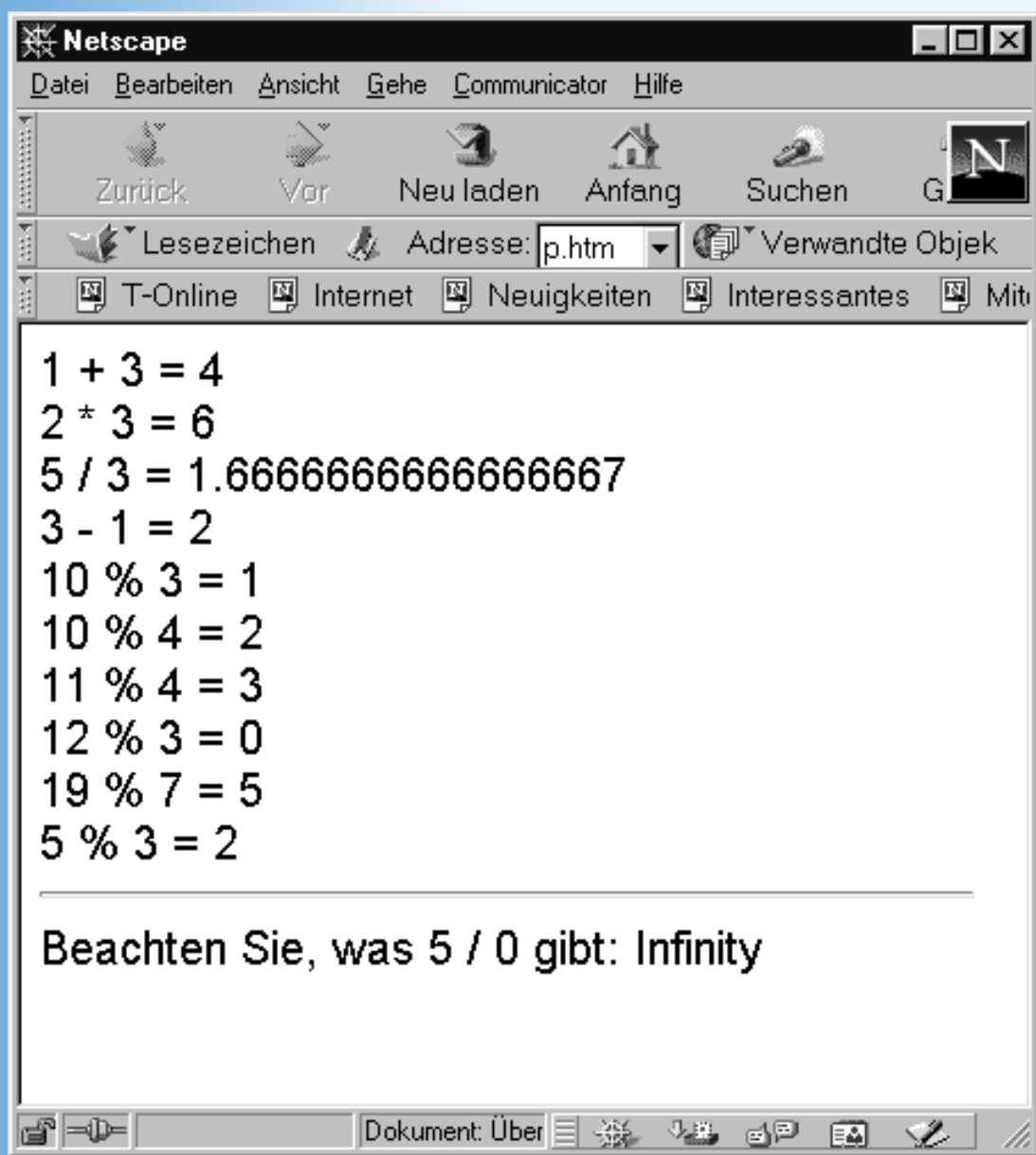


Abbildung 9.2: **Verschiedene arithmetische Operationen**



Beachten Sie, dass bei arithmetischen Operationen, welche mathematisch unendlich ergeben, ein wohldefinierter Wert, Infinity, zurückgegeben wird. Diesen Wert kann man mit geeigneten JavaScript-Funktionen sinnvoll nutzen.

Einstellige arithmetische Operatoren sind Spezialfälle der gerade genannten arithmetische Operatoren. Es gibt davon zwei. Sie verwenden nur einem Operanden, der dem Operator nachgestellt wird:

Operator	Beschreibung
-	Einstellige arithmetische Negierung, welche die arithmetische Vorzeichenumdrehung ihres nachfolgenden numerischen Operanden ergibt.
+	Der einstellige Operator + ist nur aus Symmetriegründen vorhanden.

Tabelle 9.3: Einstellige arithmetische Operatoren

Zuweisungsoperatoren und Wertzuweisung

Zuweisungsoperatoren sind die Grundlage jeder Wertzuweisung. Sie weisen das Ergebnis einer auf der rechten Seite stehenden Operation (damit ist auch die einfache Notation eines konstanten Werts gemeint) der linken Seite des Ausdrucks zu. In unseren bisherigen Wertzuweisungen haben wir immer den direkten Zuweisungsoperator = verwendet.

Bevor wir neue Zuweisungsoperatoren kennen lernen, wollen wir eine Wertzuweisung besprechen, die Einsteigern immer wieder Kopfzerbrechen bereitet. Was bedeutet die folgende Zeile, wenn a eine Variable ist?

```
a = a + 1;
```

Mathematisch gesehen kann das doch nicht stimmen, oder? Wenn man entsprechend der mathematischen Logik die Variable a durch Zahlen ersetzt, würde das

```
1 = 2;  
2 = 3;  
41 = 42;  
2000 = 2001;
```

usw. bedeuten. Selbst ich als Mathematiker könnte Ihnen das wohl schwer verkaufen ;-). Nur - diese Zeile darf so nicht gelesen werden. Es handelt sich um eine Wertzuweisung, die in zwei aufeinander folgende, logische Schritte zerlegt werden muss. Teilen wir die Zeile

```
a = a + 1;
```

in zwei Zeilen mit zwei Variablen auf:

```
b = a + 1;  
a = b;
```

Diese beiden Zeilen sind so zu lesen:

Erster Schritt: Weise der Variablen b den Wert a + 1 zu.

Zweiter Schritt: Weise der Variablen a den Wert b zu.

Was hat a danach für einen Wert? Eins mehr als vorher. Wenn nun in der zweiten Zeile b durch a ersetzt wird, was b zu dem Zeitpunkt für einen Wert hat (a + 1), steht da genau unsere mathematisch unsinnige Zeile.

Man muss also die Zeile

```
a = a + 1;
```

so lesen: Zuerst wird die rechte Seite berechnet und dann der Variablen auf der linken Seite zugewiesen. Wie die Variable auf der linken Seite konkret heißt, spielt keine Rolle. Sie bekommt nach der Auswertung der rechten Seite einfach einen neuen Wert zugewiesen.



Die Möglichkeit einer solchen Selbstzuweisung ist der vielleicht wichtigste Baustein für eine effektive Programmierung. Sie werden so etwas an unzähligen Stellen finden (gerade in Schleifen). Dabei kann diese Technik in Verbindung mit sämtlichen arithmetischen Operatoren und natürlich auch Zahlen ungleich eins verwendet werden. In der hier besprochenen Form oder über die nachfolgend noch besprochenen Wege.

Testen wir diese Geschichte mit einem Beispiel:

Beispiel 3:

Erstellen Sie die Datei Selbst.htm:

```
<HTML>  
<HEAD>  
<SCRIPT language="JavaScript">  
var a=0;  
a = a + 1;  
document.write(a);  
document.write("<BR>");  
a = a + 1;  
document.write(a);  
document.write("<BR>");  
a = a + 1;  
document.write(a);  
document.write("<BR>");  
a = a + 1;  
document.write(a);  
document.write("<BR>");  
a = a + 1;  
document.write(a);  
document.write("<BR>");  
a = a + 1;  
document.write(a);  
document.write("<BR>");  
a = a + 1;  
document.write(a);  
document.write("<BR>");  
a = a + 1;  
document.write(a);  
document.write("<BR>");  
</SCRIPT>  
</HEAD>  
<BODY>  
</BODY>  
</HTML>
```

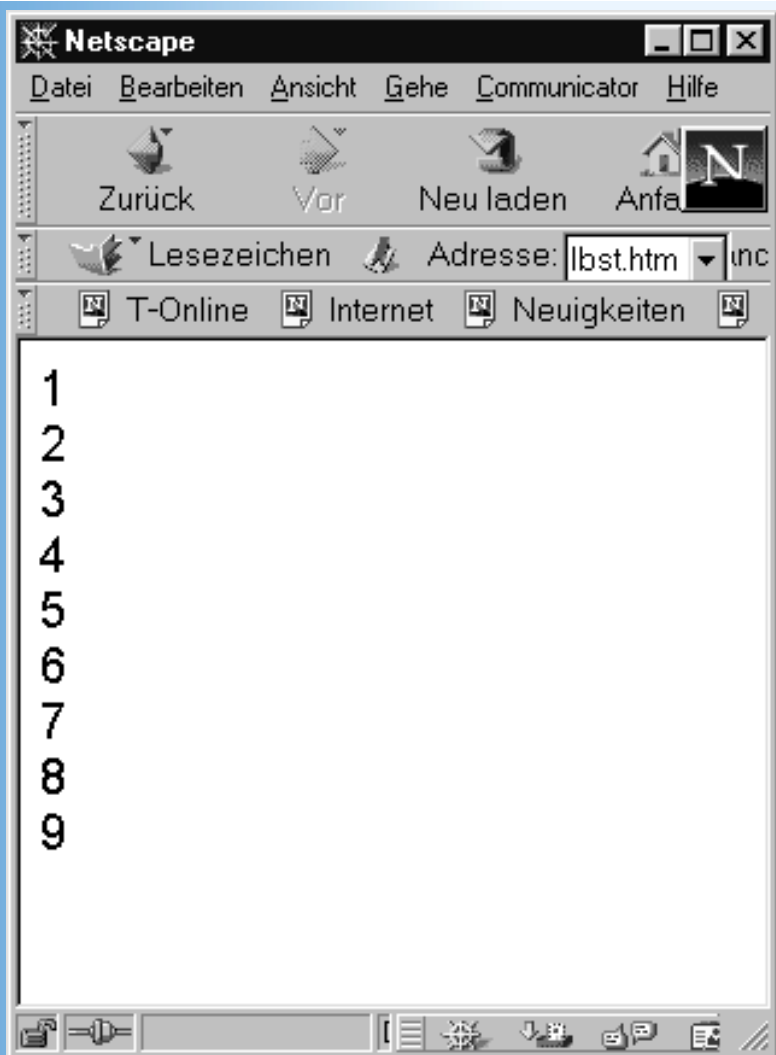


Abbildung 9.3: Immer eins mehr

Neben dem normalen Zuweisungsoperator = gibt es in JavaScript die arithmetischen Zuweisungsoperatoren. Es handelt sich nur um die verkürzte Schreibweise einer arithmetischen Zuweisung. Wie auch die arithmetischen Operatoren können sie sowohl mit ganzen Zahlen als auch mit Fließkommazahlen verwendet werden. Es gibt folgende arithmetische Zuweisungsoperatoren:

Operator	Beschreibung
+=	Additions-Zuweisungsoperator
-=	Subtraktions-Zuweisungsoperator
*=	Multiplikations-Zuweisungsoperator
/=	Divisions-Zuweisungsoperator
%=	Modulo-Zuweisungsoperator

Tabelle 9.4: Arithmetische Zuweisungsoperatoren

Das ist dann folgendermaßen zu verstehen:

a += 2; entspricht a = a + 2;

und

a *= 5; entspricht a = a * 5;

Beispiel 4:

Erstellen Sie die Datei ArithZuOp.htm:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var a=4;
document.write(a);
document.write("<BR>");
a +=1;
document.write(a);
document.write("<BR>");
a -=2;
document.write(a);
document.write("<BR>");
a *=3;
document.write(a);
document.write("<BR>");
a /=2;
document.write(a);
document.write("<BR>");
a %=4;
document.write(a);
document.write("<BR>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

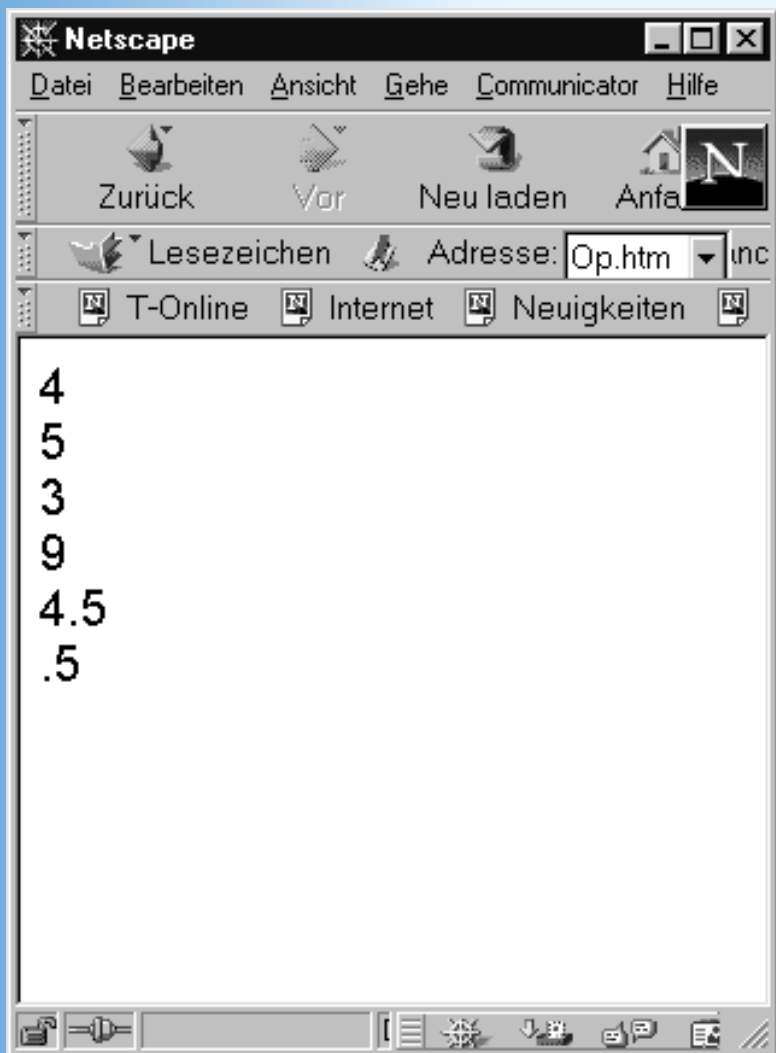


Abbildung 9.4: Verschiedene arithmetische Zuweisungsaktionen

Inkrement-/Dekrement-Operatoren

In engem Bezug zu Wertzuweisungen stehen die so genannten Inkrement-/ Dekrement-Operatoren. Diese werden zum Auf- und Abwerten eines einzelnen Wertes verwendet. Inkrement- und Dekrement-Operatoren sind einstellige Operatoren und werden nur in Verbindung mit einem ganzzahligen oder einem Fließkomma-Operanden benutzt.

Der Inkrement-Operator ++ erhöht den Wert des Operanden um 1. Die Syntax

```
++zaehler;
```

entspricht

```
zaehler = zaehler + 1;
```

oder auch

```
zaehler+=1;
```

Es bedeutet also, dass der Wert der nachstehenden Variablen um eins erhöht und dann der Variablen wieder zugewiesen wird.

Es gibt aber auch die Syntax, wo der Operator der Variablen nachgestellt wird. Die Reihenfolge von Operand und Operator ist relevant. Wenn der Operator vor dem Operanden steht, erfolgt die Erhöhung des Wertes, bevor der Wert dem Operanden zugewiesen wird. Wenn er hinter dem Operanden steht, erfolgt die Erhöhung, nachdem der Wert bereits zugewiesen wurde. Wir testen dies mit einem Beispiel:

Beispiel 5:

Erstellen Sie die Datei Dekrement.htm:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var a=4;
document.write(a);
document.write("<BR>");
document.write(a++);
document.write("<BR>");
document.write(a);
document.write("<BR>");
document.write(++a);
document.write("<BR>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

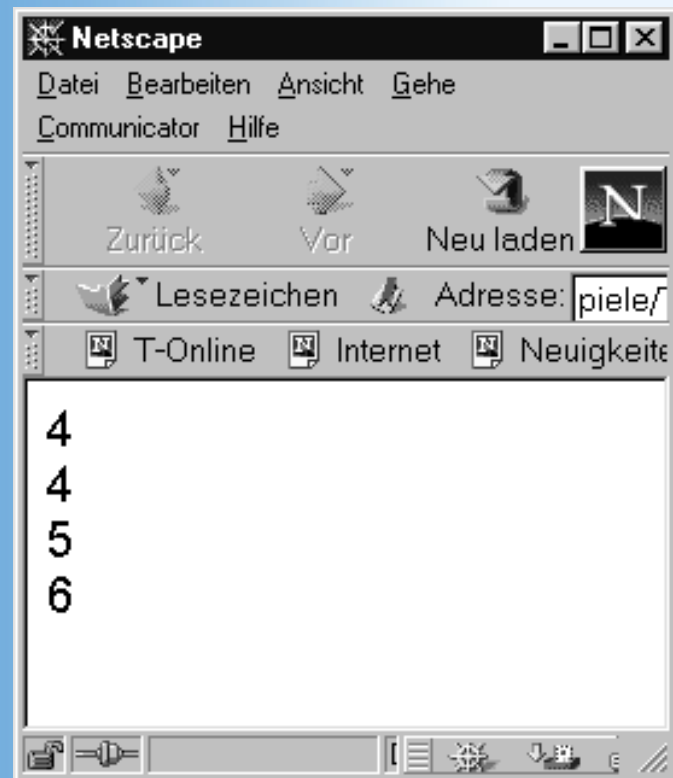


Abbildung 9.5: Optisch unscheinbar, aber alles Notwendige ist sichtbar

Das Beispiel nimmt eine Variable, weist einen Wert zu und gibt ihn aus. Die Zeile

```
document.write(a++);
```

erhöht zwar den Wert der Variablen, aber erst, nachdem (!) die Ausgabe erfolgt ist. Das bedeutet, der Wert wird erst erhöht, nachdem der Wert bereits zugewiesen wurde. Die nachfolgende Ausgabe beweist die Erhöhung, denn es wurde keine weitere Veränderung der Variablen vorgenommen. Die Zeile

```
document.write(++a);
```

zeigt, dass das Voranstellen des Operators den Wert vor der Ausgabe erhöht. Der Wert wurde also erhöht und dann erst zugewiesen.

Der Dekrement-Operator -- ist der Gegenspieler vom Inkrement- Operator. Er erniedrigt den Wert des Operanden um 1. Ansonsten ist alles identisch zum Inkrement-Operator. Die Reihenfolge von Operand und Operator ist auch

hier relevant. Wenn der Operator vor dem Operanden steht, erfolgt die Erniedrigung des Wertes, bevor der Wert dem Operanden zugewiesen wird. Wenn er hinter dem Operanden steht, erfolgt die Erniedrigung, nachdem der Wert bereits zugewiesen wurde.



Der Dekrementoperator ist die kritische Zeichenkombination, welche im Netscape Navigator in einigen Versionen Konflikte mit dem Ende des HTML-Kommentars auslöst. Der Netscape Navigator verwechselt das Ende des HTML-Kommentars beim Einschluss von Scripten in HTML-Kommentare mit diesem Operator. Für den Navigator muss das Ende-Zeichen des HTML-Kommentars vor dem Script-Interpreter mit JavaScript-Kommentarzeichen versteckt werden. Die Anweisung vor dem Ende des <SCRIPT>-Containers sieht deshalb zur Sicherheit so aus:

```
//-->
```

Das recht diffizile Verhalten des Operators macht den Umgang damit nicht ganz ungefährlich. Hauptanwendung finden die beiden Operatoren in Kontrollstrukturen und Schleifen. Dort werden die Operatoren meist so eingesetzt, dass das unterschiedliche Verhalten bei Voran- und Nachstellen keine Rolle spielt.

Vergleichsoperatoren

Vergleichsoperatoren gehören zu den wichtigsten Operatoren. Sie verwenden zwei Operanden und vergleichen diese. JavaScript kennt die nachfolgenden Vergleichsoperatoren.

Operator	Beschreibung
==	Gleichheitsoperator
!=	Ungleichheitsoperator
<	Kleiner-als-Operator
>	Größer-als-Operator
<=	Kleiner-als-oder-gleich-Operator
>=	Größer-als-oder-gleich-Operator

Tabelle 9.5: Vergleichsoperatoren

Die Operatoren werden in Verbindung mit Kontrollflussanweisungen verwendet. Wir kennen beispielsweise die if-Anweisung und wollen diese in einem Beispiel einsetzen.

Beispiel 6:

Erstellen Sie die Datei vergleichsop.htm:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function test()
{
var a=4;
if (a>4)
{
document.write(a);
document.write("<BR>");
}
else
```

```
{
document.write("Zu klein");
document.write("<BR>");
}
if (a>=4)
{
document.write("Ok");
document.write("<BR>");
}
else
{
document.write("Zu klein");
document.write("<BR>");
}
if (a<=4)
{
document.write("Ok");
document.write("<BR>");
}
else
{
document.write("Zu klein");
document.write("<BR>");
}
if (a<4)
{
document.write("Ok");
document.write("<BR>");
}
else
{
document.write("Zu gross");
document.write("<BR>");
}
if (a==4)
{
document.write("Ok");
document.write("<BR>");
}
else
{
document.write("Zu klein");
document.write("<BR>");
}
if (a!=4)
{
document.write("Ok");
document.write("<BR>");
}
else
{
document.write("Nit fier");
document.write("<BR>");
}
```

```

}
</SCRIPT>
</HEAD>
<BODY onLoad="test()">
</BODY>
</HTML>

```



Abbildung 9.6: Die Ergebnisse der Vergleiche

Das Beispiel ist recht einfach zu lesen und die Ausgabe demonstriert die Wirkung der Vergleichsoperatoren.



Beachten Sie, dass der Gleichheitsoperator == nicht mit dem Zuweisungsoperator = identisch ist. Es ist ein oft gemachter Fehler, bei einem Vergleich den Zuweisungsoperator zu notieren. Einige Sprachen wie Visual Basic oder VBScript verwenden deshalb nur das einfache Gleichheitszeichen für beide Konstellationen. Eine Vereinfachung, die wie jedwede Schwächung von strengen Syntaxregeln ihren Preis hat. Es kann dadurch zu ungewollten Wertzuweisungen kommen, wo eigentlich nur ein Vergleich geplant ist. Wer mit beiden Varianten ausreichend Programmiererfahrung hat, wird den Einsatz von unterschiedlichen Operatoren trotz der Probleme mit dem Verwechseln als die bessere Lösung schätzen.

Logische Vergleichsoperatoren

Die logischen Vergleichsoperatoren werden nur auf boolsche (Wahrheits-) Operanden angewandt und erzeugen auch nur boolsche Ergebnisse. Das bedeutet, es kann entweder true oder false als Wert erzeugt werden. JavaScript kennt folgende logische Vergleichsoperatoren:

Operator	Beschreibung
&&	logischer AND-Operator

	logischer OR-Operator
!	logischer NOT-Operator

Tabelle 9.6: Logische Vergleichsoperatoren

Einsatz finden solche Operatoren beispielsweise bei der Verknüpfung von verschiedenen Bedingungen (der NOT-Operator wird aber auch einzeln verwendet), welche in einer Kontrollstruktur gemeinsam bewertet werden müssen. Etwa in der if-Schleife, die wir schon mehrfach verwendet haben.

Beispiel 7:

Erstellen Sie die Datei LogVerglOp.htm. Die entscheidenden Stellen sind so eingerückt, dass die Abhängigkeiten gekennzeichnet werden:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function schreibeAntwort(sum,anz)
{
document.write("<H1 align=center>Provision</H1>");
document.write("<P align=left>");
if(
    (sum > 100000)
    &&
    (anz > 3)
    )
{
document.write("Schöne Einzelleistung. Sie erhalten 5% Provision");
document.write("<BR>Das sind DM " + (sum * 0.05));
document.write("<BR>");
}
if(
    (sum <= 100000)
    ||
    (anz < 4)
    )
{
document.write("Das ist nicht befriedigend. Sie erhalten nur 2% Provision");
document.write("<BR>Das sind DM " + (sum * 0.02));
document.write("<BR>");
}

}
function weiter()
{
vsum=document.forms[0].elements[0].value;
vanz=document.forms[0].elements[1].value;
schreibeAntwort(vsum,vanz);
}
</SCRIPT>
</HEAD>
<BODY>
<H1 align=center>Umsatzzahlen Versicherungen</H1>
<H3 align=center>Eingabe der abgeschlossenen Versicherungssumme</H3>
```

```
<H3 align=center>und</H3>
```

```
<H3 align=center>der Anzahl der Verträge</H3>
```

```
<FORM>
```

```
Versicherungssumme <INPUT>
```

```
<BR>
```

```
Anzahl der Verträge <INPUT>
```

```
<BR>
```

```
<INPUT type="Button" value="Weiter" onClick="weiter()">
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```

Das Beispiel generiert wieder ein Formular (wie wir am Tag 8 bereits vorgeführt haben). Die Funktionalität soll folgende sein: Ein Versicherungsvertreter gibt in dem Formular ein, wie viele Verträge er abgeschlossen hat und über welche Gesamtversicherungssumme.

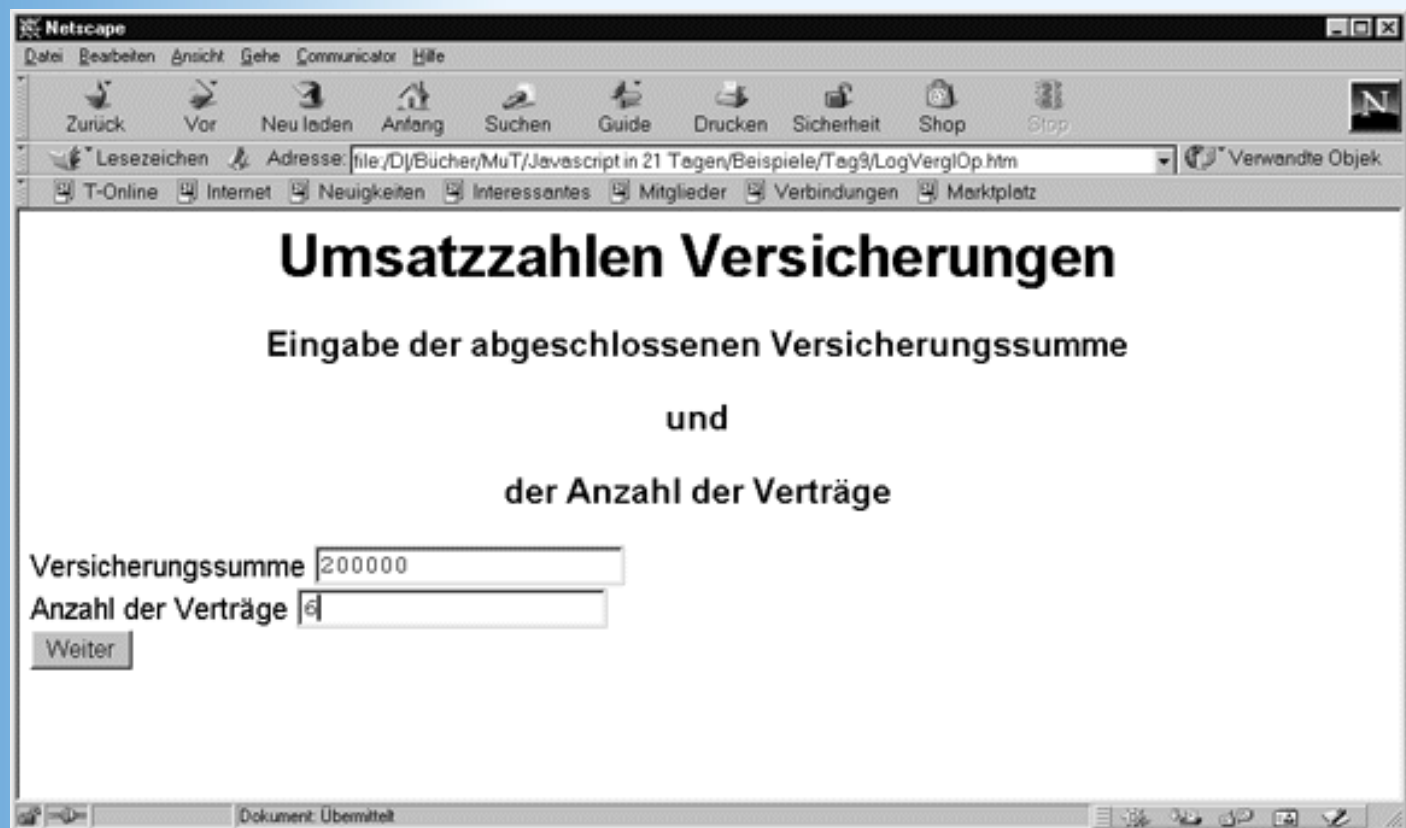


Abbildung 9.7: Die Eingaben mit beiden Bedingungen erfüllt

Nach der Bestätigung der Eingaben durch Klick auf die Schaltfläche wird eine neue Seite geschrieben. Dort werden die logischen Vergleichsoperatoren verwendet. Betrachten wir die entsprechenden Zeilen. Die if-Anweisung

```
if(
    (sum > 100000)
    &&
    (anz > 3)
)
```

kontrolliert, ob sowohl die Versicherungssumme größer als 100000 ist als auch die Anzahl der Verträge größer als 3. Nur wenn beide Bedingungen erfüllt sind, wird der nachfolgende Block ausgeführt.

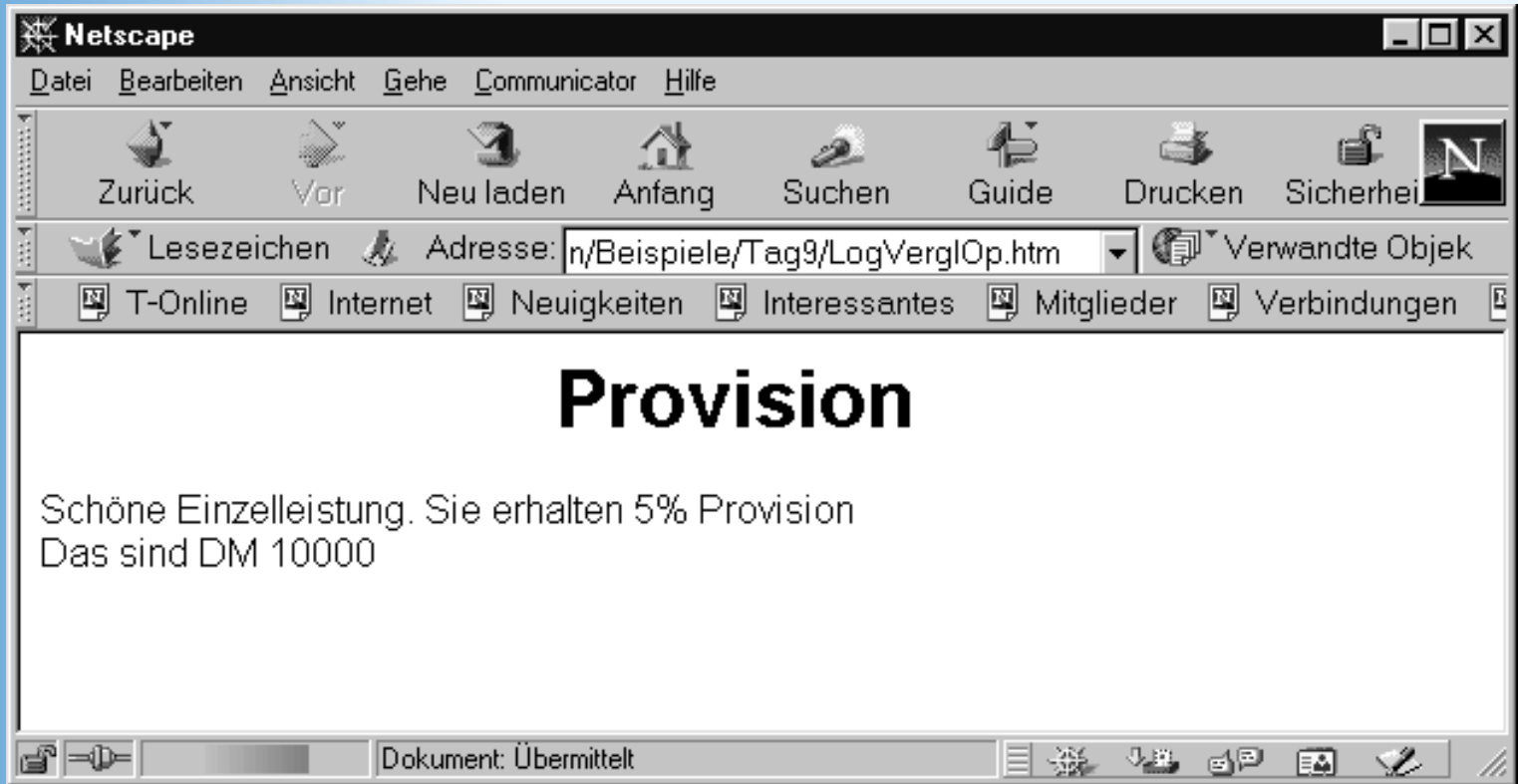


Abbildung 9.8: Das Resultat



Beachten Sie, dass die Summe und die Anzahl der Verträge als Übergabeparameter an die Funktion übergeben werden. Dabei wird bewusst beim Aufruf jeweils ein anderer Name für die Variable gewählt als für die lokal in der Funktion benutzte Variable. Erinnern Sie sich - die Namen könnten gleich sein, müssen es aber nicht. Es sind ja lokal definierte Variablen.

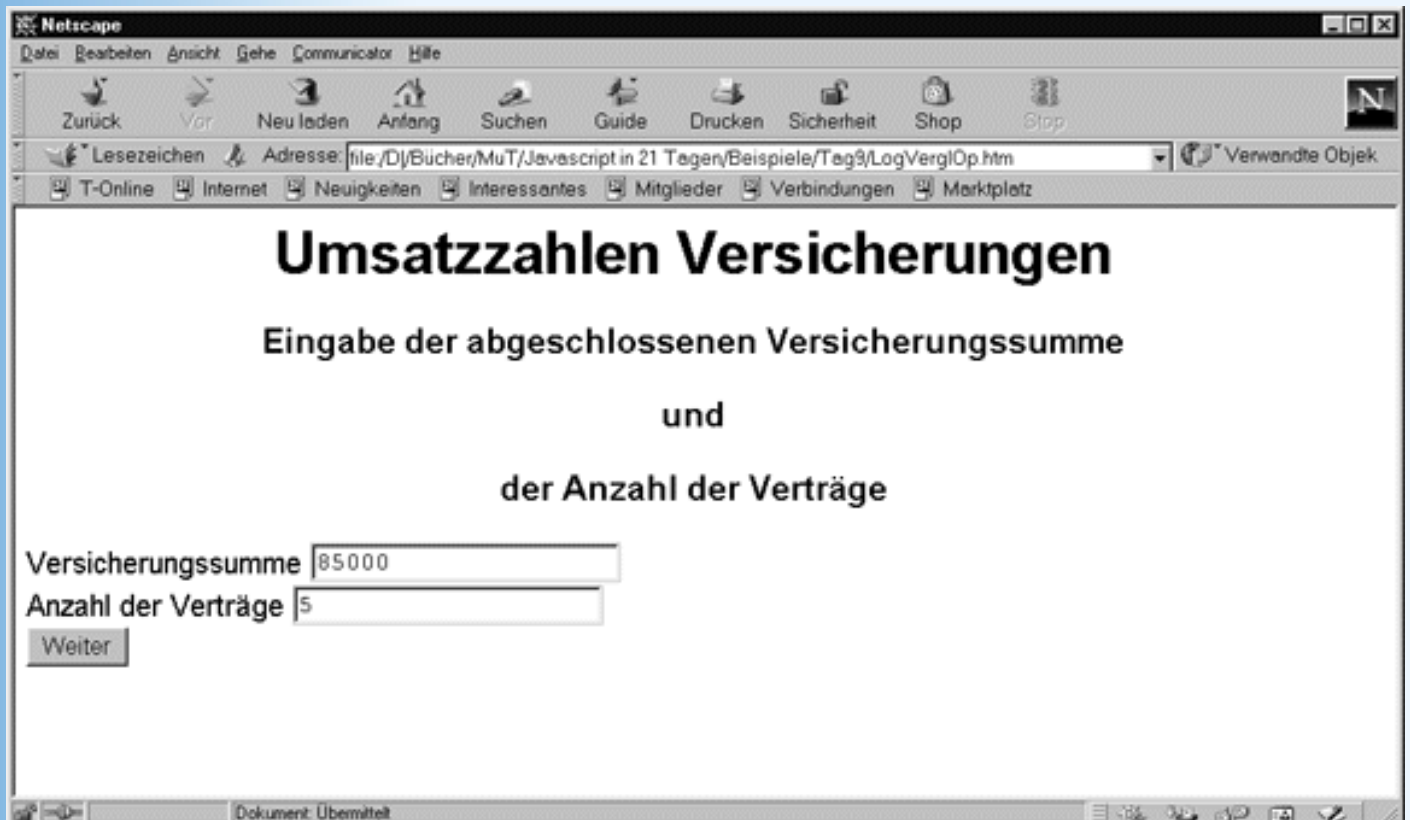


Abbildung 9.9: Die Eingaben einer Bedingung nicht erfüllt

Wenn nun eine der beiden Bedingungen nicht erfüllt ist, wird der erste if-Block nicht ausgeführt. Normalerweise lässt man nun einen else-Block folgen. Das ist hier bewusst nicht geschehen (wir wollen ja einen weiteren logischen Vergleichsoperator testen), sondern ein weiterer if-Block fängt diese Situation ab (er ist logisch so konzipiert, dass genau der else-Fall simuliert wird). Die if-Anweisung

```
if(
  (sum <= 100000)
  ||
  (anz < 4)
)
```

kontrolliert, ob eine der beiden Bedingungen nicht erfüllt ist. Dabei wurde einmal auf kleinergleich und einmal auf kleiner geprüft. Der nachfolgende Block wird ausgeführt, wenn eine oder beide Bedingungen erfüllt ist/sind.

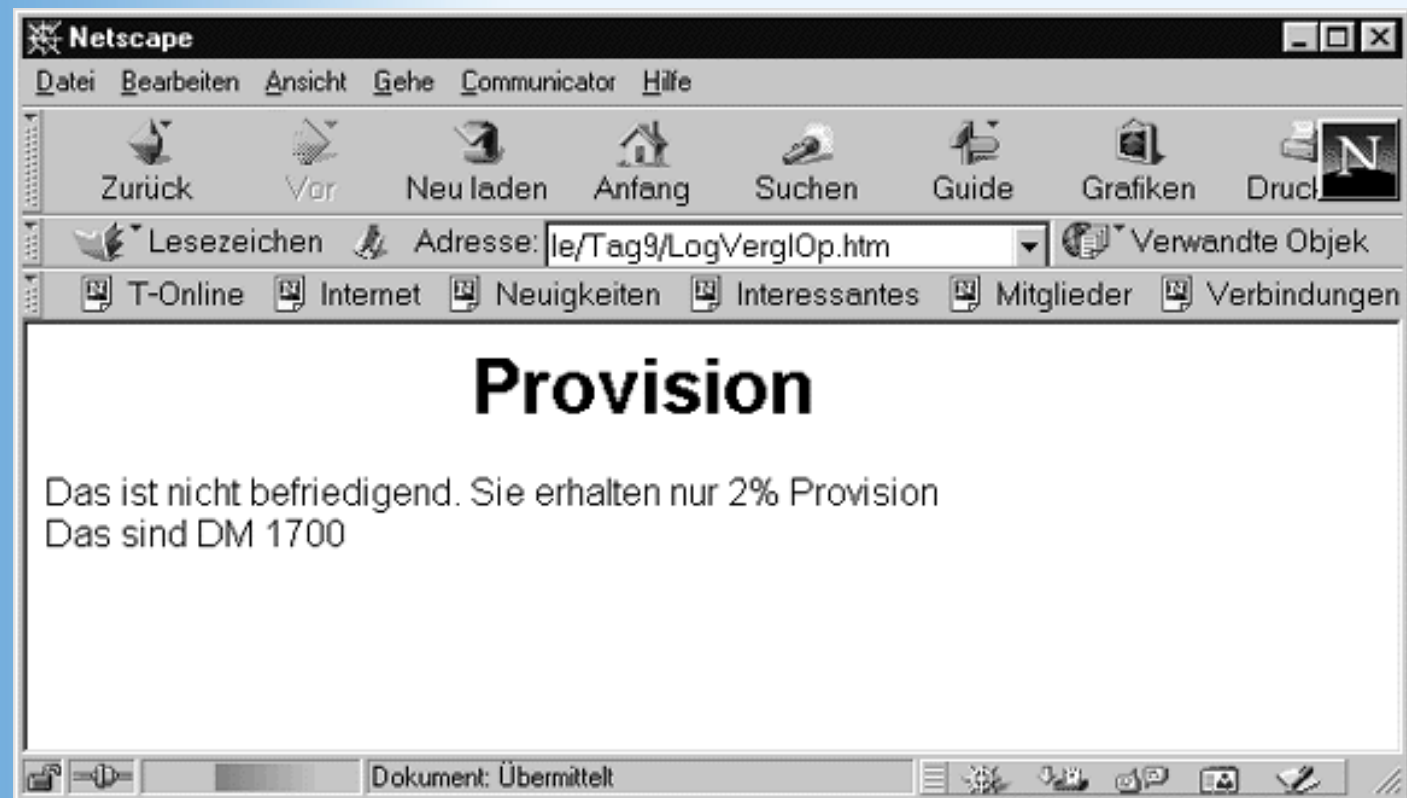


Abbildung 9.10: Das Resultat

Spielen wir noch ein Beispiel mit dem logischen Nicht-Operator durch.

Beispiel 8:

Erstellen Sie die Datei LogNot.htm:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function lade()
{
var a = false
if(!a)
{
document.write("Nicht")
document.write("<BR>")
}
}
</SCRIPT>
```

```
</HEAD>
<BODY onLoad="lade()">
</BODY>
</HTML>
```



Abbildung 9.11: Das sagt alles

Das Beispiel ist aus mehreren Gründen trotz der unscheinbaren Ausgabe sehr interessant. Es wird z.B. eine boolesche Variable verwendet. Also eine Variable, die nur true und false annehmen kann. In Kontrollstrukturen sind solche booleschen Werte von elementarer Bedeutung. Es geht nämlich immer nur darum, ob eine Prüfung wahr oder falsch ist. Man kann deshalb dort auch Variablen notieren, welche den Wert true oder false besitzen. Ganz ohne irgendwelche Vergleiche (die können vorher durchgeführt worden sein oder aber es wurde - wie in unseren Fall - der Variablen einfach ein boolescher Wert zugewiesen). Da die Variable a den Wert false hat, dürfte der nachfolgende Block in der if-Schleife nicht ausgeführt werden. Der vorangestellte Operator ! dreht den Wert jedoch auf true um.

Der konditionale und der typeof-Operator

Unter dem konditionalen bzw. if-else-Operator versteht man die Kombination aus Fragezeichen und Doppelpunkt ? :. Der Doppeloperator arbeitet mit drei Operanden. Die Operation benötigt einen booleschen Ausdruck vor dem Fragezeichen. Wenn er »wahr« liefert, wird der Wert vor dem Doppelpunkt zurückgegeben, ansonsten der Wert hinter dem Doppelpunkt. Der Operator ist eine Kurzschreibweise für ein if-else- Konstrukt. Wir wollen darauf bei der intensiveren Behandlung der if- Schleife zurückkommen.

Der in jeder Hinsicht außergewöhnliche Operator typeof ist ein unitärer Operator, der den Datentyp des nachgestellten Operanden in Form einer Zeichenkette zurückliefert. Dabei kann

- number,
- boolean,
- string,
- undefined,
- function oder
- object

als Ergebnis angegeben werden.

Beispiel 9:

Erstellen Sie die Datei typeof.htm:

```
<HTML>
```

```
<HEAD>
<SCRIPT language="JavaScript">
function lade()
{
var a = 2;
var b = true;
var c = new Date();
var d;
var e = "Das ist Text";
document.write(typeof a)
document.write("<BR>")
document.write(typeof b)
document.write("<BR>")
document.write(typeof c)
document.write("<BR>")
document.write(typeof d)
document.write("<BR>")
document.write(typeof e)
document.write("<BR>")
}
</SCRIPT>
</HEAD>
<BODY onLoad="lade()">
</BODY>
</HTML>
```

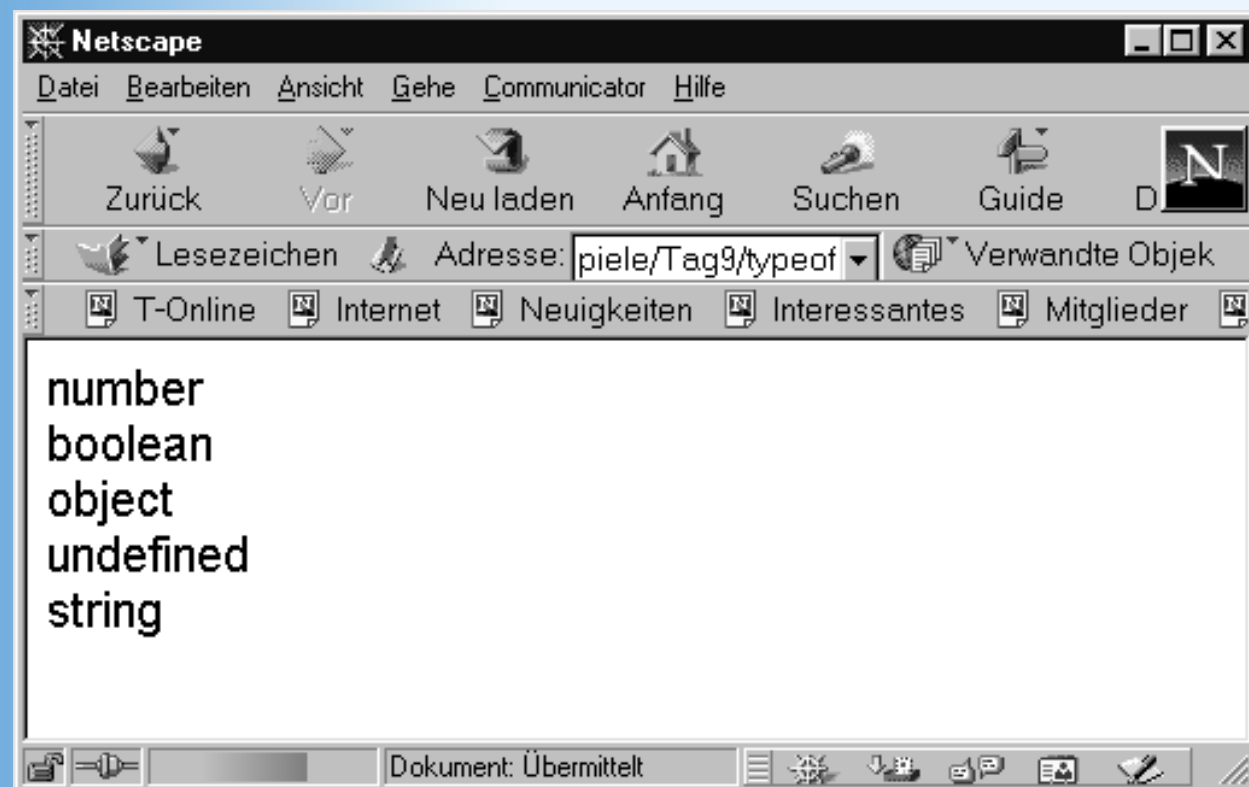


Abbildung 9.12: Typ der Variablen

Der typeof-Operator kann gut dazu genutzt werden, zu testen, ob eine Variable überhaupt definiert wurde. Wenn man den Wert undefined bei einem Test erhält, ist ihr entweder noch kein Wert zugewiesen worden oder aber es gibt sie noch gar nicht.

Bitweise Operatoren

Die letzte Form von Operatoren wollen wir hauptsächlich der Vollständigkeit halber angeben. Sie sind recht kompliziert und werden nicht oft in Scriptsprachen eingesetzt. Dennoch - JavaScript stellt wie viele andere mächtige Programmiersprachen auch Operatoren zur Verfügung, die auf Ebene der Bits arbeiten. Beispielsweise bitweise arithmetische Operatoren, welche direkt Bitwerte verändern. Oder bitweise Verschiebungsoperatoren, welche die Bits in der binären Darstellung eines Zeichens verschieben. Die Bits des ersten Operanden werden um die Anzahl an Positionen verschoben, die im zweiten Operanden angegeben wird. Im Fall der Verschiebung nach links ist es immer eine Null, mit der die rechte Seite aufgefüllt wird. Dieser Vorgang entspricht dem Multiplizieren mit 2 hoch der Zahl, die durch den zweiten Operanden definiert wird. Der normale Verschiebungsoperator nach rechts vervielfacht das Vorzeichenbit. Dieser Vorgang entspricht der Division durch 2 hoch der Zahl, die durch den zweiten Operanden definiert wird. Die Verschiebung nach rechts mit Füllnullen vervielfacht eine Null von der linken Seite.

Auch JavaScript stellt diese Operatoren zur Verfügung:

Operator	Beschreibung
&	bitweiser AND-Operator
	bitweiser OR-Operator
^	bitweiser XOR-Operator
~	bitweiser Komplement-Operator
<<	bitweise Verschiebung nach links
>>	bitweise Verschiebung nach rechts
>>>	bitweise Verschiebung nach rechts mit Füllnullen

Tabelle 9.7: Bitweise Operatoren

Der bitweise Komplement-Operator unterscheidet sich massiv von den anderen genannten bitweisen Operatoren, denn Komplementieren ist eine einstellige bitweise Operation mit nachgestelltem Operanden (im Gegensatz zu den anderen Operationen, welche zweistellig sind). Wenn ein Byte komplementiert wird, werden alle seine Bits invertiert.

JavaScript kennt auch bitweise Zuweisungsoperatoren und bitweise Vergleichsoperatoren. Bitweise Zuweisungsoperatoren führen wie normale Zuweisungsoperatoren Zuweisungen mit - jetzt bitweise zu verstehenden - Operationen durch. Bitweise Zuweisungsoperatoren verwenden einen Wert, führen eine entsprechende bitweise Operation mit dem zweiten Operanden durch und legen das Ergebnis als Inhalt des ersten Operanden ab.

Bitweise Vergleichsoperatoren vergleichen binär zwei Operanden.

Operator	Beschreibung
&=	bitweise AND-Zuweisung
=	bitweise OR-Zuweisung
^=	bitweise XOR-Zuweisung
<<=	bitweise Verschiebungszuweisung nach links
>>=	bitweise Verschiebungszuweisung nach rechts
>>>=	bitweise Verschiebungszuweisung nach rechts mit Füllnullen

Tabelle 9.8: Bitweise Vergleichsoperatoren

Operatoren-Prioritäten

Operatoren in JavaScript haben eine festgelegte Rangordnung, die immer dann angewandt wird, wenn mehrere Operatoren in einer Anweisung verwendet werden. Aus der Mathematik ist Ihnen sicher noch die Punkt-vor-Strich-Rechnung bekannt, die nichts anderes als eine Prioritätenangabe bedeutet. Die Prioritäten der JavaScript-Operatoren sind in der nachfolgenden Tabelle von der höchsten Wertigkeit bis zur niedrigsten abwärts angegeben. Bei gleicher Wertigkeit von Operatoren in einer Anweisungszeile (etwa bei Vergleichsoperatoren) erfolgt die Bewertung von links nach rechts. Beachten Sie, dass Klammern die höchste Priorität haben und Sie deshalb mit Klammernsetzen alle andere Prioritäten aufheben können. Zuweisungen haben (außer der Aufzählung mit Kommata) die niedrigste Priorität, was nichts anderes bedeutet, als dass ein Ausdruck auf der rechten Seite einer Zuweisung immer erst vollständig ausgewertet wird, bevor die Zuweisung erfolgt.

Priorität	Operatoren
1	() []
2	! ~ - <u>1</u> ++ -- typeof
3	* / %
4	+ - <u>2</u>
5	<< >> >>>
6	< <= > >=
7	== !=
8	&
9	^
10	
11	&&
12	
13	?:
14	= += -= <<= >>= &= ^= =
15	,

Tabelle 9.9: Prioritäten der Operatoren

- 1
- Im Sinn von
Negation
- 2
- Im Sinn von
Subtraktion

Beispiel 10:

Erstellen Sie die Datei Priori.htm:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function lade()
{
var a;
a = 1 + 2 * 3;
```

```
document.write(a);  
document.write("<BR>");  
a = (1 + 2) * 3;  
document.write(a);  
document.write("<BR>");  
}  
</SCRIPT>  
</HEAD>  
<BODY onLoad="lade()">  
</BODY>  
</HTML>
```



Abbildung 9.13: Ohne und mit Klammern

Das Beispiel nutzt bei der ersten Berechnung Punkt-vor-Strich-Rechnung aus, die durch die unterschiedlichen Prioritäten der Operatoren realisiert ist. Die zweite Berechnung hebt sie durch Klammern auf, da Klammern die höchste Priorität haben.

Zusammenfassung

Am heutigen Tag stand ein wohl etwas trockenes, aber nichtsdestotrotz sehr wichtiges Thema auf dem Stundenplan. Operatoren und Wertzuweisungen gehören zu den Grundtechniken, die jeder Programmierer aus dem FF beherrschen sollte. Insbesondere sollte klar geworden sein, dass Wertzuweisungen auf verschiedene Arten möglich sind. Wenn einer Variablen über einen Zuweisungsoperator ein Ausdruck zugewiesen wird, wird der Ausdruck auf der rechten Seite immer zuerst vollständig bewertet, bevor das Ergebnis zugewiesen wird.

Fragen und Antworten

Frage:

Wenn ich alle Schlüsselworte von JavaScript kenne, kann ich dann die Sprache?

Antwort:

Gegenfrage: Wenn Sie alle Worte der - sagen wir mal - spanischen Sprache kennen, können Sie dann Spanisch? Ohne die Grammatik nutzt die Kenntnis der Worte wenig. Und ohne Erfahrung und Praxis, wie man die Worte sinnvoll miteinander verbindet, nutzt auch die Kenntnis der Grammatik noch nicht viel. Es ist eher umgekehrt zu sehen. Die Kenntnis der (wichtigsten) Worte ist notwendig, aber noch lange nicht hinreichend.

Frage:

JavaScript hat eine ganze Reihe von Operatoren. Ist das nun viel oder wenig im Verhältnis zu anderen Sprachen?

Antwort:

Guter Durchschnitt. Es gibt Sprachen, die nicht so viele Operatoren bereitstellen, aber auch Sprachen wie Perl, die noch mehr Operatoren besitzen.

Frage:

Wenn ich die JavaScript-Operatoren von der Bedeutung her kenne, kenne ich dann auch die gleichen Operatoren in anderen Sprachen?

Antwort:

Ja, bis auf wenige Ausnahmen sind Operatoren in den meisten Programmier- und Scriptsprachen gleich. Eine der wichtigsten Ausnahmen ist der Vergleichsoperator, der in einigen Sprachen nur mit einem Gleichheitszeichen realisiert wird.

Workshop

Testen Sie verschiedene Schlüsselworte als Bezeichner. Der Interpreter wird keinen einzigen Fall durchlassen. Schauen Sie sich nochmals Beispiel 7 an und testen es mit verschiedenen Zahlenkonstellationen. Machen Sie sich die Verhaltensweise klar. Schreiben Sie die Vergleichsoperatoren um. Also »Vergleich auf größer« in »Vergleich auf kleiner«, »Vergleich auf kleinergleich« in »Vergleich auf kleiner« usw. Variieren Sie mit verschiedensten Konstellationen. Aber so, dass keine logische Veränderung auftritt, das Script also immer das Gleiche macht. Das heißt, Sie müssen auch die Zahlenangaben entsprechend dem Bedarf anpassen.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Betrachten Sie die folgende Anweisung:

`a + b / c - d`

In welcher Reihenfolge werden die einzelnen Schritte durchgeführt?

2. Was bedeutet der Operator `||` ?
3. Was bedeutet der Operator `!=` ?
4. Ist das Script richtig?

```
if(a=5)
{
  alert("Test");
}
```

5. Was bedeutet der Operator `&&` ?
6. Ist der nachfolgende Bezeichner ein Schlüsselwort?
`Else`
7. Was bedeutet der Operator `*=` ?
8. Ist der nachfolgende Bezeichner ein Schlüsselwort?
`funktion`
9. Was bedeutet der Operator `%` ?

10. Ist der nachfolgende Bezeichner ein Schlüsselwort?
`do`
11. Ist der nachfolgende Bezeichner ein Schlüsselwort?
`false`
12. Was bedeutet der Operator `+=` ?
13. Ist der nachfolgende Bezeichner ein Schlüsselwort?
`For`
14. Ist der nachfolgende Bezeichner ein Schlüsselwort?
`retrun`

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 2

Tag 10

Anweisungen und Kontrollfluss-Steuerung

Auf dem heutigen Tagesprogramm stehen allgemein Anweisungen unter JavaScript und insbesondere solche, die zur Steuerung des Programmflusses gesetzt werden.

Anweisungen sind sprachspezifische Befehle bzw. Befehlsfolgen, welche zur Grundsyntax einer jeden Script- bzw. Programmiersprache zählen. Sie sind das Rückgrat aller programmtechnischen Entscheidungsstrukturen.

Allgemeine Anweisungen

Anweisungen werden in einem JavaScript der Reihe nach oder aufgrund einer bestimmten Konstellation ausgeführt. Dabei kann man verschiedene Arten von Anweisungen unterscheiden.

Blockanweisungen

In JavaScript werden größere Quellcodeabschnitte zu Blockstrukturen mit geschweiften Klammern `{ ... }`

zusammengefasst, die dann einen Block bzw. eine Blockanweisung bilden. Dies haben wir in fast allen Beispielen bereits angewandt.

Deklarationsanweisungen

Deklarationsanweisungen sind die Einführung eines neuen Elements im Script. Etwa eine Variable oder eine Funktion. Es können aber auch andere Elemente eingeführt werden, wobei der Umfang der verschiedenen Elemente stark von der Programmier- oder Scriptsprache abhängt. Unter anderem gibt es die Deklaration von primitiven Datentypen, Datenfeldern, Methoden oder eines beliebigen Objekts. Zum Teil gibt es das auch in JavaScript - wir kommen noch darauf zurück.

Ausdrucksanweisungen

Ausdrucksanweisungen sind einmal das, was wir gestern intensiv behandelt haben - die Wertveränderung als Ergebnis der Verbindung von Operanden und Operatoren über die syntaktischen Regeln einer Scriptsprache. Dies ist die so genannte Zuweisungsanweisung, die rechts von einem Zuweisungsoperator einen Wert (konstant oder berechnet) stehen hat, der dem linken Ausdruck zugewiesen wird.

Alle Ausdrucksanweisungen müssen in JavaScript mit einem Semikolon beendet werden und werden immer vollständig durchgeführt, bevor die nächste Anweisung ausgeführt wird.

Die zweiten Varianten sind Auswahlanweisungen, die in einem Script in gewissen Situationen einen von mehreren möglichen Kontrollflüssen aussuchen. Sie dienen der Ablaufsteuerung von einem Script (wir gehen gleich genauer darauf ein).

Iterationsanweisungen

Iterationsanweisungen sind eine Angabe, unter welchen Voraussetzungen und wie oft nachfolgend notierte Anweisungen ausgeführt werden. Sie dienen also explizit der Ablaufsteuerung in einem Script. Im nachfolgenden Kapitel gehen wir auf diese wichtigen Kontrollfluss- Steuerungsmöglichkeiten ein.

Sprunganweisungen

Sprunganweisungen geben die Ablaufsteuerung eines Scripts an eine aufrufende Stelle zurück. Sie dienen der Ablaufsteuerung von Scripten. JavaScript kennt die Anweisung

`return`

zur Rückgabe eines Wertes und Rückgabe des Kontrollflusses sowie die Anweisung

`break`

In JavaScript 1.2 wurde eine Erweiterung von `break` vorgenommen, die aus vielen Programmiersprachen bereits bekannt ist: die `break`-Anweisung mit Angabe eines Labels, d.h. einer Sprungadresse die angibt, wo nach Ausführen der `break`-Anweisung mit der Abarbeitung des Codes fortgefahren werden soll. Das Label ist ein frei definierbarer Name an einer beliebigen Stelle im Quelltext, der mit einem nachgestellten Doppelpunkt notiert wird. Hinter dem Wort `break` können Sie dann durch Leerzeichen abgetrennt den Namen des Labels angeben. Diese Technik ist jedoch sehr kritisch zu betrachten, weil viele Browser sie nicht unterstützen. Schlimmer ist noch, dass sie die Erzeugung von so genanntem Spagetticode erlaubt, welcher in grauen Computerzeiten über die unselige BASIC-Anweisung `goto` üblich war. Wir gehen nicht weiter darauf ein. Neben `break` und `return` gibt es in JavaScript noch die Anweisung

`continue`.

Sie können damit in einer Schleife unmittelbar den nächsten Schleifendurchlauf erzwingen und die nachfolgenden Anweisungen innerhalb der Schleife ignorieren.

Wir werden alle drei Sprunganweisungen gleich im Rahmen der Behandlung von Kontrollfluss-Anweisungen einsetzen und an Hand praktischer Beispiele erläutern.

Auswahl- und Iterationsanweisungen

JavaScript kennt die in den meisten Sprachen üblichen Auswahl- und Iterationsanweisungen. Zum Teil haben wir sie schon verwendet. Da gibt es beispielsweise die if-Auswahlanweisung.

if-Anweisung

Die if-Auswahlanweisung ist uns schon in einigen Beispielen begegnet. Wir wollen sie jetzt intensiver anschauen. Diese Kontrollstruktur ist das Analogon zu der wenn()-Funktion von Excel. Wenn eine Bedingung erfüllt ist, werden die nachfolgend notierten Anweisungen ausgeführt. Grundsätzliche Syntax ist immer die folgende:

```
if ([Bedingung])
{
...Anweisungen
}
```

Unter Bedingung versteht man eine Anweisung, die entweder wahr (true) oder falsch (false) sein kann. Zu deren Erstellung verwendet man in der Regel die Vergleichsoperatoren - gegebenenfalls in Verbindung mit logischen Operatoren. Wir haben dies schon mehrfach an den vorherigen Tagen angewandt. Ein Beispiel soll dennoch nicht fehlen. Eine andere sinnvolle Variante ist, wenn eine dort aufgerufene Funktion einen booleschen Rückgabewert liefert. Auch dazu führen wir ein Exempel durch.

Beispiel 1

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var pw = prompt("Bitte geben Sie Ihr Passwort ein","");
if(pw=="geheim")
{
document.write(
"<H1 align=center>Herzlich Willkommen auf meiner Homepage</H1>");
document.write("<br>");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter PW1.html und laden Sie es in einen Browser. Beim Laden der Webseite werden Sie aufgefordert, ein Passwort einzugeben.

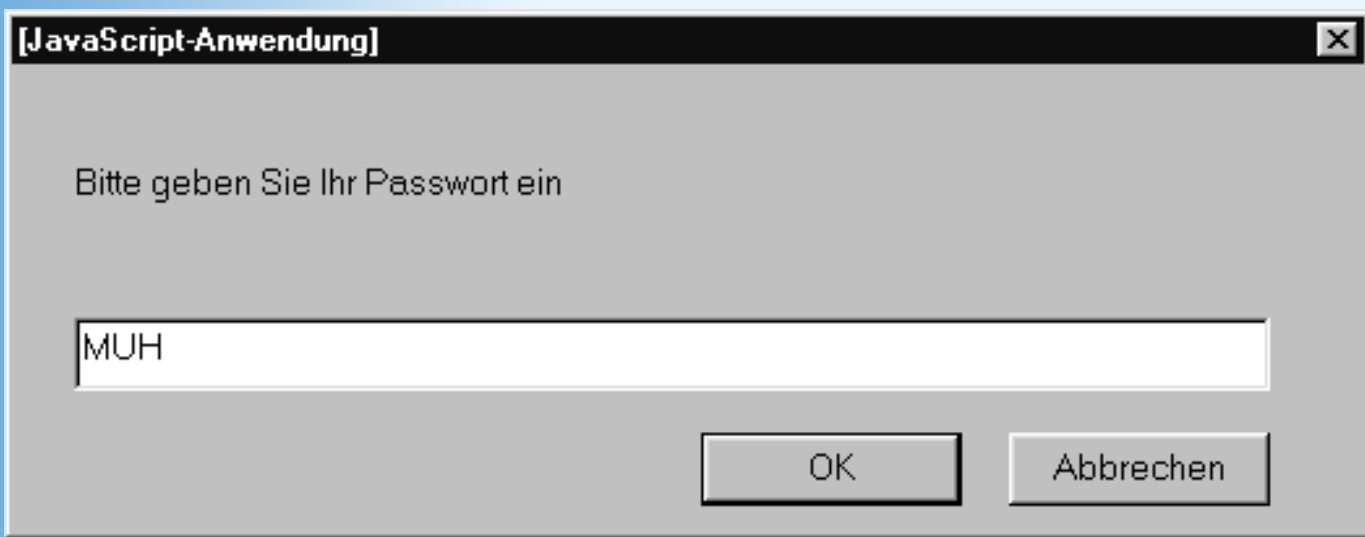


Abbildung 10.1: Eingabe des Passwortes.

Wenn das Passwort geheim eingegeben wird, wird nachfolgend eine Webseite aufgebaut. Ansonsten passiert nichts weiter. Der Anwender sieht also nur eine leere Seite.

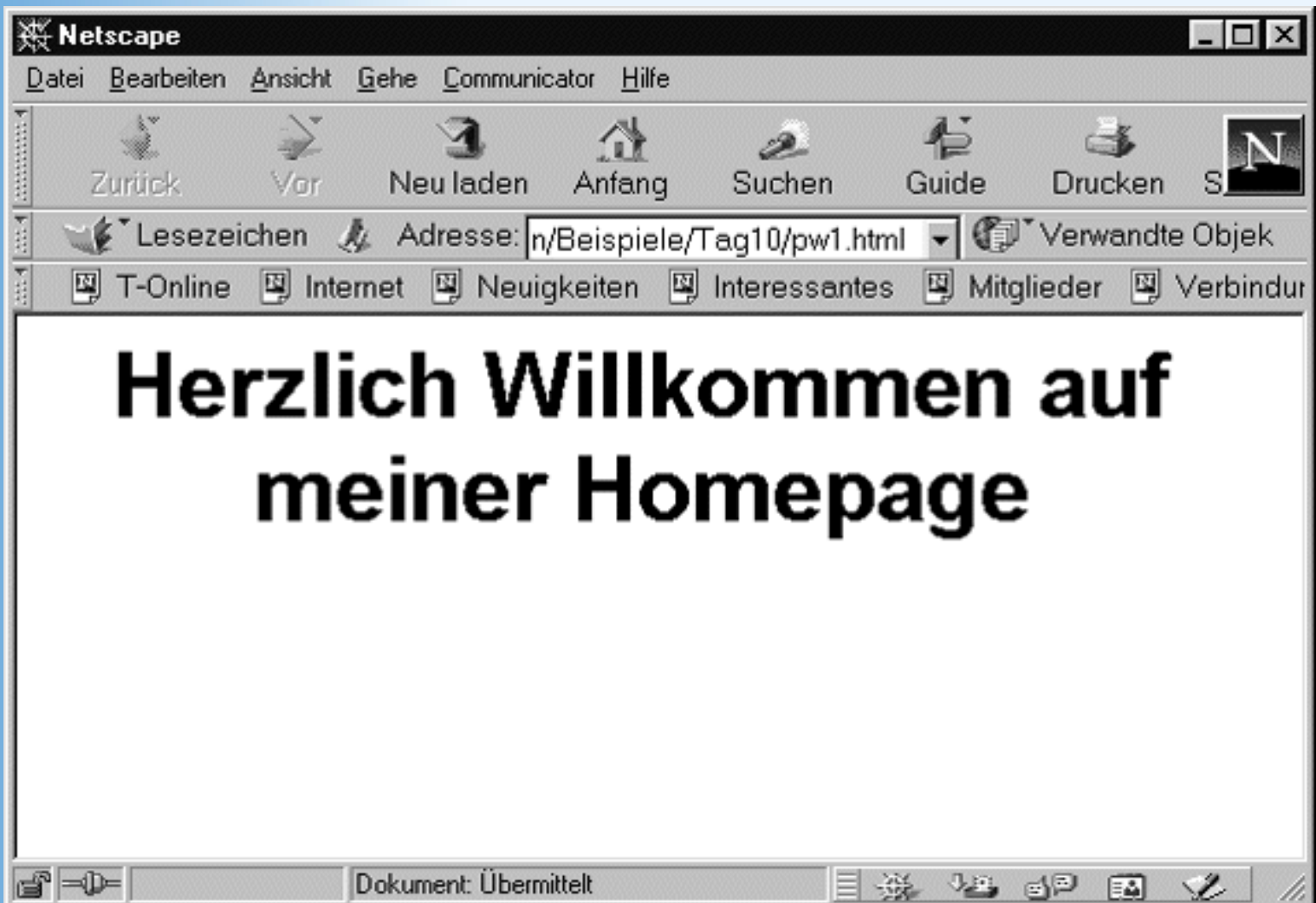


Abbildung 10.2: Bei korrekter Eingabe des Passwortes wird die Seite aufgebaut



Wie wir gestern bei logischen Operatoren gesehen haben, kann eine Bedingung aus

mehreren Bestandteilen bestehen. Das bedeutet, es muss entweder mehr als eine Bedingung notwendigerweise erfüllt sein oder es muss sich nur eine von mehreren Bedingungen bewahrheiten. In diesem Fall kommen die logischen Operatoren `&&` für eine Und-Verknüpfung und `||` für eine Oder-Verknüpfung zum Einsatz. Diese können obendrein beide gleichzeitig verwendet werden, und auch die Anzahl der Bedingungen ist im Prinzip unbeschränkt. Es ist eigentlich nur ein logisches Problem, eine gewisse Anzahl von verknüpften Bedingungen auch so zu formulieren, dass sie den gewünschten Sinn repräsentieren.

Das Beispiel soll nun so umgeschrieben werden, dass die Bedingung in einer externen Funktion überprüft und deren Aufruf direkt in die if- Bedingung notiert wird.

Beispiel 2

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function test(pass)
{
  if(pw=="geheim") return true;
}
var pw = prompt("Bitte geben Sie Ihr Passwort ein","");
if(test(pw))
{
  document.write(
"<H1 align=center>Herzlich Willkommen auf meiner Homepage</H1>");
  document.write("<br>");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter PW1b.html und laden Sie es in einen Browser. Das Script verhält sich vollkommen analog zu dem ersten Beispiel. Dennoch gibt es ein paar Stellen, die wir uns genauer anschauen wollen.

Da ist einmal die Zeile

```
if(test(pw)).
```

Diese Zeile beinhaltet die Situation, dass anstelle eines explizit dort notierten Vergleichs ein Funktionsaufruf steht. Diese dort angegebene Funktion sollte sinnvollerweise entweder true oder false zurückliefern, damit der nachfolgende Block entweder ausgeführt wird oder eben nicht.

Die Funktion test() selbst hat auch eine interessante Stelle:

```
if(pw=="geheim") return true;
```

Einmal sehen Sie die return-Sprunganweisung bei der Arbeit, aber es gibt noch etwas Interessantes. Was ist da neu gegenüber den bisherigen Fällen, wo wir die if-Anweisung verwendet haben?

Es fehlt der Block. Die Anwendung steht ohne Blockstruktur unmittelbar hinter der if-Anweisung. Das geht immer dann, wenn nur eine einzige Anweisung ausgeführt werden soll. Wir wollen dennoch in Zukunft von einem Block von Anweisungen reden.

Grundsätzlich ist eine if-Anweisung zur Behandlung einer Situation alleine oft nicht ausreichend. Meist wird eine Alternative notwendig sein. Für den Fall gibt es die Erweiterung else, die einen Block mit Anweisungen einleitet, welche immer dann (und nur dann) ausgeführt werden, wenn die vorangestellte Bedingung der if-Anweisung nicht erfüllt ist. Wenn bereits der if-Zweig ausgeführt wurde, wird der else-Zweig nicht ausgeführt. Grundsätzliche Syntax ist immer die folgende:

```
if ([Bedingung])
{
...Anweisungen
}
else
{
...alternative Anweisungen
}
```

Schreiben wir das Beispiel 1 nun so um, dass es zwei Alternativen gibt.

Beispiel 3

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var pw = prompt("Bitte geben Sie Ihr Passwort ein","");
if(pw=="geheim")
{
document.write(
"<H1 align=center>Herzlich Willkommen auf meiner Homepage</H1>");
document.write("<br>");
}
else
{
document.write(
"<H1 align=center>Leider nur für Mitglieder</H1>");
document.write("<br>");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter PW2.html und laden Sie es in einen Browser. Beim Laden der Webseite werden Sie wie gehabt aufgefordert, ein Passwort einzugeben. Wenn das eingegebene Passwort wieder geheim lautet, wird die gleiche Webseite wie in Beispiel 1 aufgebaut. Ansonsten jedoch bekommt der Anwender in diesem dritten Beispiel eine Alternativseite angezeigt.



Abbildung 10.3: Bei falscher Eingabe des Passwortes wird eine Alternativseite aufgebaut

Im else-Block können nun bei Bedarf auch weitere if-Anweisungen notiert werden. Das ist sehr oft der Fall, denn oft genügen zwei Alternativen nicht. Da in diesem Fall die if-Anweisung die einzige Anweisung des else-Blocks ist, kann das Schlüsselwort if direkt hinter else notiert werden (siehe unter Beispiel 2). Das sieht dann schematisch so aus:

```
if ([Bedingung 1])
{
...Anweisungen
}
else if ([Bedingung 2])
{
...alternative Anweisungen
}
else if ([Bedingung 3])
{
...alternative Anweisungen
}
.
.
.
else
{
...alternative Anweisungen
}
```

}

Die Anzahl der Blöcke ist beliebig und ob der abschließende else-Block vorhanden ist, ist nur eine Frage der Logik.

Beispiel 4:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var pw = prompt("Bitte geben Sie Ihre Spende ein","");
if(pw < 5000)
{
document.write("<H1 align=center>Geizhals</H1>");
document.write("<br>");
}
else if(pw < 10000)
{
document.write("<H1 align=center>Vielen Dank</H1>");
document.write("<br>");
}
else
{
document.write(
"<H1 align=center>Vielen, vielen Dank</H1>");
document.write("<br>");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter Spende.html und laden Sie es in einen Browser. Beim Laden der Webseite werden Sie aufgefordert, einen Spendenbetrag einzugeben.

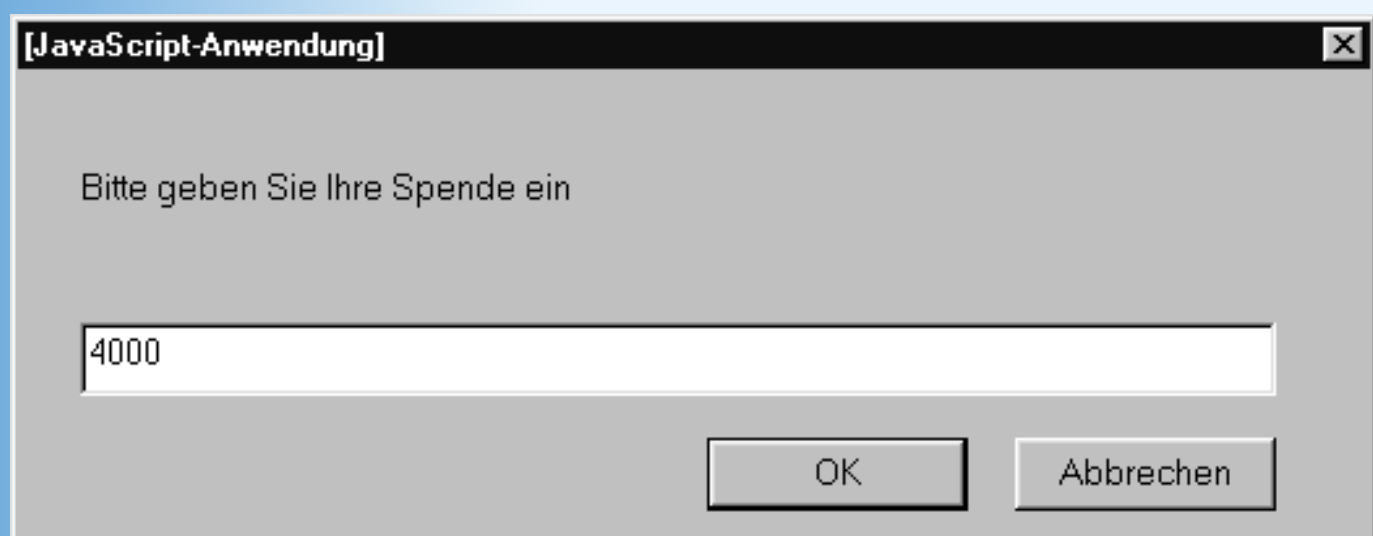


Abbildung 10.4: Höhe der mildtätigen Gabe

Wenn der Betrag unter 5.000 ist, wird der if-Block ausgeführt, ist er unter 10.000, der else-if-Block und ansonsten der else-Block.



Abbildung 10.5: So isses recht - else



Abbildung 10.6: Na, ganz okay - else if



Abbildung 10.7: Unzufriedenheit - if

Beachten Sie, dass die Struktur darauf aufbaut, dass das gesamte Konstrukt abgebrochen wird, wenn einer der Blöcke ausgeführt wurde. Das soll bedeuten, dass eine Eingabe unter 5.000 den if-Block ausführt, dann aber nicht mehr die anderen beiden Blöcke, obwohl die Eingabe auch die Bedingung kleiner 10.000 erfüllt hätte. Der erste Treffer bei kleiner 5.000 verhindert die Abprüfung weiterer Möglichkeiten.

Der Konditional-Operator als Kurzschreibweise für die if-else-Anweisung

Erinnern Sie sich an den konditionalen Operator von gestern? Es ist eine Kombination aus dem Fragezeichen und Doppelpunkt und eine Kurzschreibweise für ein if-else-Konstrukt. Vor dem Fragezeichen muss in Klammern eine Bedingung formuliert werden. Hinter dem Fragezeichen werden die Anweisungen notiert, die ausgeführt werden, wenn die Bedingung wahr ist. Hinter dem nachfolgenden Doppelpunkt folgen die Anweisungen, die sonst im else-Block stehen.

Beispiel 5:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var pw = prompt("Bitte geben Sie Ihre Spende ein","");
```

```
pw < 5000 ? alert("Geizhals") : alert("Vielen Dank");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter Konditional.html und laden Sie es in einen Browser. Beim Laden der Webseite werden Sie aufgefordert, einen Spendenbetrag einzugeben, und erhalten die entsprechende Reaktion.

Wegen der schlechten Lesbarkeit sollten Sie - zumindest in der Anfangsphase - von diesem Konstrukt Abstand nehmen.

Die switch-Fallunterscheidung

Wenn man im Ablauf eines Scripts mehr als zwei Wahlmöglichkeiten unterscheiden muss, bietet sich in einigen Situationen eine aufeinander folgende Notation von if-else-Anweisungen an. Dies kann aber bei einer größeren Anzahl von Fallunterscheidungen recht unübersichtlich und mühsam werden. Es gibt aber in JavaScript alternative Techniken, die in bestimmten Fällen eine sinnvolle Lösung erlauben.

Seit der JavaScript-Version 1.2 gibt es eine mögliche Fallunterscheidung über ein Konstrukt, welches in vielen anderen Programmiersprachen schon lange zum Standard gehört - die Fallunterscheidung über switch-case.

Wesentlicher Vorteil gegenüber der if-else-Struktur ist, dass Sie auf einfache und elegante Weise mehr als zwei Fälle unterscheiden können. Die Syntax einer switch-Fallunterscheidung sieht im Allgemeinen folgendermaßen aus:

```
switch( [auswahl] )
{
case [Fall 1]:
...
break;
case [Fall 2]:
...
break;
.
.
.
default:
...
break;
}
```

Mit dem Schlüsselwort switch leiten Sie die Fallunterscheidung ein. Als Argument wird, in runden Klammern eingeschlossen, eine Variable oder ein Ausdruck angegeben, für dessen aktuellen Wert Sie die Fallunterscheidung durchführen. Dies kann ein beliebiger Datentyp sein, der unter JavaScript erlaubt ist (etwa eine Zahl, aber auch ein Text). Er wird auf Übereinstimmung mit einem der nachfolgend hinter dem Schlüsselwort case notierten Werte geprüft.

Die einzelnen Fälle, zwischen denen unterschieden werden soll, werden untereinander aufgelistet und innerhalb geschweifeter Klammern als Blöcke hinter case notiert.



So elegant die switch-Fallunterscheidung auch ist - sie ist bei weitem nicht so flexibel wie die if-else-Struktur. Man kann nur auf feste Werte abprüfen und beispielsweise nicht direkt mit Größer- oder Kleiner-Vergleichen arbeiten. Dafür zeigt aber die switch-Fallunterscheidung nach einem Treffer eine interessante Verhaltensweise, die wir gleich kennen lernen.

Beispiel 6:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var ergebnis=prompt(
"Bitte geben Sie eine Zahl zwischen 1 und 10 ein","");
switch(ergebnis)
{
case 1:
document.write("Fall 1 wird genommen");
document.write("<P>");
break;
case 2:
document.write("Fall 2 wird genommen");
document.write("<P>");
break;
case 3:
document.write("Fall 3 wird genommen");
document.write("<P>");
break;
case 4:
document.write("Fall 4 wird genommen");
document.write("<P>");
break;
default:
document.write("Keiner der Fälle wird genommen");
document.write("<P>");
break;
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter SwitchTest1.html und laden Sie es bitte zuerst in den Netscape Navigator. Beim Laden der Webseite werden Sie aufgefordert, eine Zahl zwischen 1 und 10 einzugeben. Dieser Wert wird in einer Variablen gespeichert. In dem switch-Block wird der Fall ausgewählt, dessen

Prüfwert mit diesem Wert übereinstimmt. Die Anweisung `break` unterbricht die weitere Abarbeitung dieser `switch`-Fallunterscheidung und springt hinter den gesamten `switch`-Block. Wenn keine der Auswahlmöglichkeiten auf den Wert der überprüften Variablen zutrifft, wird der mit `default` bezeichnete Fall ausgewählt.



Abbildung 10.8: Default-Fall



Abbildung 10.9: Eingabe war 4 - je nach Wert wird ein Zweig ausgewählt

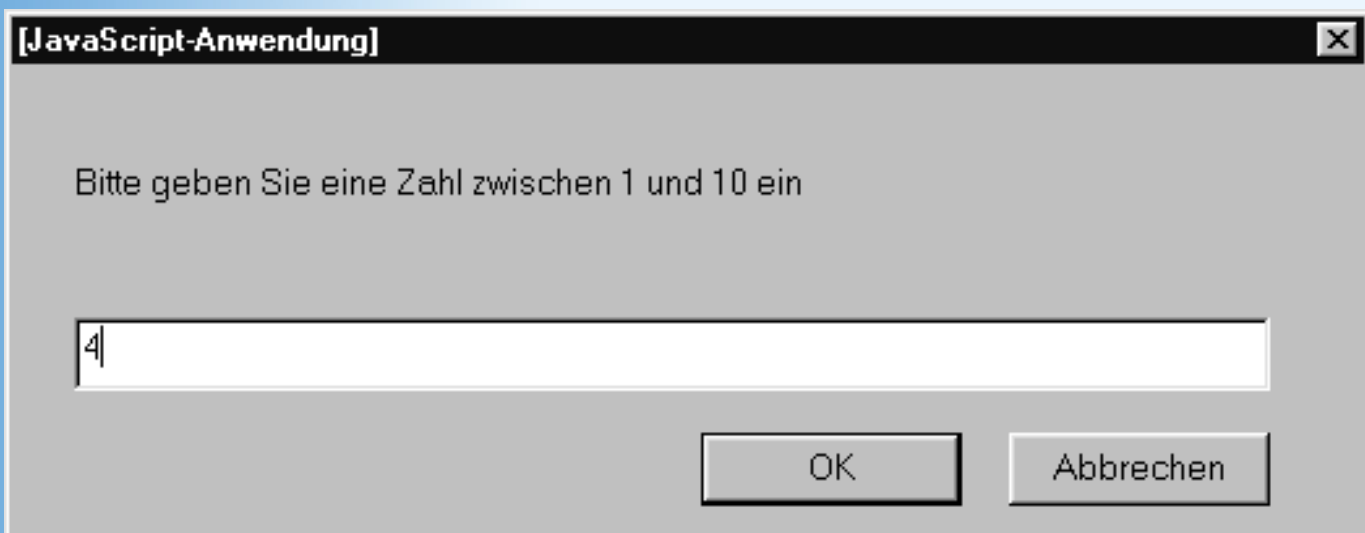


Abbildung 10.10: Eingabe des Werts

Laden Sie nun die Datei in den Internet Explorer. Es wird immer der Default-Fall genommen. Unabhängig von der Eingabe. Was könnte der Grund sein?

Das Problem basiert einmal auf dem lockeren Umgang von JavaScript mit Datentypen und außerdem darauf, dass der Internet Explorer kein JavaScript kann. Erinnern Sie sich? Er muss JavaScript erst in JScript übersetzen und kann erst dann das Script ausführen. Hier haben wir einen der Fälle, wo es dabei Probleme gibt. Der Navigator konvertiert die Benutzereingabe beim Abprüfen in der switch-Anweisung in Zahlen, der Internet Explorer bleibt bei Strings und findet deshalb keine Übereinstimmung.

Welche Lösungen gibt es für das Dilemma. Einmal ist es grundsätzlich möglich, in der switch-Anweisung auch auf Übereinstimmung mit Zeichenketten zu überprüfen (auch mehr als ein Buchstabe). Wenn Sie jeden Wert hinter der jeweiligen case-Anweisung in Hochkommata setzen (etwa so: case "3":), wird auch der Internet Explorer zurechtkommen. Der Netscape Navigator kommt ebenfalls damit zurecht, weshalb Sie für den Fall von zu testenden Benutzereingaben damit auf der sicheren Seite sind. Aber auch die Übersetzung von Benutzereingaben in numerische Werte und dann den unveränderten Test kann man für den Internet Explorer hinbiegen, ohne dass dann der Netscape Navigator Probleme bekommt. Sie wissen schon wie. Oder erinnern Sie sich nicht mehr an den Trick, Typkonvertierungen eines Strings mit nur Zahlen darin durch Multiplikation mit 1 auszulösen (Tag 8, Beispiel 6)? Die Zeile mit der Entgegennahme der Benutzereingabe könnte beispielsweise so verändert werden:

```
var ergebnis = 1 * prompt(
"Bitte geben Sie eine Zahl zwischen 1 und 10 ein","");
```

Die switch-Anweisung hat nun noch einige Feinheiten zu bieten, die wir behandeln sollten. Da ist einmal die Anordnung der case-Blöcke. Diese Anordnung muss nicht in einer bestimmten Reihenfolge erfolgen. Das ist deshalb klar, weil nur auf exakte Übereinstimmung geprüft wird. Es ist aber auch möglich, dass ein Wert in mehreren case-Blöcken überprüft wird. Das ist aber meist unsinnig, weil der erste Treffer genommen wird und nachfolgende identische case-Prüfungen irrelevant werden.

Eine andere Feinheit ist aber nicht so trivial. Haben Sie sich schon gefragt, warum am Ende jedes Blocks das Schlüsselwort break (eine Sprunganweisung) steht? Damit stellen Sie sicher, dass nach einem Treffer die nachfolgenden Fälle nicht ebenso ausgeführt werden, was in der Regel nicht gewünscht ist. Wie das zu verstehen ist, wird ein Beispiel demonstrieren. Dabei unterbleibt einfach einmal die Notation von break.

Beispiel 7:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var pw = prompt("Bitte geben Sie Ihre UserId ein","");
switch(pw)
{
case "Gast":
document.write("Willkommen als " + pw);
document.write("<P>");
document.write("Sie sehen hier ein eingeschränktes Angebot");
//break;
case "Bob":
document.write("Willkommen " + pw);
document.write("<P>");
document.write("Hier ist Ihr Angebot");
//break;
case "Admin":
document.write("Willkommen " + pw);
document.write("<P>");
document.write("Hier ist die Konfigurationsseite");
//break;
```

```
default:
document.write("<H1>Nur für Mitglieder</H1>");
document.write("<P>");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter switchTest2_ohneBreak.htm und laden Sie es in einen Browser. Beim Laden der Webseite werden Sie aufgefordert, eine UserId einzugeben. Dieser Wert wird in einer Variablen gespeichert. In dem switch-Block wird der Fall ausgewählt, dessen Prüfwert mit diesem Wert übereinstimmt. Geben Sie beispielsweise als UserId Gast ein. Wahrscheinlich erwarten Sie, nur die Seite für den Gast zu sehen.

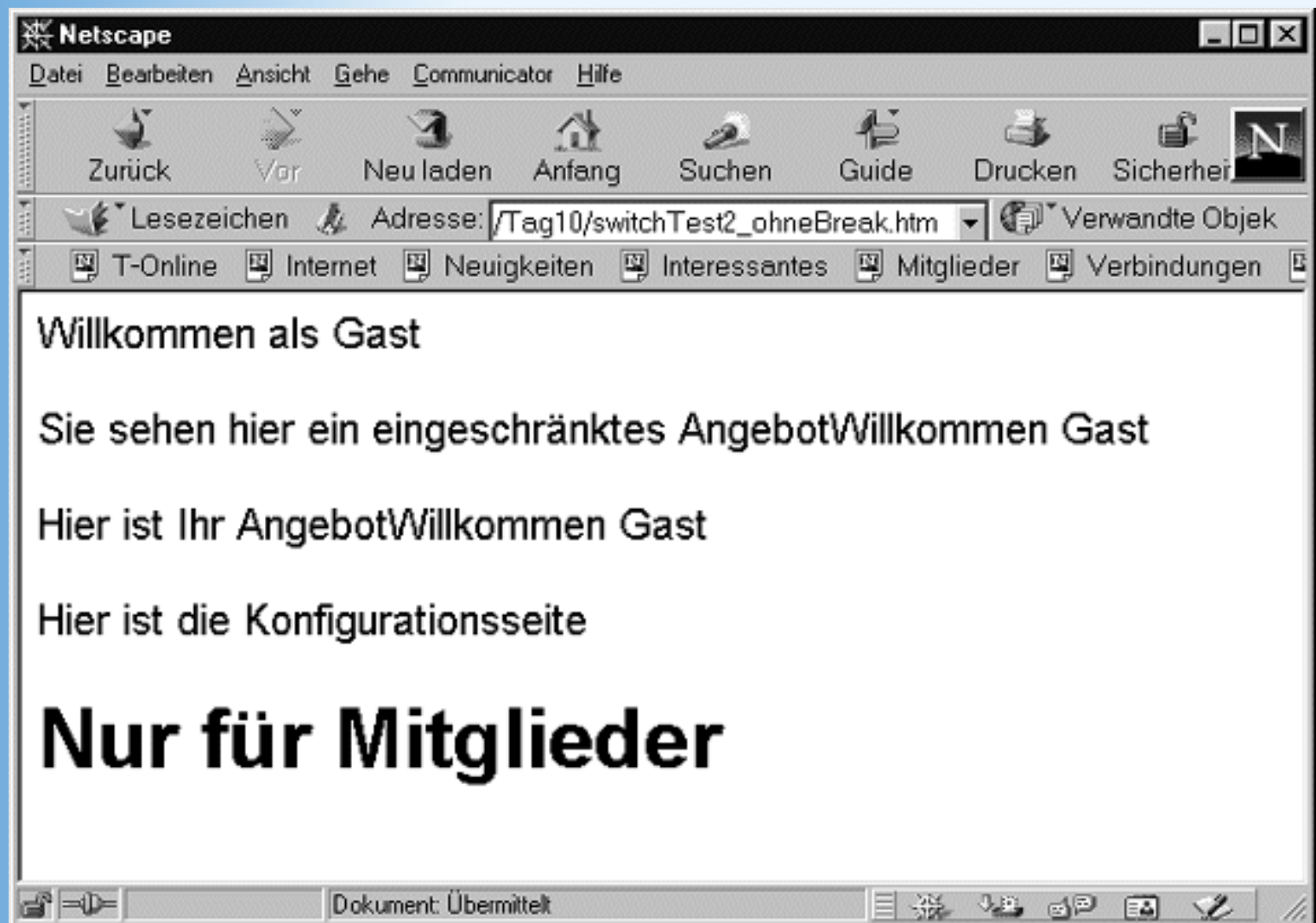


Abbildung 10.11: Das wird aber angezeigt

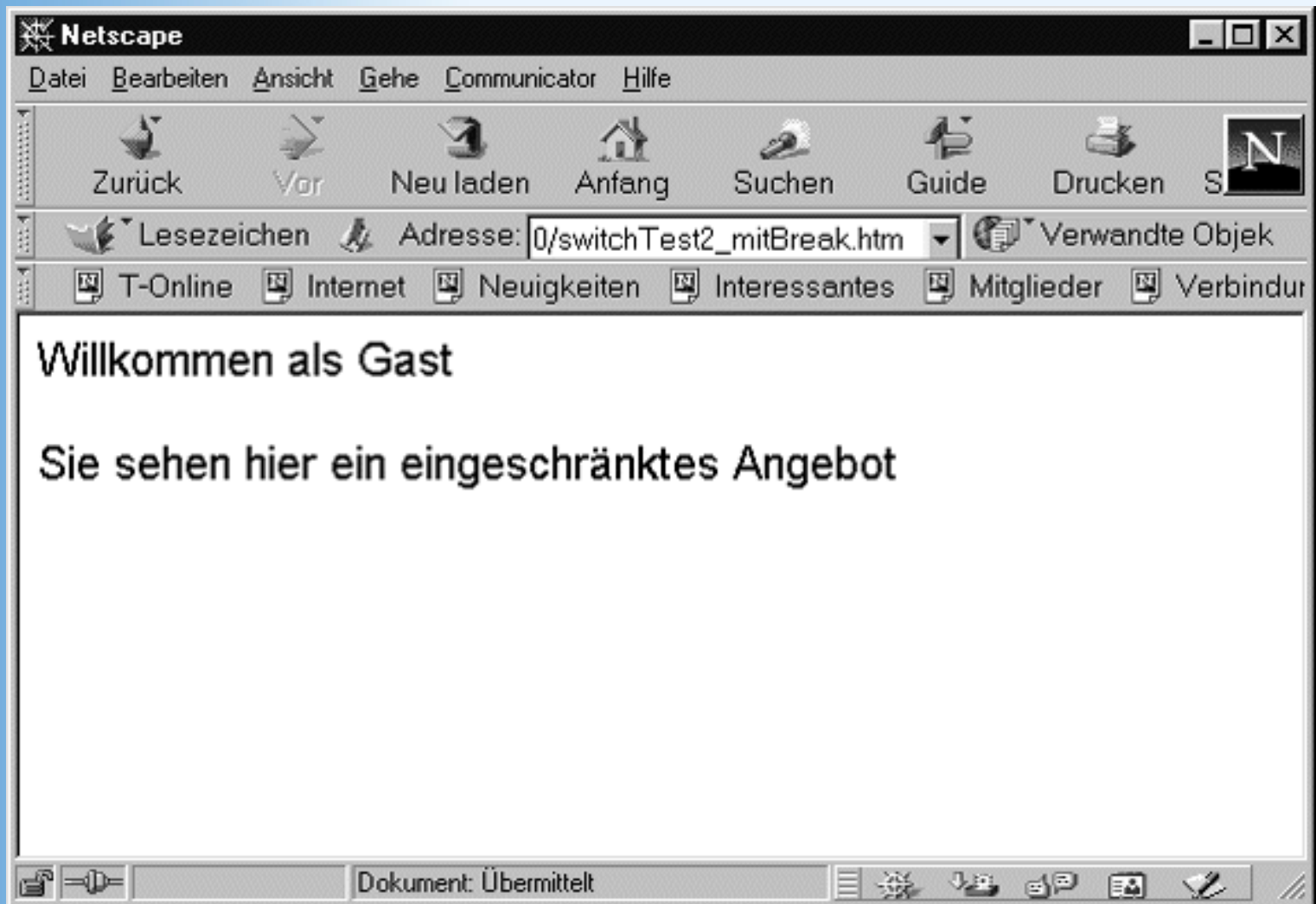


Abbildung 10.12: Das wird wahrscheinlich erwartet

In der Tat werden aber alle Blöcke nach dem ersten Treffer abgearbeitet.



Nochmals zur Sicherheit - diese Anweisung steht erst in JavaScript 1.2 zur Verfügung. Browser, die diese Anweisung nicht verstehen, werden mit einer Fehlermeldung reagieren. Dazu zählt auch der Opera 3.6. Trotz der eleganteren Lösung über switch ist es sicherer, mehrere if-Abfragen hintereinander zu schalten.

while und do-while

Zwei eng verwandte und sehr oft eingesetzte Kontrollfluss-Anweisungen sind die while- und die do-while-Schleife. Beide führen eine Prüfung einer Bedingung durch und wiederholen die im inneren Block notierten Anweisungen so lange, bis die Bedingung nicht mehr erfüllt ist. Die allgemeine Syntax ist folgende:

```
while ([Bedingung])  
{  
  ...Anweisungen  
}
```

bzw.

```
do
{
...Anweisungen
}
while ([Bedingung])
```

Die Bedingung können wie bei der if-Anweisung über Rückgabewerte von Funktionen oder direkte Vergleiche überprüft werden.

Wie unterscheiden sich nun while- und do-while-Schleifen? Bei der while- Schleife gilt, dass, wenn eine Bedingung bei Erreichen der Schleife falsch ist, die Anweisungen im Block überhaupt nicht durchgeführt werden.

Die do-while-Schleife ist eine Abart der normalen while-Schleife und steht erst seit der Java-Version 1.2 zur Verfügung. Der wichtigste Unterschied zu der normalen while-Schleife ist, dass bei der do-while-Schleife die im Inneren notierten Anweisungen auf jeden Fall einmal ausgeführt werden, bevor die Schleifenbedingung überprüft wird.



Eine kleine Eselsbrücke ist, dass bei der while-Schleife die Überprüfung im Quelltext vor dem Block mit den Anweisungen steht, bei der do-while-Schleife hinter dem Block mit den Anweisungen.

Beide Schleifen benötigen in ihrem Inneren immer eine Anweisung, die eine Situation schafft, mit der die Schleife abgebrochen werden kann. Andernfalls erzeugen Sie unter Umständen eine Endlosschleife.



Unter einer Endlosschleife versteht man eine Wiederholung von Anweisungen im Inneren, wo niemals eine Situation eintritt, die für eine Beendigung der Wiederholung sorgt. Dies kann man sinnvoll nutzen - etwa ein Werbefenster, wo eine Folge von Grafiken immer wieder abgespielt werden soll. Oder eine Hintergrundmusik. Es kann aber auch als Fehler passieren. In diesem Fall bleibt oft nur die Beendigung eines Programms oder gar ein Neustart des Rechners.

Normalerweise erfolgt die Generierung einer Abbruchsituation darüber, dass die in der Bedingung geprüfte Variable im Inneren so verändert wird, dass irgendwann die Bedingung nicht mehr erfüllt ist. Es kann aber auch im Inneren mit einer zusätzlichen Anweisung (etwa der if-Anweisung) eine Unterbrechung ausgelöst werden - etwa mit einem break oder return. Wir testen dies in mehreren Beispielen.

Beispiel 8:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var zahl=prompt(
"Geben Sie eine Zahl grösser 0 ein","");
```

```
var zaehler=0;
while(zaehler <= zahl)
{
document.write(zaehler + " ");
zaehler++;
}
document.write("<br>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter while1.html und laden Sie es in einen Browser. Beim Laden der Webseite werden Sie aufgefordert, eine Zahl größer 0 einzugeben. Dieser Wert wird in einer Variablen gespeichert. Daneben gibt es eine Zählvariable, die zuerst auf 0 gesetzt ist. Die Bedingung in der nachfolgenden while-Überprüfung ist wahr und die Anweisungen werden ausgeführt. Nach der Ausgabe des Wertes und einem Leerzeichen wird der Wert der Variablen um 1 erhöht. Wir haben hier eine Anwendung von dem Inkrement-Operator, der dem Ausdruck `zaehler = zaehler + 1;` entspricht. Das Durchlaufen der Schleife wird so lange fortgesetzt, bis die Variable den Wert erreicht, den der Anwender eingegeben hat. Dann ist die Bedingung in der while-Überprüfung falsch und die Anweisungen werden nicht mehr ausgeführt.

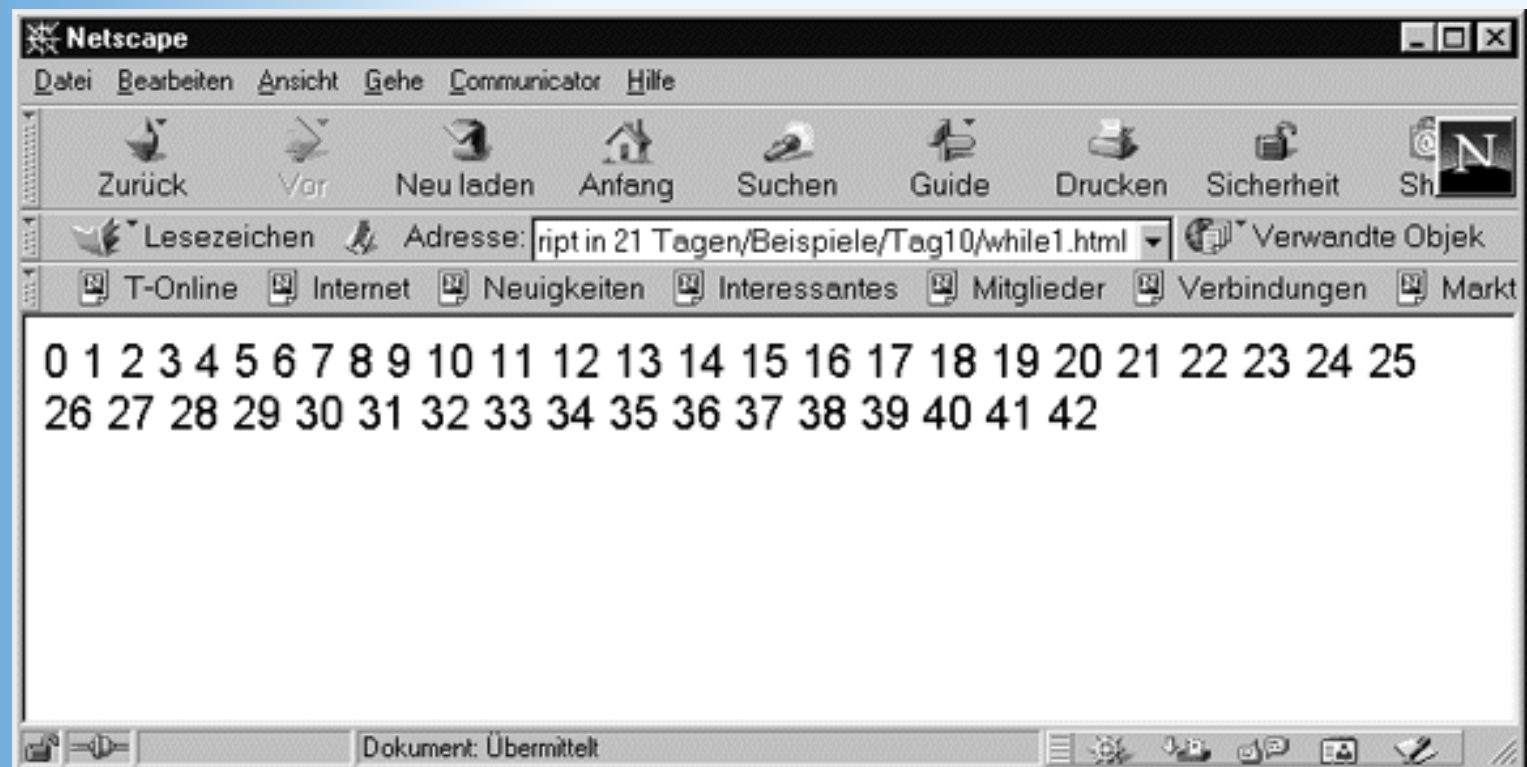


Abbildung 10.13: Die Anzeige bei Eingabe 42

Das nächste Beispiel zählt nicht bei jedem Durchlauf eine Zählvariable mit, sondern ruft eine andere Funktion auf, die in einer if-Anweisung die Übereinstimmung mit einem gewissen Wert prüft. Je nach Übereinstimmung wird mit `return` ein Rückgabewert geliefert, der dann die aufrufende Schleife abbricht oder nicht.

Beispiel 9:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var test=true;
function pruefePW(pw)
{
if(pw=="geheim") return false;
alert("Passwort leider nicht richtig");
return true;
}
function lade()
{
while(test)
{
test=pruefePW(prompt("Geben Sie das Passwort ein",""));
}
document.write("<H1>Willkommen</H1>");
}
</SCRIPT>
</HEAD>
<BODY onLoad="lade()">
</BODY>
</HTML>
```

Speichern Sie das Script unter while2.html und laden Sie es in einen Browser. Die while-Schleife überprüft eine Boolean-Variable, ob sie den Wert true hat. Die Zeile

```
while(test)
```

ist eine oft zu findende Kurzschreibweise für dies:

```
while(test==true)
```

Im Inneren der Schleife wird diese Variable nicht direkt verändert, sondern das vom Anwender eingegebene Passwort an eine Funktion als Übergabewert übergeben und dort ein boolscher Rückgabewert generiert, welcher der Testvariablen zugewiesen wird. Nur wenn Sie das Passwort geheim eingeben, wird die Webseite aufgebaut. Bei jeder anderen Eingabe erscheint die Meldung, dass das Passwort nicht stimmt. Es ist auch nicht möglich, die Schleife über den **Abbrechen**-Button des prompt()-Fensters zu beenden. Dieser beendet ja nur das Fenster, aber die Situation zum Abbrechen der Schleife ist nicht gegeben.



Wenn Sie logische Fehler in den Abbruchbedingungen der Schleife haben und diese nie erfüllt sind, erzeugen Sie eine Endlosschleife. In Verbindung mit Mitteilungs- oder Dialogfenstern kann das fatal sein. Der Browser lässt sich nicht mehr schließen. Sie müssen entweder den Task-Manager einsetzen oder gar den Rechner neu starten.

Eine weitere sinnvolle Möglichkeit, eine while-Schleife abubrechen, ist der Einsatz von break.

Beispiel 10:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var pw;
function lade()
{
while(true)
{
pw=prompt("Geben Sie das Passwort ein","");
if(pw=="geheim") break;
}
document.write("<H1>Willkommen</H1>");
}
</SCRIPT>
</HEAD>
<BODY onLoad="lade()">
</BODY>
</HTML>
```

Speichern Sie das Script unter while3.html und laden Sie es in einen Browser. Die while-Schleife überprüft bei diesem Beispiel überhaupt keine Variable oder sonst irgendetwas. Die Bedingung hat immer den Wert true, denn dieser ist als Konstante dort eingegeben. Syntaktisch ist das vollkommen okay und ein beliebter Trick, wenn man Endlos-Schleifen bewusst programmieren möchte (beispielsweise für eine Diashow, die sich endlos wiederholt). Für unseren Fall wollen wir jedoch die Schleife auch irgendwann abbrechen. Da keine Variable überprüft wird, macht ein Setzen von Variablen im Inneren der Schleife auch keinerlei Sinn. Die Zeile

```
if(pw=="geheim") break;
```

überprüft die Anwendereingabe und ruft bei Übereinstimmung die Sprung- Anweisung break auf. Diese bricht unmittelbar die umgebende Schleife ab.

Testen wir nun noch die Besonderheit der do-while-Schleife.

Beispiel 11:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var test=100;
function lade()
{
do
{
document.write("<H1>Willkommen</H1>");
test++;
}
while(test<1)
}
```



```
</SCRIPT>
</HEAD>
<BODY onLoad="lade()">
</BODY>
</HTML>
```

Speichern Sie das Script unter dowhile.htm und laden Sie es in einen Browser. Die Schleife wird einmal durchlaufen, obwohl die auf 100 gesetzte Testvariable sicher nicht der Bedingung, kleiner als 1 zu sein, genügt.

Die for-Schleife

Die letzte Programmfluss-Kontrollanweisung ist die for-Schleife. In dieser Kontrollstruktur wird eine vorgegebene Anzahl von Durchgängen festgelegt. So oft wie angegeben, werden die Anweisungen im Inneren der Schleife ausgeführt. Bei jedem Durchlauf wird die Zählvariable um eine Einheit (deren Größe in der Schleife als Letztes angegeben wird) hochgezählt. Grundsätzliche Syntax:

```
for ([Zählvar];[Bedingung];[Schrittweite der Zählvar])
{
...Anweisungen
}
```

- **Zählvar** ist die Deklaration der Zählvariablen mit gleichzeitiger Zuweisung eines Startwertes.
- **Bedingung** definiert die Bedingung, wann die Wiederholung der Anweisungen abgebrochen wird.
- **Schrittweite der Zählvar** ist eine Anweisung die angibt, wie die Zählvariable zu verändern ist. Das kann sowohl ein positives, aber auch ein negatives Intervall sein. Abhängig davon sollte die Abbruchbedingung sinnvoll gewählt werden. Das bedeutet, bei einem positiven Intervall sollte der Abbruchwert größer als der Startwert sein (Prüfung der Bedingung auf »kleiner«), bei einem negativen Intervall sollte der Abbruchwert dagegen kleiner als der Startwert sein (dann muss die Bedingung auch auf »größer als« prüfen).

Die Schleifenform eignet sich hervorragend zur Initialisierung von Arrays (darauf kommen wir noch) oder zur Beeinflussung von Situationen, wo gleichartige Dinge eine vorgegebene Anzahl-mal ablaufen.



Diese Struktur ist ein wichtiger Fall, in dem oft mit so genannten schleifenlokalen Variablen gearbeitet wird. Das bedeutet, die Zählvariable wird über die for-Anweisung deklariert und ist dann auch nur dort vorhanden.

Beispiel 12:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var zaehler;
var eingabe;
eingabe=prompt("Geben Sie bitte eine Zahl zwischen 10 und 50 ein","");
```

```
for(zaehler=0;zaehler<=eingabe;zaehler++)
{
document.write(zaehler + " ");
}
document.write("<P>");
for(zaehler=10;zaehler<=eingabe;zaehler=zaehler + 2)
{
document.write(zaehler + " ");
}
document.write("<P>");
for(zaehler=100;zaehler>=eingabe-100;zaehler=zaehler -7)
{
document.write(zaehler + " ");
}
document.write("<P>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter for1.htm und laden Sie es in einen Browser. Mit dem Wert, den der Anwender eingibt, wird der Endwert des jeweiligen Schleifendurchlaufs ermittelt. Dies kann auch über irgendwelche Berechnungen erfolgen.

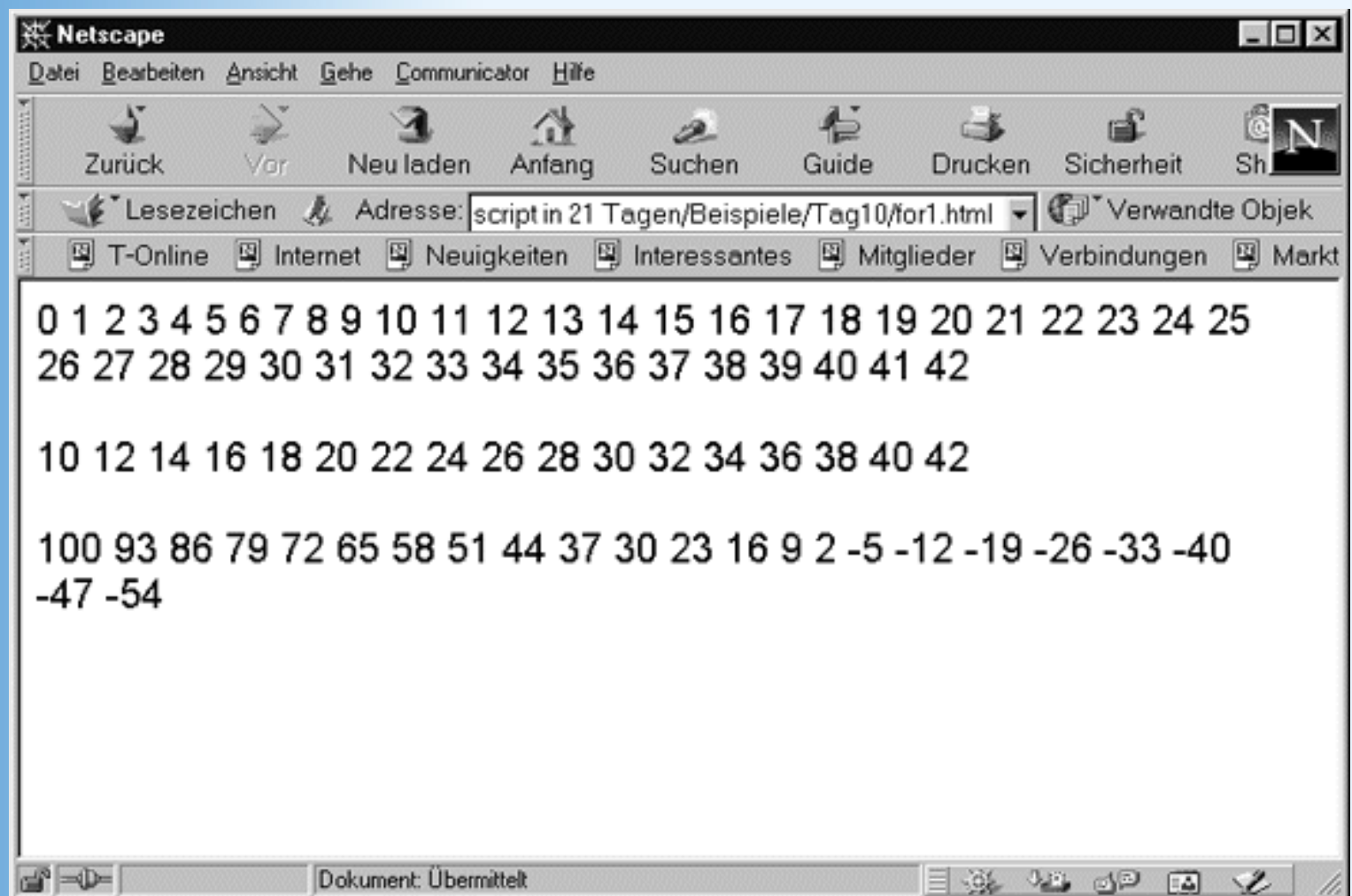


Abbildung 10.14: Die Anzeige bei Eingabe 42

Die Zeile

```
for(zaehler=0;zaehler<=eingabe;zaehler++)
```

zählt von einem Startwert 0 jeweils in Einerschritten bis zu dem Endwert, den der Anwender eingegeben hat. Dabei wird für die Intervallgröße der Inkrement-Operator eingesetzt.

Die Zeile

```
for(zaehler=10;zaehler<=eingabe;zaehler=zaehler + 2)
```

zählt von einem Startwert 10 jeweils in Zweierschritten bis zu dem Endwert, den der Anwender eingegeben hat. Das sind also weniger Durchläufe als bei der ersten Schleife.

Die Zeile

```
for(zaehler=100;zaehler>=eingabe-100;zaehler=zaehler-7)
```

zählt von einem Startwert 100 rückwärts jeweils in Siebenerschritten bis zu dem Endwert, den der Anwender eingegeben hat, minus 100. Die Überprüfung der Bedingung muss hier auf »Kleiner« bzw. »Kleiner gleich« erfolgen.



Ein Spezialfall der for-Schleife ist die for...in-Schleife. Wir werden diese im Zusammenhang mit Arrays an Tag 15 kennen lernen.

Kombinierte Anwendungen von Kontrollfluss-Anweisungen

In vielen etwas anspruchsvolleren Situationen muss man Kontrollfluss-Anweisungen kombinieren. Das kann über Verbinden von Bedingungen mit logischen Operatoren erfolgen, aber auch über die Kombination von mehreren Kontrollfluss-Anweisungen. Das haben wir etwa im Beispiel 10 gesehen - innerhalb einer while-Schleife wird da eine if-Abfrage eingebaut. Spielen wir noch ein Beispiel dieser Art durch, wo eine for- Schleife im Inneren eine if-Abfrage eingebaut hat.

Beispiel 13:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
var zaehler;
for(zaehler=0;zaehler<=100;zaehler++)
{
if((zaehler*2)%2!=(zaehler%2)*2) continue;
document.write(zaehler + " ");
}
document.write("<P>");
</SCRIPT>
</HEAD>
```

```
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter for2.htm und laden Sie es in einen Browser. Das Beispiel zeigt einen kleinen Trick, wie man gerade und ungerade Zahlen trennen kann. Die Anweisung

`(zaehler*2)%2`

ist für alle Zahlen 0. Dahingegen ist die Anweisung

`(zaehler%2)*2`

nur für gerade Zahlen 0, für ungerade 2. Die Zeile

```
if((zaehler*2)%2!=(zaehler%2)*2) continue;
```

vergleicht, ob die beiden Werte ungleich sind. Falls ja, wird die Anweisung continue ausgelöst, die Sie hier nun auch in der Praxis sehen. Unmittelbar wird ein neuer Schleifendurchlauf ausgelöst, und die Ausgabe jedes ungeraden Wertes unterbleibt.

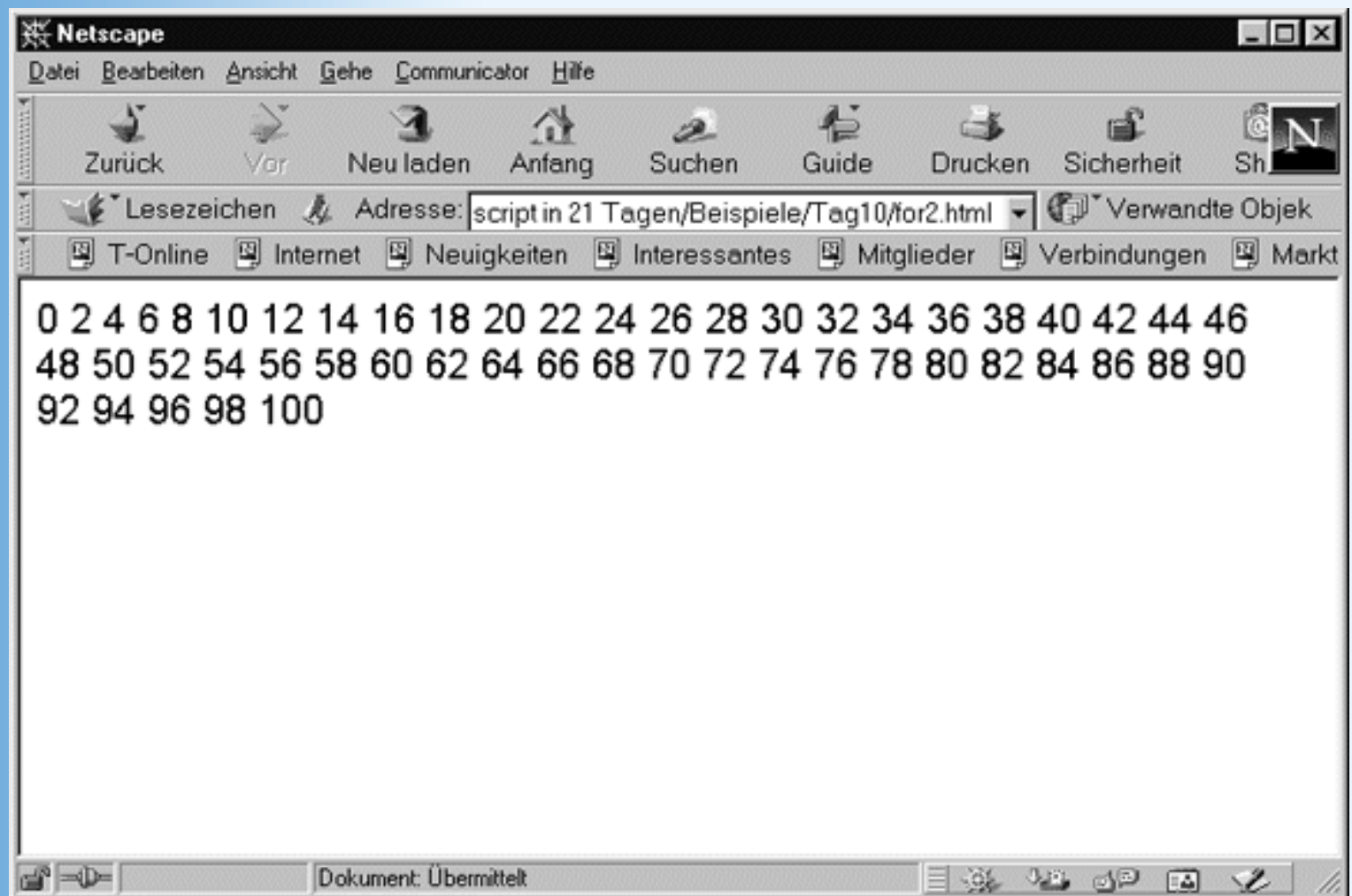


Abbildung 10.15: Nur gerade Zahlen werden ausgegeben

Auch das Verschachteln von Kontrollfluss-Anweisungen gleichen Typs findet man oft. Beispielsweise verschachtelte for-Schleifen, wie im nächsten Beispiel:

Beispiel 14:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
for(zaehler=0;zaehler<=10;zaehler++)
{
for(i=0;i<3;i++)
document.write(zaehler + " ");
document.write("<BR>");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter for3.htm und laden Sie es in einen Browser.

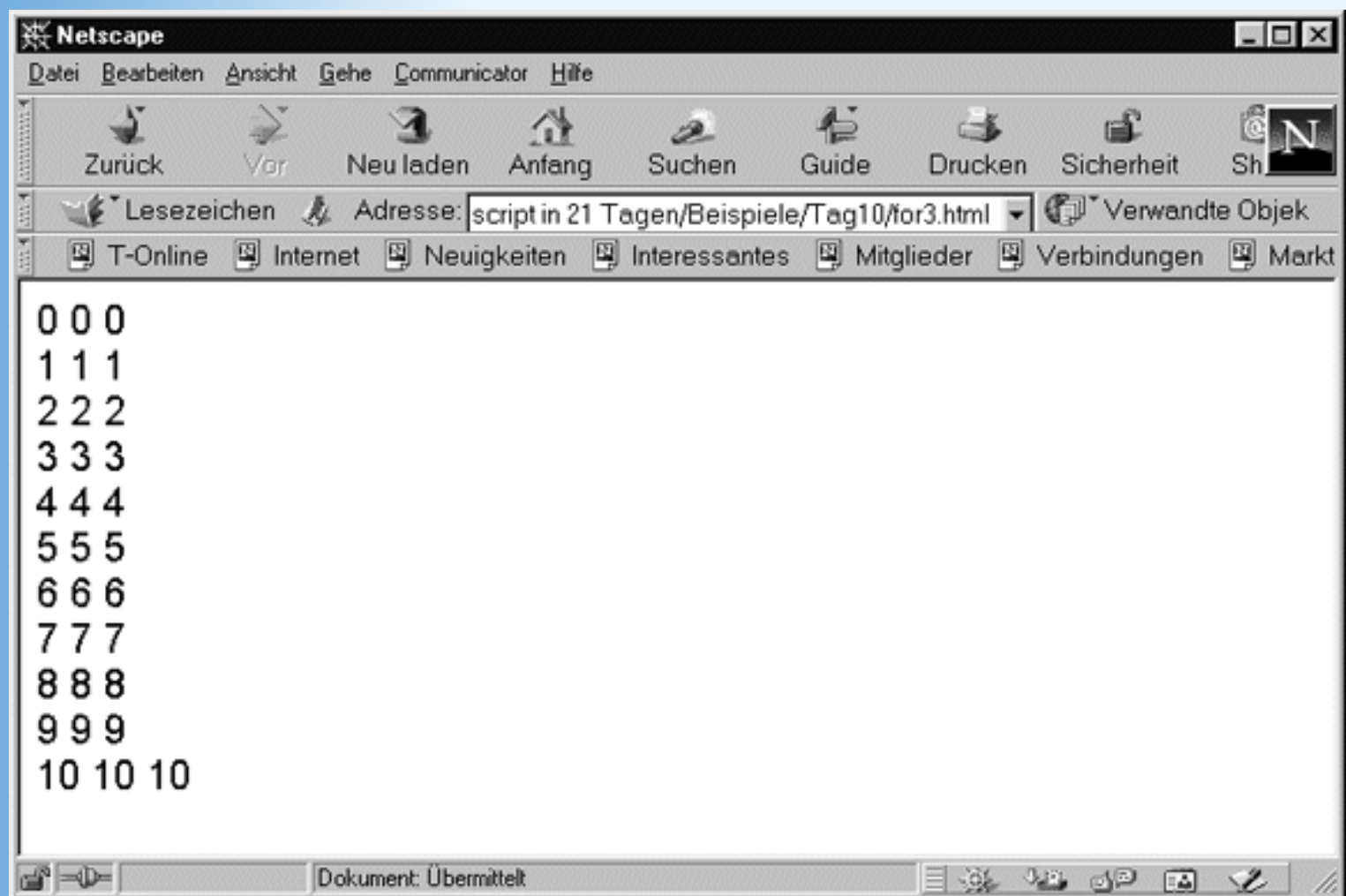


Abbildung 10.16: Die Wirkung von verschachtelten for-Schleifen

Bei jedem Durchlauf der äußeren for-Schleife wird die innere Schleife dreimal ausgeführt.

Zusammenfassung

Am heutigen Tag standen Anweisungen auf dem Programm. Insbesondere Kontrollfluss-Anweisungen, oft in Verbindung mit Sprung-Anweisungen bilden das Rückgrad von Scripten, die auf unterschiedliche

Situationen dynamisch reagieren können. JavaScript kennt folgende Anweisungen zur Kontrollfluss-Steuerung:

- if bzw. if-else
- switch-case
- while und do-while
- for
- return
- break
- continue

Fragen und Antworten

Frage:

Warum muss die break-Anweisung in einer switch-case-Anweisung verwendet werden, wenn die Blöcke nach dem Treffer nicht abgearbeitet werden sollen? Man könnte die Anweisung in der Konzeption von JavaScript doch auch so bauen, dass dies automatisch erfolgt.

Antwort:

Das stimmt natürlich, aber so, wie es konzipiert ist, kann man es auch sinnvoll einsetzen. Bei einer entsprechenden Anordnung der case-Blöcke kann man beispielsweise ab einem gewissen Treffer alle nachfolgende Schritte ausführen lassen. Etwa um Dinge mitzuzählen.

Frage:

Unterscheiden sich Anweisungen in JavaScript stark von Anweisungen in anderen Sprachen?

Antwort:

Nein. Es gibt nur wenige Details, wo sich Programmier- und Scriptsprachen in diesem Bereich unterscheiden. Eine Anweisung, wo es zwischen verschiedenen Sprachen Unterschiede gibt, ist die switch-case-Anweisung.

Frage:

Können bei verschachtelten for-Schleifen gleiche Namen für die Zählvariablen in den ineinander verschachtelten Schleifen verwendet werden?

Antwort:

Nein. Sie müssen sich unterscheiden.

Workshop

Experimentieren Sie mit den in diesem Kapitel durchgesprochenen Beispielen. Insbesondere die Verschachtelung von Anweisungen ist nicht ganz trivial und sollte ausführlich geübt werden.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am

Ende einer Lektion notierten Fragen.

1. Stimmt die Syntax?

```
for (zaehler==0; zaehler<=10; zaehler++)
```

2. Gibt es die Sprunganweisung Return?

3. Stimmt die Syntax?

```
for (zaehler=0; zaehler<=10; zaehler)
```

4. Stimmt die Syntax?

```
if (i<10)
```

5. Was ist der Unterschied zwischen break und continue?

6. Stimmt die Syntax?

```
while (a=8)
```

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 2

Tag 11

Der Aufruf von JavaScripten

Heute wollen wir uns genauer mit den Situationen auseinandersetzen, in denen JavaScripte ausgeführt werden. Wenn ein Script programmiert ist, muss dieses Script vom Browser ausgeführt werden, um zum Leben zu erwachen. Aber wann? Zufällig? Oder in einer bestimmte Situation, die Sie als Programmierer festlegen? Sicherlich das Letztere. Etwa beim Laden einer Webseite. Das Ausführen eines Scripts beim Laden erfolgt immer, wenn in einer Webseite direkt JavaScript-Anweisungen oder Funktionsaufrufe notiert werden.

Grundsätzlich ist das Laden einer Webseite jedoch nur eine von vielen Situationen, in denen die Ausführung einer Scriptfunktionalität sinnvoll ist. Sinnvolle andere Situationen können aber auch das sein:

- Das Verlassen einer Webseite
- Der Klick auf einen Hyperlink
- Der Klick auf einen Button in einem Formular
- Der Doppelklick auf eine Grafik
- Das Überstreichen einer Fläche mit dem Mauszeiger
- Das Verlassen eines Eingabefelds in einem Formular

Es gibt noch zahlreiche weitere Ereignisse, wo man sich vorstellen kann, dass bei deren Auftreten ein Script gezielt ausgeführt wird.

Das Zauberwort, was die gesamte Thematik näher fasst, steckt auch bereits in dem vorherigen Satz (ganz unscheinbar) drin - *Ereignis*. Das ist es, um was es heute im Wesentlichen geht: Ereignisse in Webseiten und wie man diese verwenden kann, um gezielt JavaScript-Funktionalitäten auszuführen.

Was ist ein Ereignis?

Während des »Lebens« einer Webseite, aber auch des Ablaufs eines Programms/Scripts, treten immer wieder spezifizierbare Situationen auf, die man konkret fassen kann. Beispielsweise:

- Ein Klick auf eine genau über die Mauskoordinaten spezifizierbare Stelle
- Ein Doppelklick auf eine genau über die Mauskoordinaten spezifizierbare Stelle
- Ein Tastendruck
- Eine Mausbewegung, wobei man genau die Mauskoordinaten unter Kontrolle hat
- Das Verlassen eines Eingabefeldes
- Automatische Vorgänge, die von einer Zeitspanne oder einem Zeitpunkt abhängen oder aufgrund von gewissen anderen Bedingungen ausgelöst werden

Wenn ein komplexeres Programm/Script (besonders in einer grafischen Umgebung) abläuft, wird es nur selten so programmiert sein, dass sämtliche Schritte vordefiniert ohne Interaktion mit der Umgebung ablaufen können. Insbesondere der Anwender greift bei umfangreicheren Programmen in den Ablauf ein. Etwa wenn er ein Fenster vergrößert, eine Schaltfläche drückt oder eine Taste bestätigt. Jede dieser Situationen kann man in einer Webseite nutzen und gezielt eine Reaktion darauf erfolgen zu lassen. Diese weiter verwendbaren Situationen werden Ereignisse oder englisch Events genannt. Unter einem Ereignis kann man speziell auf eine HTML-Seite bezogen alles verstehen, was während der Lebenszeit einer Webseite in einem Browser auftreten kann. Vom Laden in den Browser bis zum Entfernen der Seite aus dem Browser - Letzteres etwa beim Laden einer neuen Seite, aber auch beim Beenden des Browsers. Jede spezifizierbare Situation sorgt für das Eintreten eines Events, eines Ereignisses, das im System beobachtet und worauf reagiert werden kann. Die Reaktion auf solche Events wird Ereignisbehandlung oder Eventhandling genannt.



Die Ereignisbehandlung weicht in Netscape-kompatiblen Browsern von der im Internet Explorer erheblich ab. Dies liegt unter anderem daran, dass es verschiedene Ereignismodelle von den Herstellern der Scriptsprachen und der Browser gibt und HTML teils vollkommen verschieden ausgelegt wird. Dazu kommt, dass die Ereignisbehandlung von beiden Protagonisten in einem globalen Konzept gesehen wird. Dies ist jeweils in ein abstraktes Objektmodell eingebunden, das festlegt, wie auf welche Ereignisse zu reagieren ist, was passiert, wenn ein Ereignis nicht gleich ausgeführt werden kann und so fort.

Die Unterschiede zwischen dem Microsoft-Ereignismodell und dem von Netscape sind ziemlich groß. Netscape weicht in seinem Modell sehr stark von dem offiziellen W3C-Entwurf ab (das W3C-Konsortium ist u.a. verantwortlich für die Standardisierung von HTML - <http://www.w3.org>), während Microsoft die Ereignisbehandlung weitgehend so interpretiert, wie vom W3-Konsortium vorgesehen. An Tag 17 werden wir die Spezifika des Netscape-Modells noch mal in Zusammenhang mit einem globalen Ereigniskonzept durchsprechen.

Die klassische Form der Reaktion auf Ereignisse

In der althergebrachten Programmierung wurde durch jedes Programm, das auf bestimmte Ereignisse zu reagieren hatte, für jede Situation eine spezielle Funktion oder Prozedur bereitgestellt. Etwa wenn an einer bestimmten Stelle im Programmablauf auf einen Tastendruck durch den Anwender gewartet werden musste, wurde dort eine individuelle Wartestruktur erstellt. Beispielsweise so (Turbo-Pascal-Syntax):

```
REPEAT UNTIL KEYPRESSED
```

So ein Verfahren hat zahlreiche Schwachstellen. Was ist beispielsweise, wenn ein Anwender mit der Maus klickt, also etwas anderes tut, als dort vorgesehen? Jede denkbare Situation muss nachprogrammiert und entsprechend behandelt werden. Aber schlimmer noch ist die Tatsache, dass hierbei eine lineare Programmabarbeitung vorausgesetzt wird. Wie wollen Sie in diesem Konzept eine Webseite realisieren, die beispielsweise drei Buttons beinhaltet, auf die der Anwender in willkürlicher Reihenfolge klicken kann? Sie benötigen dazu eine Schleife, die endlos wiederholt wird und nichts anderes tut, als auf Benutzereingaben zu warten. Dazu muss noch parallel das möglich sein, was sonst so noch an Aktivitäten getan werden soll. Etwa das Berechnen von Werten auf Grund von Eingaben eines Anwenders in Eingabefelder oder so.

Der Ansatz ist von der Idee her auch ungefähr das, was beim Eventhandling realisiert wurde, nur ist es dort programmiertechnisch erheblich verbessert.

Die moderne Form mit einem Ereignis-Modell

In vielen der modernen, objektorientierten Programmiersprachen wird ein spezieller, standardisierter Mechanismus bereitgestellt, der auf Ereignisse im Rahmen eines Programmablaufs reagiert und gegebenenfalls eine Funktion oder eine Objektmethode auslöst. Wesentlicher Unterschied im Denkkonzept ist, dass dieses »Hören auf das Auftreten von Ereignissen« parallel zum normalen Programmablauf und vor allem im Hintergrund läuft. Es muss nicht jedes Mal neu programmiert werden. Man sagt im Fall einer Webseite dem Browser praktisch nur noch: »pass auf das Ereignis X auf« und erstellt den Rest der Webseite samt JavaScript-Elementen.

Es gibt nun verschiedene Ansätze, wie man auf Ereignisse aufpassen kann. Der gesamte Mechanismus kann beispielsweise im Rahmen einer Webseite über die Verwendung von Event-Handle (Ereignisbehandlern) realisiert werden. Dort werden Ereignisse als spezielle Objekte gesehen, auf die über standardisierte Elemente zugegriffen werden kann und die im Hintergrund beim Auftreten einer bestimmten Situation erzeugt werden.

Seit HTML 4.0 gibt es universelle Event-Handler für Ereignisse in einer Webseite. Es handelt sich einfach um Attribute, die direkt in einen HTML-Tag geschrieben werden. Das W3-Konsortium hat Event-Handler offiziell in den HTML-Sprachstandard mit aufgenommen. Dabei wurde auch festgelegt, in welchen HTML-Tags welcher Event-Handler vorkommen darf (zumindest von der Theorie her - was die Browser-Hersteller daraus machen, ist ein anderes Thema). Event-Handler sind ein wichtiges Bindeglied zwischen HTML und JavaScript, aber da sie eindeutig zu HTML zu zählen sind, spielt damit Groß- und Kleinschreibung bei der Notation in der Webseite keine Rolle. Die Grundsyntax der Verwendung von Event-Handle (n) sieht immer gleich aus:

```
<[HTML-Tag] [ATTRIBUTE]  
[onEventHandler]="Script-Anweisungen">
```

- Dabei ist [HTML-Tag] ein beliebiger HTML-Tag, welcher die Reaktion auf ein Ereignis unterstützt (das unterscheidet sich wie gesagt in verschiedenen Browsern massiv, weshalb wir darauf noch genauer eingehen müssen).
- [ATTRIBUTE] sind die normalen HTML-Attribute, welche den HTML-Tag bezüglich seiner üblichen HTML-Auswirkungen spezifizieren.
- [onEventHandler] ist der konkrete Name des Ereignisses, auf welches reagiert werden soll, und »Script-Anweisungen« sind die tatsächlichen Script-Anweisungen, welche bei Auftreten des spezifizierten Ereignisses ausgeführt werden sollen. Ein Event-Handler-Ereignis sieht aus wie jedes gewöhnliche HTML-Attribut. Gemeinhin beginnen die Namen von Ereignissen mit der Silbe on (kleingeschrieben), gefolgt von einer sprechenden Beschreibung des Ereignisses. Der besseren Lesbarkeit wegen wird meistens jeder eigenständige Begriff im Bezeichner mit einem Großbuchstaben begonnen (etwa onMouseOver, onLoad, onUnload, onClick), aber das ist nicht zwingend (es ist wie gesagt pures HTML). Hinter dem Namen des Ereignisses wird ein

Ist-gleich- Zeichen notiert, gefolgt von einer - meist zur Sicherheit in Anführungszeichen notierten - JavaScript-Anweisung oder -Funktion. In eine Event-Handler-Struktur können über Semikolon abgetrennt mehrere JavaScript-Anweisungen hintereinander geschrieben werden. Dies ist aber selten sinnvoll. Besser ist die Erzeugung einer eigenen JavaScript-Funktion, welche dann alle Anweisungen enthält, welche bei Auftreten eines Ereignisses ausgeführt werden sollen. Wir beschränken uns in der Folge auf diese Variante.

Aufruf beim Start/Verlassen der HTML-Datei

Der Aufruf einer JavaScript-Funktion beim Start der HTML-Datei ist der Fall, der in unseren bisherigen Beispielen am häufigsten mit dem Ausführen eines Scripts in Verbindung gebracht wurde. Es gibt nun zwei Varianten, wie das realisiert werden kann:

1. Mit dem Event-Handler onLoad als einer Erweiterung des <BODY>-Tags und Einschluss aller gewünschten Anweisungen in eine JavaScript- Funktion, die mit dem Event-Handler aufgerufen wird.
2. Indem die auszuführenden Script-Anweisungen einfach in einen <SCRIPT>-Container notiert werden.

Die Syntax beim Verwenden des Event-Handlers ist folgende:

```
<BODY onLoad="[Funktionsname]">
```

Damit erfolgt der Aufruf der als Wert des Attributes angegebenen JavaScript-Funktion beim Start der HTML-Datei. Die Variante mit dem Event-Handler hat einige Vorteile:

- Egal, wo die Funktion notiert ist - sie wird in dem Moment geladen und begonnen auszuführen, wo der <BODY>-Tag geladen wird. Nicht früher und nicht später. Das kann in einigen Situationen von Vorteil sein.
- Deutliche Kennzeichnung, dass beim Laden der Webseite eine Funktion ausgeführt wird und welche. In großen Seiten kann das sonst nicht ganz leicht zu sehen sein.
- Es gibt in der Regel weniger Probleme mit globalen Variablen (es gibt Situationen, wo insbesondere der Netscape Navigator mit globalen Variablen einige Mätzchen macht - wir haben das ja bereits in einigen Beispielen explizit getestet).

Allerdings hat auch die direkte Notation des Funktionsaufrufs in einen <SCRIPT>-Container durchaus Vorteile gegenüber dem Event-Handler- Einsatz. Vor allen Dingen, wenn man sämtlichen JavaScript-Code in Funktionen packt und nur den Aufruf einer Funktion - die man ansonsten im Event-Handler angeben würde - als direkt auszuführenden Code im <SCRIPT>-Container notiert (dann entfällt auch das oft für die Verwendung von Event-Handlern angeführte Argument, man könne an beliebigen Stellen die Funktion bei Bedarf erneut aufrufen - das geht dann natürlich auch):

- Weniger Code (wenngleich unwesentlich), denn onLoad= entfällt.
- In einigen Situationen wird der Event-Handler nicht richtig ausgeführt. Beispielsweise in Verbindung mit Frames und dynamisch durch JavaScript geschriebenen Webseiten wird es beim Netscape Navigator (und damit kompatiblen Browsern wie dem Opera) und dem onLoad- Event-Handler Probleme geben, wenn nach dem Laden der Seite die Größe des Browser-Fensters verändert wird (der Internet Explorer macht den Fehler nicht). Wenn in einer in einem Frame geladenen Webseite mit onLoad eine Funktion aufgerufen wird, welche die Seite dynamisch schreibt, wird der Frame-Inhalt beim Verändern der Größe neu aufgebaut, die Funktion jedoch nicht mehr ausgeführt (wir absolvieren das gleich in einem Beispiel). Die direkte Notation des Script-Aufrufs verhindert das Problem.
- Event-Handler sind erst seit HTML 4.0 offiziell und setzen damit einen Browser voraus, der diese Version auch versteht. Die direkte Notation wird von mehr Browsern verstanden.



Frames bedeuten die Aufteilung des Anzeigebereichs eines Browsers in einzelne Segmente, die unabhängig mit Inhalt gefüllt werden können. Die Frame-Technologie besteht aus zwei Ebenen. Mehrere Frames zergliedern den Anzeigebereich des Browsers. Sie bilden dennoch eigenständige Fenster mit allen HTML-Darstellungsmöglichkeiten, die dann jeweils die zweite Ebene - HTML-Dateien mit dem anzuzeigenden Inhalt - enthalten. Die Aufteilung des Browser-Anzeigefensters erfolgt über eine spezielle HTML-Datei, eine Indexdatei, welche die erste Ebene eines Projektes mit Frames bildet und die in den Browser geladen werden muss. Diese Indexdatei enthält nur Informationen über Name, Größe und Position der einzelnen im Anzeigefenster enthaltenen Frames. Insbesondere ist der eigentliche Inhalt der Frames nicht in der Indexdatei notiert. Eine oft anzutreffende Aufteilung von Frame-Strukturen besteht aus einem kleineren Frame, welches eine Indexdatei oder eine Inhaltsangabe mit lokalen Verweisen enthält, und einem größeren Frame-Bereich, welcher die eigentlichen Informationen anzeigt. Wir kommen noch genauer auf Frames und die Verbindung zu JavaScript zurück.

Das erste Beispiel soll einfach die Verwendung von onLoad zeigen. Dabei koppeln wir die Demonstration mit dem Parallelereignis zum Laden einer Seite - das ist die Situation, wenn eine Seite wieder aus dem Speicher entfernt wird. Etwa wenn eine neue Seite geladen wird. Dafür gibt das Ereignis onUnload, das analog in den <BODY>-Tag integriert wird. Sie können beide Ereignisse in dem <BODY>-Tag angeben, um sowohl beim Start einer Seite als auch bei deren Verlassen eine bestimmte Aktion auszulösen.

Beispiel 1:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function wk()
{
  alert("Herzlich Willkommen");
}
function bey()
{
  alert("Und tschüss");
}
-->
</SCRIPT>
</HEAD>
<BODY onLoad="wk()" onUnload="bey()">
</BODY>
</HTML>
```

Speichern Sie das Script unter onLoadUnLoad.htm und laden Sie es in den Internet Explorer. Sie werden mit einem Mitteilungsfenster begrüßt.



Abbildung 11.1: So werden Sie beim Laden der Seite begrüßt

Schließen Sie den Browser wieder. Eine Meldung erscheint zum Abschied.



Abbildung 11.2: Beim Verlassen der Seite



Der Netscape Navigator führt onUnload nicht aus, wenn der Browser geschlossen wird. Nur wenn Sie vorher eine Seite geladen hatten und wieder über die History zurückgehen oder eine neue Seite im Browser laden, wird dieser Event-Handler ausgeführt. Dieses Verhalten passt von der Art her zu dem etwas weiter oben angegebenen Problem, das der Navigator mit dem onLoad-Event-Handler hat (siehe das nächste Beispiel).

Wir wollen nun das Problem demonstrieren, welches der Navigator mit dem onLoad-Event-Handler hat. Dazu müssen wir ein wenig vorgehen und Frames verwenden (die intensivere Erklärung von Frames kommt noch). Erstellen Sie einfach die drei Dateien so, wie in dem Beispiel vorgegeben.

Beispiel 2:

Erstellen Sie zuerst die Datei Bild.htm (sie soll auf der linken Seite im Browserfenster angezeigt werden):

```
<HTML>
<BODY>
<IMG src="buch.gif">
</BODY>
```

</HTML>

Erstellen Sie nun die Datei Tagesdat.htm, die dynamisch eine Webseite schreibt (so etwas hatten wir an Tag 6 schon mal sehr ähnlich):

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function schreibeSeite()
{
var datum = new Date();
var tag = datum.getDate();
if(tag<10)
{
tag="0" + tag;
}
var monat = datum.getMonth() + 1;
if(monat<10)
{
monat="0" + monat;
}
var jahr = 2000 + (datum.getYear()%100);
document.write("<HTML>");
document.write("<BODY bgcolor=blue text=yellow>");
document.write("Aktueller Stand: ");
document.write(tag);
document.write('.');
document.write(monat);
document.write('.');
document.write(jahr);
document.write("</P>");
document.write("</BODY>");
document.write("</HTML>");
}
</SCRIPT>
</HEAD>
<BODY onLoad="schreibeSeite()">
Sorry, diese Seite verwendet JavaScript!
</BODY>
</HTML>
```

Geben Sie als Drittes folgenden Quelltext ein:

```
<HTML>
<FRAMESET cols="1%,3%" framespacing="0"
border="0" frameborder="0">
<FRAME name="Links" src="Bild.htm">
<FRAME name="Rechts" src="Tagesdat.htm">
<NOFRAMES>
<BODY>
<p>Diese Seite verwendet Frames.
Frames werden von Ihrem Browser aber nicht unterstützt.
</p>
</BODY>
</NOFRAMES>
</FRAMESET>
</HTML>
```

Speichern Sie diese Datei - ein Frameset - unter index.htm und laden Sie diese dann im Netscape Navigator. Sie sollten links ein animiertes GIF und rechts eine Datumsangabe in roter Schrift auf blauem Hintergrund sehen.

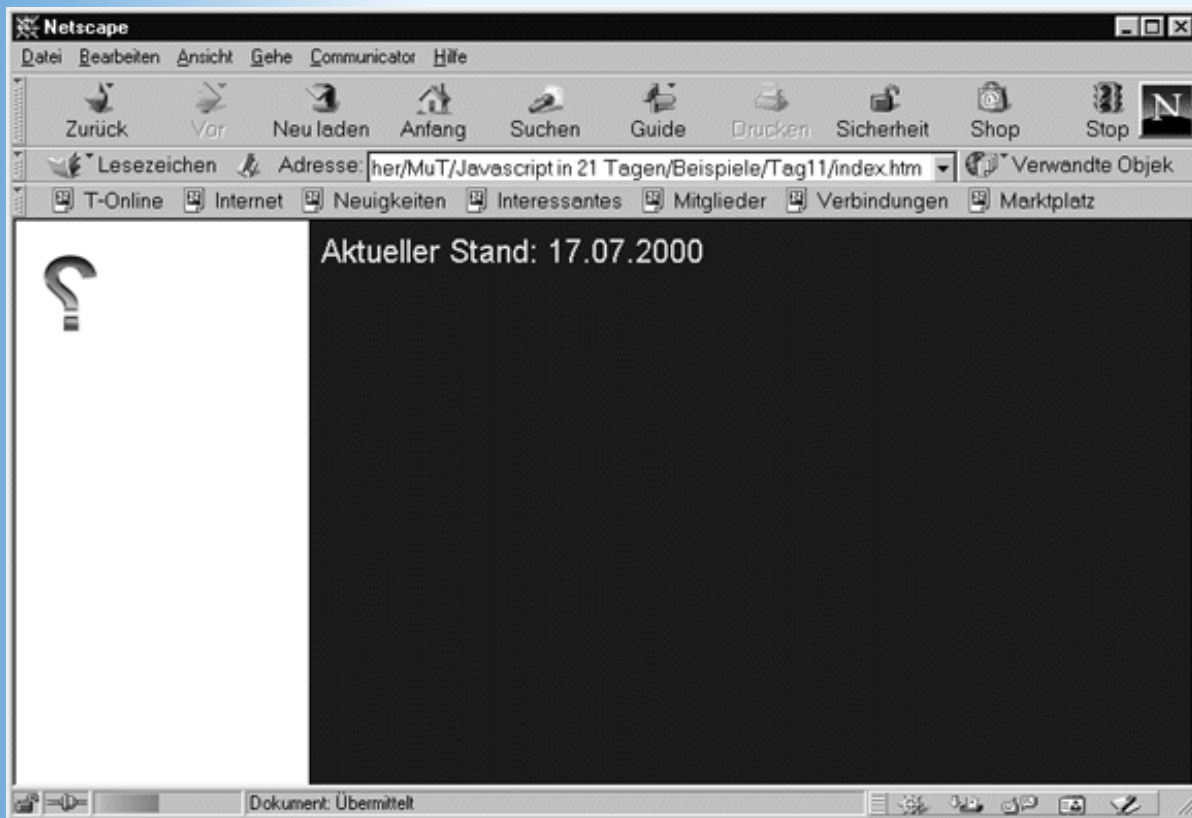


Abbildung 11.3: So sollte die Seite aussehen

Verändern Sie nun die Größe des Browsers.



Abbildung 11.4: Die Seite wird nicht mehr richtig aufgebaut

Die Seite wird nicht mehr richtig aufgebaut, d.h. die JavaScript-Funktion wird nicht ausgeführt. Was angezeigt wird ist der Alternativtext im Body, der eigentlich nur dafür da ist, JavaScript-unfähige Browser darauf hinzuweisen, dass über JavaScript die Seite dynamisch erzeugt wird. Wenn das Script beim Laden der Seite ausgeführt wird, wird dieser Text weggewischt und darf nicht zu sehen sein. Offensichtlich wird die Veränderung der Browsergröße vom Navigator nicht dazu benutzt, das Script über den Event-Handler neu aufzurufen. Das ist deshalb inkonsequent, denn die Seite wird ja neu geladen - sonst würde der Alternativtext nicht erscheinen. Wie es richtig ist, zeigt der Internet Explorer, der keinerlei Probleme mit dieser Situation hat.

Noch schlimmer als der Navigator reagiert jedoch Opera. Dort wird die Seite überhaupt nicht aufgebaut und es erscheint immer nur der Alternativtext.



Abbildung 11.5: Der Opera kommt gar nicht damit zurecht

Der Workaround unter Verzicht auf den Event-Handler sieht so aus (Name der Datei TagesdatWaR.htm):

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
function schreibeSeite()
{
var datum = new Date();
var tag = datum.getDate();
if(tag<10)
{
tag="0" + tag;
}
var monat = datum.getMonth() + 1;
if(monat<10)
{
monat="0" + monat;
}
var jahr = 2000 + (datum.getYear()%100);
document.write("<HTML>");
document.write("<BODY bgcolor=blue text=yellow>");
document.write("Aktueller Stand: ");
document.write(tag);
document.write('.');
document.write(monat);
document.write('.');
document.write(jahr);
document.write("</P>");
document.write("</BODY>");
document.write("</HTML>");
}
</SCRIPT>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
schreibeSeite();
</SCRIPT>
<NOSCRIPT>
Sorry, diese Seite verwendet JavaScript!
</NOSCRIPT>
</BODY>
</HTML>
```

Beachten Sie den Aufruf der Funktion in dem Script-Container im Body und den neuen <NOSCRIPT>-Container, der den Alternativtext beinhaltet. Mit dieser Konstruktion funktioniert die Geschichte (sogar im Opera, obwohl der bei den Frames vom Layout her unter Umständen etwas Probleme macht). Die Datei index.htm muss dann natürlich auch diese neue Seite aufrufen. Sie sieht dann so aus:

```
<HTML>
<FRAMESET cols="1%,3%" framespacing="0" border="0" frameborder="0">
<FRAME name="Links" src="Bild.htm">
<FRAME name="Rechts" src="TagesdatWaR.htm">
<NOFRAMES>
<BODY>
<p>Diese Seite verwendet Frames.
Frames werden von Ihrem Browser aber nicht unterstützt.
</p>
</BODY>
</NOFRAMES>
</FRAMESET>
</HTML>
```

Klick und Doppelklick auf eine Referenz

Die neben dem Laden bzw. Entfernen einer Webseite wohl wichtigste Situation, unter der ein Script ausgeführt werden muss, ist der Klick auf ein entsprechend sensibles Element in der Webseite. Eine Referenz. Unter einem Klick auf eine Referenz ist der Fall gemeint, wenn Sie auf ein Element wie eine Schaltfläche in einem Formular oder einen Hyperlink mit der Maus klicken. Sie erweitern den entsprechenden Tag um das Attribut onClick. Dabei gilt die allgemeine Syntax:

```
<[HTML-Tag] onClick="[Funktionsname]">
```

Dies bedeutet beispielsweise für den Fall eines Hyperlinks, dass die Syntax so aussieht:

```
<A href=[URL] onClick="[Funktionsname]">...</a>
```



Bei dem Event-Handler onClick zeigen sich die Unterschiede zwischen dem Netscape-Ereignismodell und dem von Microsoft unterstützten HTML-4.0-Ereignismodell ziemlich extrem. Während Netscape das Attribut nur in den HTML-Tags <A>, <AREA>, <INPUT> und <TEXTAREA> erlaubt, unterstützt der Explorer nahezu alle offiziellen HTML-4.0-Tags. Konkret sind das derzeit folgende:

<a>, <abbr>, <acronym>, <address>, <area>, , <big>, <blockquote>, <body>, <button>, <caption>, <center>, <cite>, <code>, <col>, <colgroup>, <dd>, <del>, <dfn>, <dir>, <div>, <dl>, <dt>, , <fieldset>, <form>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <i>, , <input>, <ins>, <kbd>, <label>, <legend>, , <link>, <map>, <menu>, <noframes>, <noscript>, <object>, , <optgroup>, <option>, <p>, <pre>, <q>, <s>, <samp>, <select>, <small>, , <strike>, , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <u>, , <var>

Die doch sehr eingeschränkte Unterstützung von Event-Handleern durch Netscape gilt leider nicht nur bei diesem Fall, sondern bei den meisten weiteren Event-Handleern. Manchmal nicht ganz so krass, aber fast immer kann man für eine Netscape-Unterstützung nur eine eng begrenzte Auswahl der offiziell für HTML 4.0 freigegebenen Elemente nutzen. Sie finden im Anhang in der Liste der HTML-Attribute eine Auflistung der HTML-Tags, bei denen der jeweils angegebene Event-Handler stehen darf. Für die meisten Event-Handler sind das fast alle Tags. Der Internet Explorer hält sich so gut wie immer daran. Der Netscape Navigator und kompatible Browser nutzen immer nur eine kleine Auswahl. Wir geben deshalb in der Folge - bis auf Ausnahmen - immer nur die Tags an, die auch Netscape unterstützt. Der Internet Explorer kann diese auf jeden Fall auch. Die Beispiele zeigen gelegentlich überdies spezielle Effekte, die nur der Internet Explorer kann.

Zur Ehrenrettung von Netscape sei gesagt, dass sie bereits vor HTML 4.0 ein Konzept entwickelt haben, das eine universelle Ereignisbehandlung gewährleistet. Dieses Konzept basiert jedoch nicht auf den Event-Handleern, sondern einem Ereignis-Objekt. Als HTML 4.0 eingeführt wurde, hat das W3C sich explizit gegen den Netscape-Entwurf entschieden. Netscape sah sich also gezwungen, einen Teil der Event-Handler zu integrieren, wollte aber nicht ganz auf sein Konzept verzichten. Es wird also vieles, was unter Netscape mit den Event-Handleern nicht funktioniert, mit dem Netscape-Ereignis-Objekt realisierbar sein. Wir gehen darauf noch an Tag 17 ein. Dennoch - die lückenhafte Unterstützung von Event-Handleern macht Netscape wenig Freunde.

Beispiel 3:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
function unsere_funktion()
{
  alert("Herzlich Willkommen");
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<A href="onclick.html" onClick="unsere_funktion()">
<H1>
Irgendwas
</H1>
</A>
</BODY>
</HTML>
```

Speichern Sie das Script unter onclick.htm und laden Sie es in einen Browser. Klicken Sie auf den Hyperlink.

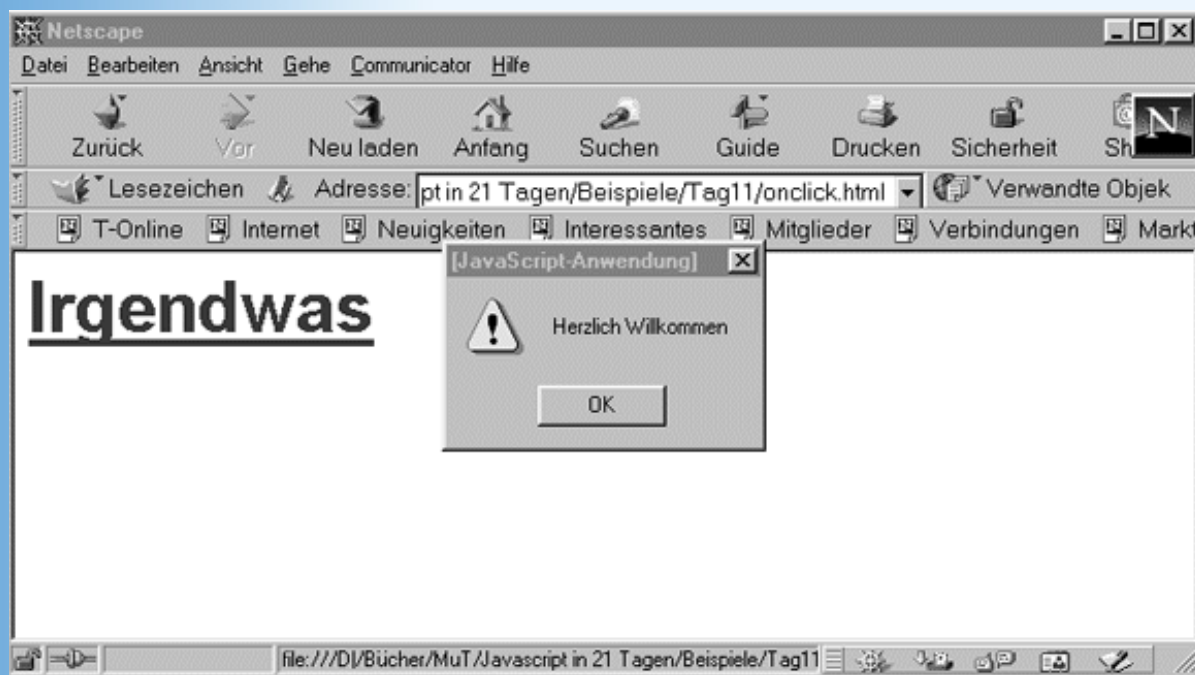


Abbildung 11.6: Beim Klick auf den Hyperlink wird das Script aufgerufen

Beachten Sie die Zeilen

```
<A href="onclick.html" onClick="unsere_funktion()">
<H1>
Irgendwas
</H1>
</A>
```

Das ist der Hyperlink, bei dem das Script ausgelöst werden soll. Um nicht zusätzlich auf eine andere Seite weitergeleitet zu werden (was ja ein Hyperlink als Hauptaufgabe hat), wird als URL einfach die Datei angegeben, in der der Link notiert ist (ein kleiner Trick, der aber sehr nützlich ist).

Die Idee, einen leeren URL anzugeben (also so was wie `<A href="" onClick = "unsere_funktion()">`), wird nicht so funktionieren, wie es gewünscht ist, wenn nur ein Script aufgerufen werden soll. Statt dessen wird der Inhalt des aktuellen Verzeichnisses angezeigt.

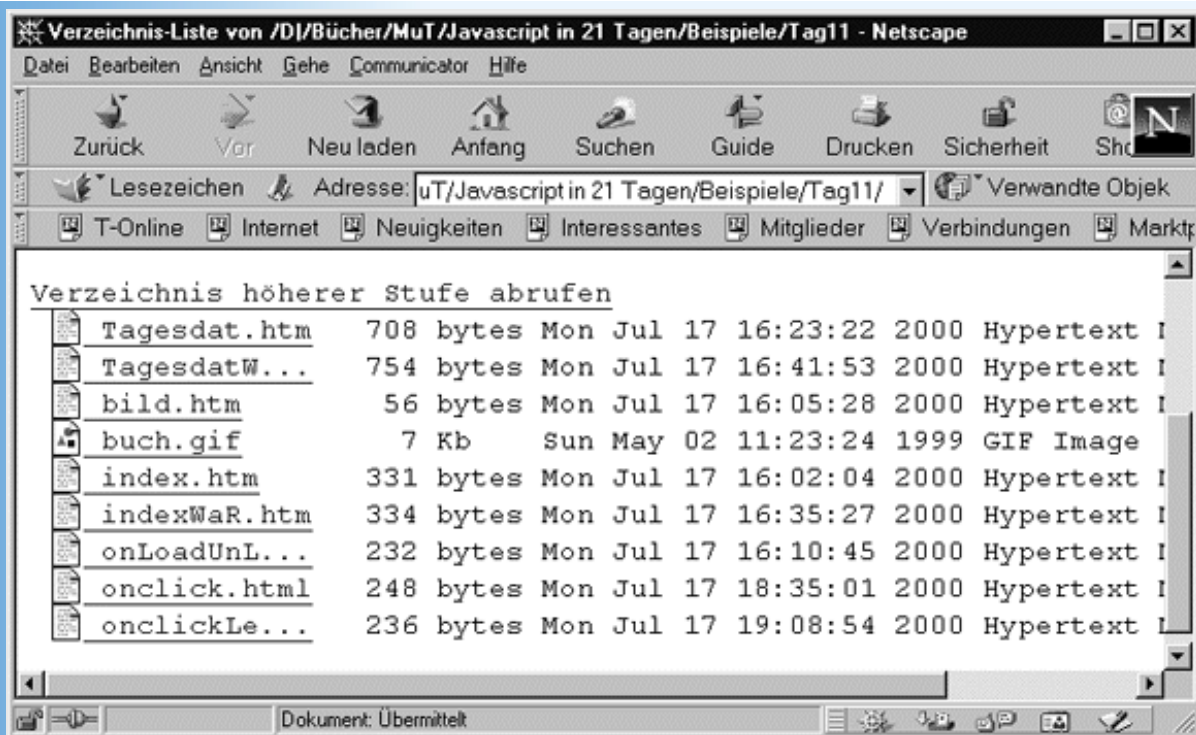


Abbildung 11.7: Bei einem leeren URL wird nach Abarbeiten des Scripts das aktuelle Verzeichnis angezeigt

Allerdings gibt es selbstverständlich eine bessere Variante zum Aufruf eines Scripts bei einem Hyperlink. Diese kennen wir auch schon. Sie verzichtet auf den expliziten Event-Handler und notiert einfach den Aufruf in Form einer Inline-Referenz als URL in den Link. Das sieht dann so aus:

```
<A href="JavaScript:unsere_funktion()">
<H1>
Irgendwas
</H1>
</A>
```

Dabei muss man aber beachten, dass diese Form in einigen Browsern Probleme machen kann.

Besonders viel Sinn macht der Event-Handler in Verbindung mit Formularen und dort dem Schaltflächen-Element. Wir haben das bereits angewandt und werden es bei der genaueren Behandlung von Formularen wieder tun.

Ein zweites Beispiel für den Event-Handler soll diesen aber bei HTML-Tags anwenden, die nur vom Internet Explorer verstanden werden. Beim Navigator passiert bei einem Klick nichts. Testen Sie es ruhig aus.

Beispiel 4:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
function a()
{
  alert("Ein animiertes GIF");
}
function b()
{
  alert("Eine Überschrift");
}
function c()
{
  alert("Eine Trennlinie");
}
//-->
</SCRIPT>
</HEAD>
<BODY>
```

```
<IMG src="buch.gif" onClick="a()">
<H1 onClick="b()">Irgendwas</H1>
<HR onClick="c()">
</BODY>
</HTML>
```

Speichern Sie das Script unter `onClickIEexplorer.html` und laden Sie es in verschiedene Browser. Klicken Sie auf das Bild, die Überschrift und die Trennlinie.

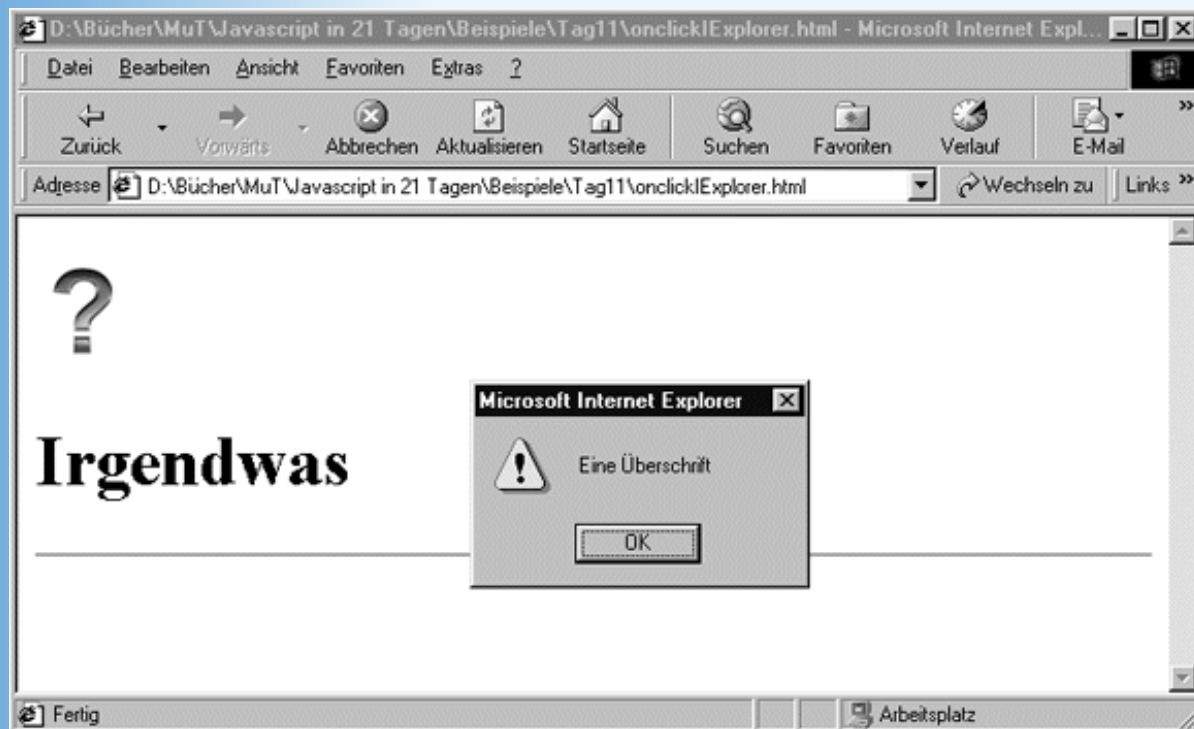


Abbildung 11.8: Auch ein Klick auf die Überschrift funktioniert im Internet Explorer



Da im Netscape Navigator die wenigsten Event-Handler funktionieren, muss man gegebenenfalls mit einem kleinen Trick arbeiten. Da der `Hyperlink` als Basiselement für fast alle Event-Handler auch im Netscape Navigator zu verwenden ist, kann man alle gewünschten sensitiven Elemente in einen Dummy-Hyperlink-Container einschließen. In diesen `Hyperlink` - am besten mit URL auf die aktuelle Datei (s.o.) - kommt dann der Event-Handler. Das wird zwar kleinere optische Probleme bewirken (etwa das Unterstreichen von Textpassagen), aber die kann man umgehen. Wenn man beispielsweise eine Grafik ohne `Hyperlink`-Kennzeichnung haben möchte, setzt man das Attribut `border=0`. Bei Texten kann man mit der zusätzlichen Anwendung von `Style Sheets` die Unterstreichung unterbinden (darauf kommen wir noch).

Eng verwandt mit dem einfachen Klick auf ein sensitives Element ist die Reaktion auf Doppelklicks. Mit dem Event-Handler `onDblClick` können Sie auf den Doppelklick eines Anwenders auf ein Element reagieren. Netscape unterstützt das Attribut wie beim einfachen Klick nur in den HTML-Tags `<A>`, `<AREA>`, `<INPUT>` und `<TEXTAREA>`, während es offiziell wieder bei nahezu allen Tags von HTML 4.0 erlaubt und auch so vom Internet Explorer realisiert ist.



Das gleichzeitige Verwenden von `onClick` und `onDblClick` ist selten sinnvoll. Meist wird der einfache Klick ausgelöst, egal wie schnell der Doppelklick erfolgt. Das ist insbesondere deshalb ein Problem, weil man als Webseitenersteller natürlich keinerlei Einfluss auf die beim Anwender eingestellte Doppelklickgeschwindigkeit hat.

In den gleichen Kontext fallen auch die beiden Event-Handler `onMouseDown` und `onMouseUp`. Sie werden zur Überwachung des Drückens einer Maustaste und deren Loslassen verwendet. Sie werden wie das Ereignis `onClick` verwendet, nur halt bei der jeweiligen Teilaktion ausgelöst.

Überstreichen mit dem Mauszeiger

Das Überstreichen eines Elements mit dem Mauszeiger ist auch ein oft zu findendes Ereignis, bei dem eine JavaScript-Funktion aufgerufen werden kann. Damit kann beispielsweise eine umfangreichere Information zu einem Link in einem extra Mitteilungsfenster angezeigt werden. Dazu muss der Tag des entsprechenden Elementes mit dem entsprechenden Event-Handler gekoppelt werden. Das kann `onMouseOver` sein, womit eine Reaktion auf den Eintritt des Mauszeigers in den Bereich eines Elementes überwacht werden kann. Der verwandte Event-Handler `onMouseOut` erlaubt die Reaktion in dem Moment, wenn der Mauszeiger den Bereich eines Elementes wieder verlässt.

Allgemein muss es sich bei dem Element um eine verweissensitive Referenz handeln, bei der beim Überstreichen mit dem Mauszeiger die JavaScript-Funktion direkt aufgerufen wird. Allerdings versteht der Netscape Navigator darunter nur einen Link bzw. den `<AREA>`-Bereich, wie er bei verweissensitiven Grafiken vorkommt. Der Internet Explorer ist bei der Interpretation von dem Ereignis viel leistungsfähiger und erlaubt das Attribut bei jedem HTML-Element. Testen wir die Event-Handler mit Grafiken (also nur Internet-Explorer-fähig), wo beim Überstreichen in der Statuszeile des Browsers eine Meldung angezeigt wird. Beim Verlassen wird über `onMouseOut` die Statuszeile jeweils wieder gelöscht (das ist notwendig, sonst bleibt die Meldung so lange stehen, bis die Statuszeile von irgendwoher wieder beschrieben wird).

Beispiel 5:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function erklaer1()
{
window.status="Der Dieb";
}
function erklaer2()
{
window.status="Das Objekt der Begierde";
}
function erklaer3()
{
window.status="Der Jäger des Diebs";
}
function loesche()
{
window.status="";
}
</SCRIPT>
<BODY>
<IMG src="cat.gif" height=120
onMouseOver="erklaer1()" onMouseOut="loesche()">
<IMG src="crystal.gif" height=180 onMouseOver="erklaer2()" onMouseOut="loesche()">
<IMG src="bat.gif" height=60
onMouseOver="erklaer3()" onMouseOut="loesche()">
</BODY>
</HTML>
```

Speichern Sie das Script unter `onMouse.html` und laden Sie es in verschiedenen Browsern. Bewegen Sie den Mauszeiger auf die Bilder und beobachten Sie die Statuszeile. Im Internet Explorer wird dort jeweils eine Meldung angezeigt; der Navigator strafft die Verwendung der Event-Handler mit Ignoranz.

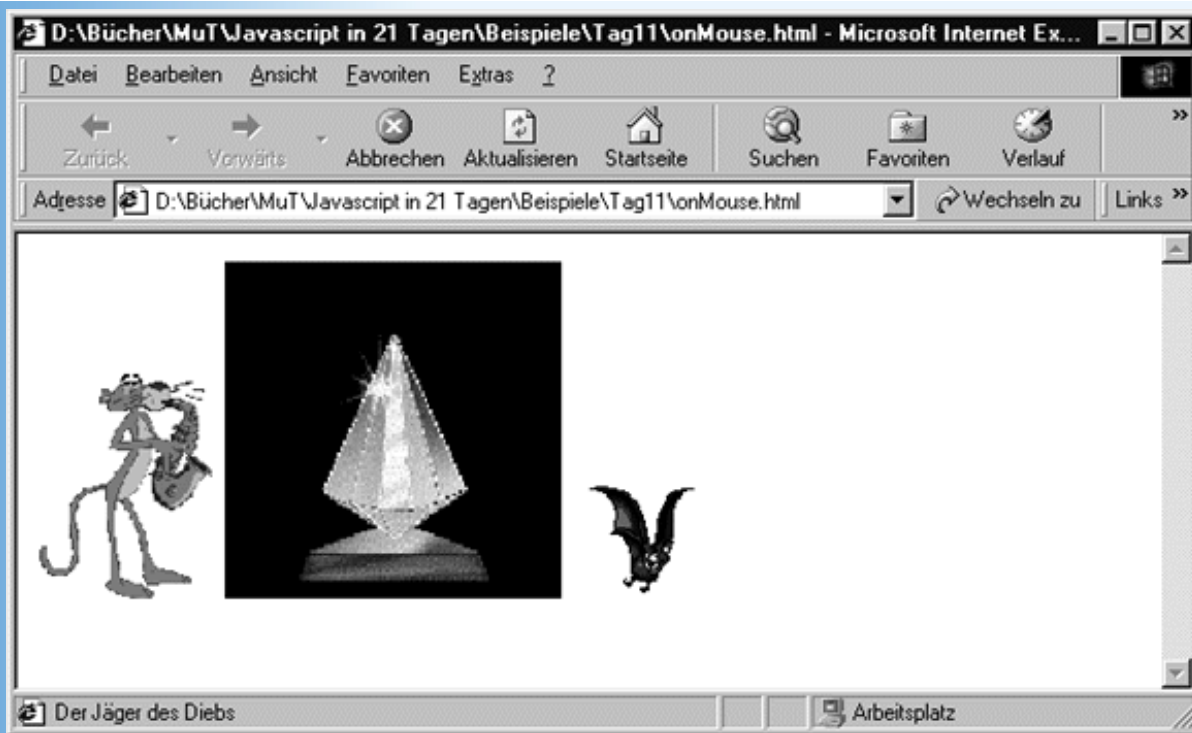


Abbildung 11.9: Meldung 3

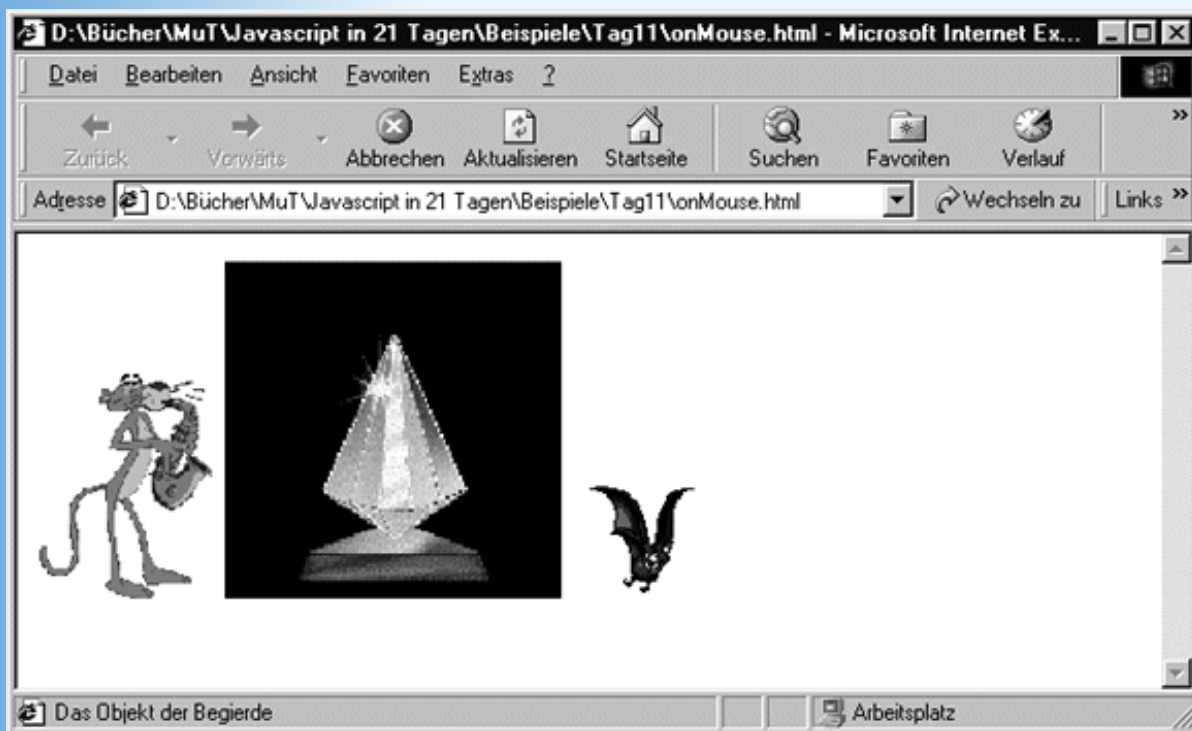


Abbildung 11.10: Meldung 2

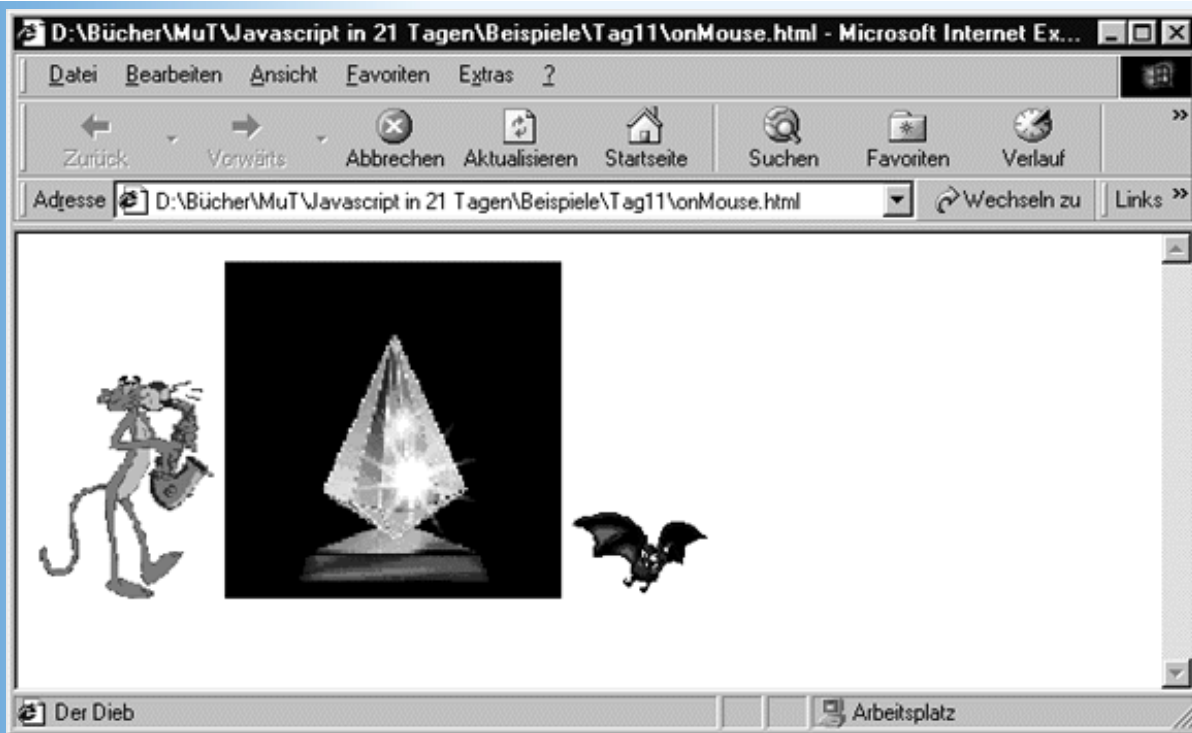


Abbildung 11.11: Meldung 1

Mausbewegungen

Die Bewegung der Maus (unabhängig davon, ob die Maustaste gedrückt ist oder nicht) innerhalb des Anzeigebereichs vom Browser erzeugt ein eigenes Ereignis. Der zugehörige Event-Handler heißt `onMouseMove`. Netscape unterstützt dieses Attribut bei HTML-Tags überhaupt nicht, sondern nur im Zusammenhang mit seinem Ereignis-Objekt. Der Internet Explorer versteht das Attribut jedoch bei allen gängigen HTML-4.0-Tags.

Auf eine Übung wollen wir hier nicht verzichten, allerdings soll nicht verheimlicht werden, dass dieser Event-Handler nicht unproblematisch ist. Immerhin wird bei jeder Mausbewegung über einem Bereich eine Funktion ausgelöst. Es gibt nicht viele sinnvolle Anwendungen, permanent eine Funktion aufzurufen, wenn sich der Mauszeiger in einem gewissen Bereich bewegt. Eine Ausnahme ist das nachfolgende Beispiel, in dem wir allerdings ein wenig vorgreifen und die Technik der Ereignis-Objekte verwenden müssen, um die Position des Mauszeigers verwenden zu können. Das Ereignis-Modell, das wir hier verwenden, ist nur im Internet Explorer zu gebrauchen (so wie auch der Event-Handler). Das Konzept samt konkurrierendem - und vollkommen inkompatiblen - Netscape- Modell bekommen wir in der dritten Woche genauer (an Tag 17).

Das Beispiel gibt in der Statuszeile die aktuelle X- und Y-Position des Mauszeigers aus, wenn er in dem Bereich der Grafik bewegt wird. Verlässt der Mauszeiger die Grafik, wird die Statuszeile gelöscht:

Beispiel 6:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function pos()
{
window.status="X-Position " + event.clientX +
", Y-Position" + event.clientY;
}
function loesche()
{
window.status="";
}
</SCRIPT>
<BODY>
<IMG src="crystal.gif" height=180
onMouseMove="pos()" onMouseOut="loesche()">
</BODY>
</HTML>
```

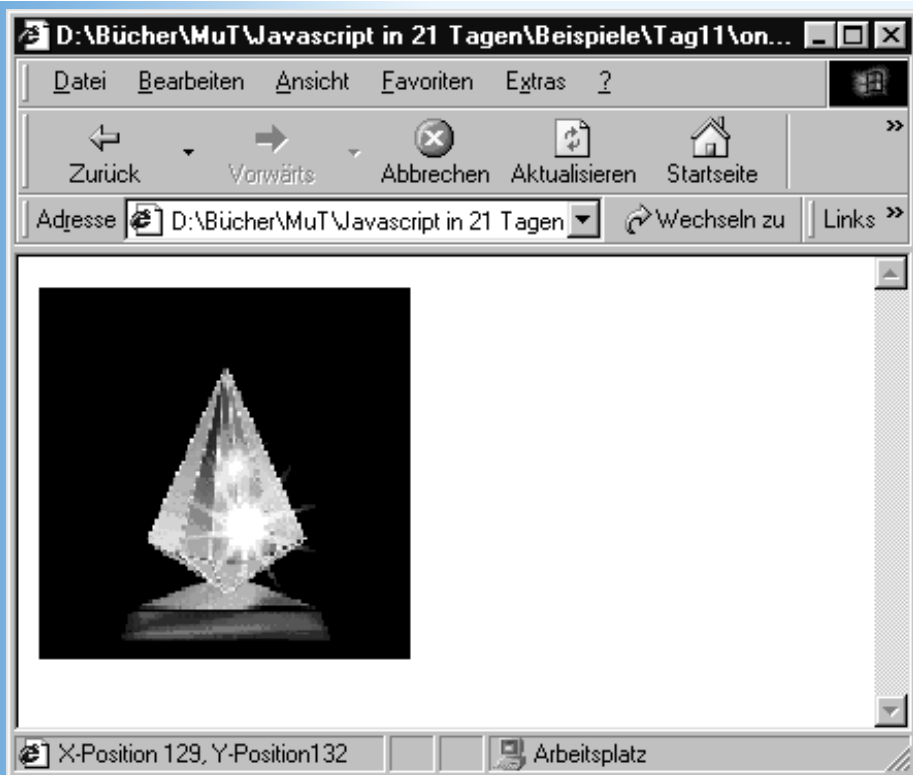


Abbildung 11.12: Positionsangabe 1

Speichern Sie das Script unter onMouseMove.htm und laden Sie es in den Internet Explorer. Wir greifen hier zur Bestimmung auf ein Objekt mit Namen event zurück, das diese Eigenschaften im Internet Explorer bereitstellt.

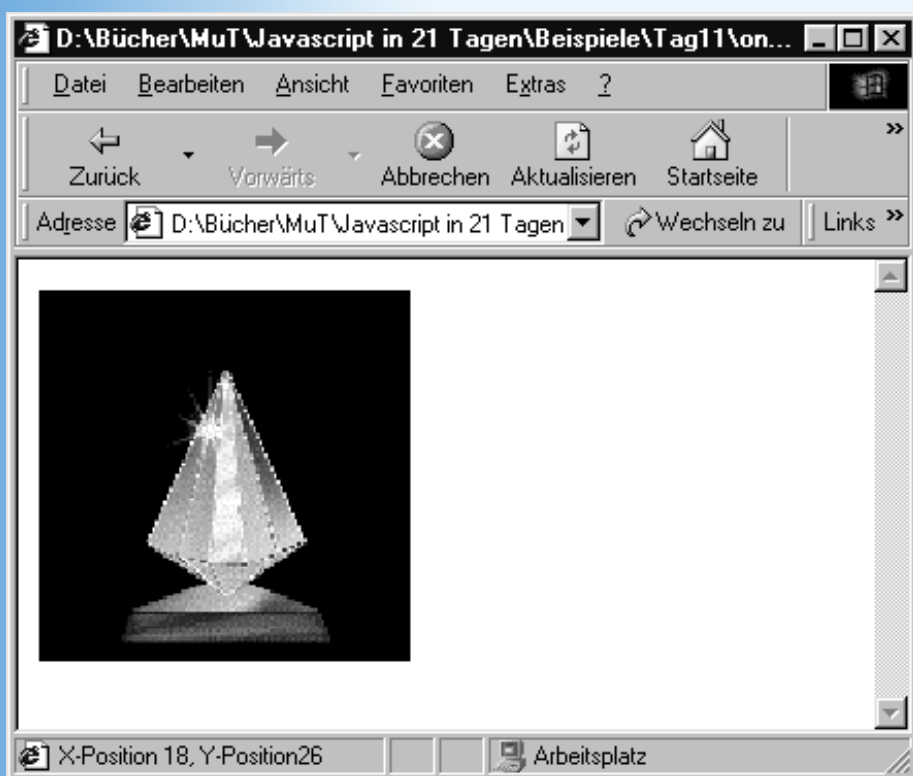


Abbildung 11.13: Positionsangabe 2

Tastaturereignisse

Die Reaktion auf das Betätigen einer (beliebigen) Taste auf der Tastatur bildet einen weiteren Kandidaten für aufzufangende Ereignisse im Rahmen einer Webseite. Diese werden aber erst seit JavaScript 1.2 offiziell unterstützt. Wie bei der Maustaste unterscheidet JavaScript beim Betätigen einer Keyboard-Taste zwischen

- dem vollständigen Tastendruck,
- dem Drücken einer Taste und
- dem Loslassen einer Taste.

Sowohl der vollständige Tastendruck als auch die jeweiligen Teilphasen des Drückens und des Loslassen einer Taste bilden jeweils ein eigenes Ereignis. onKeyPress, onKeydown und onKeyUp sind die jeweiligen Event-Handler. Alle drei Attribute sind im Netscape-Ereignismodell bislang nicht in HTML- Tags möglich, sondern nur im Zusammenhang mit ihrem Ereignis-Objekt. Das offizielle HTML 4.0 (und Microsoft weitgehend) erlauben wieder den Einsatz bei den meisten HTML-Tags, wobei es etwas schwierig ist, zu bestimmen, auf welchem Element ein Tastatur-Ereignis stattgefunden hat (es gibt ja keine X-/Y-Koordinaten wie beim Mauszeiger). Die Event- Handler haben nicht die Bedeutung der Maus-Events. Zumal auch die Bedienung einer Webseite im Wesentlichen über die Maus erfolgt.

Aktivierung und Verlassen eines Elements

Hauptsächlich in Zusammenhang mit Formularen treten die folgenden Ereignisse auf:

- Reaktion auf die Aktivierung eines Elementes
- Reaktion auf die Deaktivierung eines Elementes, d.h. das Element wird verlassen und in der Regel ein neues Element ausgewählt

Andere Situationen sind verweissensitive Grafiken, wie sie über den <AREA>-Tag realisiert werden, oder Schaltflächen, die auch ohne Formulare funktionieren und extra für die Script-Ausführung vorhanden sind - der <BUTTON>-Tag.

Bei der Aktivierung bzw. Selektion eines Elements kann über den Event- Handler onFocus das Ereignis beobachtet werden. Sie können auch auf das Verlassen eines zuvor aktivierten Elementes reagieren. Dazu gibt es den Event-Handler onBlur. Sowohl Netscape als auch Microsoft erlauben beide Attribute bei einigen HTML-Tags.

Der Navigator wird auf die Verwendung mit den HTML-Tags <BODY>, <FRAME>, <INPUT>, <LAYER>, <SELECT> und <TEXTAREA> reagieren, das offizielle HTML 4.0 sieht das Attribut bei den HTML-Tags <A>, <AREA>, <BUTTON>, <INPUT>, <LABEL>, <SELECT> und <TEXTAREA> vor. Microsoft unterstützt sowohl die meisten der von dem Netscape-Browser verstandenen Tags, aber auch die anderen offiziellen Tags (bis auf <LABEL> ohne Probleme).

Für die Beispiele mit diesen beiden Event-Handlern werden wir wieder auf einfache Formulare zurückgreifen. Deren Details folgen an späteren Tagen, aber es ist auch sicher nicht schlecht, wenn Sie vorher bereits einige Anwendungen davon gesehen haben.

Beispiel 7:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function a()
{
window.status="aaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
}
function b()
{
window.status="bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb";
}
function c()
{
window.status="klick";
}
function d()
{
window.status="klack";
}
function e()
{
window.status="Laber";
}
function f()
{
window.status="";
alert("Rababer");
}

function g()
{
```

```

window.status="hinundweg";
}
function h()
{
window.status="hochundrunter";
}

</SCRIPT>
<BODY>
<A href="onFocusBlur.htm" onFocus="a()" onBlur="b()">Click myself</A>
<BR>
<BUTTON onFocus="c()" onBlur="d()">Click und fort</BUTTON>
<FORM>
<INPUT onFocus="e()" onBlur="f()">
<BR>
<TEXTAREA onFocus="g()" onBlur="h()"> </TEXTAREA>
</FORM>
</BODY>
</HTML>

```

Speichern Sie das Script unter onFocusBlur.htm und laden Sie es in den Internet Explorer. Testen Sie die einzelnen Aktivierungs- und Deaktivierungsmeldungen. Die meisten Aktivitäten finden in der Statuszeile statt, aber auch ein Mitteilungsfenster wird verwendet. Beachten Sie, dass die Aktivierung eines Elements automatisch mit der Deaktivierung des vorher selektierten Elements einhergeht. Deshalb werden Sie in vielen Fällen zwei unmittelbar aufeinander folgende Aktionen auslösen. Wenn Sie jedoch nach der Selektion eines Elements auf eine leere Fläche im Anzeigebereich klicken, wird nur die onBlur-Aktion aufgerufen.

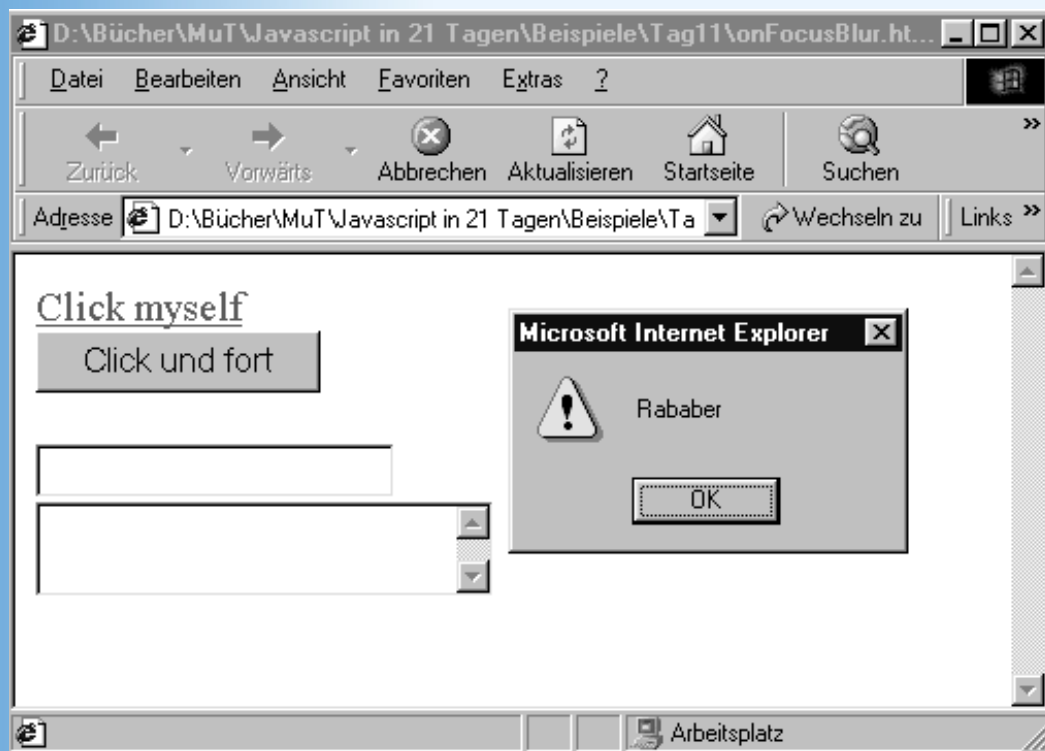


Abbildung 11.14: Es gibt auch eine Meldung als Fenster

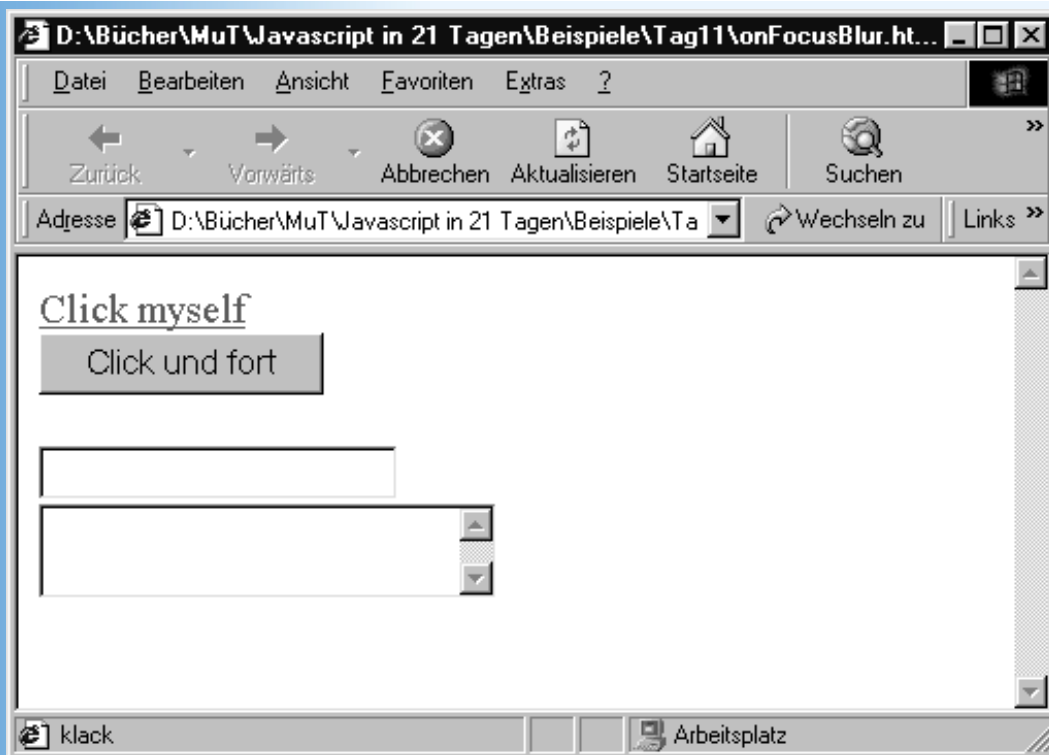


Abbildung 11.15: Meldung onBlur

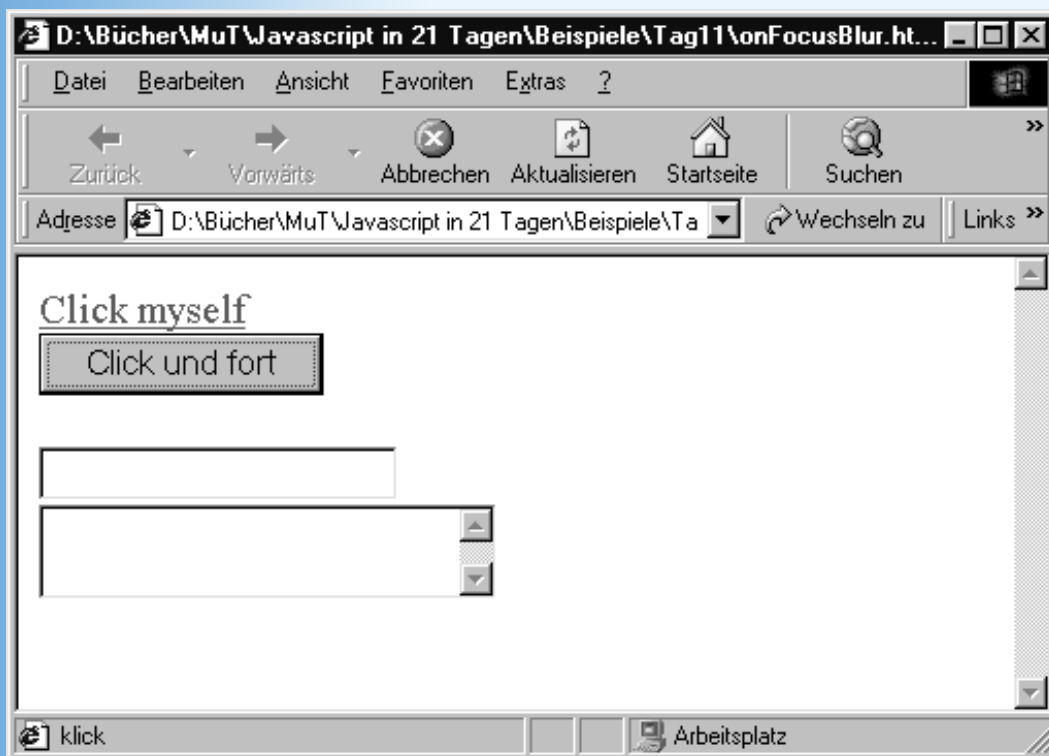


Abbildung 11.16: Meldung onFocus

Laden Sie nun die Seite in den Netscape Navigator. Sie werden sehen, dass der Button nicht angezeigt wird und nicht alle Event-Handler funktionieren.

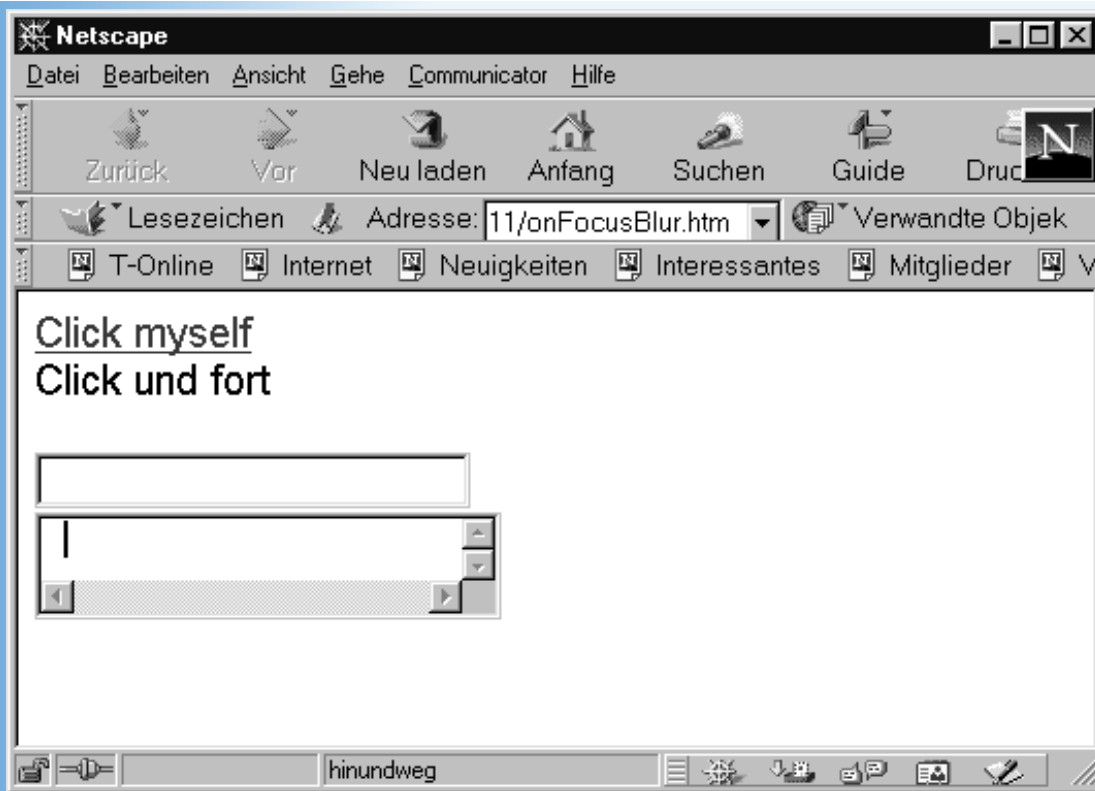


Abbildung 11.17: Weder optisch noch technisch voll funktionstüchtig



Der Event-Handler `onBlur` ist recht gefährlich. Es kann durchaus vorkommen, dass Sie sich ein ziemliches Problem einhandeln, wenn Sie beim Verlassen eines Elements eine Situation schaffen, indem unmittelbar nach dem Verlassen die Aktivierung eines Elements wieder ausgelöst wird (dies kann beispielsweise bei Verwendung von `alert()` zum Erzeugen eines Mitteilungsfensters so eintreten, wenn `alert()` gleichzeitig bei `onFocus` und `onBlur` verwendet wird), kommt Ihnen permanent die Gegenreaktion des Elements und eine entsprechende Reaktion darauf in die Quere. Besonders gefährdet ist dabei der Navigator.

Das kann aber auch bei Eingabefeldern mit einer Überprüfung passieren. Wenn dabei beispielsweise eine Fehlermeldung mit einem Mitteilungsfenster auftaucht, können Sie u.U. nicht das entsprechende Eingabefeld korrigieren. Das kann sogar dazu führen, dass Sie nicht einmal den Browser beenden können. Beachten Sie bitte dazu das nachfolgende Beispiel. Aber auch sonst ist die Verwendung von `onBlur` am besten am Ende einer vollständigen Eingabe sämtlicher relevanter Daten (an zentraler Stelle) zu empfehlen und es sollte auf keinen Fall mit einem `onFocus`-Ereignis gekoppelt werden, das bei der `onBlur`-Aktion unausweichlich ausgelöst wird.

Beispiel 8:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function wk()
{
alert("Hy");
}
function bey()
{
alert("Ciao");
}

function e()
{
alert("Laber");
}
function f()
{
```

```
alert("Rababer");
}
</SCRIPT>
<BODY onFocus="wk()" onBlur="bey()">
<FORM>
<INPUT onFocus="e()" onBlur="f()">
</FORM>
</BODY>
</HTML>
```

Speichern Sie das Script unter onFocusBlur2.htm und laden Sie es in den Internet Explorer. Testen Sie die einzelnen Aktivierungs- und Deaktivierungsmeldungen. Beachten Sie, dass ein Klick auf den Bodybereich integriert ist. Die Meldungsfenster erscheinen bei Klicks recht oft. Aber der Internet Explorer kommt noch recht gut mit der Situation zurecht. Das Problem wird erst im Netscape Navigator ernst. Laden Sie die Datei in den Browser (speichern Sie aber vorher alle wichtigen Dateien). Sie werden es (wahrscheinlich) nicht schaffen, das Frage- und Antwort- Spiel zwischen onFocus und onBlur des <BODY>-Tags zu unterbrechen. Der Browser kann nicht einmal mehr geschlossen werden. Es hilft nur noch der Taskmanager oder ein Warmstart. Sie können sich vorstellen, dass so etwas bei einem Besucher einer Webseite Freude auslöst ;-).

Reaktion auf Änderung

Das ausschließlich bei Formularen vorkommende Ereignis »Änderung« (Event-Handler onChange) erlaubt die Reaktion für den Fall, dass ein Element in einem Formular einen geänderten Wert erhalten hat. Es wird sowohl bei Netscape als auch nach offiziellem HTML 4.0 in den HTML- Tags <INPUT>, <SELECT> und <TEXTAREA> unterstützt.

Das nachfolgende kleine Beispiel erzeugt ein Formular mit zwei Eingabefeldern. Die dort eingegebenen Werte werden miteinander multipliziert in der Statuszeile des Browsers ausgegeben. Dabei wird die JavaScript-Funktion immer dann aufgerufen, wenn in einem Feld ein Wert geändert wurde und in das andere Feld geklickt wird. Auch ein Bestätigen mit der Datenfreigabetaste sollte in den meisten Browsern die Funktion auslösen.

Beispiel 9:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function change()
{
window.status =
document.forms[0].elements[0].value *
document.forms[0].elements[1].value;
}
</SCRIPT>
<BODY>
<FORM>
<INPUT onChange="change()">
<INPUT onChange="change()">
</FORM>
</BODY>
</HTML>
```

Speichern Sie das Script unter onChange.htm und laden Sie es in einen Browser.



Abbildung 11.18: Das Ergebnis wird in der Statuszeile angezeigt

Reaktion auf Zurücksetzen oder Absenden eines Formulars

Das Verständnis der beiden Event-Handler `onReset` und `onSubmit` ist ohne etwas erweiterte Kenntnisse von Formularen nicht einfach. Deshalb eine kurze Erklärung, die bereits über das hinausgeht, was wir bisher zu Formularen wissen:

In dem `<FORM>`-Container eines Formulars können Angaben gemacht werden, wohin ein Formular gesendet wird (falls die Anwendereingaben per Internet versandt werden sollen) und wie die Versendung erfolgen soll. Dabei erfolgt normalerweise die Versendung ohne irgendwelche Kontrolle, wenn eine **Absenden**-Schaltfläche betätigt wird, die über die Syntax `<INPUT type="submit" value=[Beschriftung des Buttons]>` erzeugt wird.

Beispiel:

```
<INPUT type="submit" value="Ok">
```

Des Weiteren gibt es in den meisten Formularen eine Schaltfläche, welche das Formular bei Bedarf wieder leert, d.h. zurücksetzt. Das wird über die Syntax

```
<INPUT type="reset" value=[Beschriftung des Buttons]>
```

 realisiert.

Die Reaktion auf das Zurücksetzen eines Formulars über einen Event-Handler (`onReset`) erlaubt die Erweiterung der Funktionalität, welche über die durch HTML generierte **Abbrechen**-Schaltfläche standardmäßig erfolgt. Beim Anklicken dieses Buttons werden standardmäßig alle Eingaben im Formular gelöscht. Wenn jedoch im einleitenden Formular-Tag (!) der Event-Handler `onReset` notiert ist, wird statt dessen die dort notierte JavaScript-Anweisung ausgeführt. Sie fangen also die normale Reaktion des Formulars auf Drücken der Schaltfläche ab. Damit können Sie beispielsweise eine Meldung der Form »Sind Sie wirklich sicher?« generieren, die einem Zurücksetzen eines Formulars vorangestellt wird und gegebenenfalls noch eine Abbruchmöglichkeit erlaubt.

Beispiel:

```
<FORM Name="meinFormular" onReset="return wirklich()">
```

Vollkommen analog ist der Event-Handler `onSubmit` zu sehen. Er erweitert nur statt der **Abbrechen**-Schaltfläche den Submit-Button, mit dem ein Formular weggeschickt wird. Statt es einfach abzusenden, kann eine JavaScript-Funktion aufgerufen werden, die beispielsweise überprüft, ob alle Felder eines Formulars ausgefüllt sind.

Beispiel:

```
<FORM Name="meinFormular" onSubmit="return kontrolle()">
```

Das Attribut ist ausschließlich im `<FORM>`-Tag erlaubt und wird sowohl vom Navigator als auch dem Explorer verstanden.

Sie sollten beim Funktionsaufruf über `onSubmit` und `onReset` das Schlüsselwort `return` voranstellen. Wenn Sie im Rahmen der aufgerufenen Funktion die Werte `true` oder `false` zurückgeben, ist dies eine ideale Möglichkeit, die Abarbeitung der ausgelösten Aktion noch einmal abubrechen (Rückgabe von `false` durch die aufgerufene Funktion). Das ist aber nicht zwingend.

Testen wir ein kleines Beispiel. Dabei nutzen wir eine weitere Methode des Objektes `window` - `confirm()`. Diese erzeugt ein Fenster mit zwei Schaltflächen. Je nach betätigter Schaltfläche wird der Rückgabewert `true` (bei Ok) oder `false` (bei Abbrechen) erzeugt. Dieser wird dann im Formular verwendet. Wenn das Absenden bestätigt wird, wird das Formular per Email versandt, was die in dem Beispiel verwendete Erweiterung des `<FORM>`-Tags bewirkt.

Beispiel 10:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function kontrolle()
{
// Rückfrage, ob tatsächlich Formular
// vollständig ausgefüllt
return window.confirm(
"Ist das Formular vollständig ausgefüllt?");
}
function sicher()
{
// Rückfrage, ob tatsächlich Formular leeren
return window.confirm(
"Wollen Sie dieses Formular wirklich leeren?");
}
</SCRIPT>
<BODY>
<FORM
  action="mailto:hansdampf@hotmail.com"
  method=POST
  onSubmit="return kontrolle()"
  onReset="return sicher()">
<INPUT name="eins">
<INPUT name="zwei">
<BR>
<INPUT type="submit" value="Ok">
<INPUT type="reset" value="Abbrechen">
</FORM>
</BODY>
</HTML>
```

Speichern Sie das Script unter onSubmitReset.htm und laden Sie es in einen Browser. Füllen Sie die beiden Formularfelder und betätigen Sie die **Ok**- und die **Abbrechen**-Schaltfläche. Testen Sie dann in dem Dialogfenster jeweils beide Möglichkeiten.

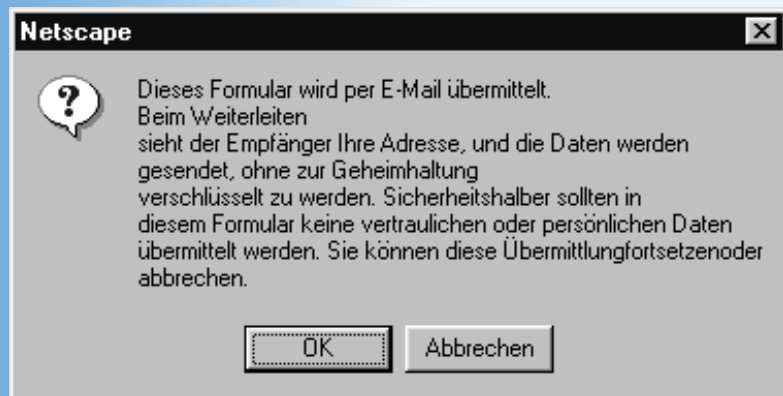


Abbildung 11.19: Bei Bestätigung würde das Formular per Email versandt

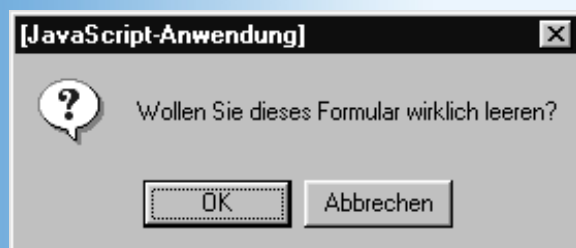


Abbildung 11.20: Rückfrage, ob das Formular geleert werden soll



Abbildung 11.21: Rückfrage, ob das Formular vollständig ist

Reaktion bei der Selektion von Text

Über den Event-Handler `onSelect` können Sie auf die Situation reagieren, wenn ein Anwender Text selektiert. Hier zeigt sich wieder einmal sehr deutlich der Unterschied zwischen dem Netscape-Ereignismodell, wo das Ereignis nur bei den Tags `<INPUT>` und `<TEXTAREA>` unterstützt wird, und der offiziell möglichen Anzahl von HTML-Tags (`<a>`, `<abbr>`, `<acronym>`, `<address>`, `<area>`, `<b>`, `<big>`, `<blockquote>`, `<body>`, `<button>`, `<caption>`, `<center>`, `<cite>`, `<code>`, `<col>`, `<colgroup>`, `<dd>`, `<del>`, `<dfn>`, `<dir>`, `<div>`, `<dl>`, `<dt>`, `<em>`, `<fieldset>`, `<form>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<hr>`, `<i>`, `<img>`, `<input>`, `<ins>`, `<kbd>`, `<label>`, `<legend>`, `<li>`, `<link>`, `<map>`, `<menu>`, `<noframes>`, `<noscript>`, `<object>`, `<ol>`, `<optgroup>`, `<option>`, `<p>`, `<pre>`, `<q>`, `<s>`, `<samp>`, `<select>`, `<small>`, `<span>`, `<strike>`, `<strong>`, `<sub>`, `<sup>`, `<table>`, `<tbody>`, `<td>`, `<textarea>`, `<tfoot>`, `<th>`, `<thead>`, `<tr>`, `<tt>`, `<u>`, `<ul>`, `<var>`).

Da die Anwendung dieses Event-Handlers nichts Neues beinhaltet, wollen wir auf ein Beispiel verzichten.

Unterbrechung eines Bildladevorgangs

Ein Ladevorgang eines Bildes oder einer Grafik in einen Browser dauert - gerade über das World Wide Wait¹ - eine gewisse Zeitspanne. Es kann dabei hauptsächlich durch zwei Situationen zu einem Abbruch des Ladevorgangs kommen:

1. Abbruch durch den Anwender
2. Fehler beim Laden

Deshalb sind zwei Event-Handler als zusätzliche Attribute des `<IMG>`-Tags möglich.

Jeder Browser verfügt über einen **Abbruch**-Button. Mit dem Event-Handler `onAbort` soll die Situation behandelt werden, wenn ein Anwender diesen **Abbruch**-Button drückt und gerade eine Grafik geladen wird.

Beispiel:

```
<IMG src="foto.jpg" width=400 onAbort="tellMeWhy()">
```

Damit lassen sich etwa Meldungen ausgeben, ob vielleicht nicht doch die Grafik geladen werden sollte - sie ist ja soooooo interessant!

Wenn beim Laden eines Bildes oder einer Grafik ein Fehler auftritt, können Sie über den Event-Handler `onError` darauf reagieren. Damit ist im Wesentlichen die Ausgabe eigener Fehlermeldungen gemeint. Eine vollständige Fehlerbehandlung wie in anderen Programmiersprachen kann damit nicht realisiert werden.

Beispiel:

```
<IMG src="foto.jpg" width=400 onError="verdammt()">
```

Beide Event-Handler können sowohl beim Navigator als auch beim Explorer angewandt werden.

Der explizit Aufruf eines Event-Handlers

Ein Event-Handler kann auch über eine andere Art ausgelöst werden. Schauen Sie sich bitte das nachfolgende Beispiel an.

Beispiel 11:


```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function unsere_funktion()
{
  alert("Herzlich Willkommen");
}
</SCRIPT>
</HEAD>
<BODY>
  <FORM NAME="meinFormular">
    <INPUT TYPE="button" NAME="meinButton" value="OK">
  </FORM>
<SCRIPT LANGUAGE="JavaScript">
document.meinFormular.meinButton.onclick =
  unsere_funktion;
</SCRIPT>
</BODY>
</HTML>
```

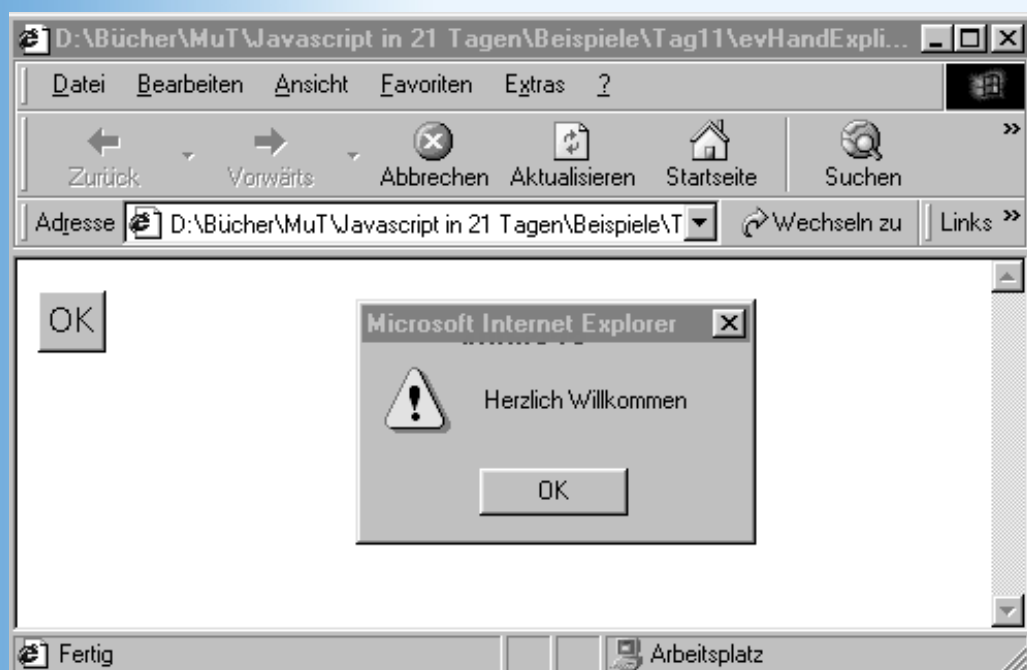


Abbildung 11.22: Klick auf den Button

Die in dem Beispiel verwendete Syntax beinhaltet eine Funktion, die in dem zweiten Script-Container dem Formular meinFormular und dort dem Element meinButton zugeordnet wird. Dabei sind zwei Dinge zu beachten:

- Event-Handler sind Funktionsreferenzen. Deshalb dürfen bei der Zuordnung keine (!) Klammern angegeben werden. Ansonsten würde die Funktion unmittelbar bei der Zuordnung ausgeführt und das Ergebnis undefined zugewiesen. Als HTML-Attribute hingegen müssen bei Event-Handlern die Klammern notiert werden.
- In JavaScript 1.1 und früheren Versionen muss der Event-Handlername (als Bestandteil des Scripts - das betrifft nicht die HTML-Attribute) vollständig in Kleinbuchstaben notiert werden (wie in dem Beispiel), wobei diese Technik so unter JavaScript 1.0 aber nicht funktioniert.

Zusammenfassung

Mittels der unterschiedlichen Event-Handler reagiert der JavaScript-Interpreter auf verschiedenartige Ereignisse, die der Event-Handler-Mechanismus über ihre festgelegten Namen identifiziert. Wesentlich ist, dass Ereignisse in dem neuen HTML 4.0 und JavaScript vorhanden und identisch sind. Das W3-Konsortium hat die Event-Handler offiziell in den HTML-Sprachstandard mit aufgenommen. Event-Handler sind ein wichtiges Bindeglied zwischen HTML und JavaScript. Ein Event-Handler-Ereignis wird wie jedes gewöhnliche HTML-Attribut angewandt. Allgemein beginnt es mit der Silbe on, gefolgt von der Beschreibung des Ereignisses. Hinter dem Namen des Ereignisses wird ein Ist-gleich-Zeichen notiert, gefolgt von einer - meist in Anführungszeichen notierten - JavaScript-Anweisung oder -Funktion.

Bei welchem HTML-Tag Event-Handler verstanden bzw. welche Ereignisse überhaupt von Browsern verstanden werden, ist leider

sehr Browser- abhängig.

Fragen und Antworten

Frage:

Wenn die Event-Handler so inkonsistent in verschiedenen Browsern gehandhabt werden - soll man sie überhaupt anwenden?

Antwort:

Sie haben wenig Alternativen. Es gibt eigentlich nur noch die Möglichkeit, auf das allgemeine Ereignis-Modell auszuweichen (dazu werden wir an Tag 17 kommen). Dort wird die Ereignisbehandlung nicht inkonsistent gehandhabt. Es ist von vornherein klar, dass das Microsoft-Modell und das Netscape- Modell inkompatibel sind. Sie benötigen also eine strikte Trennung in einer Webseite oder verschiedene Webseiten. Das bedeutet einigen Aufwand. Es kann sich jedoch dennoch rechnen, wenn man die Feinheiten von verschiedenen Ereignissen ausnutzen will. Bei Reduzierung auf die Ereignisse und HTML-Tags, wo beide Browser Event-Handler gleich behandeln, ist die Arbeit mit Event-Handlern einfacher und mit weniger Aufwand verbunden.

Frage:

Warum unterstützt gerade Netscape die Event-Handler nur so eingeschränkt?

Antwort:

Wie schon im Laufe dieses Kapitels erwähnt, priorisiert Netscape sein globales Ereignis-Modell und wurde mehr oder weniger gezwungen, das Event-Handler-Modell zu unterstützen. Man sollte auch im Auge behalten, dass Event-Handler bei vielen der offiziell erlaubten und vom Internet Explorer unterstützten HTML-Tags ziemlich überflüssig sind. Viele Kritiker werfen mit Scripten aufgepeppten Webseiten vor, nur überflüssige Spielereien damit zu erstellen. Die bei jedem Element über Event-Handler beobachtete Klickerei ist beispielsweise sehr zu hinterfragen. Was soll das eigentlich bringen? Wenn es notwendig ist, kann man es mit zahlreichen anderen Techniken realisieren.

Workshop

Testen Sie die Event-Handler. Erstellen Sie vor allem Beispiele mit den Event-Handlern, von denen keine expliziten Beispiele durchgespielt wurden.

Experimentieren Sie in Beispiel 2 mit dem Workaround für Netscape- kompatible Browser. Versuchen Sie eine andere Lösung zu finden, die zwar JavaScript-unfähigen Browsern eine Meldung anzeigt, aber dennoch gegenüber Größenveränderungen des Browsers immun ist.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

Unterstützt der Netscape Navigator diese Syntax?

1. `<BODY onLoad="a()">`
2. `<HR onClick="a()">`
3. `<A href="test.htm" onMouseOver="a()">`
4. `<BUTTON onFocus="c()" onBlur="d()">`
Click
`</BUTTON>`
5. `<INPUT onChange="change()">`
6. `> "> <FORM Name="meinFormular"`
`onReset="return wirklich()">`
7. `<BODY onUnload="a()">`
8. `<H1 onSelect ="a()">Willkommen</H1>`

:-)

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 2

Tag 12

Erweiterte Gestaltungsmöglichkeiten einer Webseite

Der gestrige Tag war sicher recht anstrengend. Am heutigen Tag steht dafür nicht ganz so viel neuer JavaScript-Stoff auf dem Programm. Uninteressant wird es dennoch sicher nicht. Wir wollen heute nur JavaScript als Hauptthema wieder ein wenig verlassen und uns um einige weitergehende Gestaltungsmöglichkeiten einer Webseite kümmern. Dabei wird also wieder HTML mehr in den Vordergrund geholt. Wir beschäftigen uns heute mit

- Listen und Aufzählungen innerhalb einer Webseite
- Tabellen

Diese HTML-Techniken sollen der neue Gegenstand für den zwölften Tag sein, aber wir wollen JavaScript natürlich auch an diesem Tag nicht vergessen. Wir werden auf diese neuen HTML-Gestaltungselemente unsere bisherigen JavaScript-Kenntnissen anwenden und damit einige Praxis erarbeiten.

Listen und Aufzählungen

Eine Aufzählung besteht aus einer Liste, in der eine Reihe von alternativen Möglichkeiten dargestellt wird. Das kennt man ja nicht zuletzt aus Textverarbeitungen, wo es diverse Formen von Aufzählungen gibt. In einer Webseite wird dazu ein äußerer Container benötigt, der den Beginn und das Ende einer Aufzählung festlegt sowie im Inneren des Containers eine Anweisung, die jeden Punkt der Aufzählung spezifiziert.

Der äußere Container

Es gibt unter HTML verschiedene Typen von Listen (wenngleich beileibe nicht so viele, wie Sie aus Textverarbeitungen unter Windows gewohnt sind), welche mit unterschiedlichen Tags definiert werden. Da gibt es zum einen eine nicht-nummerische Liste (Bullet-Liste), welche als Aufzählungszeichen einen Punkt, ein Quadrat oder ein ähnliches Zeichen verwendet. Welches Zeichen konkret verwendet wird, hängt vom Browser und dessen Einstellungen ab. Neuere Browser stellen aber bei gleicher Konfiguration die Bullet-Listen meist gleich dar (in der Regel runde Aufzählungszeichen). Diese Listenform wird in einen Container

`<UL>`

eingeschlossen.

Eine nummerierte Liste mit automatischer Durchnummerierung der Aufzählungspunkte wird in einen Container

`<OL>`

eingeschlossen. In der Grundeinstellung werden die Listenpunkte mit normalen Zahlen durchnummeriert. Die nummerierte Liste lässt gegenüber der Bullet-Liste einige Gestaltungsmöglichkeiten zu. Es gibt neben der Nummerierung mit normalen Ziffern insgesamt vier weitere Typen von nummerierten Listen - zwei alphabetisch und zwei römisch nummerierte Listen. Die Auswahl erfolgt über den Parameter `type`. Die Wertzuweisung kann sowohl in Hochkommata gesetzt werden, aber auch bei den meisten Browsern ohne diese erfolgen:

- `<OL type="i">` erzeugt eine nummerierte Liste des Typs i, ii, iii,... (römische Nummerierung mit kleinen Zahlen)

- `<OL type="I">` erzeugt eine nummerierte Liste des Typs I, II, III,... (römische Nummerierung mit großen Zahlen)
- `<OL type="a">` erzeugt eine alphabetisch nummerierte Liste (Kleinbuchstaben)
- `<OL type="A">` erzeugt eine alphabetisch nummerierte Liste (Großbuchstaben)
- Theoretisch kann man die Angabe `type="1"` setzen, aber das ist unnötig. Es ist Default-Einstellung und bewirkt eine normale numerische Liste

Bei einer numerischen Liste kann auch ein Startwert gesetzt werden, was mit einem weiteren Attribut realisiert wird: `start=[Wert]`.



Beachten Sie, dass für den Wert der type-Angabe - ungewöhnlich für HTML - Groß- und Kleinschreibung von Bedeutung ist.

Der Listenpunkt

Ein Listeneintrag kann über mehrere Zeilen gehen. Dabei ist es egal, ob der Zeilenumbruch durch die Fenstergröße oder einen erzwungenen Zeilenumbruch erzeugt wurde. Ein Listenpunkt beginnt mit dem Tag

`<LI>`.

Ein Ende-Tag ist sinnvoll, kann aber unterbleiben, denn ein nachfolgender `<LI>`-Tag erzeugt einen neuen Listenpunkt, der Ende-Tag des Listenbereich beendet einen Listenpunkt ebenso.



Die Listenpunkte sind bei allen Listen gleich. Die Unterscheidung der Liste erfolgt ausschließlich über den äußeren Container.

Beispiel 1:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<BODY>
```

```
<UL>
<LI>Erstens</LI>
<LI>Zweitens</LI>
<LI>Drittens</LI>
</UL>
```

```
<OL>
<LI>Erstens</LI>
<LI>Zweitens</LI>
<LI>Drittens</LI>
</OL>
```

```
<OL start="3" type="I">
<LI>Erstens</LI>
<LI>Zweitens</LI>
<LI>Drittens</LI>
</OL>
```

```
<OL start="1" type="i">
<LI>Erstens</LI>
<LI>Zweitens</LI>
```



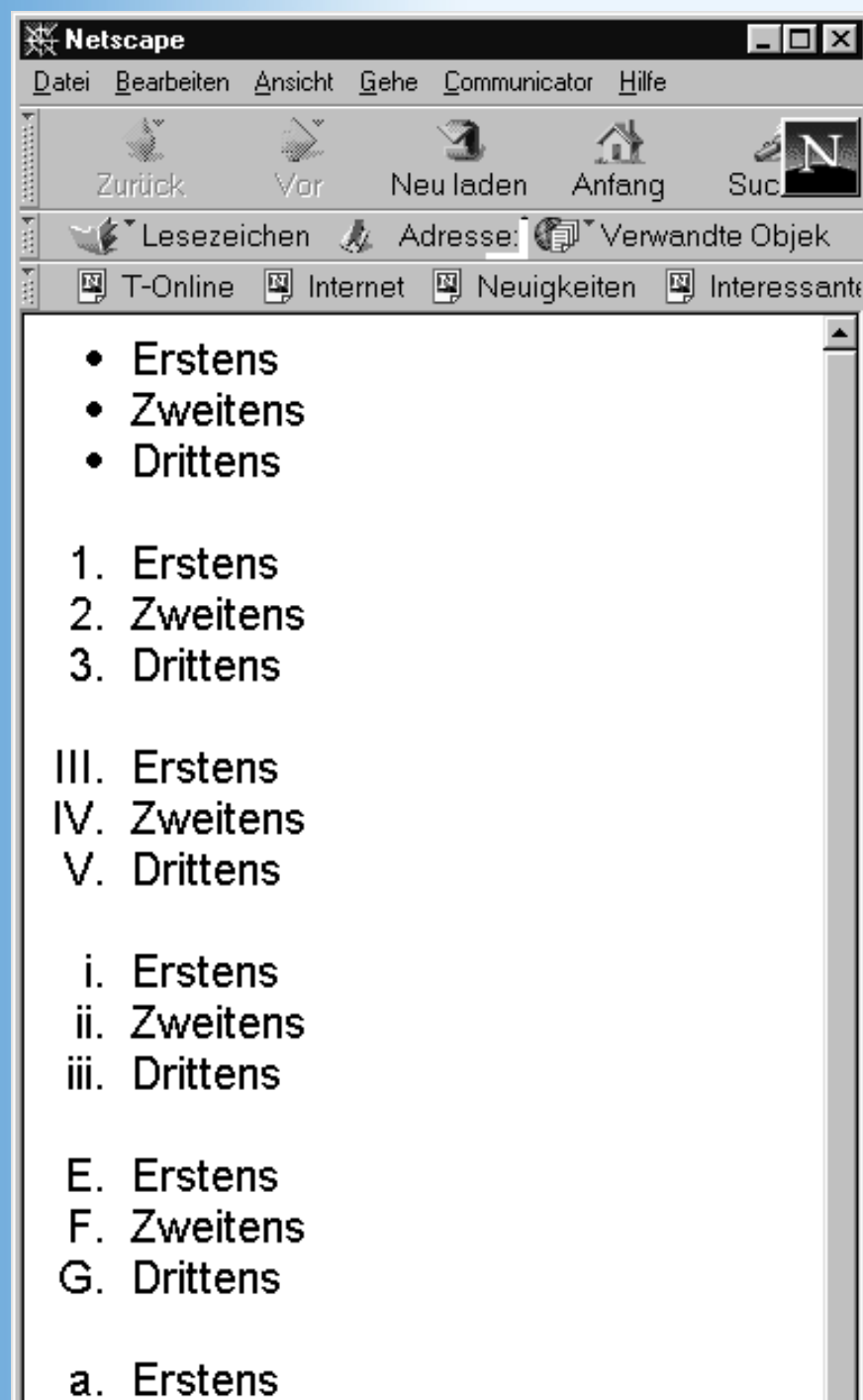
```
<LI>Drittens</LI>
</OL>
```

```
<OL start="5" type="A">
<LI>Erstens</LI>
<LI>Zweitens</LI>
<LI>Drittens</LI>
</OL>
```

```
<OL start="1" type="a">
<LI>Erstens</LI>
<LI>Zweitens</LI>
<LI>Drittens</LI>
</OL>
```

```
</BODY>
</HTML>
```

Speichern Sie das Script unter Liste.htm und laden Sie es in einen Browser.



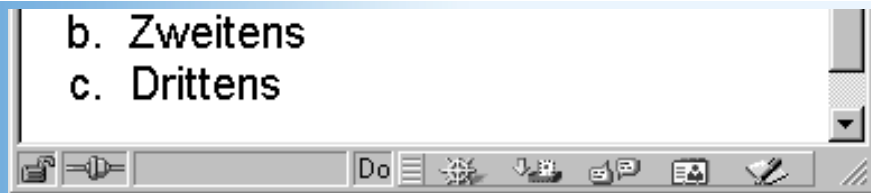


Abbildung 12.1: Verschiedene Listen

Wenn Sie sich den Quelltext ansehen, werden Sie erkennen, dass es keine Zeilenschaltung und keine Absätze gibt. Eine Aufzählung beginnt immer in einer neuen Zeile und erzeugt am Ende einen weiteren Zeilenvorschub. Ebenso erzeugt jeder Listeneintrag eine neue Zeile. Sie benötigen also keine zusätzliche Absatz- oder Zeilenschaltung.

Anwendung von JavaScript auf Listen

Listen sind nun ein neues Gestaltungselement, auf das wir unsere bisher erworbenen JavaScript-Kenntnisse anwenden können. Wir wollen zum Auslösen der JavaScript-Funktionen mit Event-Handlern arbeiten. Das nachfolgende Beispiel funktioniert jedoch nur im Internet Explorer.

Beispiel 2:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function pos()
{
window.status="X-Position " + event.clientX +
", Y-Position " + event.clientY;
}
function lp1()
{
alert("Dazu gibt es dies und jenes zu sagen");
}
function lp2()
{
alert("Dazu gibt es auch was zu sagen");
}
function lp3()
{
alert("Dazu gibt es nix zu sagen");
}

function loesche()
{
window.status="";
}
</SCRIPT>
<BODY>
<UL onMouseOver="pos()">
<LI onClick="lp1()">
Erstens
</LI>
<LI onDblClick="lp2()">
Zweitens
</LI>
<LI onMouseOut="lp3()">
Drittens
</LI>
</UL>
</BODY>
```

</HTML>

Speichern Sie das Script unter ListeJS.htm und laden Sie es in den Internet Explorer.



Beachten Sie, dass die Event-Handler sowohl im äußeren Listencontainer als auch in den einzelnen Listen-Tags untergebracht sind.

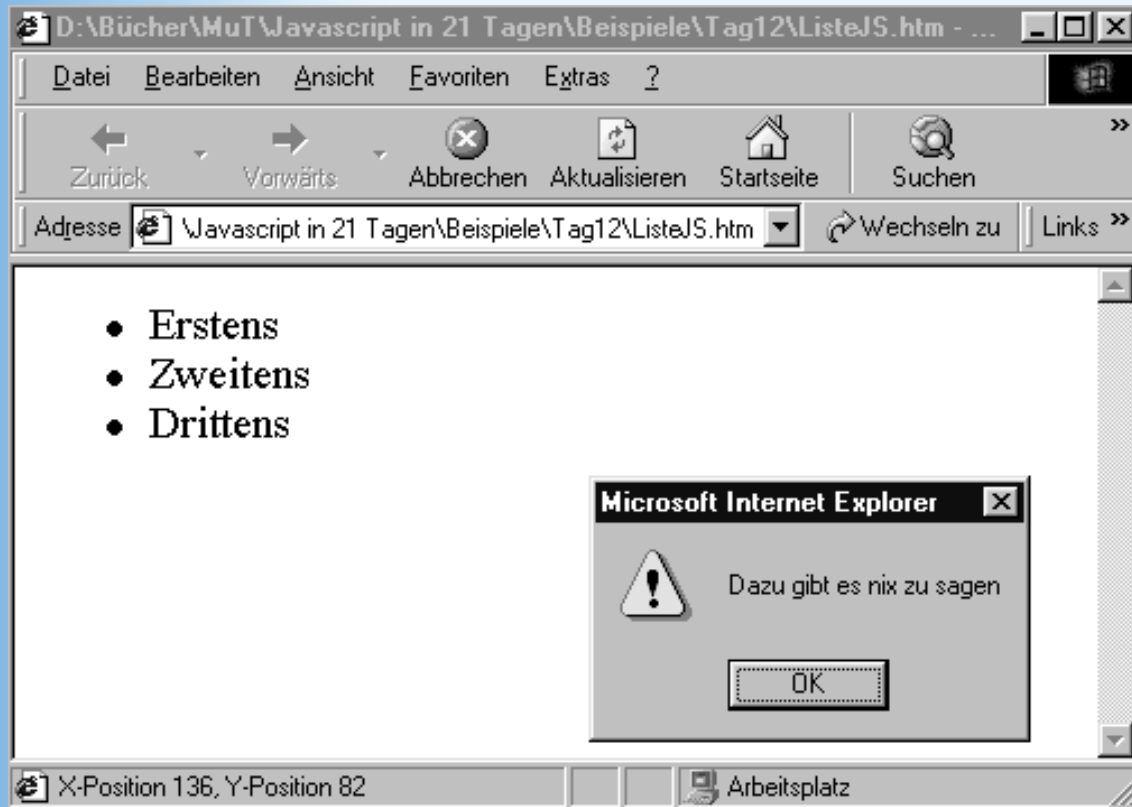


Abbildung 12.2: JavaScript-Aktivität in der Statuszeile und in Form eines Mitteilungsfensters

Bewegen Sie den Mauszeiger auf die verschiedenen Einträge in der Liste. In der Statuszeile wird jeweils die X-/Y-Position des Mauszeigers angezeigt.

Auf dem ersten Eintrag können Sie mit einem einfachen Klick eine JavaScript-Funktion aufrufen, beim zweiten Eintrag per Doppelklick und beim dritten, indem Sie den Mauszeiger einfach aus dem Bereich des Eintrags wieder herausbewegen. Beachten Sie, dass gleichzeitig das onMouseMove permanent die Statuszeile aktualisiert.

Alternative aktive Listen

Wenn man Listen aktiv machen will, sollte man nicht außer Acht lassen, dass jeder Listeneintrag auch ein Hyperlink sein kann. Von dieser Tatsache kann man vielfältig Gebrauch machen. Entweder, um die Hyperlink-Funktionalität zur Weiterleitung direkt zu verwenden, die größere Unterstützung von Event-Handlern in Hyperlinks auszunutzen oder darüber als Inline-Referenz JavaScripte aufzurufen.

Tabellen

Tabellen sind ein außergewöhnlich flexibles und wichtiges Stilmittel in Webseiten. Das Thema ist ziemlich umfangreich, und wir geben hier nur einen kleinen Überblick über das, was Sie als JavaScript-Programmierer über Tabellen in HTML wissen sollten. Für ausführlichere Details sei auf weiterführende HTML-Literatur verwiesen.

Eine Tabelle wird immer über einen sich außen befindenden Container

<TABLE>

umschlossen. Im Inneren gibt es für jede Zeile einen

<TR>

-Container, in dem dann für jede Zelle ein weiterer Container zu notieren ist. Dabei gibt es zwei Formen von Zellen. Die so genannten Kopfzellen, in denen Text hervorgehoben (zum Beispiel fett - das hängt vom Browser und den Einstellungen ab) dargestellt wird, werden in der Regel als Überschrift einer Spalte verwendet (das ist aber nicht zwingend). Kopfzellen werden mit

<TH>

festgelegt. Die andere Form von Zellen in einer Webtabelle - so genannten Datenzellen - wird mit

<TD>

festgelegt und in der Standardschriftausprägung der Seite dargestellt. Eine Tabelle ist in HTML grundsätzlich erst einmal über Zeilen und darin definierten Zellen aufgebaut.

So sieht also ein Tabellengrundgerüst aus (hier mit zwei Zeilen und zwei Spalten - die erste Zeilen beinhaltet Kopfzellen, die zweite Datenzellen):

```
<TABLE>
  <TR>
    <TH></TH>
    <TH></TH>
  </TR>
  <TR>
    <TD></TD>
    <TD></TD>
  </TR>
</TABLE>
```

Beispiel 3:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<BODY>
<TABLE>
<TR>
  <TH>Mon</TH>
  <TH>DI</TH>
</TR>
<TR>
  <TD>123</TD>
  <TD>234</TD>
</TR>
<TR>
  <TD>567</TD>
  <TD>777</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Speichern Sie das Script unter Tab1.htm und laden Sie es in einen Browser.



Abbildung 12.3: Eine einfache Tabelle

Sie werden eine 2 x 3-Tabelle erhalten. Die Tabellenstruktur ist aber kaum zu erkennen (was auch sehr oft so gewünscht ist).

Tabellen mit Linien

Wenn die Tabelle sichtbare Rahmen und Gitternetzlinien enthalten soll, müssen Sie den `<TABLE>`-Tag über den Parameter `border` erweitern. Wenn Sie das tun (`<TABLE border>`), wird die Tabelle wie folgt aussehen.



Abbildung 12.4: Eine Tabelle mit Gitterlinien

Mit der Erweiterung des `border`-Attributs um eine Wertzuweisung über die Syntax `border=[Pixel]`

können Sie auch die Dicke der umgebenden Rahmenlinien in Pixeln angeben.

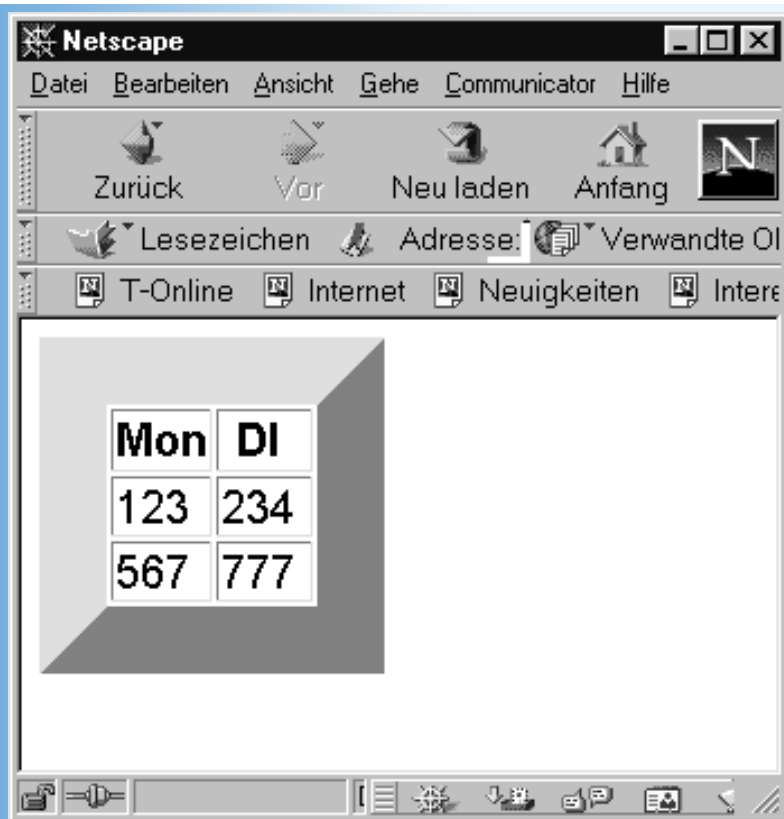


Abbildung 12.5: border=30

Die Dicke der Gitternetzlinien in Pixeln lässt sich über den Parameter cellpadding und eine Wertzuweisung festlegen. Es gilt folgende Syntax:

`cellspacing=[Pixel]`

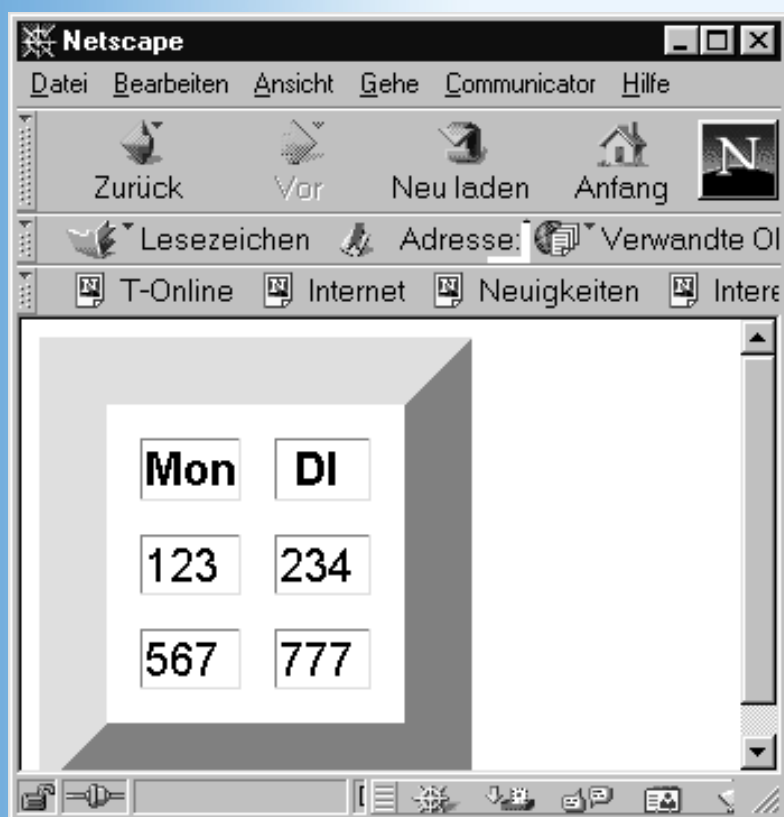


Abbildung 12.6: border=30, cellpadding=15

Tabellenausmaße

Die Höhe und die Breite von Tabellen ergibt sich normalerweise automatisch auf Grund des Inhaltes. Sie können jedoch auch die Höhe und Breite von Tabellen explizit angeben, sowohl einzeln als auch zusammen.

Dazu gibt es die allgemein üblichen Parameter

`width`

und

`height`

welche den `<TABLE>`-Tag erweitern. Sie können sowohl mit einer prozentualen Angabe als auch einer Angabe in Pixeln vorkommen. Mit einer prozentualen Angabe wird definiert, wie viel Prozent des Anzeigefensters eine Tabelle maximal einnehmen darf. Syntax:

```
<TABLE width=[Zahl]% height=[Zahl]%>
```

Damit werden die Gesamtbreite und die Gesamthöhe der Tabelle prozentual zum Anzeigefenster angegeben. Also nicht mehr der Inhalt, sondern Höhe und/oder Breite des Anzeigefensters sind relevant. Die Angaben zur Höhe und Breite können sowohl einzeln als auch zusammen in beliebiger Reihenfolge verwendet werden.

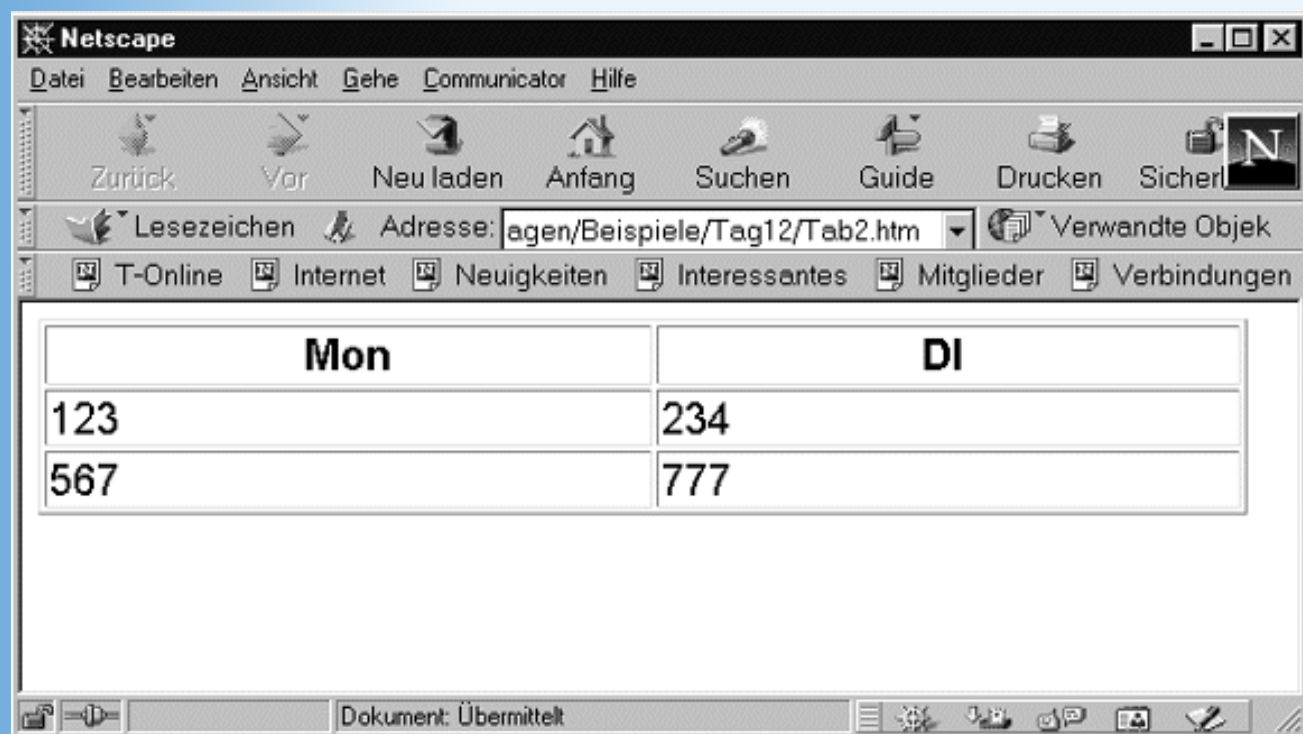


Abbildung 12.7: `width=100%`

Mit einer absoluten Angabe wird die Höhe beziehungsweise Breite in Pixeln definiert. Die Syntax ist weitgehend identisch zu der prozentualen Angabe. Es wird nur das Prozentzeichen weggelassen. Eine absolute Größenangabe der Tabelle macht diese unempfindlich gegenüber Größenveränderungen des Browserfensters. Syntax:

```
<TABLE width=[Pixel] height=[Pixel]>
```

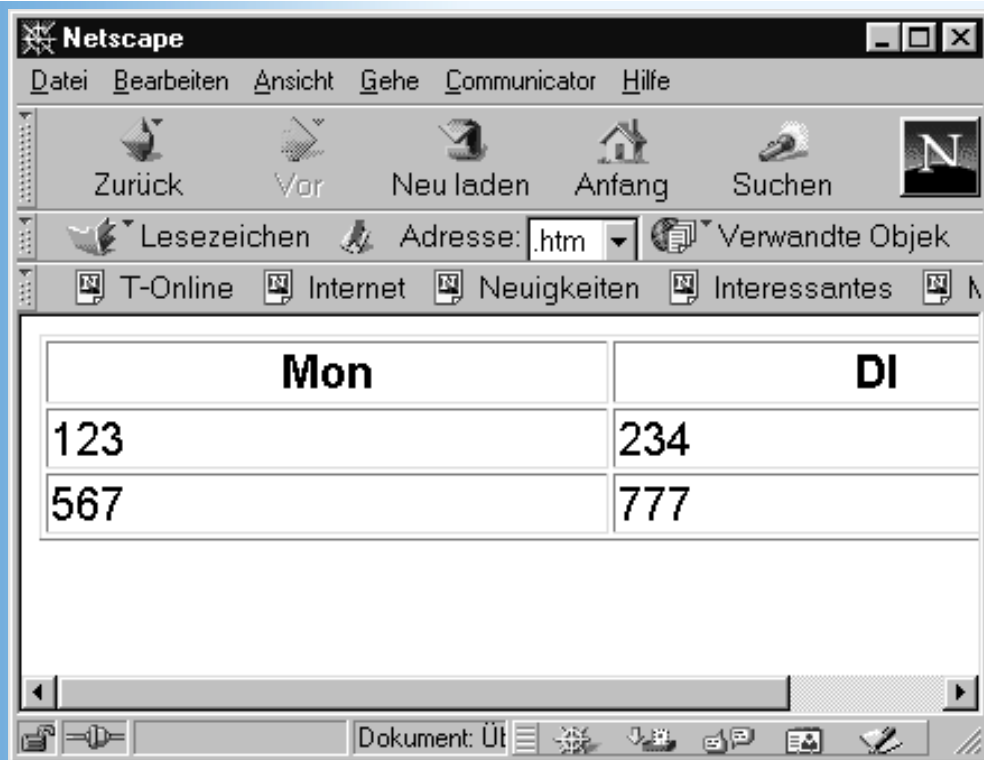


Abbildung 12.8: width=500 - das Browserfenster ist kleiner als die Tabelle

Wenn es sinnvoll erscheint, kann man Prozentangaben und absolute Größenfestlegungen auch mischen. Etwa die Breite auf 100% festlegen, die Höhe aber in Pixeln angeben.

Globale Tabellenfarben

Tabellen erlauben die Festlegung von tabellenweit gültigen Farben. Dabei gibt es zwei tabellenweite Farbangaben, welche im <TABLE>-Tag die globalen Farben in der Tabelle festlegen. Für die Farbangaben sind die üblichen RGB-Werte oder die gängigen Farbnamen erlaubt.

- Mit <TABLE bgcolor="farbe"> erfolgt die Angabe einer tabellenweiten Hintergrundfarbe.
- Mit <TABLE bordercolor="farbe"> können Sie eine tabellenweite Rahmen- und Gitternetzfarbe festlegen. Dieser Angabe macht natürlich nur Sinn, wenn sie in Verbindung mit border gesetzt wird.

Beispiel:

```
<TABLE bgcolor="red" bordercolor="green" border=10>
```

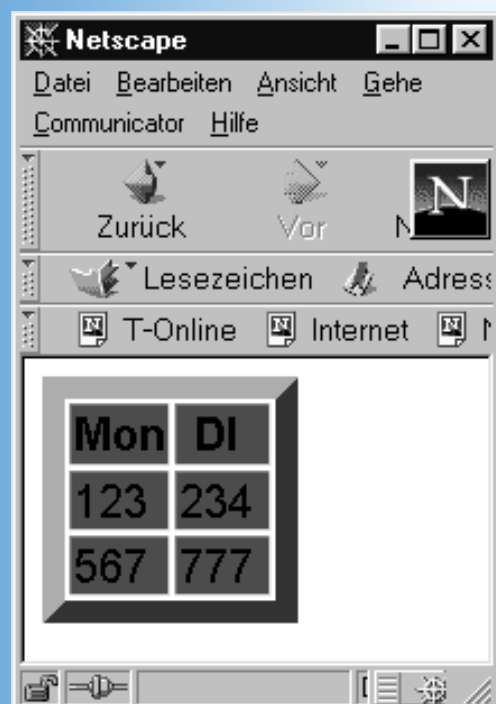


Abbildung 12.9: Verschiedene Farben beim Rahmen und Tabellenhintergrund

Es sind wieder die üblichen Farbangaben erlaubt.

Grafiken und andere Objekte in Zellen

Es können in jeder Zelle einer Tabelle statt Text auch beliebige Objekte referenziert werden. Also beispielsweise eine Grafik oder auch ein beliebiges anderes Objekt. Sie müssen nur die Referenz darauf in der Zelle notieren, wo das Objekt erscheinen soll.

Beispiel 4:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<BODY>
<TABLE
  bgcolor="red"
  bordercolor="green"
  border=10>
<TR>
  <TD>
    <IMG src="101.gif" height=100>
  </TD>
  <TD>
    <IMG src="102.gif" height=100>
  </TD>
  <TD>
    <IMG src="103.gif" height=100>
  </TD>
  <TD>
    <IMG src="104.gif" height=100>
  </TD>
</TR>
<TR>
  <TD>
    <IMG src="105.gif" height=100>
  </TD>
  <TD>
    <IMG src="106.gif" height=100>
  </TD>
  <TD>
    <IMG src="107.gif" height=100>
  </TD>
  <TD>
    <IMG src="108.gif" height=100>
  </TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Speichern Sie das Script unter TabImg.htm und laden Sie es in einen Browser.

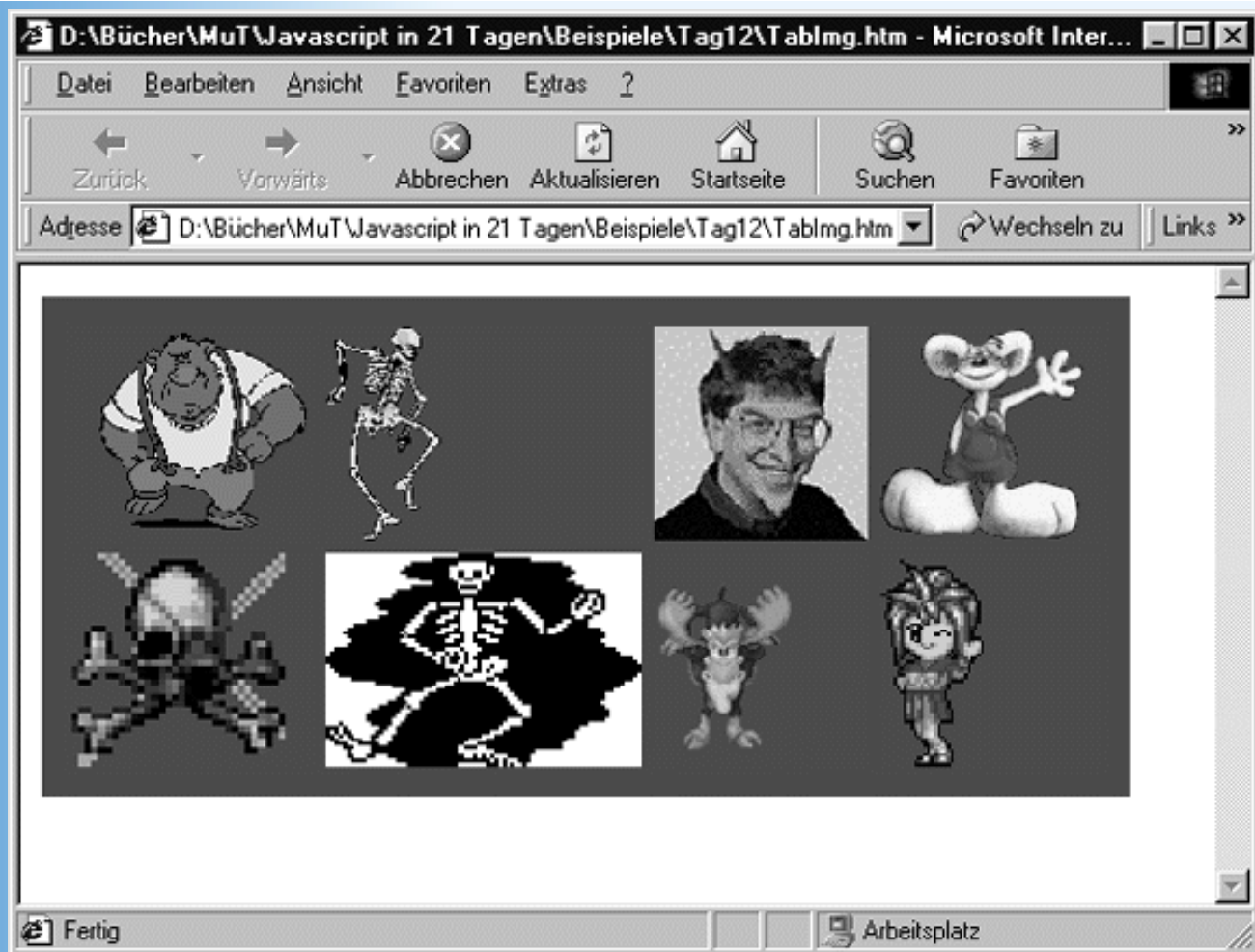


Abbildung 12.10: Bilder in einer Tabelle mit farbigem Rahmen und Zellenhintergrund

Sie werden eine 4x2-Tabelle erhalten, wo in jeder Zelle ein Bild geladen ist.

Tabellenzellengestaltung

Sie können zusätzlich zu den tabellenweiten Angaben zur Gestaltung der Tabelle auch jede Zelle individuell behandeln. Diese Angaben gelten dann entweder nur für die betreffende Zelle oder haben maximal Auswirkungen auf eine vollständige Zeile oder Spalte. Etwa wenn eine Höhenangabe für eine einzelne Zelle gemacht wird, damit aber die ganze Höhe der Zeile festgelegt ist.

Ein wichtiger Aspekt ist die Zellenausrichtung. Diese wird im Einleitungs- Tag einer Zelle mit dem üblichen Parameter `align=[Ausrichtung]`

festgelegt. Für Ausrichtung können Sie wie üblich

- left,
- center,
- justify und
- right

verwenden. Diese Angaben betreffen dann nur diese eine Zelle. Sie haben wie alle inneren Angaben eine höhere Priorität als die Angaben der externen Struktur (etwa der Zeile).

Die Festlegung von Zellenhöhe und -breite erfolgt wie üblich über die Parameter

- height und
- width.



Mit der Angabe der Höhe oder Breite einer Zelle wird auch jeweils die Höhe einer gesamten Zeile bzw. die Breite der ganzen Spalte festgelegt. Es ist deshalb sogar nur in einer Zelle der jeweilige Zeile/Spalte nötig (und sinnvoll!), eine solche Angabe zu machen.

Sie können für die Höhe und Breite von Zellen wieder prozentuale sowie Angaben in Pixeln machen. Auch Mischen ist erlaubt. Dies ist identisch zu den tabellenweiten Angaben. Sie müssen nur die gleichen Angaben innerhalb der Zellen (sowohl in Kopf- als auch in Datenzellen) möglich machen.



Grundsätzlich sind die Angaben zur Größe einer einzelnen Zelle mit Vorsicht zu verwenden, damit keine Konflikte durch widersprüchliche Angaben entstehen. Solche potentiellen Widersprüche müssen auch bezüglich der globalen Angaben einkalkuliert werden.

Der Inhalt von Zellen lässt sich auch vertikal ausrichten. Dazu werden die Zellen-Tags mit der Angabe `valign=[Ausrichtung]`

erweitert. Für Ausrichtung gibt es drei Ausprägungen:

- `top` richtet Zellen nach oben aus
- `middle` richtet Zellen mittig aus (Standardausrichtung)
- `bottom` richtet Zellen nach unten aus

Sie können in jeder Zelle die tabellenweiten Hintergrundfarben verändern. Dazu muss nur der Zellen-Tag um die Angabe `bgcolor="farbe"`

erweitert werden. Dabei darf wieder jeder darstellbare Farbwert verwendet werden.

Beispiel 5:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<BODY>
<TABLE bgcolor="yellow" bordercolor="blue"
border=10 cellpadding=5 width=90%>
<TR>
  <TD valign=top align=center>Florian</TD>
  <TD valign=middle align=Right>Der Grosse</TD>
  <TD><IMG src="101.gif" height=180></TD>
</TR>
<TR>
  <TD bgcolor=cyan valign=bottom>Felix</TD>
  <TD bgcolor=green align=center>Der Kleine</TD>
  <TD bgcolor=red>
    <IMG src="109.gif" height=180>
  </TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Speichern Sie das Script unter `Tab5.htm` und laden Sie es in einen Browser.

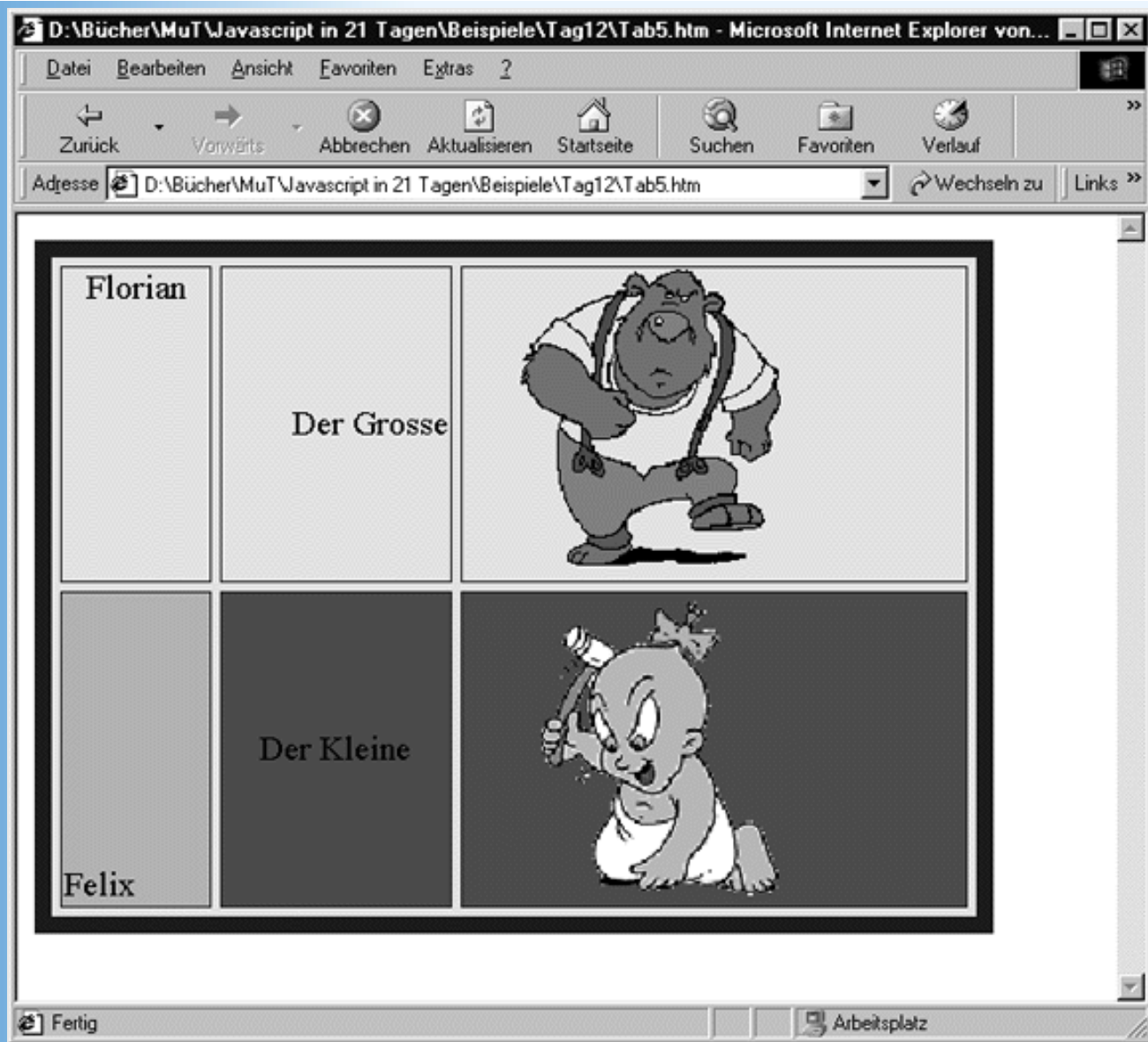


Abbildung 12.11: Verschiedene individuelle Zellausrichtungen und Farben

JavaScript auf Tabellen anwenden

Wir werden nun einen Teil unserer bisher erarbeiteten Kenntnisse in JavaScript auf Tabellen anwenden und diesen damit eine gewisse Dynamik ermöglichen. Was wir tun wollen, ist - wie bei Listen - mit Event-Handlern auf Aktionen in Tabellen reagieren.

Wir erstellen dazu eine Tabelle, die in der Statuszeile des Browsers eine nähere Erklärung des Zelleninhaltes anzeigt. Bei der zweiten Zeile der Tabelle können Sie zusätzlich auf die Zelle klicken und erhalten eine zusätzliche Erläuterung in Form eines Mitteilungsfensters.



Wir verwenden auch in der nachfolgenden Übung zwei Event-Handler, die bei den referenzierten Elementen nur vom Internet Explorer unterstützt werden. Für den Navigator müssten Sie beispielsweise die Elemente Grafik und Text durch dort unterstützte Elemente (etwa einen Hyperlink) ersetzen oder umgeben.

Beispiel 6:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function kurzText(text)
```

```
{
window.status=text;
}
function langText(text)
{
alert(text);
}
</SCRIPT>
<BODY>
<CENTER>
<TABLE width=50% height=40%>
<tr>
<th onMouseOver=
"kurzText('Wo kann man das zurückgeben?')">
Winlos
</th>
<th onMouseOver="kurzText('Watn dat?')">
Was
</th>
<th onMouseOver=
"kurzText('Wer kommt zum Essen?')">
Wer
</th>
</tr>
<tr>
<td align=center>
<img src= "201.gif" width=400
onMouseOver="kurzText('Nicht jeder mag Windows')"
onClick="langText('No Windows no cry')">
</td>
<td align=center>
<img src= "202.gif" width=80
onMouseOver="kurzText('Echt Teuflisch')"
onClick="langText('Wenn Zeit, dann bumm')">
</td>
<td align=center>
<img src= "203.gif" width=120
onMouseOver="kurzText('Sie wünschen?')"
onClick="langText('Ups')">
</td>
</tr>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Speichern Sie das Script unter TabJS.htm und laden Sie es in einen Browser.



Abbildung 12.12: Sensitive Tabellenzellen

Wir haben hier eine zentriert (Tag `<CENTER>`) im Browserfenster angezeigte Tabelle mit drei Spalten und zwei Zeilen. Die zweite Zeile enthält in jeder Zelle eine Grafik, die innerhalb der Zelle zentriert wird. In der ersten Zeile verwenden wir den Event-Handler `onMouseOver` als Attribut des Zellen- Tags, in der zweiten Zeile als Attribut der Bildreferenz. Dies hat aber beides die gleiche Auswirkung. Es wird beim Überstreichen mit dem Mauszeiger eine Funktion (`kurzText()`) aufgerufen. Damit wird in der Statuszeile des Browsers eine Meldung angezeigt. Beachten Sie, dass zwar immer nur eine Funktion aufgerufen wird, die jeweilige Meldung aber als Übergabeparameter an die Funktion übergeben wird.

Wenn Sie auf die Einträge der zweiten Zeile der Tabelle klicken, wird über den Event-Handler `onClick` eine andere Funktion aufgerufen (`langText()`). Diese zeigt den jeweils als Parameter übergebenen Wert in einem Mitteilungsfenster an (die `alert()`-Methode).

Zusammenfassung

Wir haben heute mit Listen und Tabellen zwei wichtige Gestaltungselemente des Webs kennengelernt, auf die JavaScript hervorragend angewandt werden kann. Diese HTML-Techniken sind insofern verwandt, als beide Konstruktionen mit einem sich außen befindenden Container und darin enthaltenen inneren Strukturen arbeiten.

Listen sind relativ einfach und wenig flexibel, aber das ist auch nicht deren Einsatzgebiet. Tabellen hingegen sind aus den meisten Webseiten nicht mehr wegzudenken. Darüber erhält man eine Kontrolle über eine Webseite, wie sie sonst in HTML nicht möglich ist (vielleicht mit Ausnahmen von Frames, die aber dann die Aufsplittung von Webseiten erzwingen - darauf werden wir noch kommen). Eine vergleichbare Kontrolle (und mehr) bekommt man ansonsten nur, wenn man Ergänzungstechniken wie Style Sheets hinzuzieht. Tabellen sind aber als pure HTML-Konstruktionen den Einschränkungen von Ergänzungstechniken nicht unterworfen (fehlende Unterstützung durch Browser, Ressourcenanforderungen, lange Ladezeiten, Deaktivierung in Browsern etc.), sondern werden von allen halbwegs modernen Browsern dargestellt. Dazu kommt bei aller Kontrollmöglichkeit, dass man immer noch den HTML-Interpretationsspielraum der Browser zur Verfügung hat, der komplexe Gestaltungen bei Verzicht auf absolute Festlegungen mit sehr wenigen Anweisungen ermöglicht.

Fragen und Antworten

Frage:
Ab welcher HTML-Version sind Listen dabei?

Antwort:
Ab HTML 2.0.

Frage:

Ab welcher HTML-Version sind Tabellen dabei?

Antwort:

Ab HTML 3.2. Wenn ein Browser diese Version von HTML nicht versteht, kann er Tabellen nicht darstellen. Sollten Sie diese Browser noch bedienen wollen, müssten Sie Alternativen schaffen. Den Jahrtausendwechsel konnte man aber als Ende der Schonfrist für solche Browser ansehen. Es wird wohl nicht mehr viele Browser geben, die keine Tabellen anzeigen können.

Frage:

Wie kann ich die Schrift in Tabellenzellen angeben?

Antwort:

Wie in einer normalen Webseite auch. Sämtliche Gestaltungsanweisungen können Sie auch in einer Zelle einsetzen.

Frage:

Haben wir hier alle Möglichkeiten von Tabellen behandelt?

Antwort:

Nein, aber die wichtigsten. Dennoch - mit Tabellen kann man noch viel mehr machen. Schauen Sie einfach in umfangreicheren HTML- Büchern nach.

Frage:

Tabellen und Listen sind relativ mühsam zu erstellen. Ist es da nicht einfacher, auf Tools zurückzugreifen?

Antwort:

Ja, bei der Erstellung von Listen und Tabellen erleichtern Tools die Arbeit erheblich. Allerdings müssen Sie sich im Klaren sein, dass Tool-generierte Tabellen in der Regel um ein Vielfaches größer werden als von Hand erstellte, die die Eigenschaften von HTML optimal ausnutzen. Das kann im Extremfall dazu führen, dass eine Tabelle - ohne Nutzen - um 2500 Prozent und mehr (!) aufgebläht wird. Das nachfolgende Listing entsteht beispielsweise, wenn unsere winzige Datei Tab1.htm in Excel 2000 geladen und unverändert (!) wieder gespeichert wird (wir brauchen das Listing Tab1Excel.htm gleich noch für den Workshop):

```
<html xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:x="urn:schemas-microsoft-com:office:excel"
xmlns="http://www.w3.org/TR/REC-html40">
<head>
<meta http-equiv=Content-Type content="text/html; charset=windows-1252">
<meta name=ProgId content=Excel.Sheet>
<meta name=Generator content="Microsoft Excel 9">
<link rel=File-List href="./Tab1Excel-Dateien/filelist.xml">
<link rel=Edit-Time-Data href="./Tab1Excel-Dateien/editdata.mso">
<link rel=OLE-Object-Data href="./Tab1Excel-Dateien/oledata.mso">
<!--[if gte mso 9]><xml>
  <o:DocumentProperties>
    <o:LastAuthor>RJS</o:LastAuthor>
    <o:Created>2000-07-19T08:32:30Z</o:Created>
    <o:LastSaved>2000-07-19T08:32:30Z</o:LastSaved>
    <o:Version>9.2625</o:Version>
  </o:DocumentProperties>
  <o:OfficeDocumentSettings>
    <o:DownloadComponents/>
    <o:LocationOfComponents HRef="file:E:\msowc.cab"/>
  </o:OfficeDocumentSettings>
</xml><![endif]-->
<style>
<!--table
  {mso-displayed-decimal-separator:"\,";
  mso-displayed-thousand-separator:"\.";}
@page
  {margin:.98in .79in .98in .79in;
  mso-header-margin:.49in;
  mso-footer-margin:.49in;}
```



```
tr
 {mso-height-source:auto;}
col
 {mso-width-source:auto;}
br
 {mso-data-placement:same-cell;}
.style0
 {mso-number-format:General;
 text-align:general;
 vertical-align:bottom;
 white-space:nowrap;
 mso-rotate:0;
 mso-background-source:auto;
 mso-pattern:auto;
 color:windowtext;
 font-size:10.0pt;
 font-weight:400;
 font-style:normal;
 text-decoration:none;
 font-family:Arial;
 mso-generic-font-family:auto;
 mso-font-charset:0;
 border:none;
 mso-protection:locked visible;
 mso-style-name:Standard;
 mso-style-id:0;}
td
 {mso-style-parent:style0;
 padding-top:1px;
 padding-right:1px;
 padding-left:1px;
 mso-ignore:padding;
 color:windowtext;
 font-size:10.0pt;
 font-weight:400;
 font-style:normal;
 text-decoration:none;
 font-family:Arial;
 mso-generic-font-family:auto;
 mso-font-charset:0;
 mso-number-format:General;
 text-align:general;
 vertical-align:bottom;
 border:none;
 mso-background-source:auto;
 mso-pattern:auto;
 mso-protection:locked visible;
 white-space:nowrap;
 mso-rotate:0;}
.xl24
 {mso-style-parent:style0;
 font-weight:700;
 text-align:center;
 vertical-align:middle;
 white-space:normal;}
.xl25
 {mso-style-parent:style0;
 white-space:normal;}
```

```
-->
</style>
<!--[if gte mso 9]><xml>
  <x:ExcelWorkbook>
    <x:ExcelWorksheets>
      <x:ExcelWorksheet>
        <x:Name>Tab1</x:Name>
        <x:WorksheetOptions>
          <x:DefaultRowHeight>264</x:DefaultRowHeight>
          <x:DefaultColWidth>10</x:DefaultColWidth>
          <x:Selected/>
          <x:DoNotDisplayGridlines/>
          <x:Panes>
            <x:Pane>
              <x:Number>3</x:Number>
              <x:RangeSelection>A1:B3</x:RangeSelection>
            </x:Pane>
          </x:Panes>
          <x:ProtectContents>False</x:ProtectContents>
          <x:ProtectObjects>False</x:ProtectObjects>
          <x:ProtectScenarios>False</x:ProtectScenarios>
        </x:WorksheetOptions>
      </x:ExcelWorksheet>
    </x:ExcelWorksheets>
    <x:WindowHeight>6720</x:WindowHeight>
    <x:WindowWidth>11760</x:WindowWidth>
    <x:WindowTopX>384</x:WindowTopX>
    <x:WindowTopY>264</x:WindowTopY>
    <x:ProtectStructure>False</x:ProtectStructure>
    <x:ProtectWindows>False</x:ProtectWindows>
  </x:ExcelWorkbook>
</xml><![endif]>-->
</head>
<body link=blue vlink=purple>
<table x:str border=0 cellpadding=0 cellspacing=0 width=63 style='border-collapse:
collapse;table-layout:fixed;width:47pt'>
  <col width=34 style='mso-width-source:userset;mso-width-alt:1194;width:25pt'>
  <col width=29 style='mso-width-source:userset;mso-width-alt:1024;width:22pt'>
  <tr height=18 style='height:13.2pt'>
    <td height=18 class=xl24 width=34 style='height:13.2pt;width:25pt'>Mon</td>
    <td class=xl24 width=29 style='width:22pt'>DI</td>
  </tr>
  <tr height=18 style='height:13.2pt'>
    <td height=18 class=xl25 align=right width=34 style='height:13.2pt;
width:25pt' x:num>123</td>
    <td class=xl25 align=right width=29 style='width:22pt' x:num>234</td>
  </tr>
  <tr height=18 style='height:13.2pt'>
    <td height=18 class=xl25 align=right width=34 style='height:13.2pt;
width:25pt' x:num>567</td>
    <td class=xl25 align=right width=29 style='width:22pt' x:num>777</td>
  </tr>
  <![if supportMisalignedColumns]>
  <tr height=0 style='display:none'>
    <td width=34 style='width:25pt'></td>
    <td width=29 style='width:22pt'></td>
  </tr>
  <![endif]>

```

```
</table>
</body>
</html>
```

Ohne irgendeinen Nutzen (wenn man in Bezug auf das Web argumentiert - nicht bei HTML als Ersatz für das hauseigene Dateiformat) hat sich die Größe der Datei um den Faktor 25 verschlechtert. Man kann beim Speichern diese Verschlimmerung auch nicht verhindern und eine nachträgliche Optimierung ist extrem schwer. Aber nicht nur Excel arbeitet so - gelinde gesagt - unökonomisch. Auch die meisten anderen Office-Programme und schwache HTML-Tools speichern Tabellen als extrem schlechten Code ab. Ein Großteil des Problems dürfte darauf beruhen, dass die diese Programme versuchen, Tabellen im Aussehen absolut festzulegen. Abgesehen davon, dass dies in vielen Fällen unnötig ist (eine einfache Einstellungsmöglichkeit im Programm könnte den meisten Müll verhindern) - es ist im Web, wie wir schon vielfach gesehen haben, unmöglich. Ein weiterer Effekt ist, dass ältere Browser so eine Tabelle nicht darstellen werden. Vollkommen unsinnig, denn schauen Sie noch mal in dem Beispiel selbst nach, was dargestellt werden soll.

Workshop

Erstellen Sie mit einem grafischen HTML-Editor Webseiten mit Tabellen und Listen, etwa mit dem Composer oder Frontpage. Analysieren Sie hinterher den Quelltext. Am besten erstellen Sie die Tabellen und Listen, welche in diesem Kapitel als Beispiele verwendet wurden. Achten Sie besonders darauf, wie viele - überflüssige - Anweisungen gerade bei diesen Konstruktionen durch solche Tools erstellt werden.

Wenn Sie Excel besitzen, laden Sie die Beispiele aus dem Kapitel und speichern Sie diese mit Excel wieder. Analysieren Sie den Quelltext. Falls Sie kein Excel 2000 verwenden, können Sie die Datei Tab1Excel.htm aus dem Abschnitt »Fragen & Antworten« verwenden.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Wie viele verschiedene Listenformen gibt es in HTML?
2. Was stimmt nicht an dem Listing?

```
<UL>
<LI type="i">Erstens</LI>
<LI>Zweitens</LI>
<LI>Drittens</LI>
</UL>
```

3. Was stimmt nicht an dem Listing?

```
<UL>
<LI>Erstens</LI>
<LI>Zweitens</LI>
<LI>Drittens</LI>
</OL>
```

4. Was stimmt nicht an dem Listing?

```
<TABLE>
<TR>
  <TH>Mon</TH>
  <TH>DI  </TH>
</TR>
<TD>123</TD>
  <TD>234</TD>
</TR>
</TABLE>
```

5. Ab welcher HTML-Version werden Tabellen unterstützt?

❖Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 2

Tag 13

Vorgefertigte JavaScript-Funktionen und Top-Level- Eigenschaften

In den letzten Tagen haben wir mehrfach JavaScript-Funktionen erstellt. Zum Teil haben wir damit Funktionalitäten nachprogrammiert, die über die JavaScript-Grundsyntax nicht direkt bereitstehen. Denken Sie beispielsweise an die Fakultätsberechnung oder die Quadratberechnung. Oder die Maßnahmen, die wir durchgeführt haben, damit ein String mit Zahlen darin auch von JavaScript numerisch behandelt wurde. Mit Hilfe der JavaScript-Operatoren, Variablen und Kontrollstrukturen wurden die Funktionalitäten programmiert und in Funktionen eingebunden bzw. soweit vorbereitet, dass sie nur noch in ein umgebendes Funktionsgerüst eingebunden werden mussten.

Neben diesen von Ihnen selbst geschriebenen Funktionen stellt JavaScript aber eine Reihe von integrierten Funktionen bereit, die Sie an jeder Stelle in einem JavaScript nutzen können und die Ihnen vielfach die Programmierung von Abläufen abnehmen (wozu immer das Rad neu erfinden?). Dabei soll explizit nicht an die objektgebundenen Methoden gedacht werden (diese gibt es natürlich auch noch - sie beinhalten sogar den Großteil der nützlichen JavaScript-Funktionalitäten), sondern an objektunabhängige Funktionen. Wir wollen uns heute diese Funktionen vornehmen und sehen, wie sie in der täglichen JavaScript-Programmierung genutzt werden können. Dabei nutzen einige dieser Funktionen (aber auch viele der später noch behandelten Methoden) so genannte Top-Level- Eigenschaften. Dies sind Werte mit definierter Bedeutung, die an jeder Stelle in einem JavaScript bereitstehen.

Top-Level-Eigenschaften

JavaScript kennt seit der Version 1.3 drei standardisierte Top-Level- Eigenschaften, die in mehreren vorgefertigten Funktionen von JavaScript genutzt werden und auch in selbstdefinierten Funktionen eingesetzt werden können.

- Die Eigenschaft `Infinity` ist ein numerischer Wert, der den Wert unendlich repräsentiert. Allgemein implementiert ist diese Eigenschaft erst in JavaScript 1.3, obwohl sie davor für das `Number`-Objekt vorhanden war. Der Initialisierungswert von `Infinity` ist `Number.POSITIVE_INFINITY` (positiv unendlich).
- Die Eigenschaft `NaN` ist ein Wert, der `Not a Number` repräsentiert und allgemein erst seit JavaScript 1.3 implementiert ist. Davor war die Eigenschaft nur für das `Number`-Objekt vorhanden. `NaN` ist immer ungleich jeder Zahl und auch zu sich selbst (im Sinne von einem Vergleich mit einem JavaScript-Operator). Daher kann man nie direkt in einem Vergleich auf Gleichheit mit `NaN` prüfen (dafür gibt es die nachfolgend noch besprochene Funktion `isNaN()`). Verschiedene Methoden und Funktionen liefern `NaN` als Rückgabewert, wenn ein Rückgabewert keine Zahl ergibt.
- Die Eigenschaft `undefined` ist wie die beiden anderen Eigenschaften seit JavaScript 1.3 implementiert. Sie repräsentiert den Zustand einer Variablen, der noch kein Wert zugewiesen wurde.

Die Eigenschaft `undefined` werden wir jetzt im Rahmen eines Scripts nutzen, um vor einer kritischen Situation abzufragen, ob überhaupt eine Variable bereits einen Wert zugewiesen bekommen hat. Das Beispiel ist zwar ein wenig konstruiert, aber in größeren Scripts kann es leicht zu solchen Konstellationen kommen, wie wir sie hier simulieren wollen.

Beispiel 1:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var a, b;
function ea()
{
a = prompt("Eingabe Variable A","");
}
function eb()
{
b = prompt("Eingabe Variable b","");
}
function rechne()
{
alert(a*b);
}

</SCRIPT>
<BODY>
<BUTTON onClick="ea()">Eingabe Variable A</BUTTON>
<BUTTON onClick="eb()">Eingabe Variable B</BUTTON>
<BUTTON onClick="rechne()">Berechnung</BUTTON>
</BODY>
</HTML>
```

Speichern Sie das Script unter Undefined.htm und laden Sie es in den Internet Explorer (nur da läuft es - wenn Sie es im Navigator anwenden wollen, müssen Sie ein Formular mit drei Buttons erstellen, dann funktioniert es auch da). Wenn Sie auf den Button zur Eingabe der Variablen **a** klicken, können Sie in einem Eingabedialog eine Zahl eingeben.

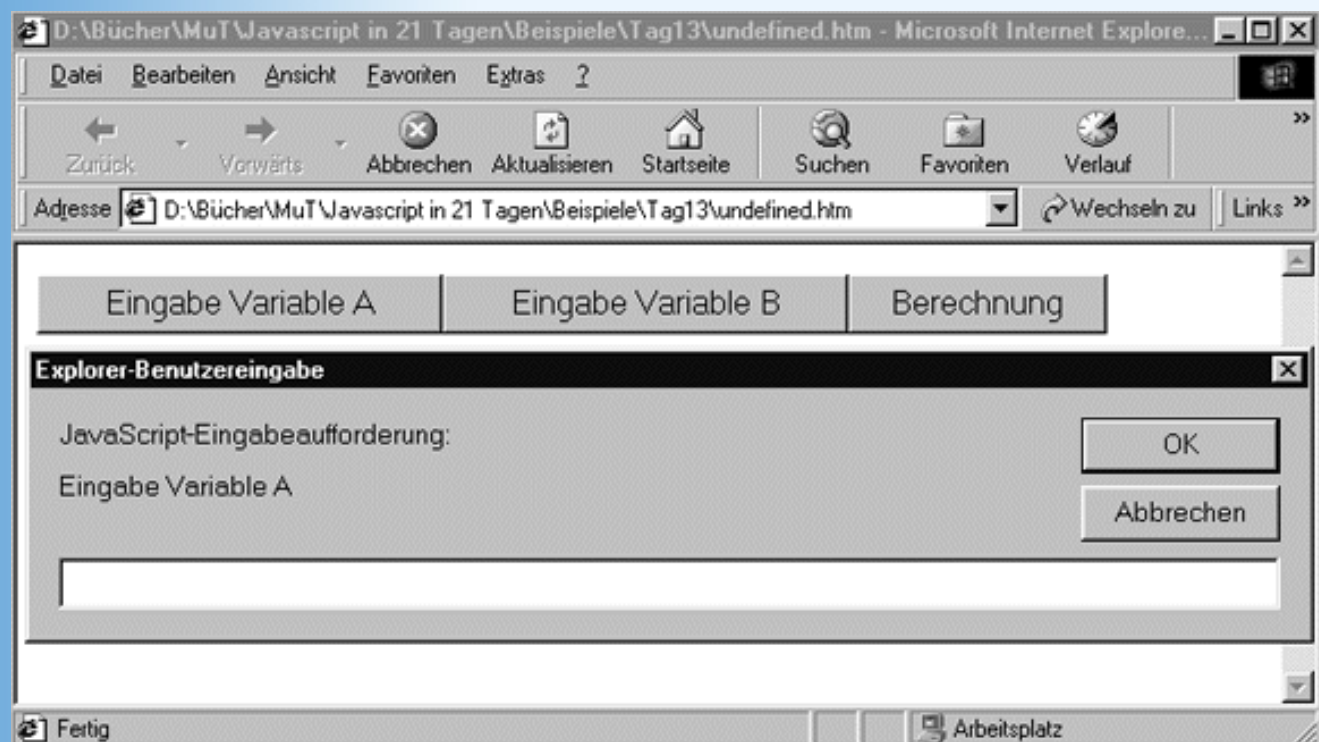


Abbildung 13.1: Eingabe der Variablen A

Analog verhält es sich mit der Variablen **b**. Wenn Sie dann nach Eingabe der beiden Variablen auf den **Berechnen**-Button klicken, werden die beiden Zahlen multipliziert und dann das Ergebnis ausgegeben.

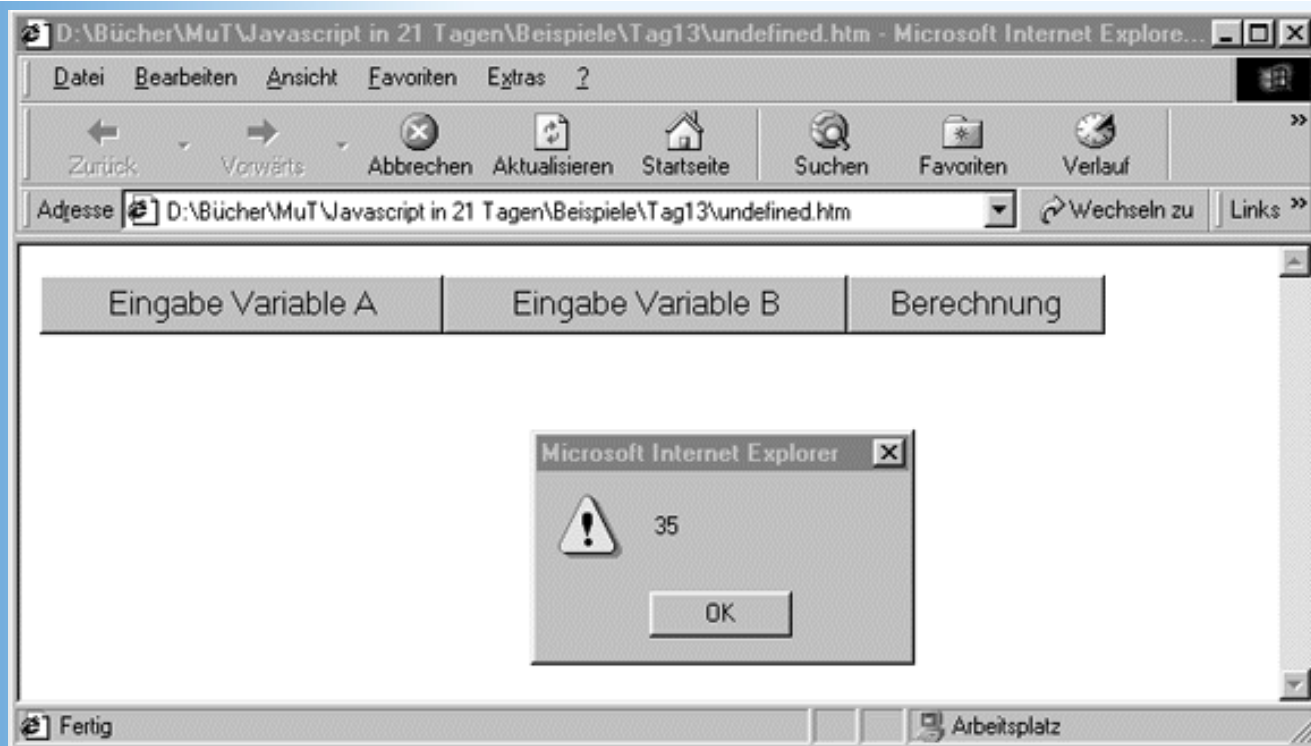


Abbildung 13.2: Ergebnis einer erfolgreichen Berechnung

So weit, so gut. Aber was ist, wenn der Anwender die eben beschriebene Reihenfolge nicht einhält? Es ist nun mal zentraler Aspekt einer grafischen Oberfläche, dass eine bestimmte Reihenfolge der Bedienung von Komponenten in der Regel nicht gefordert wird. In unserem Beispiel kann der Anwender ja auch den **Berechnen**-Button anklicken, bevor die Variablen vollständig eingegeben wurden. Dann gibt es ein Problem (NaN).

Wenn schon nicht die Reihenfolge der Bedienung der Buttons auf der Oberfläche vorgegeben werden kann oder soll, soll unser Script doch soweit optimiert werden, dass die Berechnung nur dann durchgeführt wird, wenn beide Variablen mit Werten belegt wurden (explizites Prüfen auf »numerisch« soll hier noch nicht geschehen - das machen wir in einem späteren Beispiel). Andernfalls soll eine Meldung auf einen Fehler hinweisen.

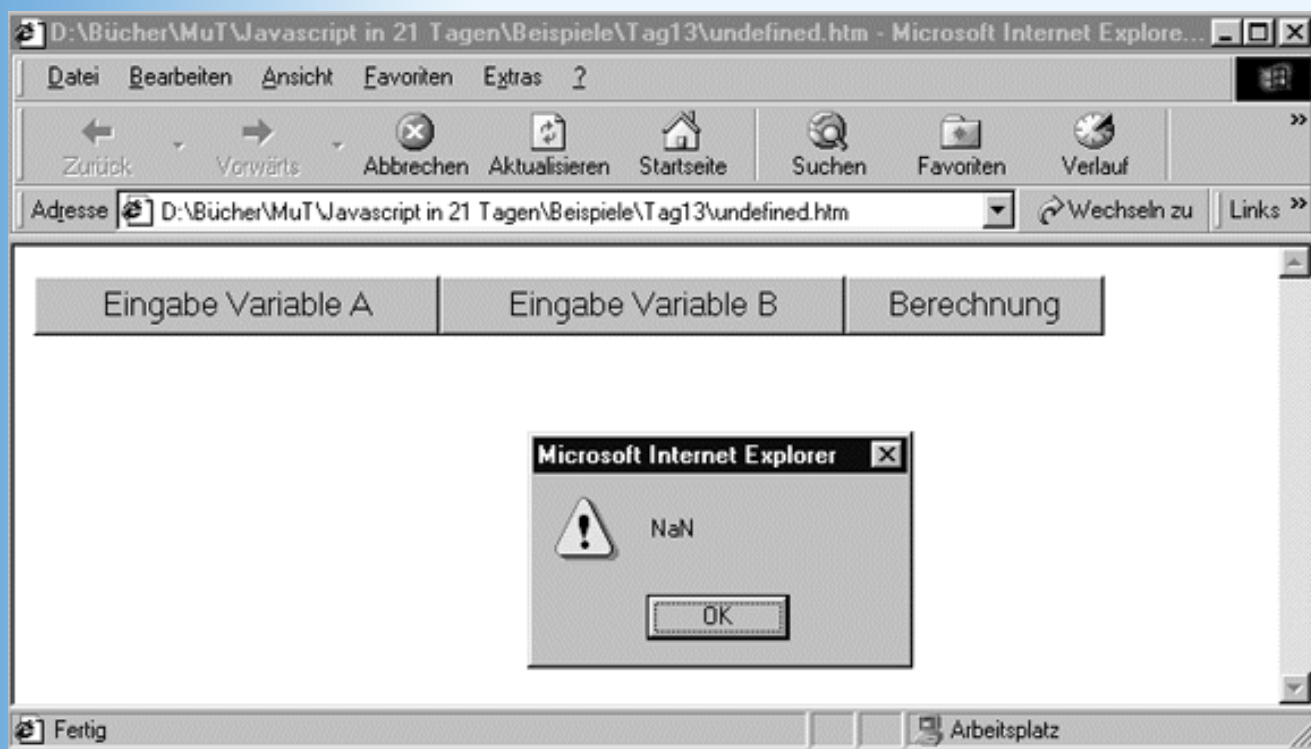


Abbildung 13.3: Ergebnis einer nicht erfolgreichen Berechnung

Beispiel 2:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var a, b;
function ea()
{
a = prompt("Eingabe Variable A","");
}
function eb()
{
b = prompt("Eingabe Variable b","");
}
function rechne()
{
if((typeof a) == "undefined")
{
alert(
"Variable A muss vor einer Berechnung gefüllt werden");
}
else if((typeof b) == "undefined")
{
alert(
"Variable B muss vor einer Berechnung gefüllt werden");
}
else
{
alert(a*b);
}
}
</SCRIPT>
<BODY>
<BUTTON onClick="ea()">Eingabe Variable A</BUTTON>
<BUTTON onClick="eb()">Eingabe Variable B</BUTTON>
<BUTTON onClick="rechne()">Berechnung</BUTTON>
</BODY>
</HTML>
```

Speichern Sie das Script unter TestUndefined.htm und laden Sie es in den Internet Explorer. Wenn Sie nun auf den **Berechnen**-Button klicken, bevor beiden Variablen ein Wert zugewiesen wurde, wird die Berechnung nicht durchgeführt. Statt dessen erscheint eine Meldung.



Abbildung 13.4: Meldung beim Fehlen des Werts der Variablen A

In der Funktion rechne() wird zu der Kontrolle, ob eine Variable bereits einen Wert zugewiesen wurde, auf den typeof-Operator zurückgegriffen. Wenn eine Variable noch keinen Wert hat, liefert er den Wert undefined.



Wir werden die anderen Eigenschaften bei den nachfolgend behandelten Funktionen in der Anwendung

sehen.

Konvertierung eines Wertes in eine Zahl

Erinnern Sie sich an unsere Beispiele, wo ein Anwender in einem Eingabeprompt eine Zahl eingegeben und JavaScript diese Zahl als String behandelt hat (beispielsweise an Tag 8 die Beispiele `add2.htm` oder `telefonnr.htm`)? Unser Trick, um eine Zahl zu erzwingen, war die Multiplikation mit der Zahl 1. JavaScript konvertiert dann das Ergebnis der Multiplikation automatisch in eine Zahl, ohne den Wert zu verändern. Das ist aber wie gesagt ein Trick, und JavaScript bietet eine eigene Funktion an, die das genauso erledigt.

Die Funktion

`Number([Objekt/Ausdruck])`

konvertiert den Inhalt eines als Parameter übergebenen Objektes oder eines Ausdrucks in eine Zahl und gibt die Zahl zurück.

Beispiel 3:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var a;
var b;
a = Number(
prompt("Geben Sie die erste Zahl ein",""));
b = Number(
prompt("Geben Sie die zweite Zahl",""));
document.write(
"Das Ergebnis von ", a, " + ", b, " ist : ");
document.write(a + b);
document.write("<br>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter `add.htm` und laden Sie es in einen Browser. Geben Sie in den beiden Eingabemasken jeweils eine Zahl ein. In der `Number()`-Funktion wird direkt die `prompt()`-Funktion (oder genauer - Methode) notiert. Der Rückgabewert als String wird vor der Zuweisung zu der Variablen konvertiert, sodass die Addition der beiden Variablen numerisch erfolgt.

Wenn sich in der `Number()`-Funktion das übergebene Objekt nicht umwandeln lässt, wird ein definierter Wert `NaN` (Not a Number) zurückgegeben. Wenn Sie also in dem Beispiel keine Zahlen eingeben, wird dieser Wert zurückgeliefert.

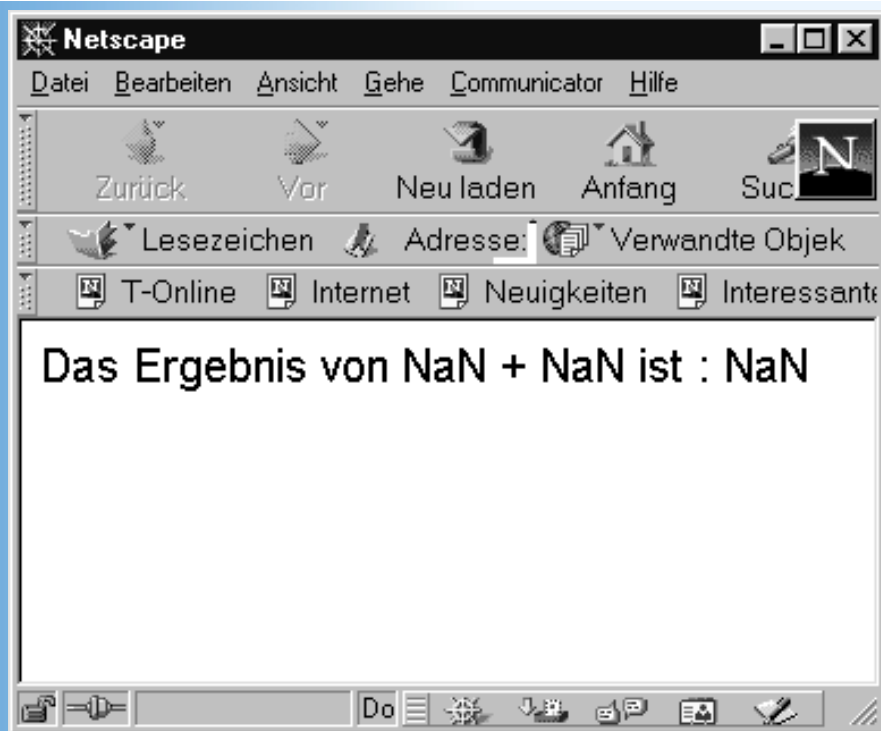


Abbildung 13.5: Keine numerischen Eingabewerte

Dieser Wert NaN kann gezielt verwendet werden. Etwa um damit im Rahmen einer Bedingung in einer Kontrollflussanweisung sinnvoll zu reagieren. Dazu gibt es eine weitere wichtige Funktion:

`isNaN([Ausdruck])`

Diese Funktion prüft den übergebenen Wert, ob er ein nichtnumerischer Wert ist (Rückgabe true) oder nicht (false). In JavaScript 1.0 wurde die Funktion nur unter Unix unterstützt, seit JavaScript 1.1 gibt es sie auf allen Plattformen.

Wenn wir uns nun unser Beispiel ansehen, könnte diese Funktion auf verschiedenste Weise ausgenutzt werden. Etwa dass eine qualifizierte Fehlermeldung ausgegeben wird. Oder es wird je nach Typ des eingegebenen Werts eine numerische Addition oder eine String-Addition ausgeführt. Wir wollen uns für eine dritte Möglichkeit entscheiden - der Anwender wird nach der Eingabe eines Strings gezwungen, seine Eingabe zu korrigieren. Andernfalls wird das Script nicht weiter ausgeführt.

Beispiel 4:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var a;
var b;
do
{
a = Number(prompt("Geben Sie die erste Zahl ein",""));
if(isNaN(a)) alert(
"Sie dürfen hier nur numerische Werte eingeben");
}
while(isNaN(a));
do
{
b = Number(prompt("Geben Sie die zweite Zahl",""));
if(isNaN(b)) alert(
"Sie dürfen hier nur numerische Werte eingeben");
}
while(isNaN(b));
document.write(
"Das Ergebnis von ", a, " + ", b, " ist : ");
```



```
document.write(a + b);  
document.write("<br>");  
</SCRIPT>  
<BODY>  
</BODY>  
</HTML>
```

Speichern Sie das Script unter NaN.htm und laden Sie es in einen Browser. Testen Sie unseren Sicherheitsmechanismus mit der do-while-Schleifen, indem Sie in den beiden Eingabemasken keine Zahl eingeben. Sowohl in der Abbruchbedingung der Schleife als auch in der if-Abfrage wird die Funktion isNaN() genutzt, um für den Fall einer nichtnumerischen Eingabe den boolschen Wert true zurückgeben zu lassen.

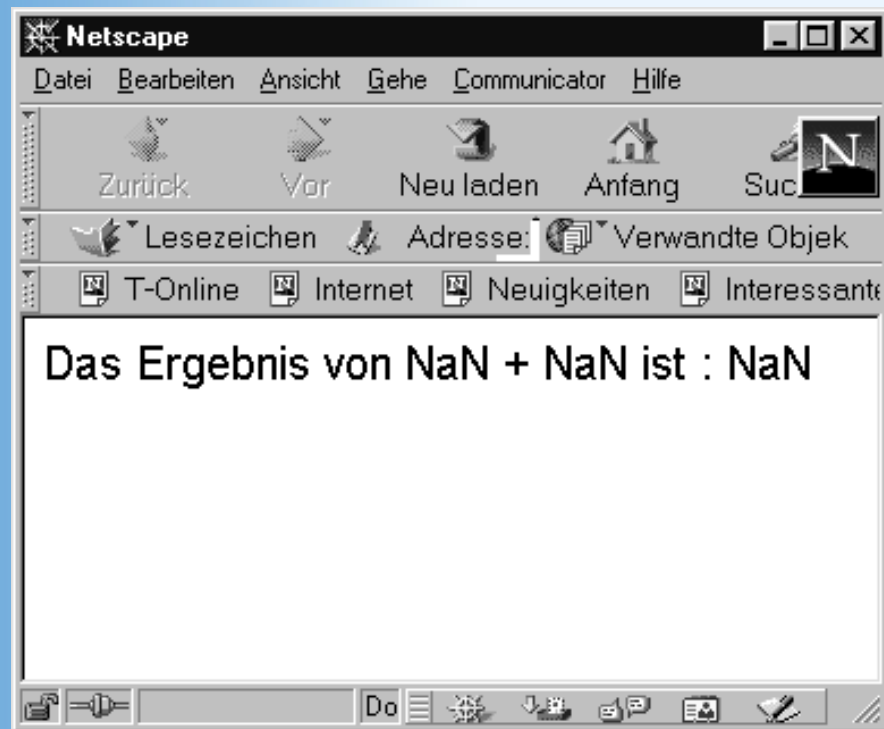


Abbildung 13.6: Zweimal leere Eingabe



Das Beispiel erzeugt keine Endlos-Schleife wie viele vergleichbare Konstruktionen, wo der Anwender nicht mehr aus der Eingabemaske heraus kann. Sowohl die Eingabe »leer« als auch die Betätigung der **Abbrechen**-Schaltfläche wird einen Leerwert erzeugen, der in 0 umgewandelt werden kann. Damit ist die Bedingung erfüllt. Für den Fall, dass sie einen numerischen Wert ungleich 0 benötigt (etwa bei einer Division), muss man zusätzliche Schritte unternehmen. Etwa zusätzlich auf ungleich 0 prüfen oder so.

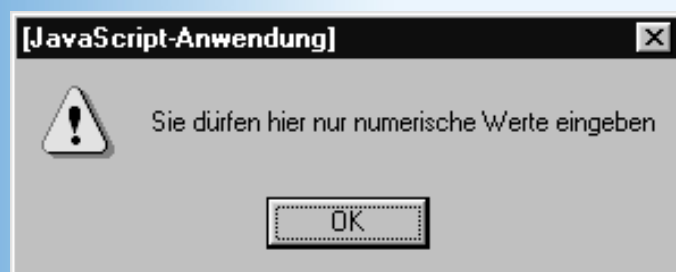


Abbildung 13.7: Die Fehlermeldung

Wir wollen das Beispiel noch in Hinsicht auf eine effektivere Programmierung optimieren. Es bietet sich an, die Eingabe samt Kontrolle des Typs der Eingabe in eine eigene Funktion auszulagern und mit Übergabe- und Rückgabewerten zu arbeiten.

Beispiel 5:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var a, b;
function typTest(text)
{
do
{
wert = Number(
prompt("Geben Sie die "+ text +" Zahl ein",""));
if(isNaN(wert))
alert("Sie dürfen hier nur numerische Werte eingeben");
}
while(isNaN(wert));
return wert;
}
a=typTest("erste");
b=typTest("zweite");
document.write(
"Das Ergebnis von ", a, " + ", b, " ist : ");
document.write(a + b);
document.write("<br>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Speichern Sie das Script unter NaN2.htm und laden Sie es in einen Browser. Beachten Sie den Aufbau der Funktion. Als Übergabewert wird ein Text übergeben, um damit eine individuelle Fehlermeldung aufzubauen. Der Rückgabewert wird dann den Variablen zugewiesen.



Der folgende Vergleich wird nicht funktionieren:

if(a==NaN)

Man muss für Vergleich auf NaN zwingend auf die Funktion isNaN() zurückgreifen.

In den gleichen Zusammenhang wie Number () sind auch die Funktionen eval([Zeichenkette]), parseFloat([Zeichenkette]) und parseInt([Zeichenkette]) zu setzen:

- eval([Zeichenkette]) betrachtet die übergebene Zeichenkette als Zahlen mit zugehörigen Operatoren und berechnet das Ergebnis. Der Name der Funktion steht für »evaluieren«, was ja auswerten bedeutet. Der String kann jede gültige JavaScript-Befehlsstruktur sein. Implementiert ist die Funktion bereits seit JavaScript 1.0, aber erst seit JavaScript 1.1 ist die Funktion für alle Objekte einsetzbar. Number() führt zwar auch Operationen aus, aber nur, wenn diese als Ergebnis eindeutig eine Zahl ergeben, es also eindeutig identifizierbare arithmetische Ausdrücke sind.
- parseFloat([Zeichenkette]) durchsucht ein String-Argument und wandelt eine übergebene Zeichenkette in eine Kommazahl um und gibt diese als Ergebnis zurück. Insbesondere kann die Zeichenkette nach den Zahlen auch Text enthalten. Ab dem ersten nichtnumerischen Zeichen wird abgebrochen und alle davor gefundenen Zahlen werden zurückgegeben (Vorzeichen und führende bzw. trennende Leerzeichen sind erlaubt.). Eventuell danach noch folgende Zahlen werden nicht mehr berücksichtigt. Das ist z.B. sehr sinnvoll, wenn ein Zahlenwert wie ein Geldbetrag samt Währungskennung in einer Zeichenkette steht. Oder eine Prozentangabe samt Prozentzeichen. Wenn bereits das erste Zeichen von links in der Zeichenkette keine Zahl ist, wird NaN zurückgegeben. Die Funktion ist seit JavaScript 1.0 implementiert, hatte aber dort nur auf Solaris- und Irix-Systemen NaN zurückgeliefert, wenn das erste Zeichen ein Buchstabe ist (sonst 0).

- `parseInt([Zeichenkette])` wandelt eine übergebene Zeichenkette in eine Ganzzahl um und gibt diese als Ergebnis zurück. Auch bei einer Gleitkommazahl wird der Nachkomma-Anteil abgeschnitten. Ansonsten ist die Funktion analog `parseFloat()` zu sehen.

Um die Unterschiede zu `Number()` zu sehen, testen wir die Funktionen in Beispielen.

Beispiel 6:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var a;
a = Number("4+5");
b = eval("4+5");
document.write("Der Wert von a ist : ", a);
document.write("<br>");
document.write("Der Wert von b ist : ", b);
document.write("<br>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

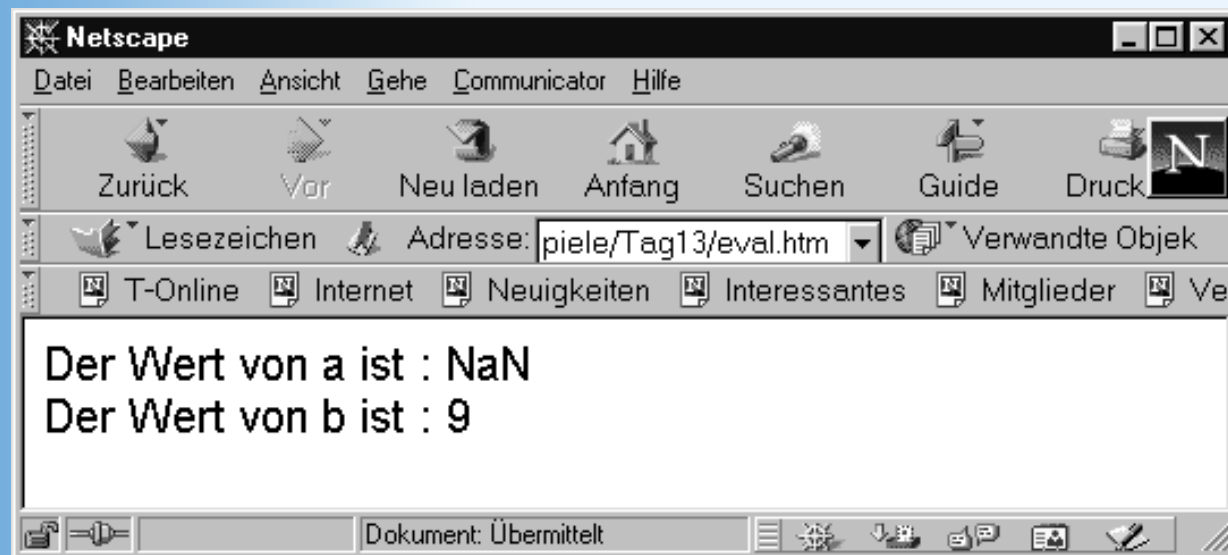


Abbildung 13.8: Einmal wird die Berechnung ausgeführt, das andere Mal nicht

Speichern Sie das Script unter `Eval.htm` und laden Sie es in einen Browser. `Number()` liefert bei einer Zeichenkette, in der eine Berechnung stattfindet, `NaN`, `eval()` führt die Berechnung aus und liefert eine Zahl zurück.



Die Funktion `eval()` sollte nicht zum Ausführen eines arithmetischen Ausdrucks verwendet werden (also, wenn der Ausdruck nicht als String oder sonst wie missverstanden werden kann). Diese werden in JavaScript automatisch ausgeführt.

Beispiel 7:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
a = Number(4.1234);
b = parseInt(4.1234);
c = Number("4.1234");
```

```
d = parseFloat("4.1234");
e = Number("4.1234%");
f = parseFloat("4.1234%");
g = parseInt("4.1234 und eine Menge Text");

document.write("Der Wert von a ist : ", a);
document.write("<br>");
document.write("Der Wert von b ist : ", b);
document.write("<br>");
document.write("Der Wert von c ist : ", c);
document.write("<br>");
document.write("Der Wert von d ist : ", d);
document.write("<br>");
document.write("Der Wert von e ist : ", e);
document.write("<br>");
document.write("Der Wert von f ist : ", f);
document.write("<br>");
document.write("Der Wert von g ist : ", g);
document.write("<br>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```



Abbildung 13.9: Die verschiedenen Wirkungen von `Number()` und `parseFloat()` bzw. `parseInt()`

Speichern Sie das Script unter `Parse.htm` und laden Sie es in einen Browser. Beachten Sie, wie `parseFloat()` und `parseInt()` wirken.

Es gibt noch eine ganz interessante Erweiterung von `parseInt()`. Als zweiten Parameter kann man eine Basis angeben. Das ist ein Wert zwischen 2 und 36. Schauen wir uns an, wie er wirkt.

Beispiel 8:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
for(i=2;i<=10;i++)
{
document.write("Basis : " , i , ": ");
document.write(parseInt("4abc",i));
document.write(" ");
document.write(parseInt("5",i));
document.write(" ");
document.write(parseInt("6c",i));
document.write(" ");
document.write(parseInt("7ddd",i));
document.write("<br>");
}
for(i=11;i<=36;i++)
{
document.write("Basis : " , i , ": ");
document.write(parseInt("a",i));
document.write(" ");
document.write(parseInt("b",i));
document.write(" ");
document.write(parseInt("c",i));
document.write(" ");
document.write(parseInt("d",i));
document.write(" ");
document.write(parseInt("e",i));
document.write(" ");
document.write(parseInt("f",i));
document.write(" ");
document.write(parseInt("g",i));
document.write("<br>");
}
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

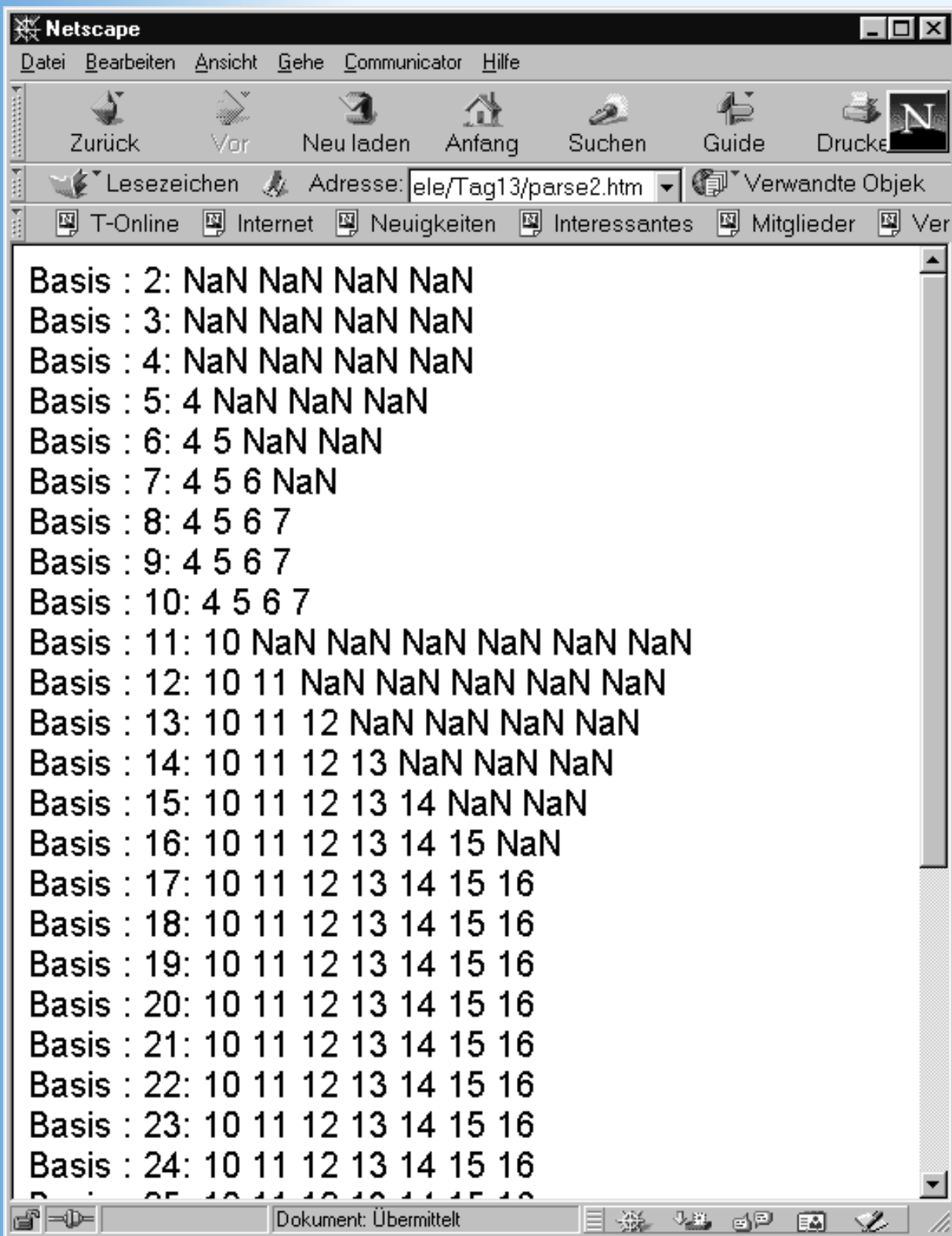



Abbildung 13.10: Die Wirkung von parseInt() mit Basis

Speichern Sie das Script unter parse2.htm und laden Sie es in einen Browser. Die Angabe der Basis bedeutet das Festlegen einer Basis für die Zahlendarstellung. Wenn die Basis kleiner als die Zahl ist, wird NaN zurückgegeben. Ist die Basis größer als 10, werden Buchstaben numerisch dargestellt. Die Basis 10 steht für dezimal, 8 für oktal oder 16 für hexadezimal. Wenn die Basis nicht angegeben wird, gelten die Regeln, dass der Anfang eines Zeichens mit 0x für hexadezimal und der Beginn mit 0 für oktal steht. Alle anderen Zahlen stehen für dezimal. Das Verhalten ist sehr leistungsfähig, jedoch recht komplex.

Textausgabe von Werten

Eine der wichtigsten Situationen im Ablauf eines Scripts ist der Fall, dass man ein Objekt vorliegen hat und Informationen benötigt, die darin enthalten sind. Um diese in eine Zeichenkette zu extrahieren, gibt es in JavaScript die Funktion `String()`¹. Diese konvertiert den Inhalt eines als Parameter übergebenen Objekts in eine Zeichenkette und gibt diese zurück.

Viele Anwendungen unter JavaScript beruhen darauf, dass Informationen in Form von Strings zur Verfügung stehen. Sei es die Ausgabe in der Webseite, sei es die Ausgabe in einem Dialogfenster. Objekte beinhalten oft Informationen nur verschlüsselt oder indirekt. Die Funktion `String()` extrahiert den Teil der Informationen, die man in einer Zeichenkette darstellen kann.

Die Bedeutung der Funktion sollte jedoch nicht überschätzt werden. Zum einen stellen einige Objekte keine sonderlich aussagekräftigen Informationen bereit, zum anderen kann man vielfach auf die Funktion verzichten und das Objekt selbst in eine andere Funktion oder Methode notieren, die dann den String automatisch extrahiert. Darüber hinaus stellen Objekte meist eine eigene Methode bereit (`toString()`), die eine analoge Aufgabe verfolgt.

Beispiel 9:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var heute = new Date();
document.write(String(heute));
document.write("<br>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```



Abbildung 13.11: String-Ausgabe des Datum-Objekts

Speichern Sie das Script unter `String.htm` und laden Sie es in einen Browser.

Kodieren und Dekodieren von Zeichenketten

Es gibt im Rahmen von verschiedenen Webvorgängen Situationen, wo die direkte Verwendung von gewissen Sonderzeichen nicht gestattet ist (dazu zählen auch das Leerzeichen und deutsche Umlaute). Etwa allgemein in URLs oder bei CGI-Prozessen. Will man sie dennoch verwenden, müssen sie maskiert, dass heißt kodiert, werden. Überall, wo ein solches kritisches Zeichen vorkommen soll, wird das entsprechende Zeichen kodiert. Diese Kodierung erfolgt über

ein vorangestelltes Prozentzeichen, gefolgt von einer hexadezimalen Angabe der Kodierung des ASCII-Zeichenwerts von dem Zeichen.

Statt dies manuell durchzuführen, gibt es die Funktion

`escape([Zeichenkette]),`

womit alle Zeichen der übergebenen Zeichenkette in ihre ASCII- Zahlenwerte umgewandelt werden (mit vorangestelltem Trennzeichen %) und die so erzeugte Zeichenkette zurückgegeben wird (als Zeichen des ISO-Latin-1-Zeichensatzes). Ausnahmen sind die Zeichen * @ - _ + . /. Implementiert ist die Funktion bereits seit JavaScript 1.0 und mit Ausnahme von Unicode-Zeichen ECMA-262-kompatibel. Unicode-Zeichen werden statt dessen entsprechend der Internet-Engineering-Task-Force- Richtlinien (IETF) für Escape-Zeichen behandelt. Spezielle Unicode- spezifizierte Escape-Sequenzen der Form %uXXXX werden nicht unterstützt.

Das Gegenstück zu `escape()` ist die Funktion

`unescape([Zeichenkette]).`

Die Funktion wandelt alle Zeichen einer übergebenen, kodierten Zeichenkette in normale ASCII-Zeichen um und gibt die lesbare Zeichenkette zurück. Die Zeichenkette muss für jedes umzuwandelnde Zeichen ein Prozentzeichen (%) und den Hexadezimalwert des Zeichens in der ASCII-Zeichentabelle in der Form %xx enthalten, wobei xx eine zweistellige Hexadezimalzahl ist.

Beispiel 10:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var a =
"Kennen Sie den kürzesten Mathematikerwitz? Epsilon < 0";
var b = escape(a);
document.write(b);
document.write("<P>");
document.write(unescape(b))
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

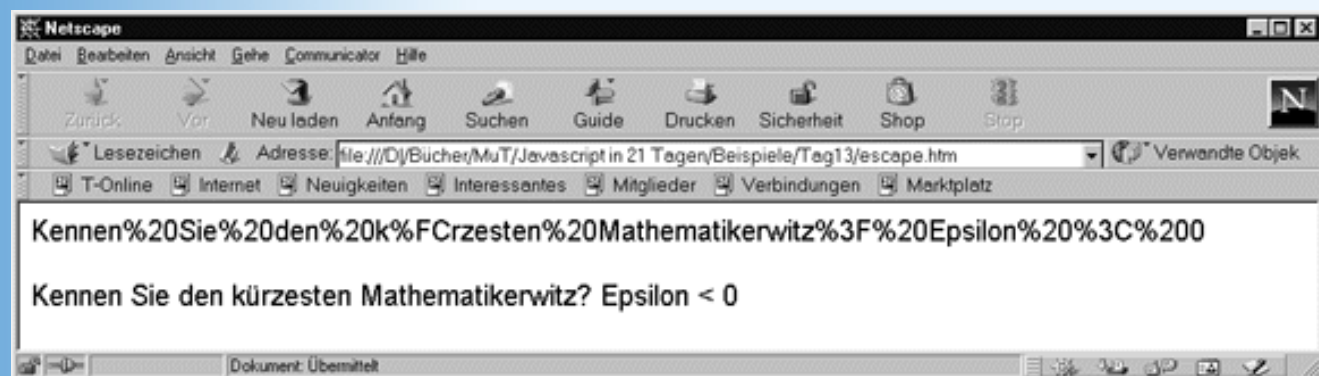


Abbildung 13.12: Ausgabe erst kodiert und dann wieder dekodiert

Speichern Sie die Datei unter `Escape.htm` und laden Sie diese in einen Browser. Das Beispiel kodiert zuerst eine Zeichenkette und dekodiert diese danach wieder. So ist das Beispiel zwar wenig sinnvoll, aber wenn der Kodierungsvorgang beispielsweise auf einem Client stattfindet, die Information dann per Internet verschickt und anschließend auf dem Server dekodiert wird, haben wir eine hochgradig sinnvolle Anwendung. (Das ist ein Teil des Prinzips, auf dem CGI basiert.)

Test auf Unendlichkeit

Eine interessante und seit JavaScript 1.3 implementierte Funktion ist `isFinite()`.

Diese evaluiert ein Argument, um zu bestimmen, ob es eine endliche Zahl ist. Syntax:

`isFinite([Zahl])`

Wenn das Argument NaN, »positiv unendlich« oder »negativ unendlich« ist, wird `false`, andernfalls `true` zurückgegeben. Diese Funktion macht insbesondere Sinn für solche Situationen, wo der definierte Wert Infinity entstehen kann. Etwa bei einer Division durch den Wert 0.

Nehmen wir unserer Beispiel 4 und machen daraus die Division von zwei Zahlen, welche der Anwender eingibt. Wenn der Divisor 0 ist, wird die Division Infinity erzeugen. Dies kann man mit der Funktion `isFinite()` abfangen und eine Meldung generieren.

Beispiel 11:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
var a, b;
function typTest(text)
{
do
{
wert = Number(prompt(
"Geben Sie die "+ text +" Zahl ein",""));
if(isNaN(wert)) alert(
"Sie dürfen hier nur numerische Werte eingeben");
}
while(isNaN(wert));
return wert;
}
a=typTest("erste");
b=typTest("zweite");
if(!isFinite(a/b))
{
alert(
"Nix gegen Unendlichkeit, aber hier wollen wir sinnvolle Werte haben.");
}
else
{
document.write(
"Das Ergebnis von ", a, " geteilt durch ", b,
" ist : ");
document.write(a / b);
document.write("<br>");
}
}
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

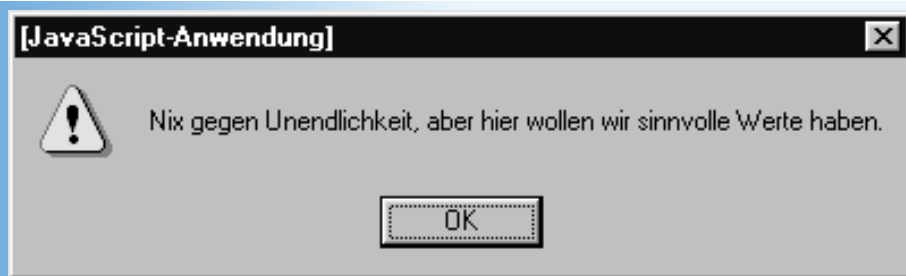


Abbildung 13.13: Fehlermeldung im Fall von Unendlich

Speichern Sie die Datei unter IsInf.htm und laden Sie diese in einen Browser. Die if-Bedingung prüft den Rückgabewert der Division mit isFinite(). Beachten Sie den Negationsoperator ! vor dem Bezeichner.

Schutzmaßnahmen

Am Rande von Interesse sind die beiden Funktionen

`taint()`

und

`untaint()`.

Sie wurden in JavaScript 1.1 eingeführt, um Datenelemente oder Scripte vor Zugriffen durch fremde Scripte verstecken zu können. `taint([dataElementName])` versteckt das Datenelement, `untaint()` entfernt den Zugriffsschutz, der mit `taint()` gesetzt wurde. Hört sich ganz nützlich an, hat aber nicht gehalten, was man sich davon versprach. Beide Funktion wurden in JavaScript 1.2 wieder entfernt.

Zusammenfassung

Wir haben heute in JavaScript integrierte Top-Level-Eigenschaften und Funktionen kennengelernt. Obwohl es nicht allzu viele sind, ist deren Nutzen enorm. Viele Situationen, in denen Sie sich sonst mühsam eine eigene Lösung zusammenstricken müssten, werden von diesen Funktionen und Eigenschaften bereits abgedeckt. Insbesondere Fälle, in denen kritische Situationen auftreten können (Datentyp-Probleme, Endlichkeitsprobleme oder Kodierungsangelegenheiten), werden damit ziemlich vollständig zu behandeln sein.

Fragen und Antworten

Frage:

Gibt es noch weitere JavaScript-Funktionen?

Antwort:

Ja. In JavaScript 1.3 gibt es noch die speziell für die Zusammenarbeit von JavaScript mit Java eingeführte Funktion `getClass()`. Diese ist aber in einem etwas anderen Zusammenhang wie die anderen Funktionen zu sehen. Ansonsten ist die Auflistung der Funktionen aber vollständig. Die geringe Anzahl von vorgefertigten Funktionen in JavaScript liegt daran, dass die meiste Funktionalität in Form von Objekten und ihren zugehörigen Methoden bereitgestellt wird. Diese bekommen wir in der dritten Woche.

Frage:

Gibt es noch weitere Top-Level-Eigenschaften in JavaScript 1.3?

Antwort:

Nein, nur die drei hier besprochenen.

Workshop

Schauen Sie sich noch einmal das Beispiel 1 an und erstellen Sie eine im Netscape Navigator lauffähige Version.

Nehmen Sie sich das Beispiel 3 vor und erstellen Sie unter Ausnutzung von `isNaN()` eine qualifizierte Fehlermeldung in

der Webseite für den Fall, dass die Multiplikation scheitert. Ansonsten soll die Ausgabe wie bisher funktionieren. Realisieren Sie auch den zweiten Vorschlag - je nach Typ des eingegebenen Werts wird eine numerische Addition oder eine String- Addition ausgeführt.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was macht `isInfinite()`?
2. Funktioniert das?
`if(a==NaN)`
3. Funktioniert das?
`if(a=="undefined")`
4. Was liefert `parseInt(45.678)`?
5. Was liefert `parseInt(DM 45.67)`?
6. Was liefert `parseFloat(45.67 DM)`?
7. Funktioniert das?
`if(a==undefined)`
8. Funktioniert das?
`if((typeof a) == undefined)`

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

1

Mit großem S. Die Funktion darf nicht mit ähnlichen Bezeichnern für Objekte, Methoden oder Datentypen verwechselt werden.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 2

Tag 14

JavaScript und Java, allgemeine Multimedia-Techniken

Am abschließenden Tag der zweiten Woche wollen wir uns in der Hauptsache mit dem Zusammenspiel von JavaScript und Java beschäftigen. Beide Techniken ergänzen sich im Rahmen des WWW hervorragend. Jede hat ihre spezifischen Aufgaben und teilweise kann man mit beiden Techniken ein Problem angehen. Es gibt überdies zahlreiche Schnittstellen, wo diese beiden zwar vom Namen eng, von der Konzeption und Technik jedoch nur sehr weitläufig verwandten Sprachen zusammenspielen können.

Heute stehen deshalb einmal die so genannten Java-Applets und ihre Referenz in einer Webseite mit allen notwendigen Fassetten auf dem Programm, aber auch, wie man zwischen einem JavaScript bzw. einer HTML-Seite und einem Java-Applet Daten austauschen kann. Dazu müssen wir zuerst einmal wieder auf HTML zurückgreifen und den Weg besprechen, wie Applets überhaupt in eine Webseite eingebunden werden können. Des Weiteren werden auch einige Aspekte von Java selbst näher zu beleuchten sein.

In einem logischen Zusammenhang mit der Einbindung von Java-Applets in Webseiten sind die Verbindungen von HTML-Gerüsten mit beliebigen Multimedia-Erweiterungen (dazu zählen wir auch ActiveX-Controls) zu sehen. Auch diese sollen in diesem Kapitel besprochen werden.

Allgemeines zur Einbindung von Java-Applets und multimedialen Objekten

Über HTML-Referenzen kann in Webseiten eine Vielzahl von Erweiterungselementen eingebunden werden. Das fängt am unteren Ende mit Grafiken an und geht bis hin zu vollständigen Programmen, die in einer Webseite referenziert werden können. In einem logischen Zusammenhang wollen wir dabei die hier folgenden Objekte sehen:

- PlugIns
- ActiveX-Controls
- Java-Applets

Allgemeines zur Einbindung von Java-Applets

Java-Applets lassen sich auf verschiedene Weise in HTML-Seiten referenzieren. Ganz am Anfang gab es den <APP>-Tag, aber der wird nicht mehr verwendet. Derzeit sind drei Tags üblich:

- der <EMBED>-Tag
- der <OBJECT>-Tag
- der <APPLET>-Tag

Wir wollen hier alle drei Varianten behandeln, uns aber im Wesentlichen auf die dritte Version konzentrieren.

PlugIns

In Webseiten lassen sich diverse Techniken einbinden, welche erst einmal durch die Standardaustattung des Browsers nicht unbedingt unterstützt werden. Das bedeutet, wenn eine Datei eines bestimmten Dateiformats im Rahmen einer Webseite geladen und dargestellt oder ausgeführt werden soll, kann ein Browser unter Umständen nicht direkt damit umgehen. Eine Erweiterung des Browsers um einen Interpreter für diese Formate löst das Problem. Netscape entwickelte schon vor geraumer Zeit zum Hinzufügen neuer Funktionen zum Browser die Technik des Installierens von so genannten PlugIns (Erweiterungsmodule für den Navigator). PlugIns werden mittlerweile auch von anderen Browsern unterstützt und lassen sich von zahlreichen Seiten aus dem Internet laden. Etwa von der Netscape-Seite <http://home.netscape.com/plugins/>, von wo zahlreiche PlugIns für die unterschiedlichsten Dateiformate (nicht nur Netscape-spezifische) geladen und dann im Browser installiert werden können.



Abbildung 14.1: Diverse PlugIns gibt es bei Netscape



Die PlugIn-Idee kann trotz ihrer unbestrittenen Pluspunkte auch kritisch gesehen werden. Das Konzept beinhaltet auch einige Fassetten, die nicht unbedingt vorteilhaft sind:

1. Die Standardisierung von Vorgängen und Formaten im Internet wird behindert (nicht gefördert, wie man vielleicht zuerst vermuten mag)! Immer mehr Anbieter von (vor allem multimedialen) Informationen stellen sich auf den Standpunkt, dass sie sich nicht an irgendwelche vorhandenen Dateistandards halten brauchen. Es gibt ja PlugIns, und der Anwender wird sich schon so ein neues PlugIn installieren, wenn er die Information nutzen möchte. Also wird flugs ein neues Dateiformat samt PlugIn entwickelt und auf die Internet-Gemeinde losgelassen.
2. Immer mehr PlugIns machen sich über die Zeit auf dem Rechner eines Anwenders breit. Der Rechner wird »zugemüllt«.
3. Der Download eines noch nicht vorhandenen PlugIns kostet zusätzliche Online-Zeit und damit das Geld des Anwenders. Wenn sich Informationsanbieter an Standardformate halten würden, müssten sich Besucher nicht immer neue PlugIns auf ihre Kosten auf den Rechner laden.

Die Installation von PlugIns kann im Übrigen über spezielle Befehle in einer HTML-Seite angestoßen werden.

Der <EMBED>-Tag ist eine Netscape-Erweiterung von HTML 3.2 zur allgemeinen Einbindung von Multimedia-Elementen in eine Webseite. Damit können neben dem Aufruf von PlugIns auch lokale Programme zur Darstellung von Nichtwebformaten (etwa Bilder mit der Erweiterung .bmp) aufgerufen werden.

Die allgemeine Syntax des HTML-Containers für PlugIns sieht so aus:

```
<EMBED
  type="[PlugIn]"
  pluginspace="[URL]"
  code="Datei"
  object="[Zusatzangaben]"
  codebase="[Verzeichnis der Datei]"
  width="[Breite]"
  height="[Höhe]">
```

- Über type wird das verwendete PlugIn angegeben. Diese Angabe kann das PlugIn über eine Version noch genauer spezifizieren (etwa "application/x-java-applet; version=1.2" für den Fall, dass ein Java-PlugIn angegeben werden soll).
- Über die optionale Angabe pluginspace kann man einen Download- Ort in Form eines URL angeben, wovon ein lokal bei einem Anwender nicht vorhandenes PlugIn bei Bedarf automatisch geladen werden kann (etwa "http://java.sun.com/products/plugin/1.1/plugin-install.html").
- Die Angabe code spezifiziert die Datei und die beiden restlichen Attribute wie üblich Breite und Höhe eines Anzeigebereichs.

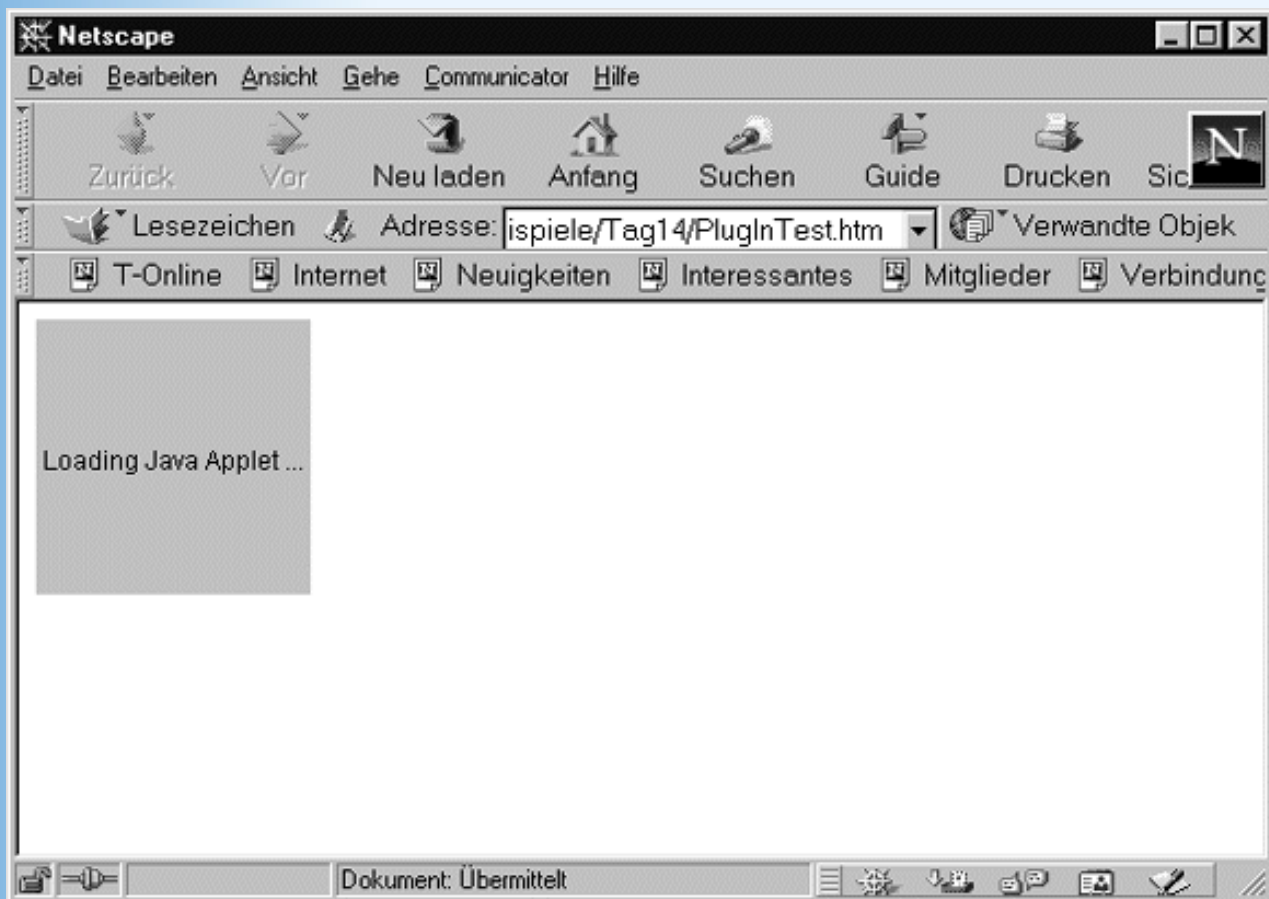


Abbildung 14.2: Das PlugIn wird geladen

Den <EMBED>-Tag gibt es auch in einer einfacheren Variante. Damit kann er auch zur Einbindung von Videos und Musik in den üblichen Standardformaten verwendet werden. Syntax:

```
<EMBED
  src="[URL]"
  width=[Breite in Pixel]
  height=[Höhe in Pixel]
  controls=console>
```

In dem optionalen <NOEMBED>-Container kann ein Alternativtext für Browser angegeben werden, die PlugIns nicht unterstützen.

Beispiel 1:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<BODY>
<EMBED src="video001.avi" width=300 height=300>
<NOEMBED>Hier läuft eigentlich ein Video.</NOEMBED>
</BODY>
</HTML>
```

Speichern Sie das Script unter Embed.htm und laden Sie es in einen Browser. Es wird beim Laden der Datei der mit dem Browser verbundene Mediaplayer gestartet (wenn so ein Player nicht vorhanden ist, kann es Probleme geben) und das angegebene Video abgespielt. Sie benötigen natürlich ein Video mit dem angegebenen Namen (oder müssen das Beispiel entsprechend anpassen) - auf der Buch-CD finden Sie ein solches. Sie können das Beispiel auch so umbauen, dass ein beliebiges anderes Video abgespielt wird.

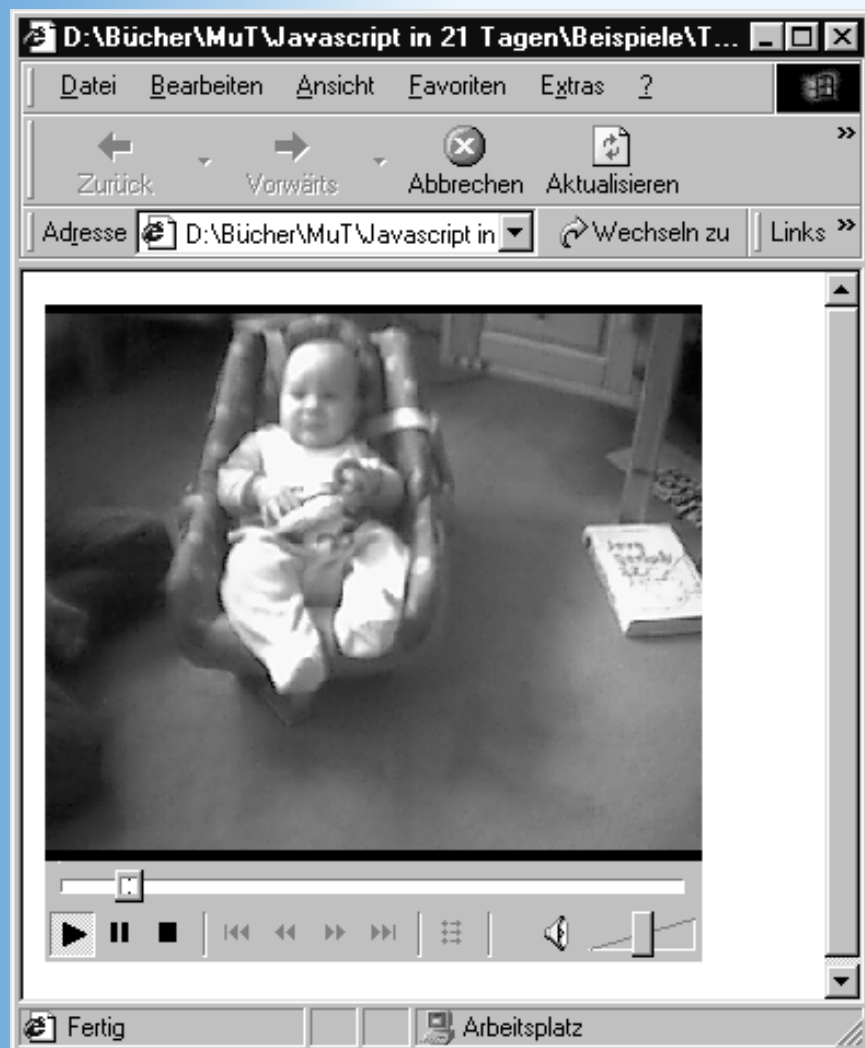


Abbildung 14.3: Abspielen eines Videos

Beispiel 2:

Das zweite Beispiel Embed2.htm spielt ein MP3- bzw. MPG-Musikstück ab (auch dieses finden Sie auf der Buch-CD). Wir nutzen das Beispiel dazu, noch mal die Verwendung von Metainformationen in einem praktischen Beispiel zu zeigen:

```
<HTML>
<HEAD>
<TITLE>Safety First - Demo<TITLE>
<META name="Author" content="Ralph Steyer">
<META name="description" content="Demo von Safety First">
```



```
<META name="keywords" content="History, MP3, Rock, Blues, Boogie">
<META name="copyright" content="2000 Safety First">
<META name="organization" content="Safety First - RJST@gmx.de">
</HEAD>
<BODY>
<EMBED src="history.mpg" width=300 height=300>
<NOEMBED>Hier sollte ein Mediaplayer das MP3-Musikstück History abspielen.
</NOEMBED>
</BODY>
</HTML>
```



Die Verwendung des <EMBED>-Tags sollte mit Vorsicht geschehen, da die Technik im Wesentlichen auf Netscape Browser ab der Version 4.x und neuere Varianten des Explorer beschränkt ist. Aber auch sonst gehört der Tag zu den Elementen, die als deprecated gelten.

Einbinden eines Java-Applets mit dem <EMBED>-Tag

Der allgemeine <EMBED>-Tag zur Einbindung von beliebigen Multimedia- Elementen lässt sich auch bei Applets anwenden. Er wird in diesem Zusammenhang gelegentlich Java PlugIn-Tag genannt, weil er genau genommen ein PlugIn zur Ausführung eines Applets angibt. Die Verwendung sollte mit Vorsicht geschehen, da die Technik wie gerade oben schon erwähnt nur eingeschränkt unterstützt wird und außerdem als deprecated erklärt ist. Grundsätzlich erfolgt die Einbindung von einem Java- Applet mit dem <EMBED>-Tag wie in dem folgenden Beispiel:

```
<HTML>
<BODY>
<EMBED type="application/x-java-applet"
  code="DasApplet.class"
  width="200" height="300">
</EMBED>
</BODY>
</HTML>
```

Java-Applets werden mit dieser Technik mittels des angegebenen Java- PlugIns gestartet. Das Problem dieser Form der Einbindung ist, dass das Applet nicht dargestellt werden kann, wenn das angegebene PlugIn nicht zur Verfügung steht und gerade keine Online-Verbindung besteht oder der Browser mit PlugIns dieser Form nicht zurechtkommt.

Der <OBJECT>-Tag

Der <OBJECT>-Tag ist der offizielle Standard-Tag unter HTML 4.0 zum Einbinden von beliebigen Multimedia-Objekten in eine Webseite. Damit soll er nach dem Willen des W3C auch zur Referenzierung von Applets verwendet werden. Seit der Einführung von HTML 4.0 soll er der Tag sein, der sämtliche bisher vorkommenden Tags zur Einbindung von multimedialen Objekten ablöst. Damit würden von der Theorie her sowohl der nachfolgend noch besprochene <APPLET>-Tag als auch der sowieso Browser-spezifische <EMBED>-Tag überflüssig.

Besondere Zielrichtung dieser verallgemeinerten Referenzierung sind die vielen Browser-spezifischen propertyierenden Tags, wie der nur vom Internet Explorer verstandene <BGSOUND>-Tag zur Einbindung von Hintergrundmusik. Syntax:

```
<BGSOUND
  src="[URL]"
  loop=[Anzahl der Wiederholungen]>
```

Die neue Philosophie möchte sämtliche Multimedia-Referenzen bündeln und alle auf nur einen Mediatyp beschränkten Tags eliminieren (darunter fällt sogar der -Tag zur Anzeige von Bildern). Positiver Effekt - die Einführung eines neuen Mediaformats erzwingt keinen neuen Tag, sondern der <OBJECT>-Tag wird nur erweitert.



Den <OBJECT>-Tag gab es schon vor HTML 4.0. Er hat aber in früheren HTML-Versionen ausschließlich zur

Einbindung von ActiveX-Controls gedient.

Die Syntax zum Referenzieren eines beliebigen Multimedia-Objekts sieht schematisch so aus:

```
<OBJECT
  classid="[ActiveX-ControllId des Plug-In]"
  codebase="[URL PlugIn]"
  width="[Breite]" height="[Höhe]">
<PARAM name="code" value="[Datei]">
<PARAM name="type" value="[PlugIn]">
<PARAM name="codebase" value="[Verzeichnis Datei]">
... alternativer Text...
</OBJECT>
```

- Der Wert des classid-Attributes ist die weltweit eindeutige ActiveX-ID für das PlugIn (ein eindeutiger 128-Bit-Wert, um den sich der Ersteller eines Multimedia-Objektes kümmern muss und der für die konkrete Objektimplementierung zuständig ist), welche zum Laden des PlugIns in den Browser über den <OBJECT>-Tag verwendet werden muss. Wenn Sie ein Multimedia-Objekt in eine Webseite einbinden wollen, müssen Sie diese Klassen-ID aus der Dokumentation des Multimedia-Objektes entnehmen oder sonst irgendwie die ID erfahren.
- Das codebase-Attribut wird dafür verwendet, das Plugin aus dem Netz zu laden, wenn es fehlt. Auch dieser URL muss aus der Dokumentation des Multimedia-Objektes entnommen werden.
- Die jeweiligen <PARAM>-Werte spezifizieren den genauen Typ des Multimedia-Objektes, eventuell die Version und möglicherweise eine Pfadangabe dorthin. Außerdem sind irgendwelche optionalen Übergabeparameter an das Multimedia-Objekt möglich.



Bei der recht komplexen und umfangreichen Angabe der ClassId und der weiteren Attribute und Parameter schlägt die Stunde von einigen HTML-Tools, die für die Einbindung von vielen Objekten eine automatische Notation von solchen Containern bereits integriert haben. Mit Menüs und Dialogmasken wird die Referenz meist recht komfortabel generiert.

Beispiel 3:

Das nachfolgende Listing (WinMediaPlayerObject2.htm) ist die Einbindung des Windows-Mediaplayers als ActiveX-Control (und damit nur Internet- Explorer-fähig):

```
<HTML>
<BODY>
<OBJECT
  classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
  id="MediaPlayer1"
  width="286" height="225">
  <PARAM name="AudioStream" value="-1">
  <PARAM name="AutoSize" value="-1">
  <PARAM name="AutoStart" value="-1">
  <PARAM name="AnimationAtStart" value="-1">
  <PARAM name="AllowScan" value="-1">
  <PARAM name="AllowChangeDisplaySize" value="-1">
  <PARAM name="AutoRewind" value="0">
  <PARAM name="Balance" value="0">
  <PARAM name="BaseURL" value>
  <PARAM name="BufferingTime" value="5">
  <PARAM name="CaptioningID" value>
  <PARAM name="ClickToPlay" value="-1">
  <PARAM name="CursorType" value="0">
  <PARAM name="CurrentPosition" value="-1">
  <PARAM name="CurrentMarker" value="0">
  <PARAM name="DefaultFrame" value>
  <PARAM name="DisplayBackColor" value="0">
  <PARAM name="DisplayForeColor" value="16777215">
```

```
<PARAM name="DisplayMode" value="0">
<PARAM name="DisplaySize" value="0">
<PARAM name="Enabled" value="-1">
<PARAM name="EnableContextMenu" value="-1">
<PARAM name="EnablePositionControls" value="-1">
<PARAM name="EnableFullScreenControls" value="0">
<PARAM name="EnableTracker" value="-1">
<PARAM name="Filename" value="">
<PARAM name="InvokeURLs" value="-1">
<PARAM name="Language" value="-1">
<PARAM name="Mute" value="0">
<PARAM name="PlayCount" value="1">
<PARAM name="PreviewMode" value="0">
<PARAM name="Rate" value="1">
<PARAM name="SAMILang" value>
<PARAM name="SAMISStyle" value>
<PARAM name="SAMIFilename" value>
<PARAM name="SelectionStart" value="-1">
<PARAM name="SelectionEnd" value="-1">
<PARAM name="SendOpenStateChangeEvents" value="-1">
<PARAM name="SendWarningEvents" value="-1">
<PARAM name="SendErrorEvents" value="-1">
<PARAM name="SendKeyboardEvents" value="0">
<PARAM name="SendMouseClickEvents" value="0">
<PARAM name="SendMouseMoveEvents" value="0">
<PARAM name="SendPlayStateChangeEvents" value="-1">
<PARAM name="ShowCaptioning" value="0">
<PARAM name="ShowControls" value="-1">
<PARAM name="ShowAudioControls" value="-1">
<PARAM name="ShowDisplay" value="0">
<PARAM name="ShowGotoBar" value="0">
<PARAM name="ShowPositionControls" value="-1">
<PARAM name="ShowStatusBar" value="0">
<PARAM name="ShowTracker" value="-1">
<PARAM name="TransparentAtStart" value="0">
<PARAM name="VideoBorderWidth" value="0">
<PARAM name="VideoBorderColor" value="0">
<PARAM name="VideoBorder3D" value="0">
<PARAM name="Volume" value="-600">
<PARAM name="WindowlessVideo" value="0">
</OBJECT>
</BODY>
</HTML>
```

Wenn Sie ein Video bereits als Parameter angeben wollen, müssen Sie die folgende Zeile ändern:

```
<PARAM name="Filename" value="">
```

Dort könnte etwa

```
<PARAM name="Filename" value="file:///C:/TEMP/VIDE0001.avi" valuetype="ref">
```

stehen, wenn Sie ein lokales Video aus dem Verzeichnis Temp auf dem Laufwerk C: referenzieren.

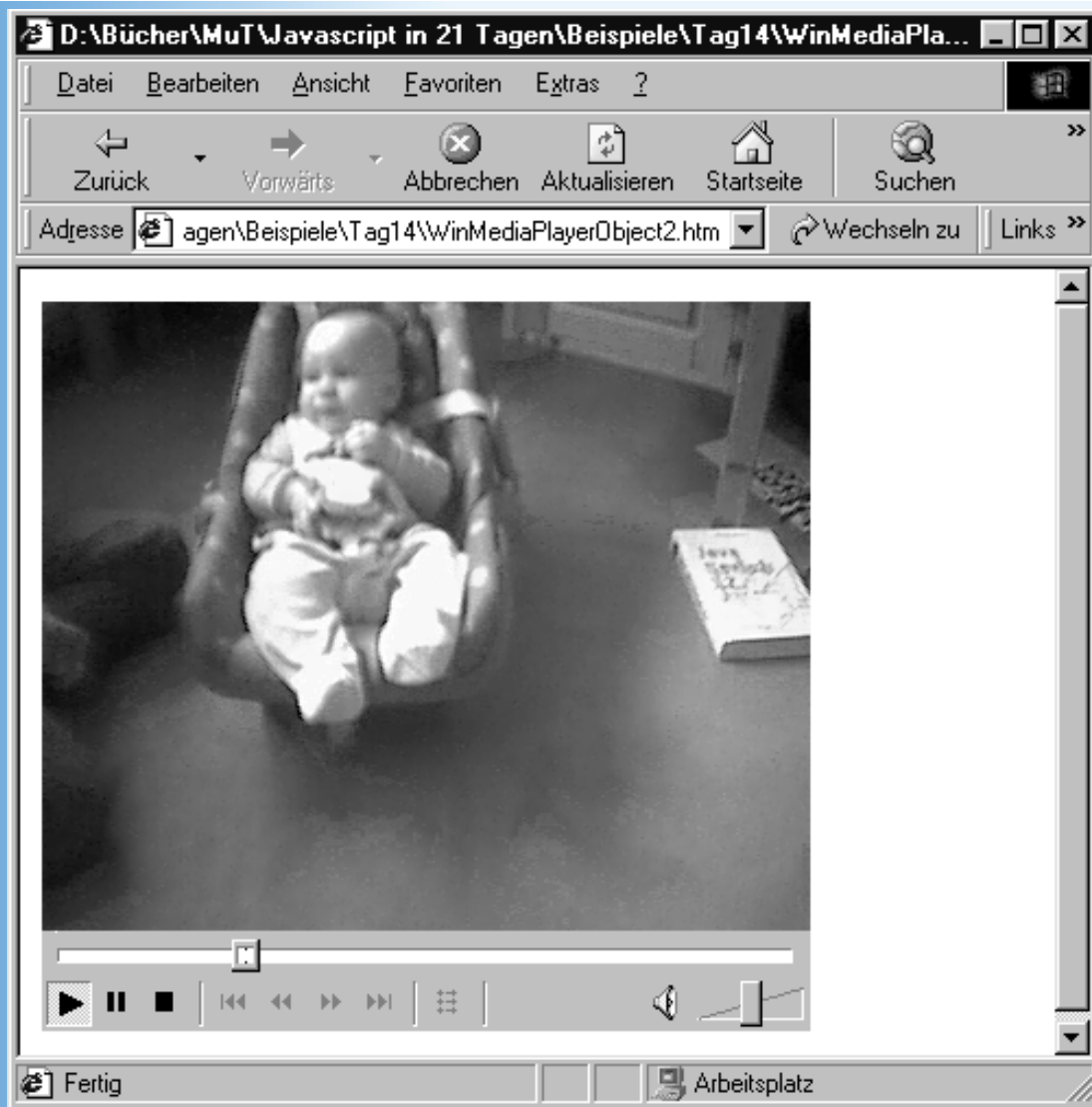


Abbildung 14.4: Der Windows Mediaplayer als ActiveX-Control

Einbinden eines Java-Applets mit dem <OBJECT>-Tag

Die Referenz eines Java-Applets mit dem <OBJECT>-Tag sieht von der Syntax her schematisch so aus:

```
<OBJECT
  classid="[ActiveX-ControllId des Java-PlugIn]"
  codebase="[URL PlugIn]"
  width="[Breite]" height="[Höhe]">
<PARAM name="code" value="[Applet].class">
<PARAM name="type" value="[PlugIn]">
<PARAM name="codebase" value="[VerzeichnissclassFile]">
... alternativer Text...
</OBJECT>
```

- Der Wert des classid-Attributes ist die weltweit eindeutige ActiveX-ID für das Java-PlugIn (dies ist immer classid="clsid:8AD9C840-044E- 11D1-B3E9-00805F499D93"), welche zum Laden des Java-PlugIn in den Browser über den <OBJECT>-Tag verwendet werden muss.
- Das codebase-Attribut wird dafür verwendet, das Java-Plugin aus dem Netz zu laden, wenn es fehlt (etwa codebase = "http:// java.sun.com/products/plugin/1.1/jinstall-11- win32.cab#Version=1,1,0,0").
- Der gleich benannte Parameter codebase (<PARAM name = "codebase" value="[VerzeichnissclassFile]">) gibt optional einen relativen URL der Applet-Klasse an.
- Die Angabe <PARAM name="JAVA_CODE" ...> ist eine Spezifizierung für den Fall, dass es Konflikte zwischen anderen Parametern der Form <PARAM name="CODE" ...> gibt.
- Der type-Parameter-Tag kann von Browsern zum Laden des PlugIn verwendet werden. In der Regel ist das wieder eine Angabe wie

- `<PARAM name="type" value="application/x-java-applet;version=1.1">`
- oder
`<PARAM name="type" value="application/x-java-applet">`

Beispiel 4:

Das nachfolgende Listing ist die Einbindung eines Java-Applets mit dem `<OBJECT>`-Tag (identisch zu dem `<EMBED>`-Beispiel):

```
<HTML>
<BODY>
<OBJECT
  classid=
    "clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
  codebase=
    "http://java.sun.com/products/plugin/1.1/
    jinstall-11-win32.cab#Version=1,1,0,0"
  width="200" height="300">
<PARAM name="type" value="application/x-java-applet">
<PARAM name="JAVA_CODE" value="DasApplet.class">
</OBJECT>
</BODY>
</HTML>
```



Der `<OBJECT>`-Tag ist wie gesagt der offiziell vom W3C angegebene Tag für die Referenz auf Java-Applets. Sie werden ihn dennoch in kaum einer Webseite mit Java-Applets finden. Gegen die Verwendung des `<OBJECT>`-Tags spricht neben der recht aufwendigen Syntax vor allem, dass viele Anwender ActiveX-bezogene Technik aus Sicherheitsgründen nicht akzeptieren. Der speziell für Applets vorhandene Tag `<APPLET>` ist einfacher zu handeln und unproblematischer.

Die Applet-Referenz mit `<APPLET>`

Die konkrete Referenzierung eines Java-Applets mittels des `<APPLET>`-Containers ist die mit Abstand populärste Variante (deprecated hin oder her). Der `<APPLET>`-Container beinhaltet in seinem Inneren gegebenenfalls benötigte Parameter für das Applet, er kann aber auch leer sein, wenn das Applet keine Parameter benötigt. Die Syntax sieht so aus:

```
<APPLET
  code="[Klasse]"
  src=[URL]
  codebase=[URL]
  width=[Wert]
  height=[Wert]>
</APPLET>
```

- Dabei bezeichnet [Klasse] die Applet-Klasse, also das Applet selbst. Dies ist immer eine Datei mit der Kennung .class, wobei Groß- und Kleinschreibung unbedingt zu beachten ist. Wenn die Dateierweiterung angegeben wird, ist sie immer (!) klein zu schreiben (sie kann aber auch unterbleiben).
- Die optionale Angabe src gibt zusätzlich die Quelltext-Datei (eine Datei mit der Kennung .java) eines Java-Applets an (auch hier muss die Dateierweiterung unbedingt klein geschrieben werden). Die Angabe wird in der Regel nicht benötigt. Die Angabe hat keinen Effekt bezüglich der Referenz in einer Webseite, sondern ist nur als Dokumentation zu sehen. Gegebenenfalls kann man mit dem Quelltext das Applet modifizieren und neu übersetzen.
- Das optionale Attribut codebase bestimmt eine Suchpfadangabe für das Applet (relativ oder absolut), wenn das Applet sich nicht im gleichen Verzeichnis wie die Webseite befindet.
- Die beiden anderen Attribute width und height sind wie üblich die Breite und die Höhe des Applets in Pixeln.

In einer Applet-Referenz lassen sich noch Angaben über den Textfluss um das Applet herum machen. Der `<APPLET>`-Tag

kann entsprechend um die folgenden optionalen Angaben erweitert werden:

- align gibt die Ausrichtung von einem folgenden Text an und kann u.a. die Werte `left`, `right` und `middle` annehmen.
- vspace definiert den vertikalen Abstand zwischen Applet und Text oberhalb und unterhalb des Applets.
- hspace definiert den horizontalen Abstand links und rechts vom Applet.

Innerhalb des `<APPLET>`-Containers selbst (nicht als Attribute des Tags selbst) können zwei Angaben stehen:

- freier Text
- Parameter

Der freie Text im `<APPLET>`-Container wird dazu verwendet, einem Anwender mit einem nicht Java-fähigen Browser (oder mit deaktiviertem Java) deutlich zu machen, dass an dieser Stelle ein Java-Applet steht. Java-unfähige Browser werden nach dem Prinzip der Fehlertoleranz den `<APPLET>`-Tag (ebenso natürlich den Abschluss-Tag) ignorieren und den im Container notierten Text als reinen ASCII-Text auf dem Bildschirm anzeigen. Java-fähige Browser (mit aktivierter Java-Option) sind so konzipiert, dass sie den Text innerhalb des Containers nicht anzeigen, sondern nur das Applet selbst.

Die zweite optionale Angabe im `<APPLET>`-Container sind Parameter, die über HTML-Tags an das Applet übergeben werden. Dabei hängt es natürlich von der Programmierung des Applets ab, ob solche Parameter benötigt werden oder nicht.



Ein oft gemachter Denkfehler bei der Referenzierung von Applets ist es, zu vermuten, dass es doch irgendeine Logik aus Sichtweise der Webseite geben muss, welche Applets Parameter benötigen und welche nicht. Als Webseiten-Ersteller oder JavaScript-Programmierer haben Sie darauf keinerlei Einfluss. Es gilt das Prinzip »Vogel friss oder stirb«. Wenn ein Applet Parameter benötigt, müssen Sie diese korrekt angeben. Kommt es ohne aus, müssen halt keine Parameter im `<APPLET>`-Container notiert werden (falls doch welche da stehen, werden sie wie üblich ignoriert). So wenig, wie ein Museum, das ein Bild ausstellt, dem Maler sagen kann, wie er das Bild vor Jahren malen muss¹ (die Sache ist gelaufen - das Bild ist fertig), so wenig kann derjenige, der ein Applet verwendet, festlegen, ob Parameter anzugeben sind oder nicht.

Die Angabe von Parametern erfolgt innerhalb des `<APPLET>`-Containers über `<PARAM>`-Tags - gegebenenfalls mehrere hintereinander (soviel ein Applet halt benötigt - das muss der Beschreibung des Applets entnommen werden). Im Fall von mehreren Parametern spezifiziert jeder einzelne Parameter-Tag einen anderen Applet-Parameter. Die Syntax lautet wie folgt:

```
<PARAM name=[name] value=[wert]>
```

Dabei bezeichnet name den Namen des Parameters, wie er auch intern im Java-Quellcode verwendet wird, und value den Wert des angegebenen Parameters.



Wichtig ist zu beachten, dass Java zwischen Groß- und Kleinschreibung unterscheidet. Im Gegensatz zu den meisten anderen Situationen unter HTML muss deshalb der Wert von name unbedingt so geschrieben werden, wie die Dokumentation des Applets es angibt. Das Java-Applet nimmt die Parameternamen und deren Werte in speziellen Methoden entgegen und die unterscheiden Groß- und Kleinschreibung (unter Umständen spielt das ebenfalls beim Wert selbst eine Rolle).

Erstellen wir nun verschiedene Webseiten, die Java-Applets referenzieren, als Beispiele.

Beispiel 5:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<BODY>
<APPLET
  code=ArcTest.class
  width=400
  height=400>
Hier sollte ein Applet stehen
```

```
</APPLET>
</BODY>
</HTML>
```

Speichern Sie die Datei unter `applet1.htm` und laden Sie es in einen Browser. Dabei müssen Sie natürlich sicherstellen, dass Ihnen ein Java-Applet dieses angegebenen Namens zur Verfügung steht. Auf der Buch-CD finden Sie das hier verwendete Applet.



Für die Beispiele verwenden wir teilweise freie Applets, das als Demos mit der Java-Entwicklungsumgebung von Sun, dem JDK 1.2, ausgeliefert werden.

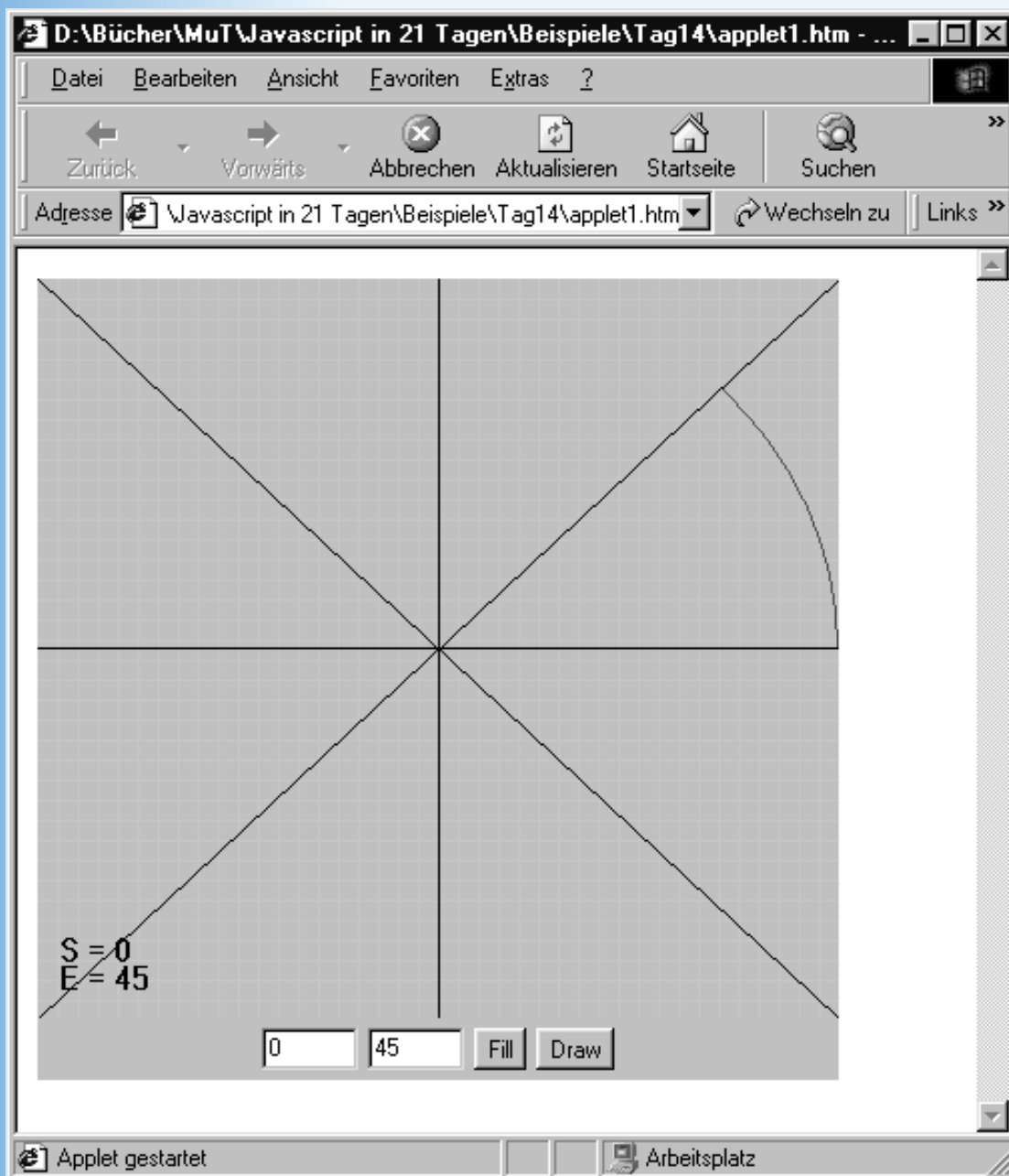


Abbildung 14.5: Das Applet nach dem Laden



Versuchen Sie nicht, ein Applet umzubenennen. Der Name ist nicht wie bei Dateien anderen Typs willkürlich, sondern in vielen Fällen integraler Bestandteil des Applets.

Das nächste Beispiel arbeitet nun mit Parametern, die an das Applet übergeben werden

Beispiel 6:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<BODY>
<APPLET
  codebase="."
  code="Chart.class"
  width=251
  height=125>
<PARAM name=c2_color value="green">
<PARAM name=c2_label value="Q2">
<PARAM name=c1_style value="striped">
<PARAM name=c4 value="30">
<PARAM name=c3 value="5">
<PARAM name=c2 value="20">
<PARAM name=c4_color value="yellow">
<PARAM name=c1 value="10">
<PARAM name=c4_label value="Q4">
<PARAM name=title value="Performance">
<PARAM name=c3_style value="striped">
<PARAM name=columns value="4">
<PARAM name=c1_color value="blue">
<PARAM name=c1_label value="Q1">
<PARAM name=c3_color value="magenta">
<PARAM name=c3_label value="Q3">
<PARAM name=c2_style value="solid">
<PARAM name=orientation value="horizontal">
<PARAM name=c4_style value="solid">
  <PARAM name=scale value="5">
  Hier sollte ein Applet stehen
</APPLET>
</BODY>
</HTML>
```

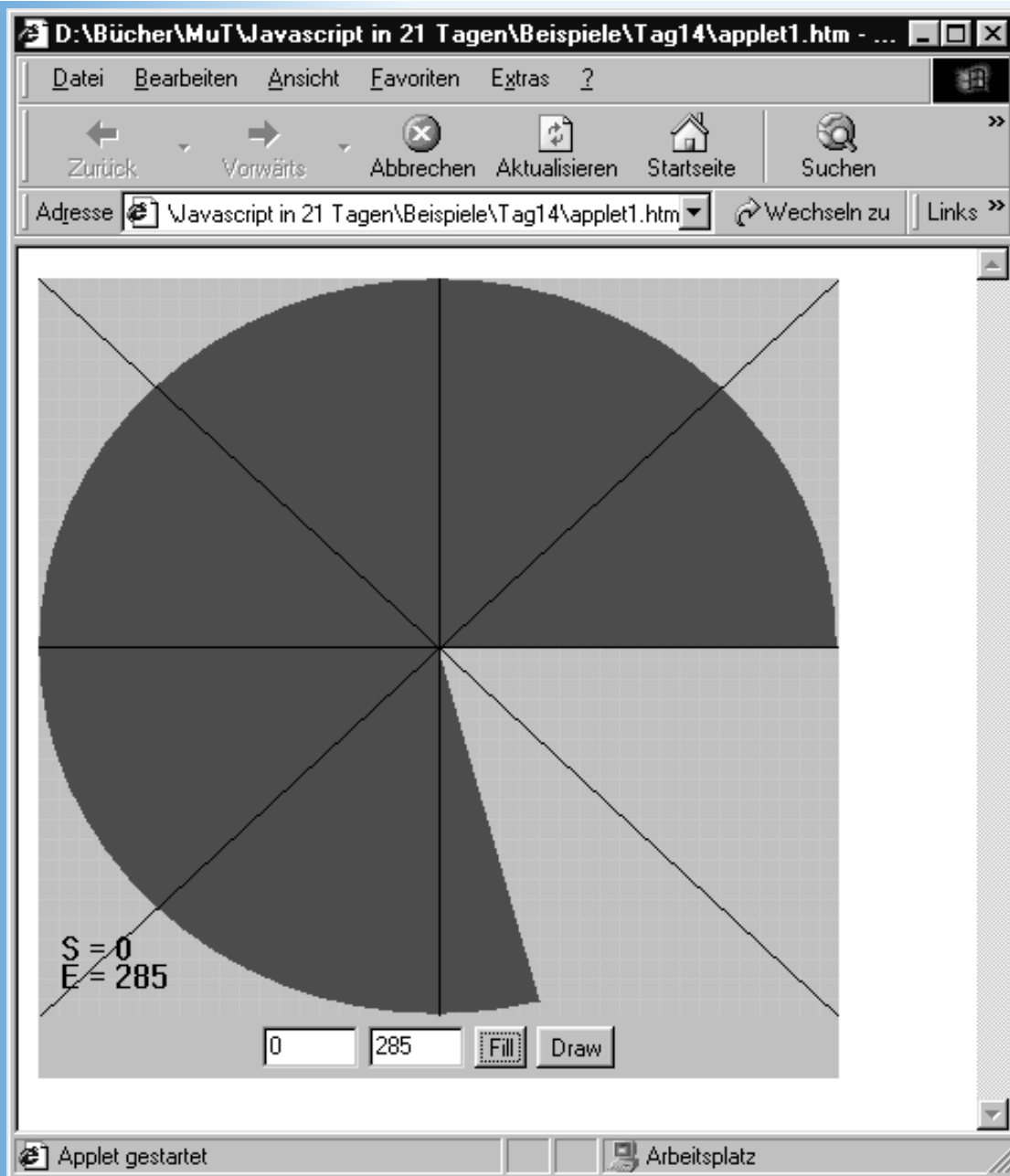


Abbildung 14.6: Das Applet, nachdem eine Zeichenaktion ausgelöst wurde

Speichern Sie die Datei unter applet2.htm und laden Sie es in einen Browser. Auch hier müssen Sie wieder sicherstellen, dass Ihnen ein Java-Applet dieses angegebenen Namens zur Verfügung steht. Auf der Buch-CD finden Sie das hier verwendete Applet (siehe Abbildung 14.7).

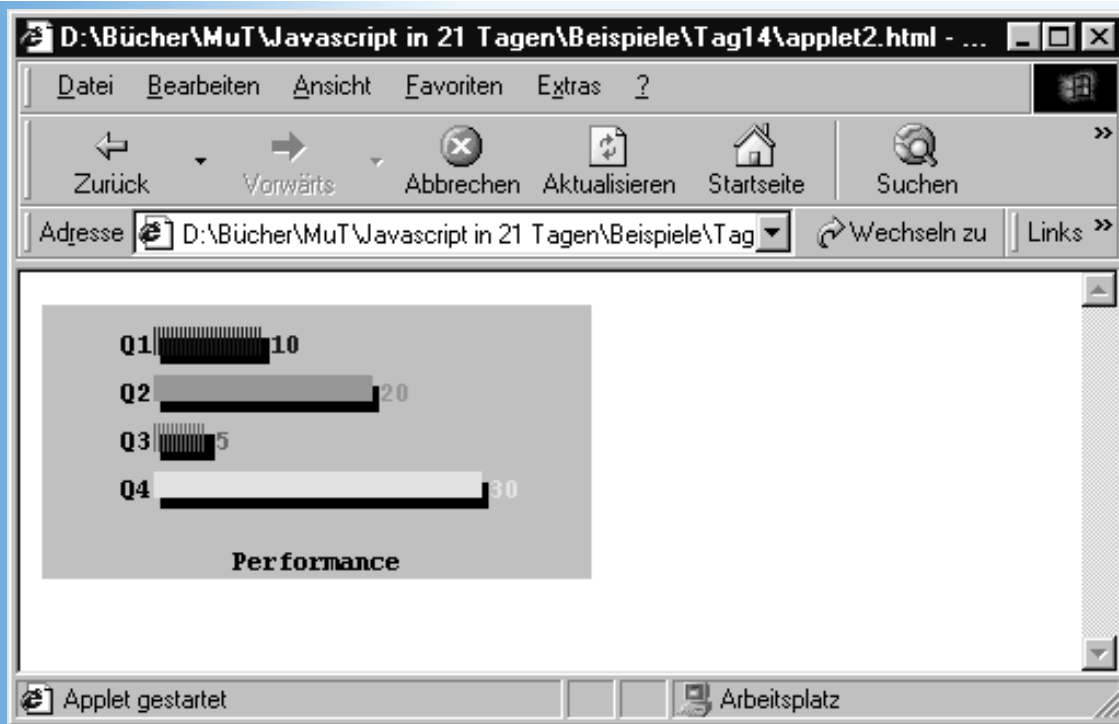


Abbildung 14.7: Das Applet mit den Parametern des Beispiels

Verändern Sie die Werte der Parameter und schauen Sie, was passiert (siehe Abbildung 14.8).

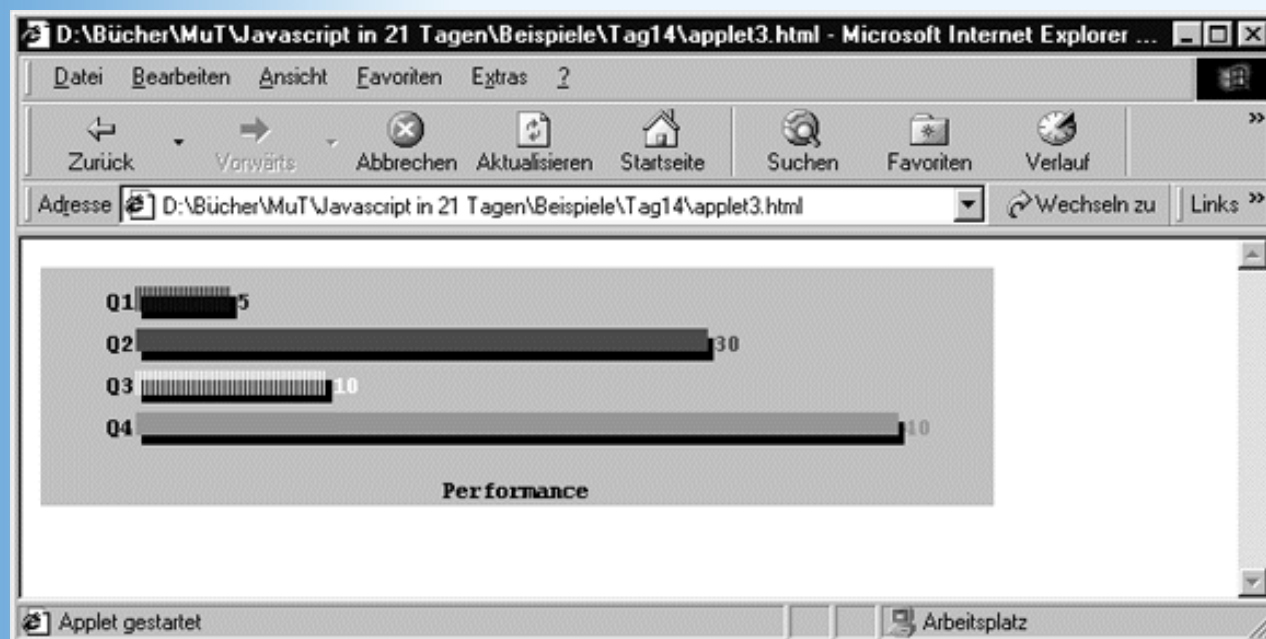


Abbildung 14.8: Das Applet mit anderen Parametern

Zugriff auf Java-Funktionalitäten aus Scripten

Die Verbindung von JavaScript mit Java bzw. Java-Applets ist nicht auf den gemeinsamen, aber sonst unabhängigen Einsatz innerhalb einer Webseite beschränkt. Es gibt mehrere Möglichkeiten, wie JavaScript auf ein Java-Applet Einfluss nehmen kann, aber auch umgekehrt Java-Applets auf Scripte zugreifen können.

Dynamische Generierung der Applet-Referenz

Die einfachste Möglichkeit, um aus JavaScript heraus auf Java-Applets zugreifen zu können, ist die dynamische Generierung einer Webseite mittels `document.write()`. Damit wird einfach die gesamte Applet-Referenz samt eventuell notwendiger Parameter dynamisch geschrieben. Beim Laden der Webseite wird die Referenz dann verwendet und eventuelle Parameter werden vom Applet aufgenommen.

Beispiel 7:

Wir nehmen hier ein Applet, das in einem Parameter übergebenen Text zerschneidet und in zufälligen Farben die Teilstücke anzeigt. Es entsteht ein blinkender Text, wobei dessen Blinkgeschwindigkeit über einen weiteren Parameter manipuliert werden kann. Wenn die Referenzierung in HTML hardcodiert erfolgt, sieht die HTML-Datei samt Applet-Referenz mit Beispielwerten so aus (App.htm):

```
<HTML>
<BODY>
<APPLET
  code="Blink.class" width=300 height=100>
  <PARAM
    name=lbl value="Die Antwort ist 42!">
  <PARAM
    name=speed value="4">
  Hier sollte ein Applet ablaufen
</APPLET>
</BODY>
</HTML>
```

Wenn Sie die Datei in einen Java-fähigen Browser laden, wird der in der Zeile

```
<PARAM name=lbl value="Die Antwort ist 42!">
```

angegebene Text mit der in Zeile

```
<PARAM name=speed value="4">
```

angegebenen Geschwindigkeit blinkend angezeigt.

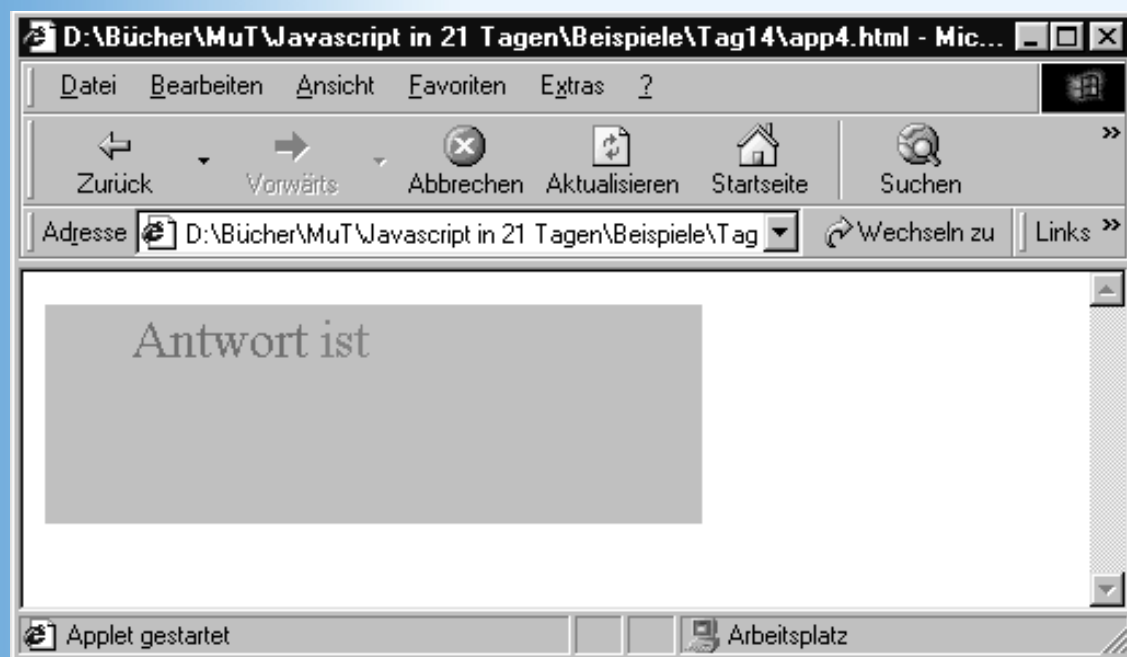


Abbildung 14.9: Das Applet mit den fest in HTML vorgegebenen Parametern

Sie können jetzt in HTML selbst die Werte der Parameter verändern (und auch die Größe des Applets), aber das wollen wir einem Anwender überlassen, der ein Webformular ausfüllen und dort die Eckdaten des Applets eingeben kann.

Beispiel 8:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
function belegeVor()
{
document.forms[0].elements[0].value=300;
document.forms[0].elements[1].value=200;
document.forms[0].elements[2].value="Ist das gut?";
```

```
document.forms[0].elements[3].value="4";
}
function schreibeSeite(breite, hoehe, text, speed)
{
var appBeginn = '<APPLET code="Blink.class" width=' +
    breite +
    " height=" +
    hoehe +
    ">";
var appPara1 = '<PARAM name=lbl value="' +
    text +
    '">';
var appPara2 = '<PARAM name=speed value="' +
    speed +
    '">';
document.write("<HTML>");
document.write("<BODY>");
document.write("<CENTER>");
document.write(appBeginn);
document.write(appPara1);
document.write(appPara1);
document.write("Hier sollte ein Applet ablaufen");
document.write("</APPLET>");
document.write("</CENTER>");
document.write("</P>");
document.write("</BODY>");
document.write("</HTML>");
}
</SCRIPT>
</HEAD>
<BODY onLoad="belegeVor()">
<CENTER>
<H1>Geben Sie die Eckdaten des Applets ein</H1>
</CENTER>
<FORM>
Breite <INPUT size=4 maxlength=4>
Höhe <INPUT size=4 maxlength=4>
<BR>
Text <INPUT size=60 maxlength=60>
<BR>
Geschwindigkeit <INPUT size=5 maxlength=3>
<P>
<CENTER>
<INPUT type="Button" value="Starte Applet"
onClick="schreibeSeite(
    document.forms[0].elements[0].value,
    document.forms[0].elements[1].value,
    document.forms[0].elements[2].value,
    document.forms[0].elements[3].value)">
</CENTER>
</FORM>
</BODY>
</HTML>
```

Speichern Sie die Datei unter appletDynamisch.htm und laden Sie diese dann in einen Java-fähigen Browser. Zuerst haben Sie in einem Formular die Möglichkeit, die Breite und Höhe des Applets sowie den für die Anzeige zu verwendenden Text und die Blinkgeschwindigkeit einzugeben. Experimentieren Sie mit verschiedenen Eingaben.

Beachten Sie, dass die Felder vorbelegt sind. Das erledigt die beim Laden der Webseite aufgerufene Funktion belegeVor(). Dabei verwenden wir - wie schon öfter - eine Objektnotation für die Felder des Formulars (das bekommen wir samt einiger

anderer Formularfeinheiten in den nächsten Tagen genauer).

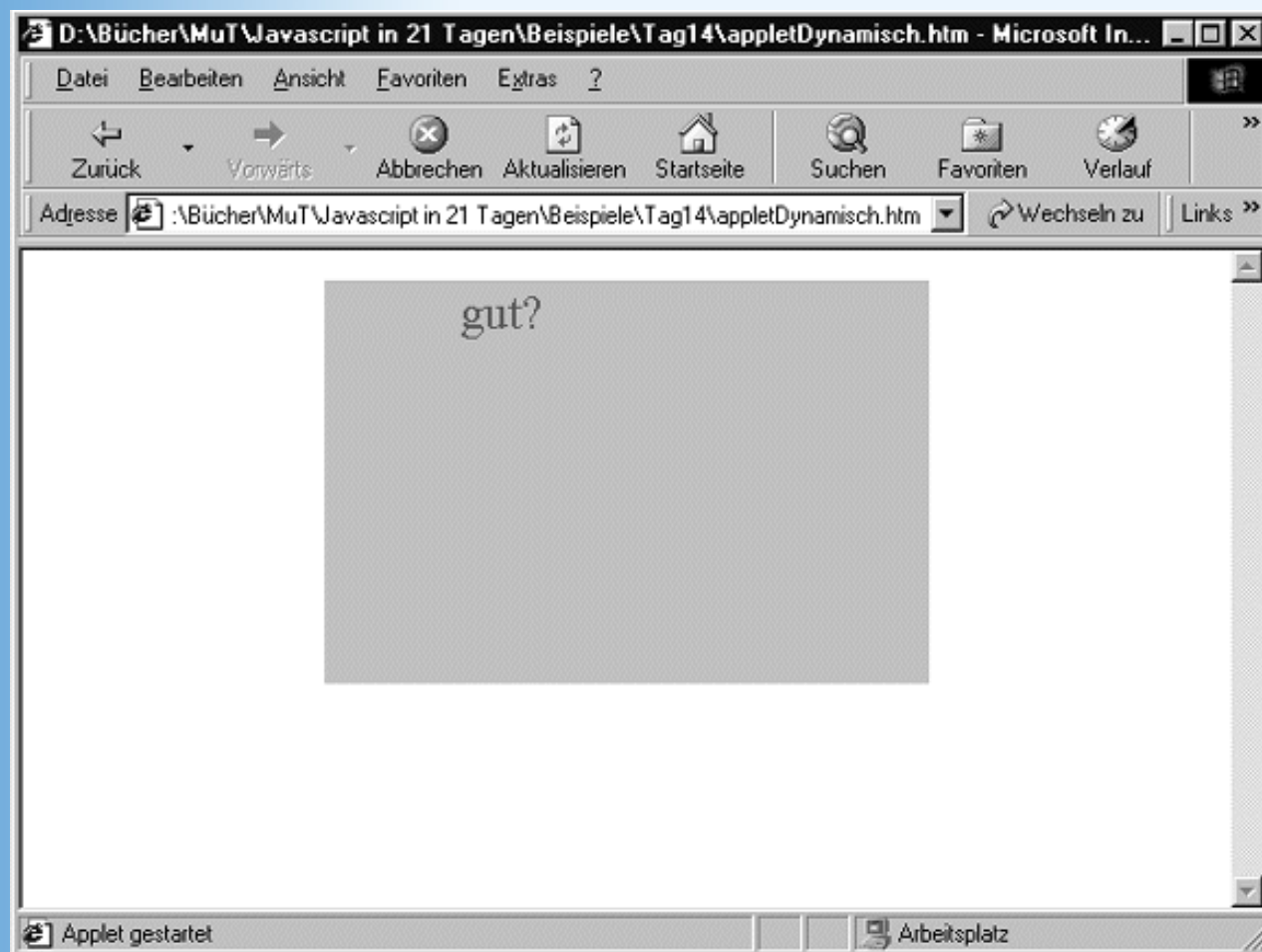


Abbildung 14.10: Das Applet mit den unveränderten Vorgabewerten



Abbildung 14.11: Das Formular wird vorbelegt

Die Werte, welche der Anwender in das Formular einträgt (beachten Sie, dass die Eingabefelder bezüglich der Größe und der maximal einzugebenden Zeichen begrenzt sind), werden als Übergabewerte an die Funktion `schreibeSeite()` übergeben.

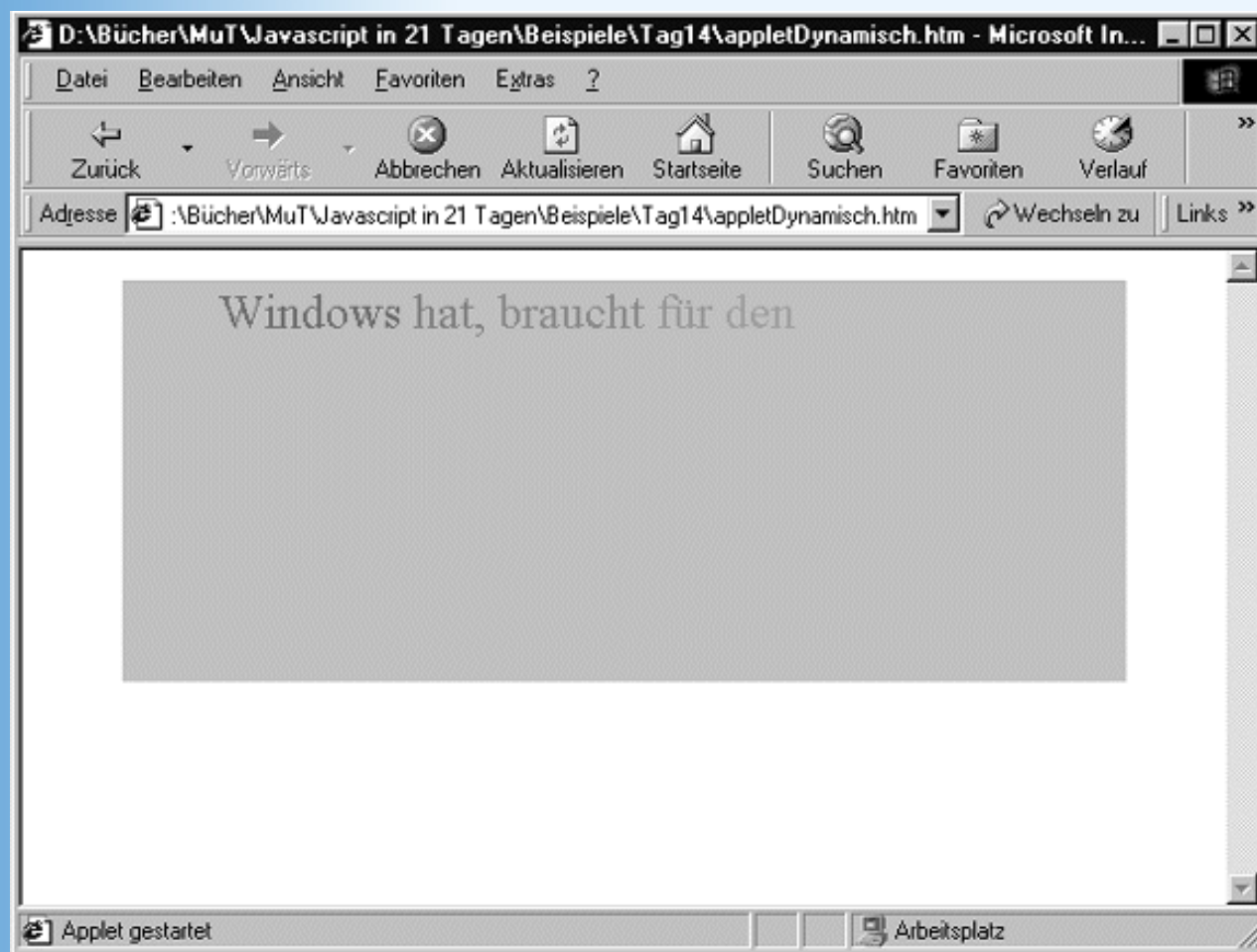


Abbildung 14.12: Das resultierende Applet



Abbildung 14.13: Im Formular werden die Parameter geändert

Dort werden die Übergabewerte mit String-Bausteinen verkettet und diese dann mit den übrigen HTML-Anweisungen über `document.write()` geschrieben.



Beachten Sie, dass die direkte Notation der Übergabewerte in der Anweisung `document.write()` nicht sinnvoll ist. Das vorherige Zusammensetzen eines Strings außerhalb der Anweisung ist auf jeden Fall besser.

LiveConnect

Über das dynamische Erstellen der Applet-Referenz aus JavaScript heraus gibt es auch die Möglichkeit, direkt zwischen Quellcodes von einem Java- Applet und einem JavaScript Verbindungen zur Laufzeit zu schaffen. Beide Sprachen sind dynamisch, d.h. noch zu Laufzeit können Dinge hinzugeladen werden, die ein bereits fertiges Script oder Applet beeinflussen können. Um dies zu realisieren, gibt es verschiedene Ansätze, von denen wir exemplarisch einen behandeln.



Der Datenaustausch von JavaScript nach Java und umgekehrt ist nicht ganz trivial. Im Wesentlichen deshalb, weil die Datentypen und Objekte beider Sprachen nicht vollkommen gleich sind. Zwar liegen die allgemeinen Regeln für die Umwandlung von Datentypen und Objekten fest, aber durch die unterschiedlichen Längen der Datentypen und deren Struktur ist eine Konvertierung nicht immer ohne Datenverlust möglich.

Es gibt von Netscape für den externen Zugriff auf Java-Funktionalitäten unter JavaScript innerhalb eines so genannten Objektmodells (das bekommen wir nächste Woche genauer) die Objekte (unbedingt Groß- und Kleinschreibung exakt beachten)

- Packages,
- java,
- JavaArray,

- `JavaObject` und
- `JavaPackage`.

Damit haben Sie Zugriff auf das so genannte Core-Java. Das sind die zum reinen Java gehörenden Funktionalitäten. Dazu gibt es zusätzlich die unter JavaScript ansprechbaren Objekte

- `netscape` und
- `sun`,

die den Zugriff auf die herstellerspezifischen Java-Pakete `netscape.*` bzw. `sun.*` ermöglichen. Das gesamte Konzept wird von Netscape unter dem Namen *LiveConnect* zusammengefasst und beinhaltet auch den umgekehrten Zugriff aus Java heraus auf JavaScript.



LiveConnect funktioniert ausschließlich mit dem Netscape Navigator oder kompatiblen Browsern. Das gesamte Verfahren setzt zudem für eine tiefer gehende Anwendung einige Grundkenntnisse in Java voraus, auf die wir hier nicht eingehen können und weshalb wir ziemlich an der Oberfläche bleiben müssen. Zudem müssen einige Java-Details einfach so hingenommen werden.

Über das `Packages`-Objekt bzw. die anderen genannten Objekte hat man aus JavaScript heraus Zugriff auf alle unter Java als `public` deklarierten Methoden und Felder. Dies sind solche Methoden und Eigenschaften, die über dieses Java-Schlüsselwort für den Zugriff von außen freigegeben sind. Auf Elemente, welche nicht so deklariert sind, hat man keinerlei Zugriff. Das ist aber nicht als ein Nachteil zu verstehen - wenn Sie ein Auto fahren wollen, brauchen Sie ja auch keinen direkten Zugriff auf die Motorelektronik. Im Gegenteil - das wäre gefährlich und könnte die gesamte Funktionalität in Frage stellen, wenn man da was veränderte. Ein Objekt sollte nur das nach außen geben, was notwendig ist, und den Rest im Hintergrund und vor dem Anwender verborgen erledigen.

Aus JavaScript heraus können Sie nun mittels der Standard-Punktnotation auf die Methoden und Eigenschaften von Java zugreifen, die in den angegebenen Paketen bereitgestellt werden (sofern die JVM zur Verfügung steht, was in neueren Browsern aber so gut wie immer der Fall ist - sonst würden auch Java-Applets nicht laufen). Die Objekte `java`, `netscape` und `sun` sind einerseits Eigenschaften von `Packages`, andererseits wieder selbst Objekte, welche die Pakete `java.*`, `netscape.*` und `sun.*` repräsentieren.

Selbstverständlich kann man mit dem Aufruf von Java-Funktionalitäten aus JavaScript heraus auch auf Applets zugreifen (über die nächste Woche noch intensiver auszudiskutierende Syntax

```
document.[Applet].[Appletmethode/Appleteigenschaft]),
```

aber man kann noch viel mehr machen. Aus JavaScript heraus kann man auf den Konstruktor (das ist eine besondere Methode, mit der ein Objekt erzeugt werden kann - das werden wir auch bei JavaScript in der nächsten Woche sehen) einer beliebigen Java-Klasse mit folgender Syntax zugreifen:

```
var [JavaScriptObjektvariable] = new Packages.[Klasse];
```

Das sieht etwa für die Java-Klasse `Frame` zum Erstellen eines Fensters aus JavaScript wie folgt aus:

```
var meinFrame = new Packages.java.awt.Frame();
```

Es geht sogar noch einfacher, denn die jeweiligen JavaScript-Objekte `netscape`, `sun` und `java` stehen als Synonyme für die Pakete gleichen Namens. Daher ist ein Zugriff auch ohne das `Packages`-Schlüsselwort möglich. Das Beispiel von oben sieht dann so aus:

```
var meinFrame = new java.awt.Frame();
```



Das `Package`-Objekt stellt überdies mit der Eigenschaft `className` den Pfadnamen zu jeder aus JavaScript erreichbaren Java-Klasse zur Verfügung. Darüber kann dann sogar auf Java-Klassen außerhalb der direkt auf Toplevel-Ebene zur Verfügung stehenden Pakete zugegriffen werden.



Nochmals der Hinweis, dass die LiveConnect-Technik explizit auf das Netscape-Objektmodell abgestimmt ist.

Beispiel 9:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function ausgabe()
{
java.lang.System.out.println(
"Erbarme - zu spät. Die Hesse komme.");
}
</SCRIPT>
</HEAD>
<BODY onLoad=ausgabe(>
</BODY>
</HTML>
```



Abbildung 14.14: Nicht eindrucksvoll, aber die Ausgabe erfolgt über Java

Speichern Sie das Script unter Java1.htm und laden Sie es in den Netscape Navigator. Das erste einfache Beispiel demonstriert die Ausgabe auf Systemebene über die Java-Standardausgabemethode `println()`, welche innerhalb eines JavaScripts aufgerufen wird. Die Ausgabe erfolgt in der Java-Console, welche im Communicator (unter dem Menüpunkt **Communicator** und **Extras**) erst geöffnet werden muss, bevor man etwas sehen kann.

Das nächste, etwas komplexere Beispiel greift über Java auf die Systemeinstellungen zu und gibt diese Informationen in der Java-Console aus:

Beispiel 10:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function ausgabe()
```

```
{
var Syst = java.lang.System;
var OS = Syst.getProperty("os.name");
var OSver = Syst.getProperty("os.version");
var JavaVer = Syst.getProperty("java.version");
java.lang.System.out.print("Die Java-Version: ");
java.lang.System.out.println(JavaVer);
java.lang.System.out.print("Das Betriebssystem: ");
java.lang.System.out.println(OS);
java.lang.System.out.print(
"Die Betriebssystemversion: ");
java.lang.System.out.println(OSver);
}
</SCRIPT>
</HEAD>
<BODY onLoad=ausgabe()>
</BODY>
</HTML>
```

Speichern Sie das Script unter Java2.htm und laden Sie es in den Netscape Navigator. Über die Methode `getProperty()` werden verschiedene Eckdaten der Plattform abgefragt, die dann wieder in der Java-Console ausgegeben werden. Beachten Sie bitte, dass wir hier zwei verschiedene Methoden zur Ausgabe verwenden. Einmal `println()`, aber auch `print()`. Während `println()` einen Zeilenvorschub bewirkt (dieser wirkt sich hier aus - die Ausgabe erfolgt nicht im Rahmen der Webseite, wo ein expliziter HTML-Tag benötigt würde), wird `print()` eine Ausgabe des spezifizierten Inhalts ohne Zeilenvorschub auslösen.



Abbildung 14.15: Systemdaten unter Java ermittelt

Das dritte Beispiel zur Anwendung von LiveConnect verwendet eine JavaScript-Funktion, um darin ein Java-Fenster zu erstellen und anzuzeigen. Dabei wird mittels Java die Titelzeile des Fensters beschriftet, die Hintergrundfarbe gesetzt, ein Layout festgelegt und zwei Buttons im Fenster platziert. Bitte beachten Sie, dass der Java-Code keinen Schließbefehl für das erzeugte Fenster enthält (Schließen des Browsers beendet auch das Fenster) und auch die Buttons noch keine weitergehende Funktionalität haben (das würde zu weit in Java hineinführen).

Beispiel 11:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function fenster()
{
var meinFenster = new java.awt.Frame();
var meinButton1 = new java.awt.Button("Okay");
var meinButton2 = new java.awt.Button("Nicht okay");
var evt = new java.awt.Event();
meinFenster.setSize(350,200);
meinFenster.setTitle("Mein Java-Fenster");
meinFenster.setLayout(new java.awt.FlowLayout());
meinFenster.setBackground(java.awt.Color.red);
meinFenster.add(meinButton1);
meinFenster.add(meinButton2);
meinFenster.setVisible(true);
}
</SCRIPT>
</HEAD>
<BODY onLoad=fenster(>
</BODY>
</HTML>
```

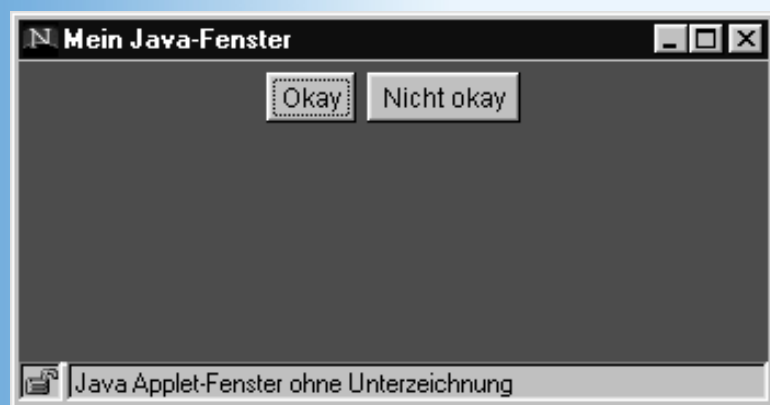


Abbildung 14.16: Ein per JavaScript-Funktion aufgerufenes Java-Fenster

Zusammenfassung

HTML kann mittels Referenzen und zahlreicher Techniken ergänzt werden. Dabei gibt es von der Theorie her seit HTML 4.0 einen zentralen HTML- Tag, der alle diese Aufgaben erledigt: <OBJECT>. In der Praxis zeigt sich jedoch, dass dieser Tag die vielen, für jede Technik individuellen, Tags noch lange nicht verdrängt hat. Insbesondere für die Referenz von Java-Applets ist der <APPLET>-Tag das Maß aller Dinge.

Um aus HTML auf Java-Applets zugreifen und sie beeinflussen zu können, enthält die Applet-Referenzierung bei Bedarf spezielle Parameter-Tags. Diese Beeinflussung über Parameter aus HTML heraus ist jedoch nur statisch, d.h. einmal fest in die Webseite geschrieben. Wenn man das Java- Applet weitergehend manipulieren möchte, bietet JavaScript (aber auch andere Scriptsprachen wie VBScript) einen idealen Ansatz. Da man mit JavaScript dynamisch Webseiten schreiben kann, ist es auch leicht möglich, die komplette Applet-Referenz neu zu schreiben.

Fragen und Antworten

Frage:

Kann man nur den <APPLET>-Tag mit JavaScript neu schreiben?

Antwort:

Nein. Selbstverständlich kann eine Applet-Referenz auch mit den anderen Tags dynamisch generiert werden. Und nicht nur Applet- Referenzen.

Frage:

Kann man auch andere multimediale Elemente aus JavaScript heraus beeinflussen?

Antwort:

Ja. Alle Elemente, die per HTML Parameter übergeben bekommen, können damit auch über JavaScript dynamisch manipuliert werden. Etwa ActiveX-Controls. Aber auch Bilder, Tabellen usw. Gerade Letzteres werden wir aber noch genauer sehen. Der Zugriff direkt auf Techniken jenseits von Java auf Source-Ebene wie unter LiveConnect ist jedoch unter JavaScript nicht-trivial. (Wobei auch die Java-JavaScript-Verbindung nicht als trivial angesehen sollte - sie ist nur bei LiveConnect durch die Oberhoheit von Netscape stark standardisiert.)

Frage:

Brauche ich für die Anwendung von LiveConnect irgendwelche Java- Entwicklungstools oder sonstige Programme oder Dateien?

Antwort:

Nein. Jeder neuere Netscape Browser beinhaltet bei richtiger Installation bereits alles, damit Java so unterstützt und so angewandt werden kann. Das bedeutet aber nicht, dass Sie bereits Java-Applets oder vollständige Java-Programme erstellen können. Dafür benötigen Sie dann Java-Entwicklungstools wie das JDK.

Frage:

Kann ich mit reinen JavaScript-Kenntnissen LiveConnect sinnvoll nutzen?

Antwort:

Nein. Vermutlich nicht. Java unterscheidet sich doch erheblich von JavaScript. Ohne ausreichende Java-Kenntnisse wird der Einsatz kaum effektiv. Aber man sollte beachten, dass das Lernniveau von Java erheblich über dem von JavaScript liegt. Insbesondere ist die Anfangsschwelle von Java sehr hoch. Es wird insbesondere viel Hintergrundwissen aufgebaut werden müssen, bevor man die ersten Erfolge sieht und vor allem deren Vorgänge versteht.

Workshop

Nehmen Sie sich die Video-Referenz in Beispiel 1 mit den statischen Parametern vor und schaffen Sie analog dem Beispiel 7 ein Formular, in dem ein Anwender Parameterwerte (sprich den Namen eines Videos) auswählen kann. Daraus schreiben Sie dann die konkrete Video-Referenz. Sie sollten natürlich ein paar Videos parat haben.

Verändern Sie in Beispiel 8 die Ausgabe. Fügen Sie weitere Ausgabezeilen hinzu.

Verändern Sie Beispiel 9 so, dass die Ausgabe als neu generierte Webseite erfolgt (mit document.write()).

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was stimmt hier nicht?

```
<EMBED  
src="video001.avi"  
width=300  
height=300  
<NOEMBED>>  
Hier läuft eigentlich ein Video.  
</NOEMBED>
```

2. Ist das korrekt?

```
<EMBED  
type="application/x-java-applet"  
code="MeinApplet.class"  
width="300"  
height="200">
```

3. Was tut folgender HTML-Tag?


```
<BGSOUND  
src="test.wav"  
loop=2>
```

4. Was muss bei der Angabe von Parametern für ein Java-Applet unbedingt beachtet werden?
5. Versteht der Internet Explorer LiveConnect?
6. Ist das korrekt?

```
<APPLET  
code=ArcTest.class  
height=400>  
Hier sollte ein Applet stehen  
</APPLET>
```

7. Was ist der Unterschied zwischen den Java-Methoden `println()` und `print()`?
8. Was bewirkt folgende Syntax?
`var meinFrame = new Packages.java.awt.Frame();`

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

1

Die unsinnige zeitliche Formulierung ist genauso gemeint.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 2

Sie haben nun das Ende der Woche zwei erreicht. Die Woche stand ganz im Blickpunkt eines systematischen Aufbaus von JavaScript und dessen syntaktischen Grundlagen. Fassen wir wieder knapp zusammen, was an Stoff behandelt wurde. In dieser Woche wurde besprochen,

- was Variablen, Datentypen, Literale und Ausdrücke sind,
- was Funktionen und Prozeduren sind,
- was Objekte sind und wie Methoden und Eigenschaften in Bezug zu Funktionen und Variablen zu sehen sind,
- was Wertzuweisungen, Operatoren und Operanden sind,
- was Schlüsselworte sind und welche es in JavaScript gibt,
- welche Anweisungen und Kontrollfluss- Steuerungsmöglichkeiten JavaScript kennt,
- wie der Aufruf von JavaScripten erfolgt,
- wie Listen, Aufzählungen und Tabellen innerhalb einer Webseite realisiert werden (als kleiner HTML- Exkurs),
- welche vorgefertigten Funktionen und Top-Level- Eigenschaften JavaScript bereitstellt und
- wie die Verbindung von JavaScript mit Java sowie allgemeinen Multimedia-Techniken erfolgt.

Sie haben nun einen Punkt erreicht, wo sie die syntaktischen Grundlagen von JavaScript beherrschen.

Woche 3

In der nächsten Woche werden wir die Arbeit mit Objekten unter JavaScript in den Mittelpunkt stellen. Bisher haben wir Objekte einfach verwendet und uns keine weiteren Gedanken um deren Hintergründe gemacht. Das müssen wir aber, wenn wir JavaScript effektiv einsetzen wollen. Nur so werden Vorgänge logisch, die bisher weitgehend vom Himmel zu fallen scheinen. Dabei soll einmal geklärt werden, was Objekte genau sind und wie sie in JavaScript allgemein eingesetzt werden, aber auch, wie eigene Objekte erstellt werden können.

Unter den gesamten Themenkomplex fallen auch die globale Ereignisbehandlung sowie die Arbeit mit Formularen und Stilvereinbarungen in einer Webseite - mit und ohne JavaScript-Verbindung.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 3

Tag 15

Objekte in JavaScript - Grundlagen

Bisher haben wir JavaScript hauptsächlich unter dem Aspekt der Sprachsyntax gesehen. Aber bereits viele Vorgänge ließen sich nur dann anschaulich demonstrieren, wenn auf Objekte wie `document` oder `window` zurückgegriffen wurde. Objekte sind im Grunde die zentralen Elemente in der JavaScript-Programmierung, worüber erst JavaScripte ihre volle Bedeutung entfalten können. Einige besonders wichtige Standardobjekte haben wir ja wie gesagt bereits kennen gelernt (ohne deren Hintergründe gründlicher »aufzudröseln«). In der dritten Woche stehen Objekte und der potentielle Nutzen für JavaScripte auf dem Stundenplan. Sie werden dabei den Grundaufbau der JavaScript-Objektstruktur kennen lernen und natürlich Anwendungen der wichtigsten Objekte.

Der Umgang mit Objekten setzt etwas mehr Theorie voraus, als es in vielen anderen Schritten in unserem Lehrgang bisher der Fall war. Deshalb werden die Beispiele an diesem Tag auch hauptsächlich im hinteren Teil des Kapitels zu finden sein.

Was sind Objekte?

Wir wissen schon einiges über Objekte und wollen dies Know-how noch mal zusammentragen und ergänzen.

Objekte sind grundsätzlich zusammengehörende Anweisungen und Daten, die eine in sich abgeschlossene und eigenständige Einheit bilden. Unter einem Objekt in der EDV stellt man sich ein Softwaremodell vor, das ein Ding aus der realen Welt mit all seinen Eigenschaften und Verhaltensweise beschreiben soll. Etwa das Objekt Webseite, einen Teil dieser Seite (z.B. eine Überschrift, einen Absatz, eine Grafik, eine Schaltfläche, ein Eingabefeld usw.), einen Bildschirm oder einen Browser. Aber auch Teile von einer Software selbst können ein Objekt sein. Etwa der gesamte Anzeigebereich des Browsers, der unter dem Objekt `window` für JavaScript bereitsteht. Eigentlich ist in dem objektorientierten Denkansatz alles als Objekt zu verstehen, was sich eigenständig erfassen und ansprechen lässt.

Objekte bestehen im Allgemeinen aus zwei Bestandteilen, den Objektdaten, d.h. den Attributen beziehungsweise Eigenschaften, und den Objektmethoden.

Eigenschaften sind die Details, die ein Objekt charakterisieren und durch welche sich ein Objekt von einem anderen unterscheidet. Etwa die Größe, die Beschriftung oder die Farbe.

Methoden sind die aktiven Algorithmen, die von einem Objekt bereitgestellt werden, damit bei einem Aufruf bestimmte Dinge getan werden. Methoden sind so etwas wie einem Objekt fest zugeordnete Funktionen, die auch nur über das Objekt verwendet werden können.

In streng objektorientierten Sprachen wie Java gibt es keinerlei Elemente außerhalb von Objekten. Weder globale Variablen noch Funktionen oder Prozeduren. JavaScript, wie auch nahezu alle anderen Scriptsprachen, und weitere hybride Sprachen wie C/C++ stellen hingegen neben den OO- Elementen solche zusätzlichen Elemente bereit.

Damit Objekte aktiv werden können, tauschen sie so genannte *Botschaften* aus, welche die Kommunikation von Objekten regeln. Das sendende Objekt schickt dem Zielobjekt eine Aufforderung, eine bestimmte Aktion auszuführen. Das Zielobjekt reagiert entsprechend. Die genaue formale Schreibweise folgt in der Regel dem Schema »Empfänger Methodenname Argument«. Punkt und Klammer trennen dabei in den meisten Sprachen (insbesondere JavaScript) die drei Bestandteile der Botschaft (die so genannte *Punkt-Notation* oder *DOT-Notation*) in der Form

`Empfänger . Methodenname (Argument)`

Nach außen ist ein Objekt nur durch seine Schnittstellen zu den Methoden definiert; es ist gekapselt, versteckt seine innere Struktur vollständig vor anderen Objekten und erst recht vor objektfreien Elementen wie globalen Variablen, Funktionen

oder Prozeduren (Information Hiding). Dies bietet einige Vorteile:

- Wer ein Objekt verwenden will, muss überhaupt nichts über seinen inneren Aufbau wissen. Er kann sich einfach auf die Anwendung des Objekts konzentrieren. Das einzige, was ein Anwender wissen muss, ist der Name des Objekts und die bereitgestellten Eigenschaften und Methoden (bei Methoden ist es natürlich noch notwendig zu wissen, wie sie angewandt werden, welche Parameter sie benötigen und ob eventuell ein Rückgabewert vorhanden ist).
- Ein Objekt kann von dem Ersteller intern vollständig verändert werden. So lange es sich nur nach außen unverändert zeigt, wird das veränderte Objekt problemlos in einem System funktionieren, wo es in seiner alten Form funktioniert hatte.

Die Standardobjekte von JavaScript

Über JavaScript kann man auf zahlreiche vordefinierte Objekte zugreifen, die in Form einer Objekt-Bibliothek bereitgestellt werden. Dabei wollen wir uns das Konzept in der Tat wie eine Bibliothek mit Büchern vorstellen. JavaScript kann man sich als einen dort registrierten Leser vorstellen, der sich bei Bedarf ein Buch ausleiht.

Der Denkansatz beinhaltet die nahe liegende Vermutung, dass es noch weitere Leser geben könnte, die in der Kundenkartei der Bibliothek registriert sind. Das ist in der Tat richtig. Die für JavaScript zur Verfügung stehenden und von uns nachfolgend behandelten Objekte sind nicht für JavaScript reserviert, sondern eine offene Welt, auf die auch mittels anderer Techniken zugegriffen werden kann.

Die Objekt-Bibliothek sollte man sich auch nicht als eine Selbstbedienungsbibliothek vorstellen, sondern es gibt Bibliothekare, bei denen man nach einem bestimmten Buch fragen muss und die es dann herausgeben (oder auch nicht).

Wie konkret nach einem bestimmten Buch gefragt wird, wissen Sie schon. Über den Namen des Objekts. Aber der Denkansatz führt weiter. Die Vorstellung einer Bibliothek erlaubt auch die Erklärung einer so genannten Objekthierarchie.

Wenn Sie sich eine Sammlung von Büchern vorstellen - etwa ein Lexikon mit zwölf Bänden -, langt die Angabe der Sammlung nicht, wenn Sie ein einzelnes Exemplar benötigen. Sie müssen beim Bibliothekar auch exakt angeben, welches Buch der Sammlung Sie wollen. Wie geht das in der OO- Theorie? Na, über die Punktnotation. Angenommen, das Lexikon mit den gesammelten Werken heißt `document` und ist auch nur so als eigenständiges Werk im Hauptverzeichnis der Bibliothek registriert. Sie wollen nun das dritte Buch daraus (die einzelnen Bücher sollen über einen Bezeichner, `forms`, und einen Index beschriftet sein), dann sagen Sie dem Bibliothekar, er möge das Lexikon `document` aufsuchen (das findet er über das Hauptverzeichnis der Bibliothek) und dort das dritte Buch von `forms` heraussuchen. Wenn man das knapp in der DOT-Notation hinschreibt (und den Index bei 0 anfangen lässt), sieht das so aus:

```
document.forms[2]
```

Das kennen Sie schon, oder?! Wenn nicht, schauen Sie sich beispielsweise nochmals an Tag 14 das Beispiel 8 an.

Wenn nun auch jedes Buch `forms` aus einzelnen Abschnitten (sie sollen mal mit - sagen wir - `elements`¹ beschriftet sein) besteht und Sie nur einen der Abschnitte ausleihen können oder wollen, würde der zweite Abschnitt in der DOT-Notation so hingeschrieben:

```
document.forms[2].elements[1]
```

Stellen wir uns nun einmal den umgekehrten Weg vor. Wenn Sie dem Bibliothekar einfach nur sagen, Sie wollen Buchabschnitt `elements[1]` ausleihen, wird er wohl rückfragen, in welchem Buch er bitte suchen solle. Sagen Sie dann `forms[2]`, wird er immer noch nicht zufrieden sein, denn er wird kein eigenständiges Exemplar dieses Namens im Hauptverzeichnis der Bibliothek finden. Nur, wenn Sie ihm auch sagen, dass dieses Buch `forms[2]` zu dem Lexikon `document` gehört, wird er es finden und Ihnen geben.

Eine Sondersituation sollte noch besprochen werden: Wenn Sie bereits nach einem Buchabschnitt wie gerade beschrieben nachgefragt haben und Sie einen weiteren aus dem gleichen Buch benötigen, können Sie dem Bibliothekar unter Umständen sagen, dass er noch ein weiteres Buch mitbringen soll. Wenn Sie das dem Bibliothekar zurufen, während er vor dem Lexikon mit den zwölf Bänden steht, langt es, wenn Sie dann nur das Buch angeben und den sonst notwendigen Namen des Lexikons selbst weglassen. Nach dem Motto: »Bitte bringen Sie mir noch das Buch `forms[5]` mit«. Wenn Sie nicht gerade wegen Ruhestörung aus der Bibliothek geworfen werden, wird er damit was anfangen können und das Buch mitbringen. Das soll nur schon verdeutlichen, dass unter gewissen Umständen die DOT-Notation um die Angabe der Objekte verkürzt werden kann, die ohnehin klar sind. Wo man also schon unweigerlich drin ist (wie im Fall einer Webseite in `window`) oder es sonst irgendwie eindeutig ist, dass ein bestimmtes Objekt im Prinzip vorangestellt werden müsste.

Für JavaScript sind verschiedene Bibliothekare zuständig. Die Bibliothekare, an deren Schalter sich JavaScript anstellen muss, heißen alle mit zweitem Namen *Browser*. Die jeweiligen Vornamen sind *Netscape Navigator*, *Internet Explorer*,

Opera usw. Und jeder Bibliothekar hat so einige Bücher, die er seinem Kunden *JavaScript* nicht herausgibt. Und wenn es um eine bestimmte Fragestellung geht, gibt es unter Umständen mehr als ein Buch, in dem die Lösung enthalten ist. Nicht jeder Bibliothekar entscheidet sich dann für das gleiche Buch.

Verlassen wir nun den Denkansatz einer Bibliothek für Bücher und kommen zu den EDV-Objekten zurück.

Unter JavaScript stehen u.a. die folgenden Objekte für eine Verwendung in einer Webseite zur Verfügung.

Objekt	Beschreibung
all	Das Objekt ermöglicht den direkten Zugriff auf alle Elemente einer HTML-Datei. Es gehört aber nicht zum offiziellen JavaScript-Standard, sondern ist eine Implementation für den Internet Explorer ab der Version 4.0.
anchor	Das Objekt beinhaltet alle Verweisanker in einer HTML-Datei.
applet	Das Objekt beinhaltet alle Java-Applets in einer HTML-Datei.
Array	Über dieses Objekt werden Arrays erzeugt. Dessen Elemente können über einen gemeinsamen Bezeichner und einen Index angesprochen werden.
Boolean	Ein Objekt mit Wahrheitswerten.
Date	Das Objekt enthält Informationen zu Datum und Uhrzeit.
document	Dieses Objekt repräsentiert die Webseite selbst.
event	Ein Objekt, das bei Anwenderereignissen erzeugt wird und für die (zentrale) Ereignisbehandlung genutzt werden kann.
form	Objekt, das die Formulare einer HTML-Seite repräsentiert.
frame	Objekt, das die Framesets und Frames einer HTML-Seite repräsentiert.
Function	Ein Objekt mit JavaScript-Funktionen.
history	Dieses Objekt enthält Informationen über die URLs, die ein Anwender besucht hat.
image	Ein Objekt, über das auf die Grafiken in einer HTML-Datei zugegriffen werden kann.
layer	Die Layer in einer HTML-Datei (Netscape-spezifisch).
link	Das Objekt, das die Verweise in der aktuellen HTML-Datei repräsentiert.
location	In diesem Objekt werden Informationen über URL-Adressen geführt.
Math	Ein Objekt mit zahlreichen mathematischen Konstanten und Methoden.
mimeType	Ein Objekt mit MIME-Type-Informationen.
navigator	Die Objektrepräsentation mit Informationen über den verwendeten WWW-Browser.
Number	Ein Objekt mit numerischen Werten.
plugin	Ein Objekt, das die vorhandene Plugins in einem Browser repräsentiert.
RegExp	Ein Objekt mit regulären Ausdrücken.
screen	Ein Objekt mit Informationen über den verwendeten Bildschirm.
String	Ein Objekt für die Manipulation von Zeichen und Zeichenketten.
Style	Die Objektrepräsentation der Stilattribute eines Elements.
window	Dieses Objekt enthält Statusinformationen über das gesamte Browser-Fenster. Jedes Fenster hat sein eigenes window-Objekt. Das window-Objekt ist das höchste Objekt in der Objekthierarchie der Objekte, welche den Browser direkt betreffen.

Tabelle 15.1: Objekte unter JavaScript.

Mehr zu diesen Objekten und zahlreiche konkrete Anwendungen werden wir in dieser Woche noch sehen.

Objektfelder und Arrays allgemein

Es gibt neben den hier aufgeführten Objekten weitere Objekte, die sich in der Notation ein wenig von den anderen Objekten unterscheiden, aber sonst ganz »normale« Objekte sind. Dies sind so genannte Objektfelder. Charakteristisch dafür ist, dass

diese über einen Feldnamen sowie eine Indexnummer identifiziert werden. Ansonsten ist die Anwendung von Eigenschaften und Methoden vollkommen identisch und wir werden im Folgenden meist von Objekten sprechen. Beispiele für solche Objektfelder sind `forms[]` oder `elements[]`. Es handelt sich dabei um Arrays, einen Begriff, dem wir uns gleich ausführlich widmen wollen. Vorerst klären wir die Frage, wie Objektfelder mit den obigen Objekten in Beziehung stehen?

Einige Objektfelder entstehen automatisch, wenn eine Webseite geladen wird und Objekte eines bestimmten Typs darin enthalten sind. Wenn beispielsweise eine Webseite ein Formular enthält, bedeutet dies, ein Objekt des Typs `form` ist darin enthalten. Wenn nun mehr als ein Formular in einer Webseite vorhanden ist, muss der Browser diese Formulare irgendwie identifizieren und speichern. Jedes Formular wird in einem Feld eines Objektfeldes gespeichert, das automatisch generiert wird und das vom Bezeichner meist dem erzeugenden Objekt sehr ähnlich ist (im Fall von Formularen ist das beispielsweise `forms` - beachten Sie das `s`). Die Indexnummern entstehen automatisch, wenn der Browser das Objekt beim Abarbeiten der HTML-Seite erzeugt und in einen Schlitz des Arrays einordnet. Das erste im Dokument auftretende Objekt jeden vorkommenden Typs erhält den Index 0, das zweite den Index 1 und so fort.

Für den Fall von Formularen wird das erste Objekt vom Typ `form` im Array- Eintrag `forms[0]` gespeichert, das zweite in `forms[1]` usw.

Die nachfolgende Tabelle gibt die wichtigsten Objektfelder an, deren potentiellen Inhalt sowie eine kleine Beschreibung.

Objektfeld	Typ der enthaltenen Objekte	Beschreibung
<code>anchors</code>	<code>anchor</code>	Die im Objektfeld enthaltenen Objekte repräsentieren eine Liste aller Hypertext-Anker in einer Webseite.
<code>applets</code>	<code>applet</code>	Die enthaltenen Objekte repräsentieren eine Liste aller Applets in einer Webseite.
<code>elements</code>	[Eingabelemente eines HTML-Formulars]	Die enthaltenen Objekte repräsentieren eine Liste aller Eingabelemente, welche sich in einem als übergeordnetes Objekt angegebenen Formular befinden. Diese werden in JavaScript durch die folgenden Objekte repräsentiert: <code>Button</code> , <code>Checkbox</code> , <code>FileUpload</code> , <code>Hidden</code> , <code>Password</code> , <code>Radio</code> , <code>Reset</code> , <code>Select</code> , <code>Submit</code> , <code>Text</code> und <code>Textarea</code> (das bekommen wir bei der Behandlung von Formularen umfassend).
<code>forms</code>	<code>form</code>	Die enthaltenen Objekte repräsentieren eine Liste aller Formulare in einer Webseite.
<code>frames</code>	<code>frame</code>	Die enthaltenen Objekte repräsentieren eine Liste aller Frames in einer Webseite.
<code>images</code>	<code>image</code>	Die enthaltenen Objekte repräsentieren eine Liste aller Bilder in einer Webseite.
<code>links</code>	<code>link</code>	Die enthaltenen Objekte repräsentieren eine Liste aller Hyperlinks in einer Webseite.
<code>mimeTypes</code>	<code>mimeType</code>	Die enthaltenen Objekte repräsentieren eine Liste aller MIME-Typen in einer Webseite.
<code>options</code>	[Liste der Optionen eines Eingabefeldes vom Typ <code>select</code>]	Die enthaltenen Objekte repräsentieren eine Liste aller erlaubte Optionen, die bei dem als übergeordnetes Objekt angegebenen Objekt vom Typ <code>select</code> vorkommen.
<code>plugins</code>	<code>plugin</code>	Die enthaltenen Objekte repräsentieren eine Liste aller in dem Browser installierten Plugin-Module.

Tabelle 15.2: Objektfelder

Arrays

Wir wollen jetzt auf ein Thema zu sprechen kommen, das viele Quellen bei der Behandlung von Variablen einordnen. Das ist auch durchaus sinnvoll - ein Array oder Datenfeld ist eine Sammlung von Variablen, die alle über einen Namen und einen numerischen Index angesprochen werden können. Wir führen die Details zu Arrays aber erst hier bei der Behandlung von Objekten bzw. Objektfeldern ein. Warum gerade hier? Warum nicht später, warum nicht früher? Dies hat natürlich seine Gründe:

1. Objektfelder sind Arrays mit Objekten. Zwar haben wir bereits Objektfelder verwendet, aber es ist spätestens jetzt höchste Zeit, die Hintergründe zu dem Thema zu klären.
2. Arrays sind in JavaScript Objekte! Das ist der Grund, warum wir bisher noch nicht intensiver darauf eingegangen sind. Die Erzeugung von Arrays lässt sich nur in Verbindung mit der Erzeugung von Objekten einleuchtend behandeln.

Es ist also in unserem Aufbau des JavaScript-Lehrgangs der ideale Zeitpunkt zum Einführen von Arrays.

Array sind immer dann von großem Nutzen, wenn eine Reihe von gleichartigen oder logisch zusammenfassbaren Informationen gespeichert werden soll. Das könnten beispielsweise die Monate des Jahres, die Applets in einer Webseite oder die zu einer Person gehörenden Daten sein.

Wenn Sie für jede einzelne Information eine eigene Variable definieren, müssen sie viele sinnvolle Namen vergeben und man hat viel Schreibarbeit. Ein Name und ein Index sind viel effektiver. Der Hauptvorteil ist aber, dass der Zugriff auf die einzelnen Einträge im Array über den numerischen Index erfolgen kann. Das kann man in Programmkontrollfluss-Anweisungen und sonstigen automatisierten Vorgängen nutzen. Erst damit werden Dinge möglich, die man bei der Nutzung von Objektfeldern in einer Webseite einsetzt. Etwa der Fall, dass in einer Webseite fünf Applets vorkommen. Mit den Vorschriften, wie sie im unter JavaScript verwendeten Objektmodell realisiert sind, ist klar, dass diese in einem Array mit definierten, vorgegebenen Bezeichnern (applets) gespeichert werden und in der Reihenfolge ihres Vorkommens in der Webseite im Array einsortiert und durchnummeriert werden.

Ein Array wird anders erzeugt als eine normale Variable. Einmal beinhaltet ein Array eine gewisse Anzahl von Feldern, die man angeben kann (was aber keine verbindliche Festlegung ist - in JavaScript können dynamisch neue Felder hinzukommen). Was aber noch ein erheblicher Unterschied zu »normalen« Variablen ist, ist die Tatsache, dass Arrays eben Objekte sind. Sie werden mit Hilfe eines JavaScript-Schlüsselworts erzeugt: `new` (die Hintergründe werden unmittelbar nach diesem Abschnitt geklärt). Und zwar mit folgender Syntax:

```
var [Arraybezeichner] = new Array([Anzahl Einträge]);
```

Das erzeugt ein Array mit der vorgegebenen Anzahl von Einträgen.

Alternativ kann man die Arrayeinträge bereits vorbelegen (mit identischen Werten):

```
var [Arraybezeichner] =  
new Array([Anzahl Einträge], "[Vorbelegungswert]");
```

Eine Erzeugung mit gleichzeitiger Vorbelegung und damit auch festgelegter Anzahl von Elementen geht auch so:

```
var [Arraybezeichner] =  
new Array([Element0], [Element1], ..., [ElementN]);
```

Auf diese Weise wird ein Datenfeld mit $N + 1$ Elementen erstellt.



Die Größe eines Arrays lässt sich mit der Eigenschaft (es sind ja Objekte) `length` überprüfen.

Da sich aber Arrays in JavaScript zur Laufzeit noch vergrößern lassen, legt man oft nur den Array-Bezeichner fest und fügt dann bei Bedarf die Felder hinzu. Ein Array ohne Einträge wird einfach so erzeugt:

```
var [Arraybezeichner] = new Array();
```

Dies entspricht

```
var [Arraybezeichner] = new Array(0);
```

Einem Element eines Arrays weisen Sie einfach einen Wert zu, indem Sie den Namen des Arrays, eine eckige Klammer (auf), den Index und eine eckige Klammer (zu) angeben und dann wie gewöhnlich einen Wert zuweisen. Syntax:

```
[Arraybezeichner][[Index]] = [Wert];
```

Beispiel:

```
test = new Array();
test[0] = 1;
test[1] = 10;
test[2] = 100;
test[3] = 42;
```

Ein Array kann Variablen verschiedener Datentypen aufnehmen (das können die meisten Programmiersprachen nicht). Sie können beispielsweise in einem Array die Daten von einer Person aufnehmen, die teilweise aus Zahlen bestehen, teilweise aus booleschen Werten und aus Text.

Beispiel 1:

Erstellen Sie die folgende Datei (Array1.htm):

```
<HTML>
<SCRIPT language="JavaScript">
function meinArray()
{
var adresse = new Array();
adresse[0] = "Hauptstrasse";
adresse[1] = 42;
adresse[2] = 12345;
adresse[3] = "Hinterdemonde";
adresse[4] = "00612345678";
adresse[5] = true;
adresse[6] = "Wüst";
adresse[7] = "Willi";
document.write("Mein Name ist " + adresse[7] +
" " + adresse[6] + ".");
document.write("<BR>");
document.write("Ich wohne in der ",adresse[0],
" ", adresse[1]);
document.write("<BR>");
document.write(adresse[2]," ", adresse[3]);
document.write("<BR>");
document.write("Meine Telefonnummer: ", adresse[4]);
document.write("<BR>");
document.write("Fax vorhanden? ", adresse[5]);
document.write("<BR>");
}
</SCRIPT>
<BODY onLoad="meinArray()">
</BODY>
</HTML>
```

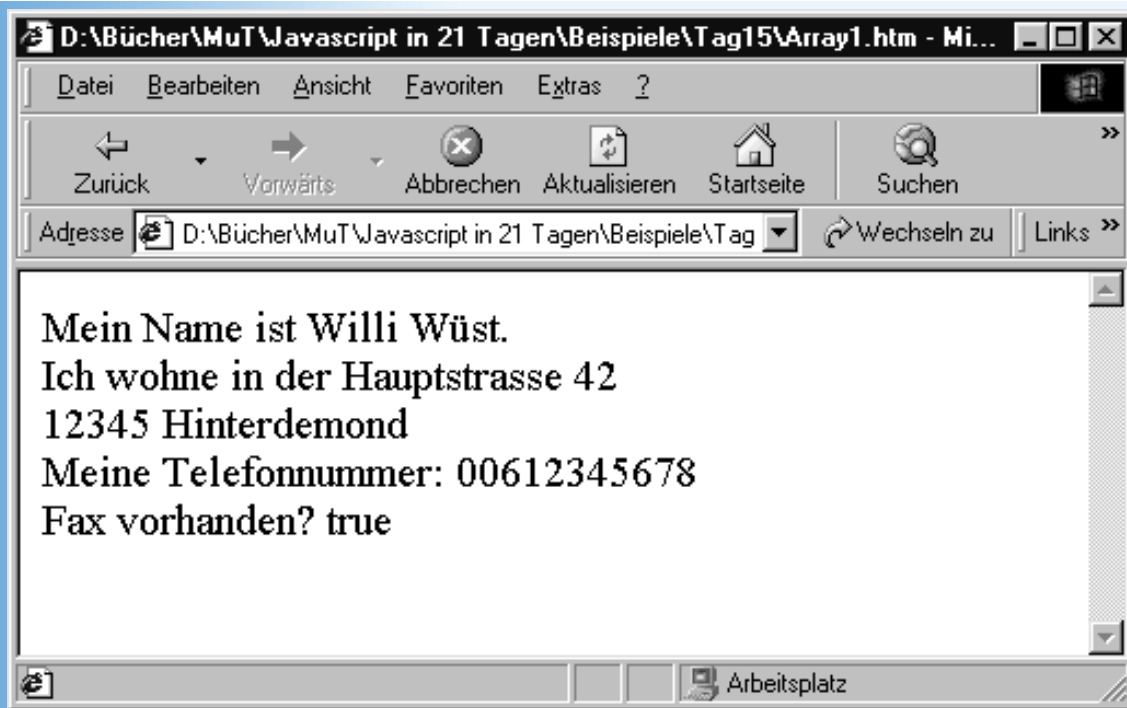


Abbildung 15.1: Es lassen sich mit Arrays verschiedene Typen von Werten speichern



Bei der Deklaration des Arrays wird in dem Beispiel und auch bei der Einführung der Syntax var vorangestellt. JavaScript ist da bei Variablendeklarationen normalerweise sehr lässig. Das soll bedeuten, die Deklaration könnte normalerweise auch so funktionieren:

```
adresse = new Array();
```

oder allgemein:

```
[Arraybezeichner] = new Array();
```

Der Internet Explorer wird auch damit zurechtkommen, aber der Netscape Navigator erkennt dann beim Zugriff auf die Array-Elemente unter Umständen den Array-Bezeichner nicht richtig. Nutzen Sie zur Sicherheit auf jeden Fall die vollständige Syntax.

Ein ziemlich wichtiger Grund für den Einsatz von Arrays ist der Zugriff mit Schleifen. Testen wir die Verwendung von Arrays in Verbindung mit der for-Schleife.

Beispiel 2:

Zuerst wird ein leeres Array erzeugt (Test der Größe mit der Eigenschaft length), das dann mit 20 Elementen über die for-Schleife gefüllt wird (beachten Sie den Indexbeginn 0). Die zweite for-Schleife zählt die Elemente rückwärts durch und gibt deren Werte aus (Name der Datei Array2.htm):

```
<HTML>
<SCRIPT language="JavaScript">
function meinArray()
{
var meinArray = new Array();
var zaehler;
// Kontrolle der Arraygroesse
document.write("Die Groesse des Arrays: " +
meinArray.length);
document.write("<BR>");
for (zaehler = 0; zaehler<20;zaehler++)
{
meinArray[zaehler] = zaehler + 1;
}
// Kontrolle der Arraygroesse
```



```
document.write(
"Groesse des Arrays nach der Zuweisung: " +
meinArray.length);
document.write("<BR>");
// Ausgabe der Werte
for (zaehler = 19; zaehler>=0;zaehler--)
{
document.write(meinArray[zaehler]);
}
document.write("<BR>");
}
</SCRIPT>
<BODY onLoad="meinArray()">
</BODY>
</HTML>
```

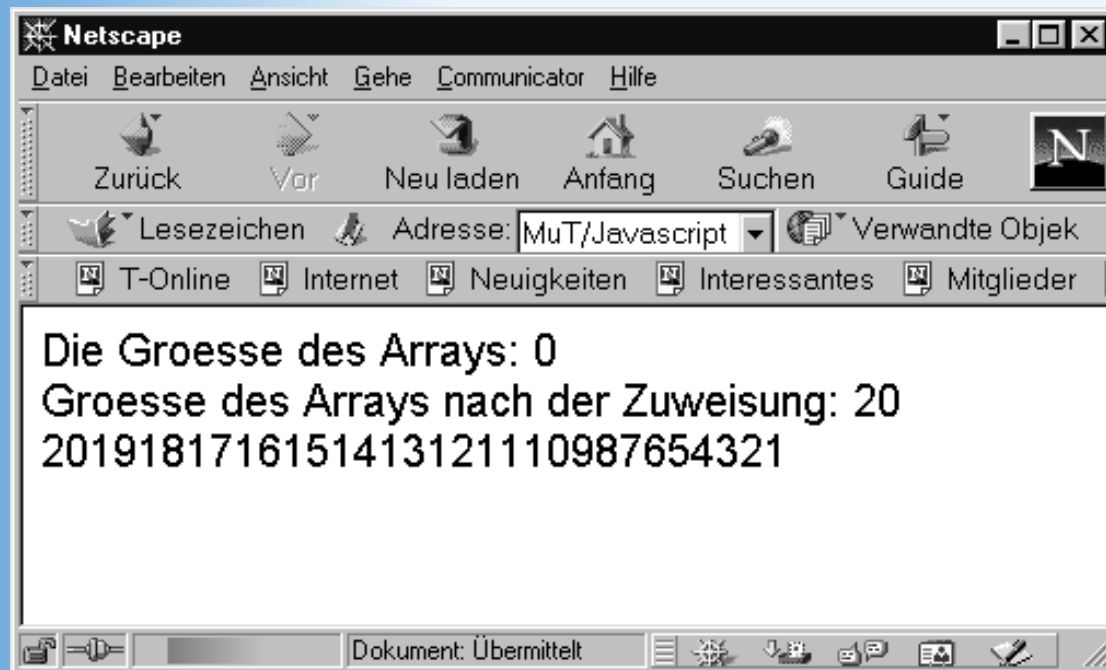


Abbildung 15.2: Nutzen des Arrays bei der Verwendung von Schleifen

Beispiel 3:

Wir wollen in diesem Zusammenhang eine interessante Variante der for- Schleife demonstrieren. Wenn man bei dem Beispiel die 2. for-Schleife durch die for...in-Schleife ersetzt, kann man ohne explizite Rücksicht auf die Größe des Arrays dies vollständig verwenden (diese Schleifenform durchläuft automatisch alle Einträge eines Arrays). Das ist dann sinnvoll, wenn ein Array während der Laufzeit dynamisch auf Grund nicht vorhersehbarer Einflüsse (etwa Benutzereingaben) erzeugt werden muss (dieses Mal aber von vorn nach hinten die Elemente durchlaufen). Der Name der Datei soll Array3.htm sein:

```
<HTML>
<SCRIPT language="JavaScript">
function meinArray()
{
var meinArray = new Array();
var zaehler;
document.write("Die Groesse des Arrays: " +
meinArray.length);
document.write("<BR>");
for (zaehler = 0; zaehler<30;zaehler++)
{
meinArray[zaehler] = zaehler + 1;
}
document.write(
```

```
"Groesse des Arrays nach der Zuweisung: " +  
meinArray.length);  
document.write("<BR>");  
// Ausgabe der Werte  
for (zaehler in meinArray)  
{  
document.write(meinArray[zaehler]);  
}  
document.write("<BR>");  
}  
</SCRIPT>  
<BODY onLoad="meinArray()">  
</BODY>  
</HTML>
```

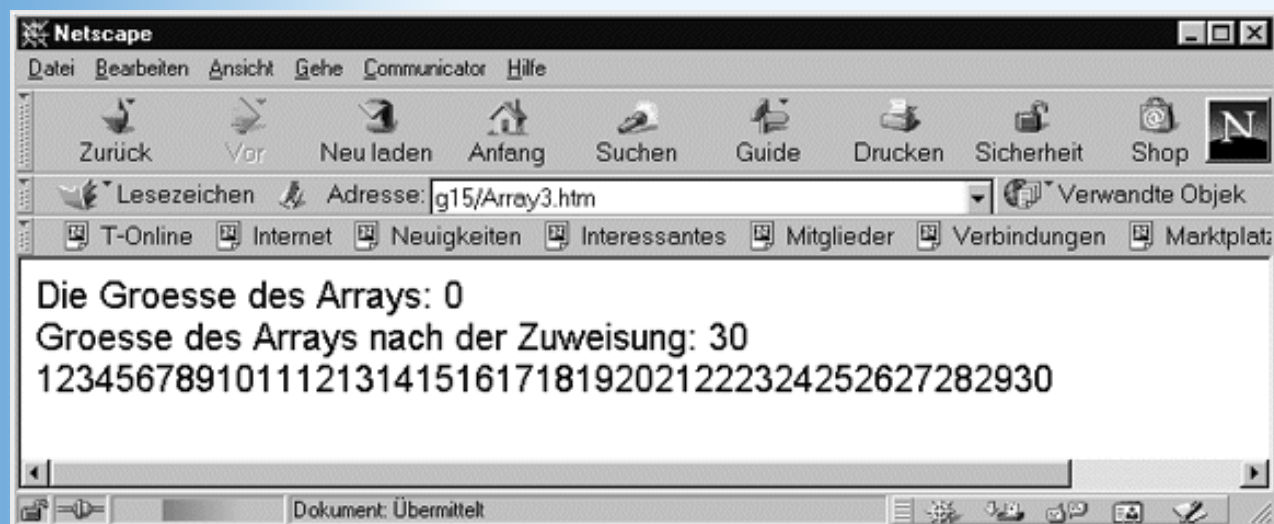


Abbildung 15.3: Verwendung von for...in

Die Objekthierarchie

Kommen wir nach dem Objektspezialfall Arrays wieder zu den Objekten allgemein zurück. Viele (aber nicht alle) der in JavaScript nutzbaren Objekte stehen in einer Objekthierarchiebeziehung zueinander. Dies bedeutet, ein Objekt ist in einem anderen Objekt als Eigenschaft enthalten. Dies ist eine ziemlich logische Abbildung der Realität. So wie eine Webseite im Browserfenster enthalten ist, müssen die die Realität beschreibenden Objekte dieser Tatsache Rechnung tragen. Oder das schon mehrfach behandelte Thema Formular - in einem Formular sind die einzelnen Elemente des Formulars enthalten. In der Objektabbildung heißt dies, dass das document in window enthalten ist und elements in einem Objekt in forms enthalten sein muss.

Ein Unterobjekt erbt immer dessen nach außen bekannten Eigenschaften und Methoden und erweitert diese sinnvollerweise um irgendwelche zusätzlichen Funktionalitäten. Wenn ein solches in der Objekthierarchie tiefer angesiedeltes Objekt angesprochen werden soll, muss einfach dessen Elternobjekt über die Punktnotation vorangestellt werden (wie in unserem Bibliothek-Denkansatz).

Beispiel:

```
window.document
```

Das Objekt window ist in diesem Beispiel das Elternobjekt von document.

Allerdings sind nicht sämtliche der Objekte in einer einzigen Hierarchiebeziehung miteinander verbunden. Das wäre auch nicht sinnvoll, denn auch in der Realität gibt es unzählige unabhängige Situationen. Die beiden nachfolgenden Grafiken zeigen zwei wesentliche (aber nicht vollständige) Objekthierarchiebeziehungen der Objekte, die in JavaScript verwendet werden können.

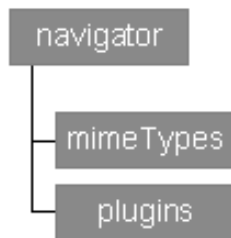


Abbildung 15.4: Die Objekthierarchie des navigator-Objekts

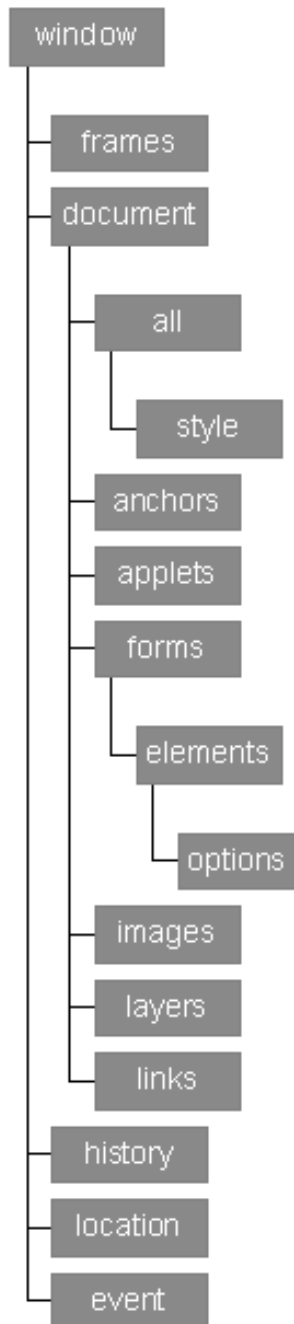


Abbildung 15.5: Ein Ausschnitt der Objekthierarchie des window-Objekts



Ein Unterobjekt wird in Bezug auf das Elternobjekt als Eigenschaft dieses Objekts betrachtet.

Alle diejenigen Objekte, welche Sie nicht in den beiden Grafiken finden, sind hierarchisch unabhängig.

Wenn Sie ein Unterobjekt von window ansprechen wollen, müssen Sie streng genommen mit window beginnen. Wenn Sie also das Objekt document ansprechen wollen, müssten Sie demnach

```
window.document
```

schreiben und entsprechend für eine Methode darin beispielsweise

```
window.document.write();
```

Da greift aber auch der Vergleich mit der Bücherbibliothek: Es kann darauf verzichtet werden, wenn es ohnehin klar ist, dass man ein spezifisches Objekt notieren müsste. Das ist ja auch die Regel, der wir uns bisher immer bedient haben. Ob es alert() oder prompt() als Methoden von window oder document.write() ohne vorangestelltes window war.

Zugriff auf Objekt-Eigenschaften und - Methoden

Wie wir wissen, besitzt jedes Objekt seine spezifischen Eigenschaften und objektgebundenen Methoden, auf die Sie innerhalb des JavaScript-Codes zugreifen können.

Eigenschaften

Zugreifen bedeutet im Fall von Eigenschaften entweder Lesen der Werte, aber in vielen Fällen ist auch eine Veränderung der Eigenschaftswerte möglich. In jedem Fall wird die Objekteigenschaft über ihren Namen, den Namen des Objekts und die Punktnotation angesprochen.

Wenn man Werte von Eigenschaften auslesen möchte, kann man die Objektnotation direkt an die Stelle im Script notieren, wo sonst ein Literal oder eine Variable steht. Das hatten wir schon in einigen Beispielen verwendet. Etwa an Tag 11 in Beispiel 9 die Zeile

```
window.status =  
document.forms[0].elements[0].value *  
document.forms[0].elements[1].value;
```

In diesem Fall haben wir die Werte der beiden Objekte direkt miteinander multipliziert. Oft wird aber der Wert einer Objekteigenschaft oder die Eigenschaft selbst einer Variablen zugewiesen. Dies erfolgt wie sonst auch über die Syntax
`var [Variable] = [Objekt].[Eigenschaft];`

Die Veränderung von Eigenschaften ist - falls das Objekt es zulässt - genauso möglich, wie Sie den Wert einer normalen Variablen ändern:

```
[Objekt].[Eigenschaft] = [Wert];
```

Methoden

Wie wir bereits wissen, zeichnet ein Objekt neben den spezifischen Eigenschaften aus, was seine zugeordneten Objektmethoden sind. Zwar muss nicht jedes Objekt Methoden besitzen, aber die meisten Objekte stellen welche zur Verfügung. Objektmethoden werden vollkommen analog den Eigenschaften über die Punktnotation angesprochen. Dabei geben Sie wieder zuerst den Namen des Objekts an und dahinter durch einen Punkt getrennt den Namen der Methode. Zusätzlich - und das ist wichtig - gehören immer eine öffnende und eine schließende Klammer zu einem Methodenaufruf. Dabei sind keine Leerzeichen zwischen den einzelnen Bestandteilen erlaubt!

Beispiel:

```
window.close();
```

Es ist nun auch so, dass man an Methoden Werte übergeben kann, die dann die Arbeit der Methode beeinflussen. Dies erfolgt wie bei normalen Funktionen über Parameter, die innerhalb des Klammerspaars stehen. Falls eine Methoden einen Rückgabewert liefert, kann er direkt verwendet werden, oder man speichert ihn in einer Variablen. Die direkte Verwendung erfolgt, indem die Methode dort notiert wird, wo sonst auch ein Ausdruck, ein Literal oder eine Variable stehen kann.

Die with()-Anweisung

In vielen Fälle ist es notwendig, mit ein und demselben Objekt mehrere Anweisungen in Folge auszuführen. Mit der Anweisung with() kann man sich dann Schreibaarbeit sparen. Über die Syntax

```
with([Objektname])
```

```
{  
... [mehrere Objektmethoden oder Eigenschaften] ...  
}
```

sparen Sie die mehrfache Notation des Objekts, denn im Inneren kann man dann auf die jeweils vorangestellte Notation des Objekts verzichten.

Beispiel:

```
with(document)  
{  
write("Hallo");  
write("<BR>");  
write("Welt");  
write("<BR>");  
}
```

Das Entstehen von Objekten

Wenn wir bisher von Objekten und deren Eigenschaften und Methoden gesprochen haben, haben wir uns noch gar keine genauen Gedanken darum gemacht, wie Objekte eigentlich entstehen. Damit Objekte entstehen, werden zwei Dinge benötigt:

1. Einen Bauplan für das Objekt und seine Eigenschaften. Das nennt man die Objektdeklaration bzw. Klasse.
2. Irgendetwas, das mit Hilfe dieses Bauplans ein konkretes Objekt erstellt. Dies »irgendetwas« bezeichnet man als Konstruktor oder Konstruktormethode. Dieser Konstruktor erzeugt aus der Objektdeklaration die konkrete Objektinstanz, die dann auch erst verwendet werden kann.

Alle JavaScript-Methoden eines spezifischen Objektes sind erst dann anwendbar, wenn Sie zuvor eine Objektinstanz von dem zugehörigen Objekt erzeugt haben oder - was der Regelfall ist - irgendwie im Hintergrund automatisch eine Instanz erzeugt wurde. Der letzte Fall passiert fast immer, wenn Sie eine dieser oben besprochenen vordefinierten Objekte mit ganz gewöhnlichen HTML-Tags innerhalb der Datei definieren. In wenigen Fällen (wenige Standardobjekte und selbstdefinierte Objekte) müssen Sie eine Objektinstanz mit Hilfe von JavaScript erzeugen.

Eine Objektinstanz explizit erstellen

Um aus einer Objektdeklaration explizit eine neue Objektinstanz und seine Eigenschaften anzulegen, verwendet man in der Regel das reservierte JavaScript-Schlüsselwort `new`, gefolgt von dem Bezeichner der Objektedeklaration und anschließend einem Klammerpaar (eventuell mit Parametern darin). Dies sieht von der Syntax her so aus:

```
new [Objektdeklaration]([optionale Parameter]);
```

Der Bezeichner der Objektdeklaration samt Klammernpaar und Inhalt ist das, was als Konstruktormethode bezeichnet wird. Eine besondere Methode, welche vom Bezeichner her immer identisch mit der Objektdeklaration ist und deren einzige Aufgabe die Erzeugung des Objekts ist. Dabei werden bei Bedarf Initialisierungen vorgenommen und sonstige notwendige Schritte ausgeführt und vor allen Dingen wird Speicherplatz für das Objekt reserviert.

Sofern ein Objekt seine Arbeit nicht dadurch vollständig erledigt hat, dass es erzeugt wurde und einige Arbeitsschritte durchgeführt hat, weist man es in der Regel einer Variablen zu. Über den Namen der Variablen kann dann bei Bedarf auf das Objekt zugegriffen werden. Die Zuweisung erfolgt meist in einem Schritt mit der Erzeugung. Das sieht dann meist so aus:

```
var [ObjektInstanz] =  
new [Objektdeklaration]([optionale Parameter]);
```

Spiele wir den Vorgang einmal an Hand konkreter Beispiele durch. Dabei werden wir die Objektdeklaration `Date` verwenden, welche mit der Konstruktormethode `Date()` das aktuelle Tagesdatum samt sekundengenauer Uhrzeit zurückgibt.

Beispiel 4:

Geben Sie nachfolgenden Quelltext ein (Name der Datei `Objekt1.htm`):

```
<HTML>  
<SCRIPT language="JavaScript">  
function obErzeug()  
{  
alert(new Date());  
}
```



```

}
</SCRIPT>
<BODY onLoad="obErzeug()">
</BODY>
</HTML>

```

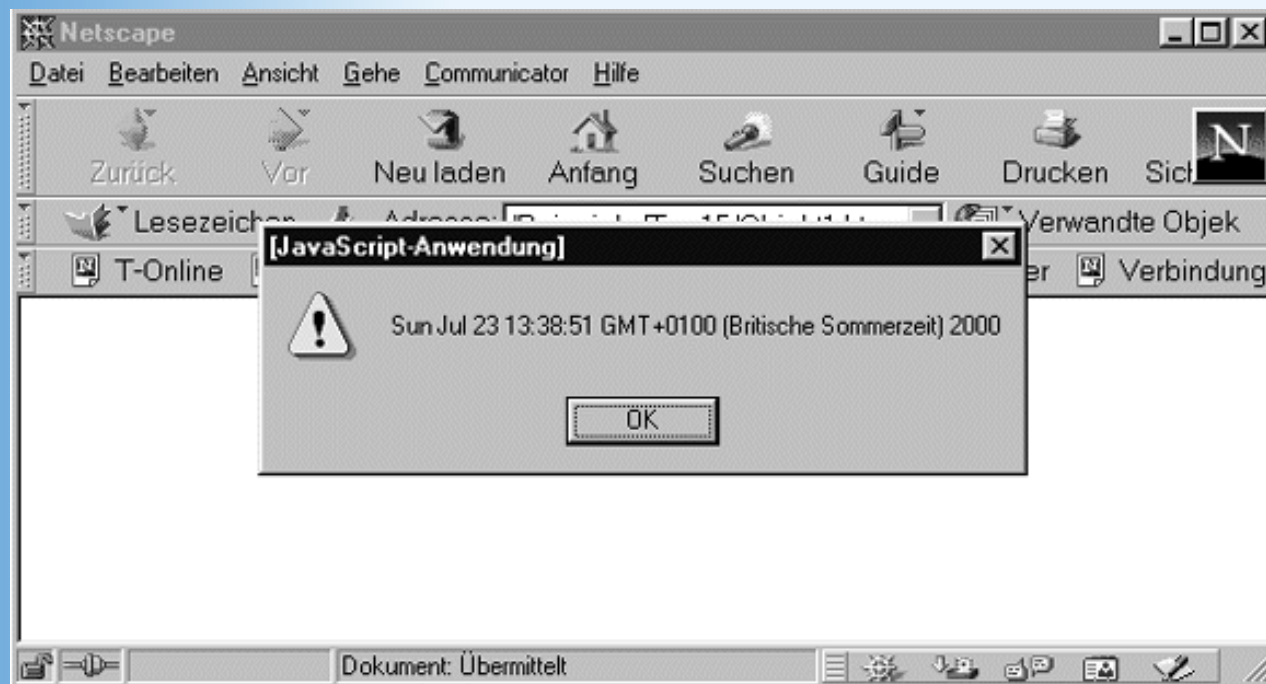


Abbildung 15.6: Direkte Verwendung eines Datumobjekts

In dem Beispiel haben wir das erzeugte Objekt direkt in der alert()- Methode verwendet und den Rückgabewert einfach ausgegeben. Sofern wir nicht nochmals auf irgendwelche Dinge aus dem Objekt zugreifen müssen, ist so eine Notation ausreichend.

Was ist aber, wenn wir erneut Informationen des Objekts benötigen? Etwa wenn das Objekt den genauen Zeitpunkt des Ladens der Webseite kennzeichnet und man an späterer Stelle berechnen möchte, wie lange die Seite angezeigt wurde. Eine erneute Erzeugung über Verwendung von new und dem Konstruktor ist keine Lösung, denn dann wird ja ein neuer Zeitpunkt abgefragt. Die Lösung kennen Sie aber schon - bei der Erzeugung wird das Objekt einer Variablen zugewiesen.

Das nachfolgende Beispiel macht genau dies unter Verwendung einiger Methoden von Date, die wir noch genauer bekommen. Im Wesentlichen werden zum Ladezeitpunkt der Seite und beim Verlassen der Seite die Stunden, Minuten und Sekunden jeweils aus dem Datumobjekt extrahiert. Beim Verlassen werden die jeweiligen Werte voneinander abgezogen. Dass wir diese Werte hier einfach anzeigen, ist natürlich nicht die typische Anwendung von dieser Technik. Sie kann aber vielfach sinnvoll verwendet werden. Denken Sie nur an die Auswertung, wie lange welche Webseiten in einem Webprojekt angesehen werden. Wenn man das via Cookies z.B. bei einem Anwender speichert oder gar in einer Datenbank auf Serverseite, kann man recht einfach genaue Benutzerprofile erstellen oder einfach nur interessante, gut gemachte Seiten von Ladenhütern trennen.

Beispiel 5:

Geben Sie nachfolgenden Quelltext ein (Name der Datei Objekt2.htm):

```

<HTML>
<SCRIPT language="JavaScript">
var Stunden, Minuten, Sekunden;
function ladeZeitPunkt()
{
    aktuelleZeit = new Date();
    Stunden = aktuelleZeit.getHours();
    Minuten = aktuelleZeit.getMinutes();
    Sekunden = aktuelleZeit.getSeconds();
    alert("Ladezeitpunkt der Seite: " +
    Stunden + ":" + Minuten + ":" + Sekunden);
}

```

```
function verlassZeitpunkt()  
{  
  verlassZeit = new Date();  
  var anzeigStd = verlassZeit.getHours() - Stunden;  
  var anzeigMin = verlassZeit.getMinutes() - Minuten;  
  var anzeigSek = verlassZeit.getSeconds() - Sekunden;  
  alert("Zeit, die Sie sich die Seite angesehen haben: "  
        + anzeigStd + " Stunden, " + anzeigMin +  
        " Minuten und " + anzeigSek + " Sekunden.");  
}  
</SCRIPT>  
<BODY onLoad="ladeZeitPunkt()"  
onUnLoad="verlassZeitpunkt()">  
</BODY>  
</HTML>
```

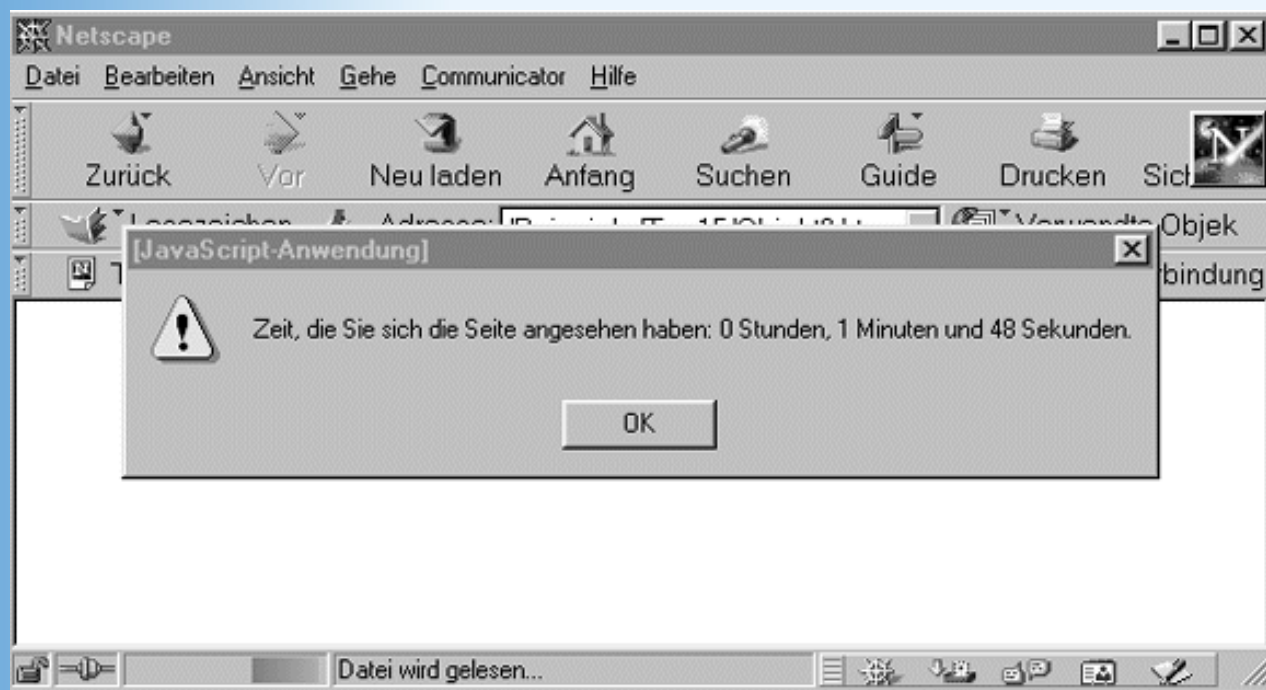


Abbildung 15.7: Die Dauer der Betrachtung der Webseite

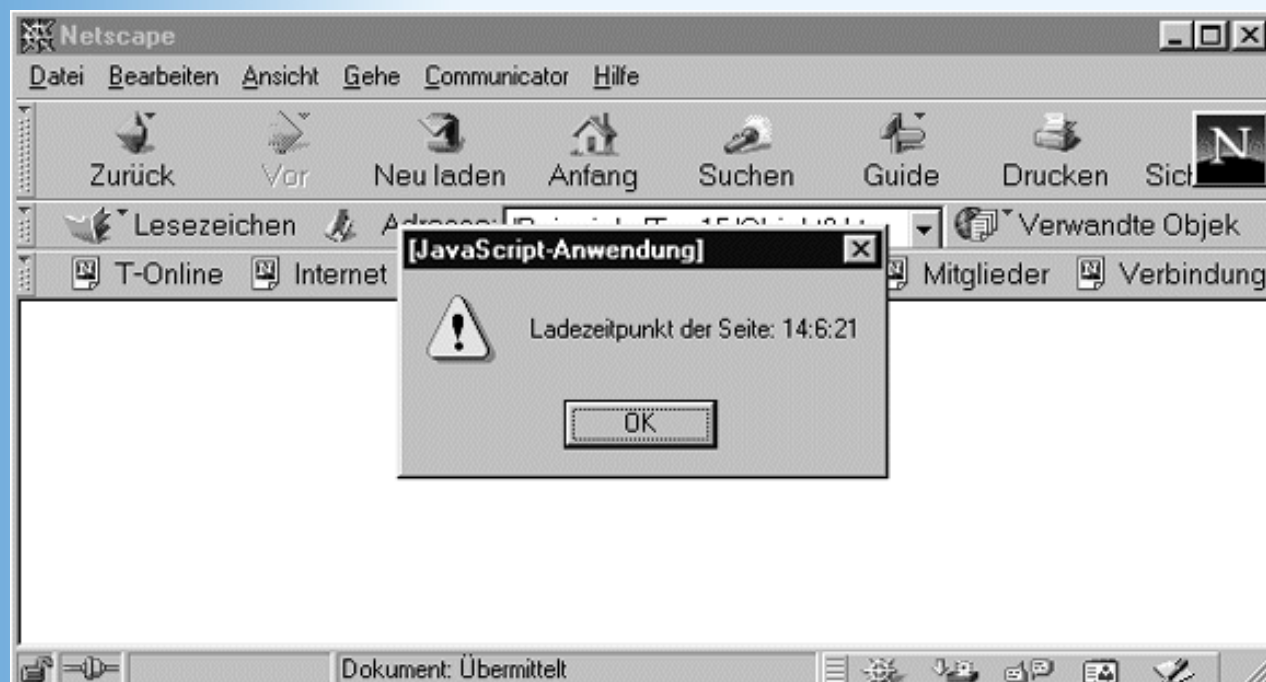


Abbildung 15.8: Die Uhrzeit des Ladens der Webseite

Das abschließende Beispiel zu diesem Thema soll zeigen, dass es zu einer Objektdекlaration durchaus mehr als eine Konstruktormethode geben kann. Date hat nämlich noch weitere Konstruktormethoden mit anderen Aufgaben, als das aktuelle Tagesdatum samt Uhrzeit aus dem Rechner des Webseitenbetrachters auszulesen. Man kann darüber auch ein Datumsobjekt mit vorgegebenem Datum erzeugen, wie wir es in dem nächsten Beispiel tun.

Beispiel 6:

Geben Sie nachfolgenden Quelltext ein (Name der Datei Objekt3.htm):

```
<HTML>
<SCRIPT language="JavaScript">
function obErzeug()
{
alert(new Date(2001,11,31));
}
</SCRIPT>
<BODY onLoad="obErzeug()">
</BODY>
</HTML>
```



Abbildung 15.9: Das vorgegebene Datum

Beachten Sie die Zeile

```
alert(new Date(2001,11,31));
```

Dort wird das Datum auf das Jahr 2001, Dezember, 31. gesetzt. Der Monat beginnt mit 0 für Januar.

Eigene Objekte und das Schlüsselwort this

Neben der Verwendung von den vordefinierten Standardobjekten für JavaScript steht es Ihnen frei, eigene Objekte zu definieren. Dies ist von grundsätzlicher Bedeutung, wenn Sie objektorientiert in JavaScript programmieren wollen.

Es stellt sich nun aber vielleicht die Frage, warum Sie das überhaupt tun sollen? Die Antwort ist nicht ganz einfach. Viele Argumente, die für die objektorientierte Programmierung im Allgemeinen sprechen, werden bei JavaScript nicht greifen. JavaScript ist nicht streng objektorientiert², sondern nur objektbasierend. Statt Objekte zu erzwingen, gibt es so gut wie immer einen Weg herum. Das soll bedeuten, die Erstellung von eigenen Objekten ist in JavaScript so gut wie nie zwingend und oftmals mühsamer als konventionelle Wege, die ohne Objekte auskommen. Damit das nicht so in der Luft stehen bleibt, wollen wir die Angelegenheit etwas umfangreicher erklären.

Was ist der Sinn eines Objekts? Ein Objekt fasst Eigenschaften und Funktionalitäten zusammen, die gemeinsam zur Beschreibung einer Situation sinnvoll oder notwendig sind. Nun, das kann man mit Funktionen auch, wenn man mit lokalen

Variablen für die inneren Abläufe und globalen Variablen als Analogon der Eigenschaften von Objekten arbeitet. Wenn eine Sprache keine globalen Variablen bereitstellt, kann man dies nicht so einfach simulieren, aber die Zusammenfassung von bestimmten Eigenschaften und Funktionalitäten als Argument für die zwingende Erstellung von eigenen Objekten ist bei hybriden Sprachen wie JavaScript ziemlich schwach.

Stärkstes Argument für die Verwendung von Objekten ist normalerweise die Vererbung. So etwas gibt es ohne Objekte³ nicht. Das bedeutet, wenn man ein Unterobjekt eines Objektes anlegt, besitzt dieses bereits dessen Methoden und Eigenschaften. Man muss das Unterobjekt nur noch um die Methoden und Eigenschaften spezialisieren, die es dann von dem Elternobjekt unterscheidet. Bevor Ihnen jetzt jedoch der Mund wässrig gemacht wird - JavaScript ist in Bezug auf Vererbung abgemagert worden. Sowohl Vererbung als auch viele andere wesentliche Charakteristika der objektorientierten Programmierung fehlen. JavaScript ist ja auch explizit nur objektbasierend und nicht objektorientiert.

Es gibt aber andere Argumente, die das Verwenden selbstdefinierter Objekte auch in JavaScript sinnvoll machen.

- Eine Objektdeklaration samt Konstruktormethode ist nicht unbedingt auf die Anwendung in der gleichen Sprache beschränkt. So einen Fall hatten wir gestern, als wir in JavaScript eine Java-Konstruktormethode aufgerufen haben (Tag 14), um damit ein Objekt zu erzeugen. Objekte sind also immer dann sinnvoll, wenn verschiedene Technikwelten zusammenspielen.
- Die Wartbarkeit von Programmstrukturen wird erhöht, wenn man konsequent (!) mit Objekten arbeitet. Konsequenz bedeutet insbesondere den Verzicht auf globale Variablen. Aber auch sonst lässt sich mit Objekten ziemlich effektiver und stabiler Code entwickeln.
- Statt allgemeine Funktionen zu definieren, die an jeder Stelle und zu jeder Gelegenheit aufgerufen werden können, kann man mit Methoden bestimmte Funktionalitäten an ein Objekt koppeln und damit verhindern, dass sie unsinnig angewandt werden. Etwa wenn man eine Funktion bellen() (mit genau der Funktionalität, die der Bezeichner aussagt) programmieren würde, könnte ein Objekt Fisch (ebenfalls mit genau der Funktionalität, die der Bezeichner aussagt) diese aufrufen. Wenn bellen() aber eine Methode wäre, die einem Objekt Hund (wieder sprechender Bezeichner) zugeordnet wäre, wäre diese missbräuchliche Anwendung schon gar nicht mehr möglich.

Um in JavaScript nun ein eigenes Objekt anzulegen, sind effektiv zwei Schritte nötig:

1. Eine Objektdeklaration als Bauplan für das Objekt und seine Eigenschaften muss erstellt werden (deklarieren), denn es gibt noch keine vordefinierte Schablone.
2. Mit Hilfe dieses Bauplans wird dann wie bei vordefinierten Objektdeklarationen eine konkrete Objektinstanz erstellt.

Um ein eigenes Objekt und seine Eigenschaften anzulegen, müssen Sie innerhalb JavaScript einfach eine spezielle Funktion definieren, die als Deklaration des Objekts, als seine Konstruktormethode, fungiert. Nachdem die Objektdeklaration angelegt ist, können Sie an anderen Stellen innerhalb Ihres JavaScripts Instanzen dieses Objekts definieren. Dies geschieht wieder mit Hilfe einer Variablen, aus der über Zuweisung des Objektes mit dem reservierten JavaScript-Schlüsselwort `new` eine Objektinstanz wird. Dies sieht also von der Syntax her schematisch so aus:

```
function [Funktionsname]()
{
  ... [Beschreibungen von Eigenschaften] ...
}
...
var [ObjektInstanz] = new [Funktionsname]();
```

In der Konstruktormethode ist ein JavaScript-Schlüsselwort von zentraler Bedeutung - das `this`-Schlüsselwort. Es ist ein wesentliches Element jeder objektorientierten Programmiersprache, denn darüber hat man Zugriff auf das Objekt, welches eine Funktionalität aufruft. Das nachfolgende Beispiel erzeugt eine Objektdeklaration `tier`, die als Objekt dann ein reales Tier soweit beschreiben soll, wie es notwendig ist (wir beschränken uns erst einmal auf drei Eigenschaften). Dabei charakterisieren diese gewissen Eigenschaften das konkrete Tier, aber in der Objektdeklaration ist ja noch nicht bekannt, wie das später daraus erzeugte Objekt heißen soll. Eine Objektdeklaration ist ja eine Bauvorschrift, aus der jeder, der darauf zugreifen kann, ein Objekt erzeugen kann. Wie soll man in der Bauvorschrift schon wissen, welchen Variablenbezeichner derjenige später nimmt? Er wird in der Deklaration durch einen Platzhalter für *diesen* Namen ersetzt. *Dieser* heißt ins Englische übersetzt *this*.

Beispiel 7:

Geben Sie nachfolgenden Quelltext ein (Name der Datei `string.htm`):

```
<HTML>
<SCRIPT language="JavaScript">
// Konstruktor
```



```
function tier(typ, fell, schwanz)
{
this.typ = typ;
this.fell = fell;
this.schwanz = schwanz;
}
//Objekterzeugung
var tier1 = new tier("Hund",true,true);
var tier2 = new tier("Vogel",false,true);
var tier3 = new tier("Fisch",false,true);

document.write("Das erste Tier ist ein ");
document.write(tier1.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier1.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier1.schwanz);
document.write("<P>");
document.write("Das zweite Tier ist ein ");
document.write(tier2.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier2.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier2.schwanz);
document.write("<P>");
document.write("Das dritte Tier ist ein ");
document.write(tier3.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier3.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier3.schwanz);
document.write("<P>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

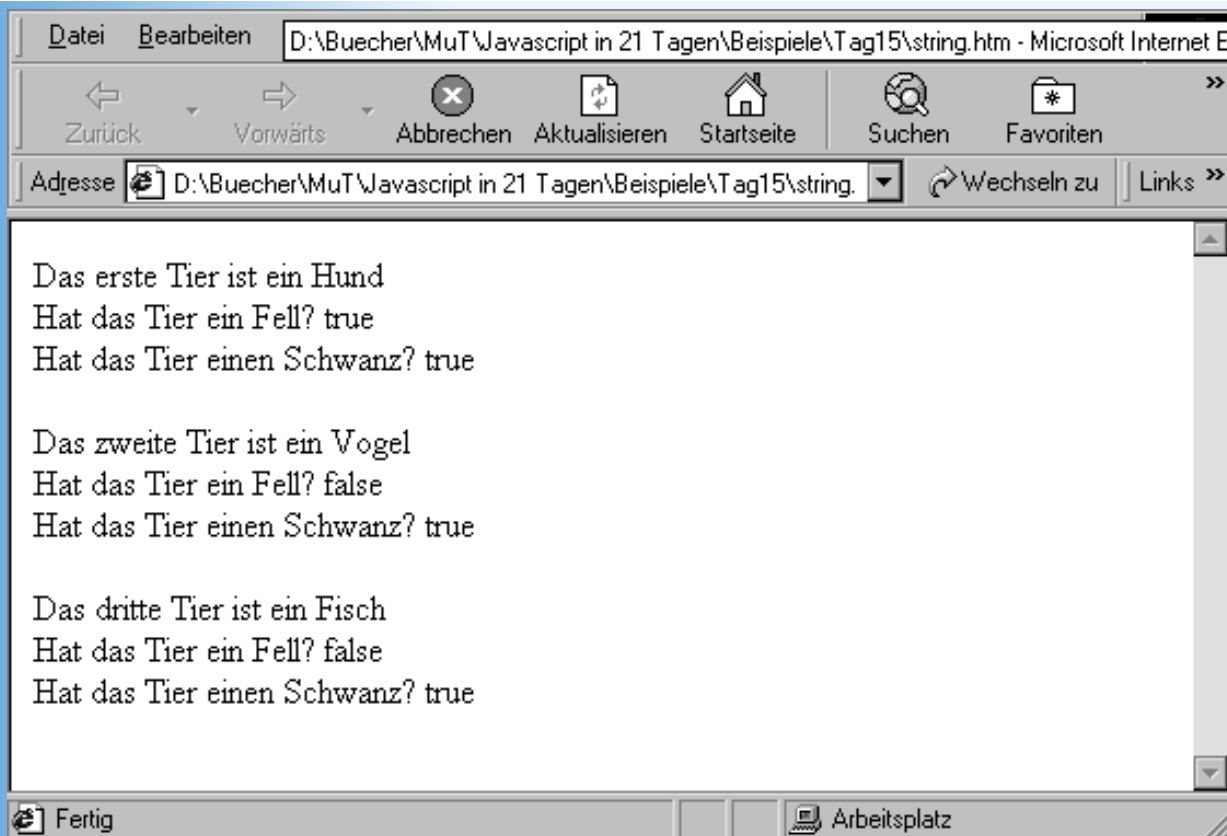



Abbildung 15.10: Drei Objekte aus einer Objektdeklaration

Die drei in dem Beispiel erzeugten Objekte haben alle andere Eigenschaften. Über den `this`-Stellvertreter der Objektdeklaration wurden bei der Erzeugung jedem konkreten Objekt die jeweiligen Eigenschaften zugewiesen.



Beachten Sie, dass wir an einer Stelle eine Methode `toString()` verwenden. Diese wandelt Objekthinhalte in Strings um. Da aber die Ausgabemethode das in unserer Situation auch automatisch macht, kann man auch darauf verzichten.

Ein Objekt kann nun neben Eigenschaften auch Methoden besitzen. Diese werden in JavaScript wie normale Funktionen definiert, nur verwenden auch sie in der Regel das Schlüsselwort `this`, um bei einem Zugriff auf Werte des aufrufenden Objektes auch auf das konkrete Objekt zu verweisen. Die Deklaration einer Methode erfolgt außerhalb der Konstruktormethode wie eine gewöhnliche Funktion. Sie muss allerdings danach noch mit der Konstruktormethode verbunden werden. Verankert wird sie innerhalb der Objektdeklaration über die Syntax

```
this.[Methodenname] = [Methodenname];
```



Beachten Sie, dass hinter dem Methodennamen keine Klammern stehen!

Schreiben wir das letzte Beispiel so um, dass unser Objekt ebenfalls eine Methode bereitstellt. Je nachdem, wie eine - noch neu hingenommene - Eigenschaft gesetzt ist, soll beim Aufruf der Methode über ein konkretes Objekt ein Mitteilungsfenster angezeigt werden oder nicht.

Beispiel 8:

Geben Sie nachfolgenden Quelltext ein (Name der Datei `eigOb2.htm`):

```
<HTML>
<SCRIPT language="JavaScript">
// Die Methodendeklaration
function lautgeben()
{
```

```
if(this.laut==true) alert("Wuff Chipp Miau");
}
function tier(typ, fell, schwanz,laut)
{
this.typ = typ;
this.fell = fell;
this.schwanz = schwanz;
this.laut=laut;
// Die Implementierung der Methode im Objekt
this.lautgeben = lautgeben;
}

var tier1 = new tier("Hund",true,true,true);
var tier2 = new tier("Vogel",false,true,true);
var tier3 = new tier("Fisch",false,true,false);
tier1.lautgeben();
document.write("Das erste Tier ist ein ");
document.write(tier1.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier1.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier1.schwanz);
document.write("<P>");
tier2.lautgeben();
document.write("Das zweite Tier ist ein ");
document.write(tier2.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier2.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier2.schwanz);
document.write("<P>");
tier3.lautgeben();
document.write("Das dritte Tier ist ein ");
document.write(tier3.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier3.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier3.schwanz);
document.write("<P>");
</SCRIPT>
<BODY></BODY>
</HTML>
```

Wir erweitern das Exempel noch ein wenig. Zum einen schreiben wir unsere Methode um, aber vor allem kommt in der neuen Erweiterung eine weitere Konstruktormethode `besitzer(name,alter)` ins Spiel. Diese erzeugt bei Anwendung ein neues Objekt, das den Besitzer des jeweiligen Tiers repräsentiert (eigentlich nur eine Person, aber wir wenden es als Besitzer eines Tiers an). Wir erweitern in dem folgenden Beispiel in einem weiteren Schritt die Konstruktormethode zum Erzeugen des Tierobjekts um den Besitzer. Beachten Sie dabei, dass der Besitzer als Objekt an die Konstruktormethode übergeben wird (als eine Variable, welche das Objekt mit zwei Eigenschaften enthält)! Entsprechend erhält man dann die Eigenschaften des Besitzers über das Tierobjekt und den Bezeichner des übergebenen Objekts. Also über eine doppelte Punktnotation. Wir haben hier den Fall der Verwendung eines Unterobjekts.



Abbildung 15.11: Aufruf der selbstdefinierten Objektmethode

Beispiel 9:

Geben Sie nachfolgenden Quelltext ein (Name der Datei eigOb3.htm):

```
<HTML>
<SCRIPT language="JavaScript">
// Die Methodendeklarationen
function besitzer(name,alter)
{
this.name=name;
this.alter=alter;
}
function lautgeben()
{
document.write("<br><i>Methodenaufruf</i><br>");
if(this.laut==true) document.write("Wuff Chipp Miau")
else document.write("Das Schweigen der Lärmer.");
}
function tier(typ, fell, schwanz, laut,eigentumVon)
{
this.typ = typ;
this.fell = fell;
this.schwanz = schwanz;
this.laut=laut;
this.eigentumVon=eigentumVon;
```

```
// Die Implementierung der Methode im Objekt
this.lautgeben = lautgeben;
}
// Die neuen Besitzerobjekte
var hans = new besitzer("Hans Dampf",86);
var willi = new besitzer("Willi Wüst",12);
// Die Tierobjekte mit den Besitzerobjekten als letzten Übergabewert
var tier1 = new tier("Hund",true,true,true,hans);
var tier2 = new tier("Vogel",false,true,true,willi);
var tier3 = new tier("Fisch",false,true,false,hans);
document.write("Das erste Tier ist ein ");
document.write(tier1.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier1.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier1.schwanz);
document.write("<br>");
// Zugriff auf die Unterobjekte
document.write("Das Tier gehört ");
document.write(tier1.eigentuMvon.name);
document.write(". Alter: ");
document.write(tier1.eigentuMvon.alter);
tier1.lautgeben();
document.write("<P>");
document.write("Das zweite Tier ist ein ");
document.write(tier2.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier2.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier2.schwanz);
document.write("<br>");
document.write("Das Tier gehört ");
document.write(tier2.eigentuMvon.name);
document.write(". Alter: ");
document.write(tier2.eigentuMvon.alter);
tier2.lautgeben();
document.write("<P>");
document.write("Das dritte Tier ist ein ");
document.write(tier3.typ.toString());
document.write("<br>");
document.write("Hat das Tier ein Fell? ");
document.write(tier3.fell);
document.write("<br>");
document.write("Hat das Tier einen Schwanz? ");
document.write(tier3.schwanz);
document.write("<br>");
document.write("Das Tier gehört ");
document.write(tier3.eigentuMvon.name);
document.write(". Alter: ");
document.write(tier3.eigentuMvon.alter);
tier3.lautgeben();
document.write("<P>");
</SCRIPT>
<BODY></BODY>
</HTML>
```

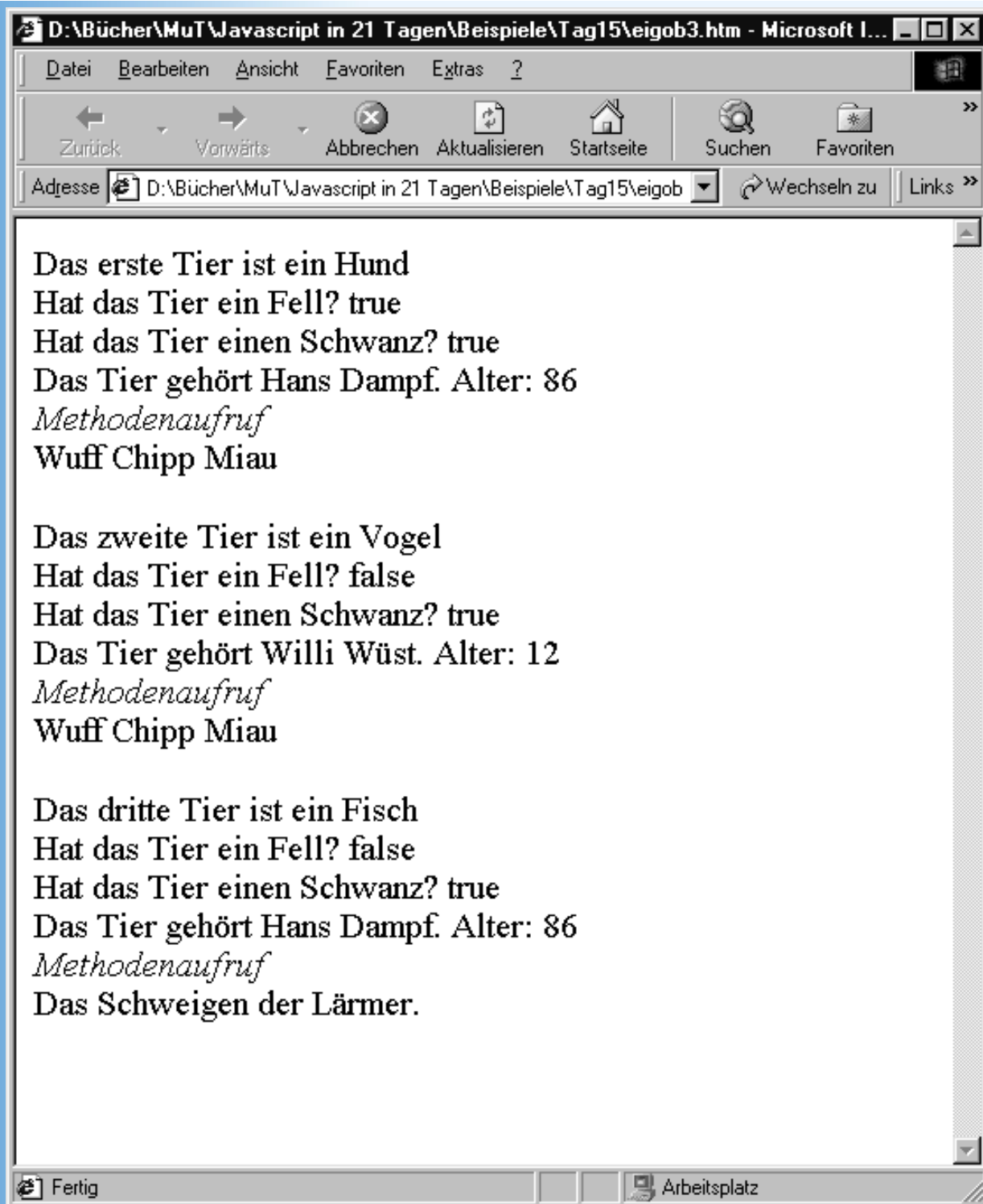


Abbildung 15.12: Zwei Objekte und eine Methode

Prototype

JavaScript bietet in neuen Versionen die Möglichkeit, nachdem ein Objekt bereits mit `new` definiert wurde, es nachträglich um neue Eigenschaften und Methoden zu erweitern. Das JavaScript-Schlüsselwort für diesen Zweck heisst `prototype`. Allgemeine Syntax dazu ist die folgende:

```
[Objekttyp].prototype.[Eigenschaft] = [Wert];
```

bzw.

```
[Objekttyp].prototype.[Methode] = [Methode];
```

Sobald diese Aktion durchgeführt wurde, besitzen alle zukünftigen, aber auch alle vorherigen Objekte die neue Eigenschaft bzw. Methode. Insbesondere ist zu beachten, dass nach der `prototype`-Anweisung erzeugte Objekte für die zugefügte Eigenschaft immer den über `prototype` angegebenen Wert besitzen. Ein Erweitern der Konstruktormethode um die Eigenschaft wird zwar keinen Laufzeitfehler erzeugen, und auch die Eigenschaft wird bereitgestellt (allerdings wegen

prototype), aber der Wert ist der, der bei der prototype-Zuweisung vergeben wurde. Er muss explizit mit einer Zuweisung verändert werden. Das nachfolgende Beispiel zeigt dies.

Beispiel 10:

Erstellen wir ein Beispiel, das mit der prototype-Technik arbeitet. Geben Sie nachfolgenden Quelltext ein (Name der Datei eigOb4.htm):

```
<HTML>
<SCRIPT language="JavaScript">
function tier(typ, fell, schwanz)
{
this.typ = typ;
this.fell = fell;
this.schwanz = schwanz;
}
var tier1 = new tier("Hund",true,true);
document.write("Eigenschaften von dem Objekt: ");
document.write(tier1.typ.toString());
document.write("<br>");
document.write(tier1.fell);
document.write("<br>");
document.write(tier1.schwanz);
document.write("<P>");
tier.prototype.besitzer = "Hans Dampf";
document.write("Besitzer ist ");
document.write(tier1.besitzer);
document.write("<P>");
var tier2 = new tier("Katze",true,true,"Willi Wüst");
document.write("Besitzer Tier 2 ? ");
document.write(tier2.besitzer);
document.write("<P>");
document.write("Besitzer Tier 2 ? ");
tier2.besitzer = "Willi Wüst";
document.write(tier2.besitzer);
document.write("<P>");
document.write("Besitzer Tier 1 ? ");
document.write(tier1.besitzer);
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

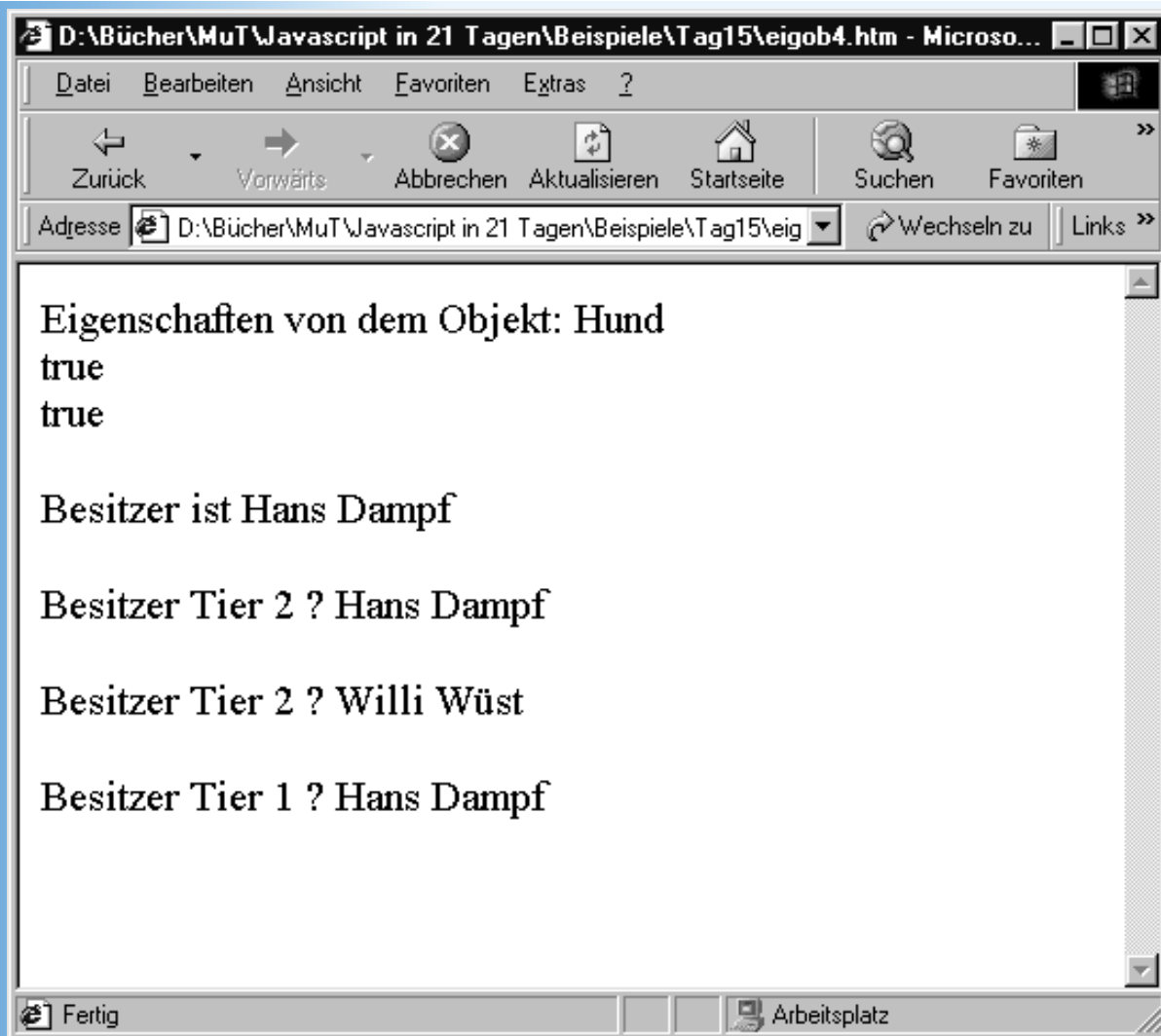


Abbildung 15.13: Veränderung mit prototype

Zusammenfassung

Heute haben wir uns das erste Mal bewusst JavaScript unter dem Aspekt von Objekten gewidmet. Da JavaScript keine streng objektorientierte Sprache ist (sondern nur objektbasierend), gibt es dort nicht nur Objekte wie beispielsweise in Java, sondern globale Variablen, Funktionen und all die prozeduralen Elemente, die wir die letzten zwei Wochen kennengelernt haben. Objekte sind aber die zentralen Elemente in der JavaScript- Programmierung, worüber erst JavaScripte ihre volle Bedeutung und Leistung entfalten können.

Ein wichtiger Spezialfall von Objekten sind Arrays - eine Sammlung von logisch zusammengehörenden Variablen und Objekten. Allgemein sind Objekte grundsätzlich zusammengehörende Anweisungen und Daten, die eine in sich abgeschlossene und eigenständige Einheit bilden und als Softwaremodelle Dinge aus der realen Welt mit all ihren für die Behandlung eines Problems notwendigen Eigenschaften und Verhaltensweise beschreiben sollen.

Die Eigenschaften eines Objekts sind die Details, die es charakterisieren und von anderen Objekten unterscheidet. Sie können aus einem Script heraus gelesen und manchmal auch gesetzt werden. Methoden sind die aktiven Algorithmen, die von einem Objekt bereitgestellt werden, damit bei einem Aufruf bestimmte Dinge getan werden. Methoden sind so etwas wie einem Objekt fest zugeordnete Funktionen, die auch nur über das Objekt verwendet werden können.

Die Verwendung von Objektmethoden und Eigenschaften erfolgt grundsätzlich über die DOT-Notation, wo links das Objekt, dann ein Punkt und dann die Eigenschaft oder Methode notiert wird. Über JavaScript kann man auf zahlreiche vordefinierte Objekte zugreifen, die in Form einer Objekt-Bibliothek bereitgestellt werden (wir werden sie intensiv die nächsten Tage behandeln). Viele (aber nicht alle) der in JavaScript nutzbaren Objekte stehen in einer Objekthierarchiebeziehung zueinander. Dies bedeutet, ein Objekt ist in einem anderen Objekt als Eigenschaft enthalten.

Objekte werden durch eine Objektdeklaration beschrieben und durch die Anwendung eines Konstruktors daraus explizit erzeugt. Dies kann automatisch erfolgen, aber auch explizit durch den Programmierer. Um aus einer Objektdeklaration explizit eine neue Objektinstanz und seine Eigenschaften anzulegen, verwendet man das reservierte JavaScript-

Schlüsselwort `new`, gefolgt von dem Bezeichner der Objektdeklaration und anschließend einem Klammerpaar (eventuell mit Parametern darin). In der Regel weist man das erzeugte Objekt einer Variablen zu, über deren Namen man dann bei Bedarf auf das Objekt zugreifen kann. Die Zuweisung erfolgt meist in einem Schritt mit der Erzeugung. Das sieht dann meist so aus:

```
var [ObjektInstanz] =  
new [Objektdeklaration]([optionale Parameter]);
```

Neben der Verwendung von den vordefinierten Standardobjekten für JavaScript kann man auch eigene Objekte definieren. Um in JavaScript nun ein eigenes Objekt anzulegen, sind effektiv zwei Schritte nötig.

1. Eine Objektdeklaration als Bauplan für das Objekt und seine Eigenschaften muss erstellt werden (deklarieren), denn es gibt noch keine vordefinierte Schablone. Schlüssel zu diesem Konstruktor ist das Schlüsselwort `this`, um darüber den Zugriff auf das später daraus erzeugte Objekt zu ermöglichen.
2. Mit Hilfe dieses Bauplans wird dann wie bei vordefinierten Objektdeklarationen eine konkrete Objektinstanz erstellt. Dies geschieht wieder mit Hilfe einer Variablen, aus der über Zuweisung des Objektes mit dem reservierte JavaScript-Schlüsselwort `new` eine Objektinstanz wird. Dies sieht also von der Syntax her schematisch so aus:

```
function [Funktionsname]()  
{  
    ... [Beschreibungen von Eigenschaften] ...  
}  
...  
var [ObjektInstanz] = new [Funktionsname]();
```

Fragen und Antworten

Frage:

Sind die in JavaScript bereitgestellten Objekte wie `window` oder `document` mit JavaScript erstellt worden?

Antwort:

Nein. Diese Objekte liegen nicht in JavaScript-Quelltext vor. Sie würden sie ja sonst auch als Scriptcode auf Ihrem Rechner finden. Außerdem ist dort der Mechanismus der Vererbung aktiv. Das kann mit JavaScript nicht geleistet werden.

Frage:

Verwenden viele JavaScripte selbstdefinierte Objekte?

Antwort:

Nein. Die meisten Scripte in Webseiten sind so einfacher Natur, dass man ohne selbstdefinierte Objekte auskommt. Man sollte immer im Hinterkopf behalten, wo der Einsatzzweck von Webscripten liegt. Selbstdefinierte Objekte fangen an, in Revieren zu wildern, die mächtigeren Sprachen wie Java vorbehalten sein sollten.

Frage:

JavaScript reserviert Schlüsselworte wie `class` oder `super`. Haben die nicht etwas mit Objektorientierung zu tun?

Antwort:

In der Tat. Sie sind aber, wie einigen weitere, nur reserviert und werden in JavaScript (noch) nicht eingesetzt.

Workshop

Betrachten Sie die Beispiele der vergangenen Tage in Hinsicht auf Objektnotationen. Kontrollieren Sie, welche Objekte und welche zugehörigen Methoden und Eigenschaften wir bereits verwendet haben.

Erstellen Sie eigene Objekte mit Eigenschaften und Methoden.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was macht `this`?

2. Ist das korrekt?

```
var test = New Date();
```

3. Stimmt das?

```
window.status =  
document.forms[0].elements.value *  
document.forms[0].elements.value;
```

4. Stimmt der Methodenaufruf?

```
window.close;
```

5. Stimmt das?

```
with(document)  
{  
write("Mircoweich");  
write("<BR>");  
}
```

6. Was ist ein Objekt?

7. Was ist ein Konstruktor?

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

1
Was ein Zufall ;-)

2
Nicht einmal sanft objektorientiert ;-)

3
Oder genauer - Klassen.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 3

Tag 16

Frames und das window-Objekt

Die an diesem Tag durchzuarbeitende Lektion wird ziemlich umfangreich. Wir haben einen der zentralen Punkte für den effektiven Einsatz von JavaScript vor uns. Am heutigen Tag wollen wir uns dem window-Objekt und einem seiner wichtigsten und eng verwandten Unterobjekte, frame, intensiver widmen. Das Objekt window ist auf der obersten Objektebene in der Objekthierarchie angesiedelt und beinhaltet alle diejenigen Objekte als Unterobjekte, die mit der Anzeige von Informationen im Browser irgendetwas zu tun haben.

Über das Objekt frame haben Sie Kontrolle über Frames, einer Technik zur Aufteilung des Anzeigebereichs eines Browsers in einzelne Segmente, die unabhängig von einander mit Inhalt gefüllt werden können. Dazu müssen wir Frames erst einmal über ihre HTML-Technik einführen, um uns dann mit dem JavaScript-Zugriff darauf zu beschäftigen.

Das window-Objekt

Das Objekt window beinhaltet wie gesagt alle diejenigen Objekte als Unterobjekte, die mit der Anzeige von Informationen im Browser irgendetwas zu tun haben (siehe dazu Tag 15). Insbesondere steht das immer vorhandene Hauptanzeigefenster eines Browsers darüber zur Verfügung. Sie können aber auch zusätzliche Instanzen des Objekts window erzeugen. Dies erfolgt im Allgemeinen über die window-Methode open(). Dabei wird das neue Fenster als Client des aufrufenden Fensters direkt geöffnet. Es gilt folgende Syntax:

```
var [Fenstervariable] =  
window.open("[URL]", "[Fenstername]", "[Optionen]")
```

- URL ist eine beliebige Datei, die in dem Fenster angezeigt werden soll. Es kann eine HTML- oder Grafikdatei sein, aber auch jede andere Datei, welche der Browser darstellen kann. Dabei gelten die üblichen Regeln für Verweise. Die Adresse muss in Anführungszeichen stehen. Ein leeres Fenster wird erzeugt, wenn anstelle einer URL-Adresse einfach »« angegeben wird.
- Fenstername ist ein weitgehend frei wählbarer Variablenbezeichner, der den üblichen Regeln für Variablennamen genügen muss und nicht mit einigen reservierten Token (window, self, parent, top) übereinstimmen darf, die eine feste Bedeutung haben. Der Name des Fensters muss in Anführungszeichen stehen.
- Die Angabe Optionen ist optional. Der gesamte »Optionen-Ausdruck« muss in Anführungszeichen stehen. Angaben zu einzelnen Optionen werden durch Kommata getrennt.

Beispiel:

```
open("Meinedatei.htm", "meinFenster");
```

Sinnvoll (aber nicht zwingend) ist es wie gesagt, bei der Erstellung einer Fensterobjektinstanz diese in einer Variablen zu speichern. Sie können dann an späterer Stelle noch darauf zugreifen (etwa zum Schließen des Fensters).

Beispiel:

```
var fenster1 = open("Meinedatei.htm", "meinFenster");
```

Das Schließen von Fensterinstanzen erfolgt über die Methode close(), welche das vorangestellte Fensterobjekt schließt. Das Fensterobjekt wird einfach mit dem Namen der Variablen angesprochen, in der die Fensterinstanz bei der Erstellung über die open()-Methode gespeichert wurde. Sie können sowohl von einem anderen Fenster aus ein fremdes Fenster schließen als auch das gerade angezeigte Fenster. Dies funktioniert aber nur innerhalb vorgegebener Randbedingungen.

Sie müssen eine Variable (sprich: Namen) zur Verfügung haben, die eine Objektreferenz des zu schließenden Fensters beinhaltet. Insbesondere ist dies für das Hauptfenster des Browsers erst Mal ein Problem. Es gibt aber die reservierten Fensternamen `self` oder `window`, die immer für das gerade aktive Fenster stehen. Damit kann man den Browser selbst schließen. Jedes aktive Fenster (auch das Hauptfenster) lässt sich über die Anweisungen

```
window.close();
```

oder

```
self.close();
```

schließen, wenn Sie innerhalb des Fensters ausgelöst wird. Probieren Sie einfach die nachfolgende Übung aus.

Beispiel 1:

Erstellen Sie bitte nachfolgende HTML-Datei (Name der Datei `window1.htm`):

```
<HTML>
<HTML>
<SCRIPT language="JavaScript">
var meinFenster;
function fenster_auf()
{
  meinFenster = open("bild1.htm","DiesistkeinleeresFenster");
}
function fenster_zu()
{
  meinFenster.close();
}
</SCRIPT>
<BODY>
<A href="JavaScript:fenster_auf()">
Neues Fenster auf
</A>
<BR>
<A href="JavaScript:fenster_zu()">
Neues Fenster zu
</A>
<BR>
<A href="JavaScript:self.close()">
Aktives Fenster zu
</A>
<BR>
</BODY>
</HTML>
```

Über drei Links rufen wir hier JavaScript auf. Einmal als Inline-Referenz. Diese schließt das aktive Fenster mit den Links. Bitte erstellen Sie die zweite Datei (`Bild1.htm`), welche im gleichen Verzeichnis abgespeichert werden muss und die in der neu erzeugten Objektinstanz angezeigt werden soll:

```
<HTML>
<BODY>
<IMG src="131.gif">
</BODY>
</HTML>
```

Laden Sie die Datei `window1.htm` und probieren Sie die Links aus.

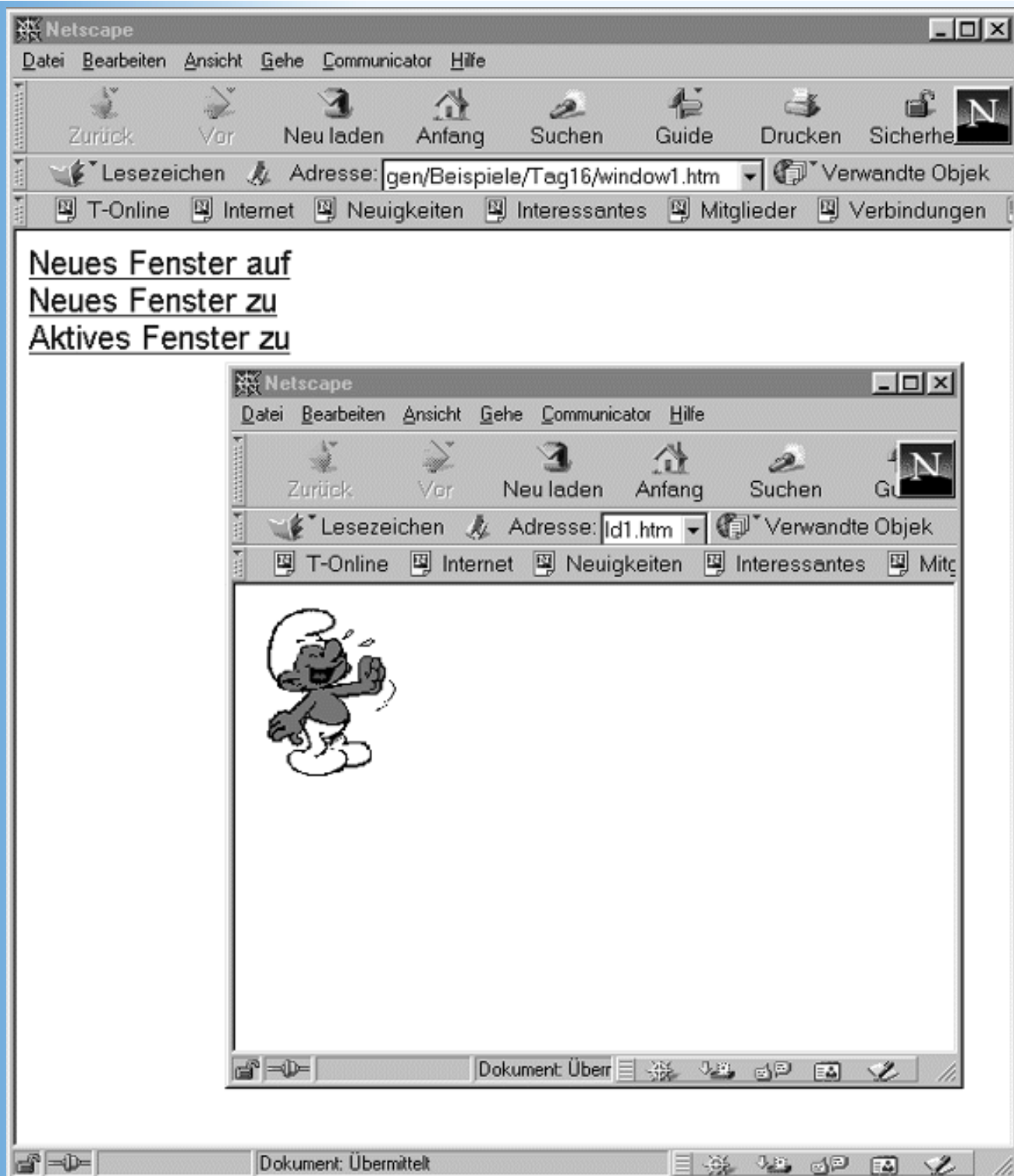


Abbildung 16.1: Das per JavaScript geöffnete zweite Fenster

Der Navigator wird übrigens auch das aktive Fenster ohne weiteres schließen, während der Internet Explorer rückfragt.



Abbildung 16.2: Warnung, dass das aktive Fenster geschlossen werden soll

Ihnen wird sicher auffallen, dass Folgefenster beim Öffnen in der gleichen Größe wie das aufrufende Fenster dargestellt werden und die Position (wenn kein Vollbild vorliegt) relativ zur dessen Position gesetzt wird. Dies kann natürlich

individuell angepasst werden, wozu im Wesentlichen einige der Eigenschaften der `open()`-Methode genutzt werden können (wobei gleich vorausgeschickt werden muss, dass dies von einigen Browsern leider nicht umgesetzt wird).

Es können sowohl Parameter beim Aufruf einer Fensterinstanz verwendet als auch bei bereits erzeugtem Fenster den Fenstereigenschaften neue Werte zugewiesen werden. Die erlaubten Optionen beim Öffnen sind folgende:

Option	Beschreibung
<code>width=(Pixel)</code>	Die Option erzwingt eine in Pixeln angegebene Fensterbreite.
<code>height=(Pixel)</code>	Die Option erzwingt eine in Pixeln angegebene Fensterhöhe.
<code>resizable=yes/no</code>	Die Option legt fest, ob die Größe fest oder veränderbar ist (Voreinstellung <code>yes</code>).
<code>scrollbars=yes/no</code> <code>toolbar=yes/no</code> <code>status=yes/no</code> <code>menubar=yes/no</code> <code>location=yes/no</code>	Die Optionen legen jeweils fest, ob das Fenster eine fenstereigene Bildlaufleiste, Buttonleiste, Statuszeile, Menüleiste und URL-Adresszeile hat. Voreinstellung ist immer <code>no</code> .
<code>directories=yes/no</code>	Die Option legt fest, ob das Fenster über fenstereigene Directory-Buttons (Netscape) verfügt.

Tabelle 16.1: Optionen beim Öffnen

Beispiel 2:

Erstellen Sie bitte nachfolgende HTML-Datei (Name der Datei `window2.htm`):

```
<HTML>
<HTML>
<SCRIPT language="JavaScript">
var meinFenster;
function fenster_auf()
{
    meinFenster =
    open(
        "bild1.htm",
        "DiesistkeinleeresFenster",
        "width=600, height=250, resizable=yes,
        directories=yes, menubar=yes, location=yes");
}
function fenster_zu()
{
    meinFenster.close();
}
</SCRIPT>
<BODY>
<A href="JavaScript:fenster_auf()">
Neues Fenster auf
</A>
<BR>
<A href="JavaScript:fenster_zu()">Neues Fenster zu</A>
<BR>
<A href="JavaScript:self.close()">
Aktives Fenster zu
</A>
<BR>
</BODY>
</HTML>
```

Laden Sie die Datei window2.htm in verschiedene Browser und probieren Sie die Links aus. Es wird auffallen, dass nicht alle Optionen in jedem Browser umgesetzt werden.

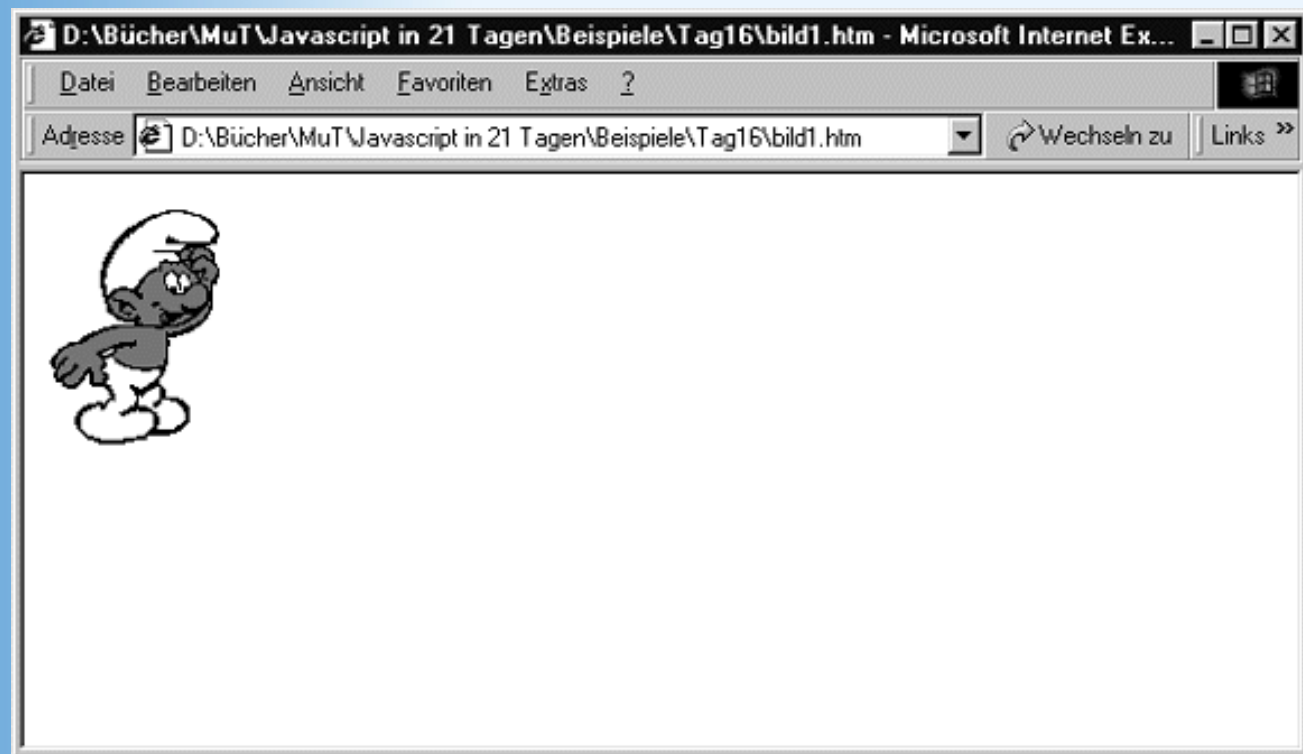


Abbildung 16.3: Der Internet Explorer zeigt auch eine Menüleiste an



Abbildung 16.4: Das per JavaScript geöffnete zweite Fenster im Navigator

In JavaScript 1.3 gibt es noch einige zusätzliche Optionen, die aber sehr inkonsistent von den Browsern behandelt werden. So kann über

`dependent=yes/no`

festgelegt werden, ob das Elternfenster geschlossen werden soll, wenn das Clientfenster geschlossen wird (yes). Über

`hotkeys=yes/no`

lassen sich die Browser-Hotkeys deaktivieren (no). Die Angaben

`innerHeight=(Pixel)`

und

`innerWidth=(Pixel)`

ergänzen die bisherigen Angaben zur Höhe und Breite, indem Sie explizit den Anzeigebereich des neuen Fensters festlegen (nicht die äußere Größe, welche den inneren Anzeigebereich auf Grund optionaler Statuszeile und Menüzeile nicht genau festlegt). Besonders wichtig ist jedoch, dass über

`screenX=(Pixel)`

und

`screenY=(Pixel)`

die linke obere Ecke des Browserfensters festgelegt werden kann.

Beispiel 3:

Erstellen Sie bitte nachfolgende HTML-Datei (Name der Datei `window3.htm`):

```
<HTML>
<HTML>
<SCRIPT language="JavaScript">
var meinFenster;
function fenster_auf1()
{
  meinFenster =
  open(
    "bild1.htm",
    "DiesistkeinleeresFenster",
    "width=300,height=250,screenX=100,screenY=100");
}
function fenster_auf2()
{
  meinFenster =
  open(
    "bild1.htm",
    "DiesistkeinleeresFenster",
    "width=300,height=250,screenX=200,screenY=300");
}
function fenster_zu()
{
  meinFenster.close();
}
</SCRIPT>
<BODY>
<A href="JavaScript:fenster_auf1()">
Neues Fenster auf 100, 100
</A>
<BR>
<A href="JavaScript:fenster_auf2()">
Neues Fenster auf 200, 300
</A>
<BR>
<A href="JavaScript:fenster_zu()">
Neues Fenster zu
</A>
<BR>
</BODY>
</HTML>
```

Laden Sie die Datei `window3.htm` in verschiedene Browser und probieren Sie die Links aus. Der Navigator kann mit den Positionierungen gut umgehen (wenn das Folgefenster schon offen ist, wird es bei einem Klick auf die alternative Positionierung einfach dorthin verschoben - probieren Sie es aus). Der Internet Explorer wird die Positionierung nicht sauber ausführen.

Die Eigenschaften von window

Die Eigenschaften von dem Objekt window stehen immer dann zur Verfügung, wenn zuvor eine Objektinstanz von window erzeugt wurde oder ein bereits vorhandenes Fenster über eine der gültigen Fensterreferenzen angesprochen wird. Hier eine Auswahl der wichtigsten Eigenschaften samt Beschreibung:

Eigenschaft	Beschreibung
closed	Wenn ein Fenster geöffnet und einer Variablen zugewiesen wurde, kann man über diese, das Objekt bezeichnende, Variable und diese Eigenschaft bestimmen, ob das Fenster mittlerweile wieder geschlossen wurde. Falls ja, hat die Eigenschaft den Wert true.
defaultStatus	Hierüber können Sie die Standardanzeige in der Statuszeile des Browsers ermitteln oder festlegen. Diese ist unter Umständen nicht permanent sichtbar, denn die Statuszeile des Browsers kann vielfältig beeinflusst werden, etwa über Setzen der Eigenschaft status.
innerHeight, innerWidth	Höhe und Breite des Anzeigebereichs eines Browsers. Die Angaben sind verschieden zu den Werten der äußeren Höhe und Breite, da dort die Größe optionaler Elemente wie Statuszeile, Bildlaufleiste oder Menüzeile abgezogen werden muss.
frames	Ein Objektfeld als Eigenschaft von window. Wir kommen gleich bei Frames darauf zurück.
length	Anzahl von offenen Fenstern der aktuellen Instanz des Browsers.
name	Über diese Eigenschaft kann der Name eines Fensters bestimmt werden.
outerHeight, outerWidth	Die Höhe und Breite des gesamten Fensters.
opener	Das Fenster, von dem aus das aktuelle Fenster geöffnet wurde.
pageXOffset, pageYOffset	Startposition des sichtbaren Dokuments im Fenster (Angabe links oben). Scrollen verändert den Wert.
parent	Übergeordnete Fenster der Objekthierarchie (read-only).
locationbar, menubar, personalbar, scrollbars, statusbar, toolbar	Angabe, ob die Fensterinstanz eine eigene URL-Adresszeile, eine eigene Menüleiste, eine Zeile für die Favoriten (Internet Explorer) bzw. persönliche Symbolleiste, Bildlaufleisten, Statuszeile oder Werkzeugleiste anzeigt. Es handelt sich jeweils um ein Unterobjekt von window, das über die Eigenschaft visible (jeweils Wert true, falls das Objekt sichtbar gesetzt ist) zu manipulieren ist.
self	Das aktuelle Fenster (read-only).
status	Die Eigenschaft ermittelt oder setzt die aktuelle Anzeige der Statuszeile eines Fensters.
top	Oberstes window-Objekt in der Objekthierarchie (read-only).

Tabelle 16.2: Eigenschaften von window



Das window-Objekt gibt es bereits seit JavaScript 1.0, aber da waren noch nicht alle jetzt verfügbaren Eigenschaften implementiert. In JavaScript 1.1 wurden die Eigenschaften closed, history und opener hinzugefügt. JavaScript 1.2 brachte crypto, innerHeight, innerWidth, locationbar, menubar, offlineBuffering, outerHeight, outerWidth, pageXOffset, pageYOffset, personalbar, screenX, screenY, scrollbars, statusbar und toolbar.

Beispiel 4:

Erstellen Sie bitte nachfolgende HTML-Datei (Name der Datei window4.htm):

```
<HTML>
<SCRIPT language="JavaScript">
var meinFenster;
```

```
function fenster_auf()
{
  meinFenster =
  open("bild1.htm","DiesistkeinleeresFenster");
  meinFenster.outerHeight=300;
  meinFenster.outerWidth=300;
}
function fenster_zu()
{
  meinFenster.close();
}
</SCRIPT>
<BODY>
<A href="JavaScript:fenster_auf()">
Neues Fenster auf
</A>
<BR>
<A href="JavaScript:fenster_zu()">
Neues Fenster zu
</A>
<BR>
</BODY>
</HTML>
```

Laden Sie die Datei in verschiedene Browser und probieren Sie die Links aus. Der Navigator kann mit dem Setzen der Größe gut umgehen. Der Internet Explorer wird die Größenangaben nicht ausführen.

Die Event-Handler von window

Offiziell werden die folgenden Event-Handler beim window-Objekt unterstützt:

- onBlur
- onDragDrop
- onError
- onFocus
- onLoad
- onMove
- onResize
- onUnload

Die Methoden von window

Das window-Objekt besitzt einige sehr wichtige Methoden, die wir in den vorherigen Tagen auch vielfach schon verwendet haben. Nachfolgend finden Sie die wichtigsten Methoden des window-Objektes:

Methode	Eigenschaft
<code>alert([Meldung])</code>	Öffnen eines Anzeigefensters mit der in den Klammern spezifizierten Nachricht. Die auszugebende Meldung muss als Parameter in Hochkommata gesetzt werden. Rückgabewerte von eventuell dort notierten Funktionen oder Methoden werden als String ausgegeben.
<code>back()</code>	Entspricht weitgehend einem Klick auf die Zurück -Schaltfläche im Browser.
<code>blur()</code>	Entzieht einem Fenster den Focus (ähnlich dem Event-Handler onBlur).
<code>captureEvents()</code>	Auffangen von Ereignis-Objekten - mehr dazu morgen (Netscape).

<code>clearTimeout</code> <code>([TimeoutVar])</code>	Diese Methode macht den Aufruf von <code>setTimeout()</code> wieder rückgängig, sofern dieser in der Form <code>[TimeoutVar] = setTimeout()</code> erfolgte. Diese Variable <code>[TimeoutVar]</code> muss <code>clearTimeout()</code> als Parameter übergeben werden.
<code>close()</code>	Die Methode schließt ein Fenster.
<code>confirm([Meldung])</code>	Erzeugt ein Dialogfenster zum Bestätigen oder Abbrechen einer Aktion. Es gibt zwei Schaltflächen (OK und Abbruch), welche die Rückgabewerte <code>true</code> und <code>false</code> liefern. Diese können Sie dann in der aufrufenden Funktion auswerten. Die auszugebende Meldung muss als Parameter in Hochkommata gesetzt werden. Rückgabewerte von eventuell dort notierten Funktionen oder Methoden werden als String ausgegeben.
<code>find([string</code> <code>[,caseSensitive,</code> <code>backward])</code>	Sucht im aktuellen Dokument nach der angegebenen Zeichenkette und gibt <code>true</code> zurück, wenn sie gefunden wurde. Über die optionalen booleschen Werte <code>caseSensitive</code> und <code>backward</code> kann angegeben werden, ob Groß- und Kleinschreibung relevant ist und ob vom aktuellen Punkt aus bis zum Anfang eines Dokumentes gesucht werden soll.
<code>focus()</code>	Aktiviert ein Fenster (ähnlich dem Event-Handler <code>onFocus</code>).
<code>forward()</code>	Entspricht weitgehend einem Klick auf die Vorwärts -Schaltfläche im Browser.
<code>home()</code>	Ruft die Startseite des Browsers auf.
<code>moveBy(horizontal,</code> <code>vertikal)</code>	Verschiebt ein Fenster um die angegebenen Pixel in horizontale oder vertikale Richtung.
<code>moveTo(x-Koordinate,</code> <code>y-koordinate)</code>	Verschiebt ein Fenster auf die angegebene X-/Y-Position.
<code>open(URL, Fenstername, [Optionen])</code>	Die Methode öffnet ein neues Fenster.
<code>print()</code>	Druckt den Inhalt einer Seite aus (analog dem Menübefehl zum Drucken).
<code>prompt(</code> <code>[Aufforderungstext], [Feldvorbelegung])</code>	Die Methode fordert den Anwender in einem Dialogfenster mit einer OK - und einer Abbrechen -Schaltfläche zu einer Eingabe in einem Feld auf und gibt diesen Wert an die aufrufende Funktion zurück, wenn Sie die OK -Schaltfläche betätigen (bei Abbrechen den Wert <code>null</code>). Als Parameter geben Sie das Label des Eingabefeldes und optional einen Vorbelegungswert an ("" für keine Vorbelegung).
<code>resizeBy</code> <code>(horizontal,</code> <code>vertikal)</code>	Verändert die Größe von einem Fenster um die angegebenen Pixel in horizontale oder vertikale Richtung.
<code>resizeTo(Breite, Hoehe)</code>	Setzt die Größe von einem Fenster auf die angegebenen Pixel in der angegebenen Breite und Höhe.
<code>scrollBy(horizontal, vertikal)</code>	Scrollt den Inhalt von einem Fenster um die angegebenen Pixel in horizontale oder vertikale Richtung.
<code>scrollTo(x-Koordinate, y-Koordinate)</code>	Scrollt den Inhalt von einem Fenster auf die angegebene X-/Y-Position.

<code>setTimeout([Anweisung], [Millisek])</code>	Diese Methode ruft die angegebene Anweisung nach der angegebenen Anzahl von Millisekunden auf. Der Rückgabewert kann einer Variablen zugewiesen werden, die von <code>clearTimeout()</code> zum Widerrufen einer verzögerten Anweisung genutzt werden kann.
<code>stop()</code>	Bricht den Ladevorgang einer Seite ab. Entspricht weitgehend einem Klick auf die STOP -Schaltfläche im Browser.

Tabelle 16.3: Methoden von window

Die Methoden `blur()`, `focus()` und `scroll()` und die unterstützten Eventhandler `onBlur`, `onError` und `onFocus` wurden in JavaScript 1.1 eingeführt. JavaScript 1.2 brachte die neuen Methoden `atob()`, `back()`, `btoa()`, `captureEvents()`, `clearInterval()`, `crypto.random()`, `crypto.signText()`, `disableExternalCapture()`, `enableExternalCapture()`, `find()`, `forward()`, `handleEvent()`, `home()`, `moveBy()`, `moveTo()`, `releaseEvents()`, `resizeBy()`, `resizeTo()`, `routeEvent()`, `scrollBy()`, `scrollTo()`, `setHotKeys()`, `setInterval()`, `setResizable()`, `setZOptions` und `stop()`. Als deprecated gilt seitdem die `scroll()`-Methode.

Ein Klassiker, der die Eigenschaft `status` ausnutzt, ist das nachfolgende Beispiel. Die Statuszeile wird für eine Laufschrift genutzt, die von links nach rechts beliebigen Text Zeichen für Zeichen aufrollt. Ist der Text vollständig angezeigt, wird die Statuszeile gelöscht und es geht wieder von vorne los.

Dabei wird die Methode `setTimeout()` verwendet, um einen so genannten rekursiven Aufruf zu gewährleisten. Die Methode `setTimeout()` ruft nach der vorgegebenen Zeit (vom Zeitpunkt des Aufrufs an gerechnet) die als Parameter angegebene Anweisung auf. Damit die gleiche Anweisung erst dann ausgeführt wird, wenn die erste erledigt ist und ein immer gleiches, vorgegebenes Zeitintervall gewartet wurde, ruft man sich die aufgerufene Methode mit der Methode `setTimeout()` am Ende wieder selbst auf.

Beispiel 5:

Erstellen Sie bitte nachfolgende HTML-Datei (Name der Datei `window5.htm`):

```
<HTML>
<HTML>
<SCRIPT language="JavaScript">
var text = "Herzlich Willkommen bei RJS-EDV-KnowHow";
var anzeige = "";
var i=0;
function textRoll()
{
i=i+1;
anzeige = text.substring(0,i);
window.status=anzeige;
if (i == text.length) i = 0;
scrolltext();
}
function scrolltext()
{
setTimeout('textRoll()',100);
}
</SCRIPT>
<BODY onLoad="scrolltext()">
</BODY>
</HTML>
```

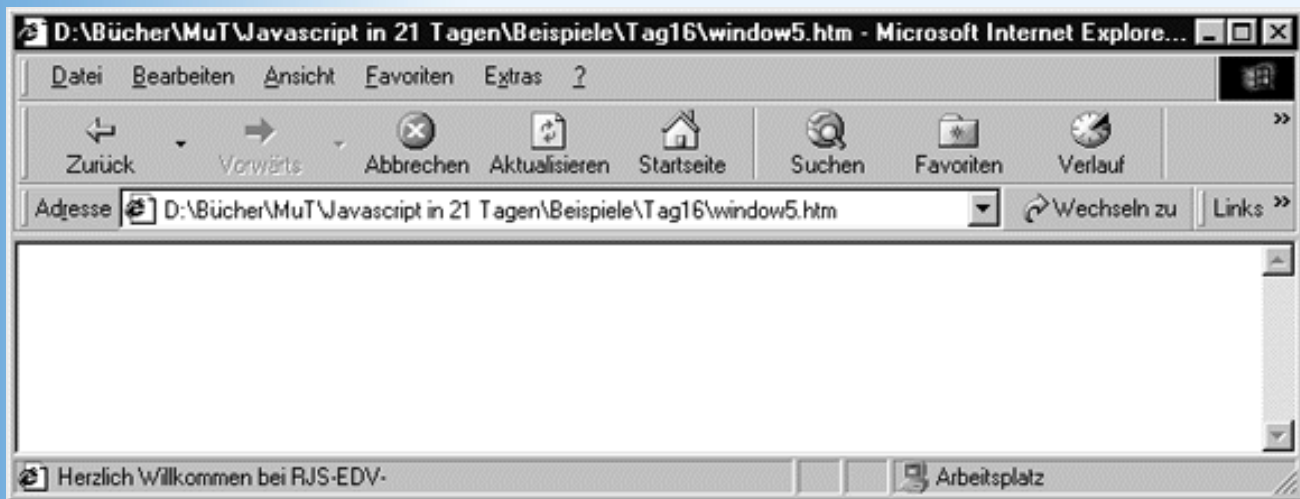


Abbildung 16.5: Die Statuszeile wird dynamisch geschrieben

Im nächsten Beispiel wollen wir das Problem über Methoden angehen, dass eine Fenstergröße und eine Positionierung eines Fensters weder mit der `open()`-Methode noch mit deren Eigenschaften so gesetzt werden kann, dass alle Browser das gleich behandeln. Versuchen wir, inwieweit die passenden Methoden vom `window`-Objekt funktionieren. Dabei wenden wir auch das Schlüsselwort `self` an, um beim Laden einer Webseite das Hauptfenster des Browsers bereits in Größe und Position anzugeben.

Beispiel 6:

Erstellen Sie bitte nachfolgende HTML-Datei (Name der Datei `window6.htm`):

```
<HTML>
<SCRIPT language="JavaScript">
var meinFenster;
function fenster_auf()
{
    meinFenster =
    open("bild1.htm","DiesistkeinleeresFenster");
    meinFenster.moveTo(100, 200);
    meinFenster.resizeTo(400, 300);
}
function positionierung()
{
    self.resizeTo(400, 300);
    self.moveTo(150, 150);
    self.scrollTo(0,200)
}
function fenster_zu()
{
    meinFenster.close();
}
</SCRIPT>
<BODY onLoad="positionierung()">
<IMG src="131.gif">
<P>
<IMG src="132.gif">
<P>
<A href="JavaScript:fenster_auf()">
Neues Fenster auf
</A>
<BR>
<A href="JavaScript:fenster_zu()">
Neues Fenster zu
</A>
```



```
<BR>  
</BODY>  
</HTML>
```

Testen Sie die Datei in verschiedenen Browsern. Leider zeigt sich auch hier, dass nicht alle Methoden vollständig identisch funktionieren (aber doch recht gut).

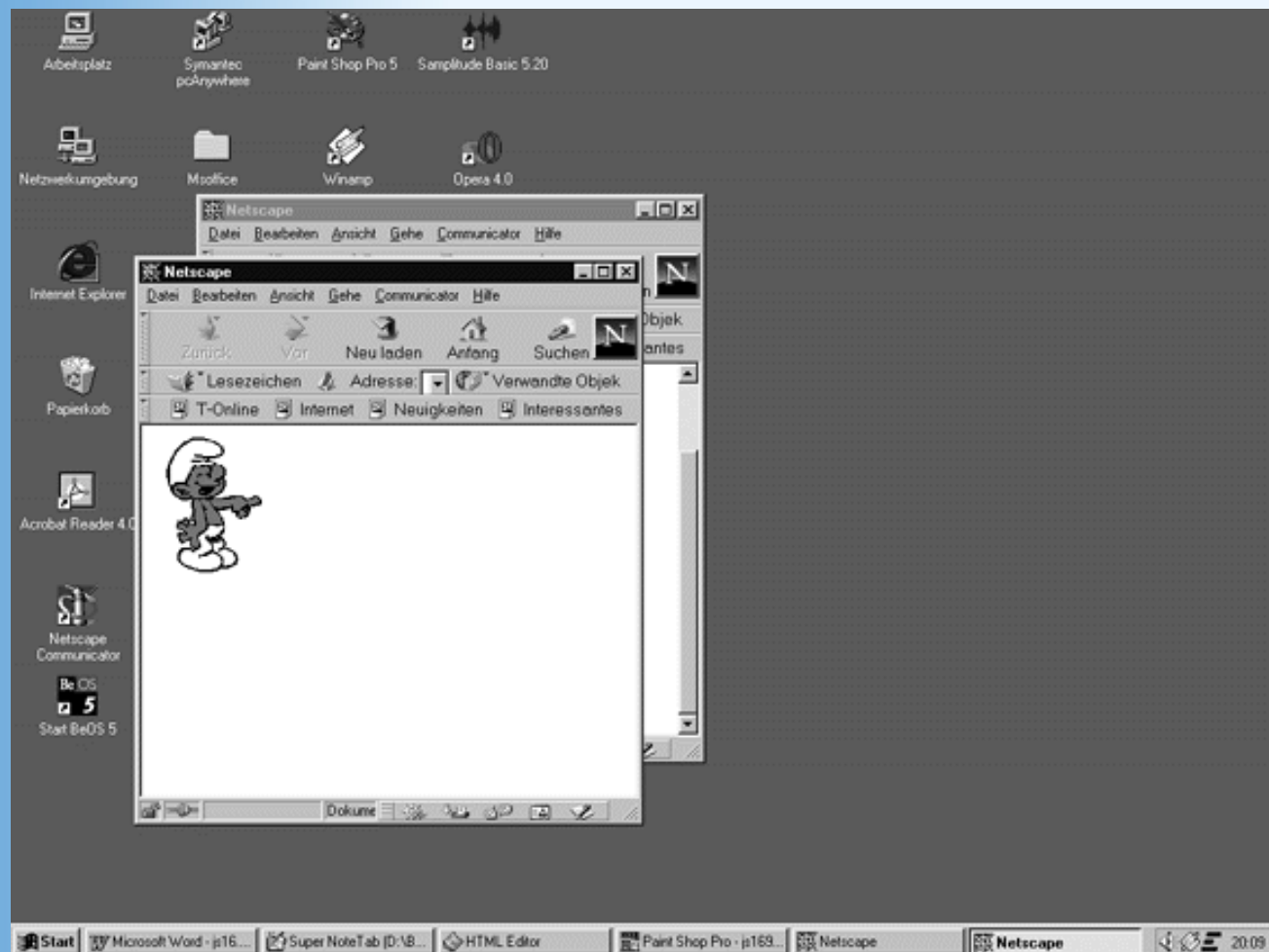


Abbildung 16.6: Das Folgefenster in Originalgröße und -positionierung im Netscape



Abbildung 16.7: Die Startdatei im IE



Abbildung 16.8: Die Startdatei im Netscape

Testen wir nun noch eine weitere wichtige Methode von window - confirm().

Beispiel 7:

```
<HTML>
<SCRIPT language="JavaScript">
var meinFenster;
function fenster_auf()
{
    meinFenster =
    open("bild1.htm","");
}
function fenster_zu()
{
    if(window.confirm(
    "Wollen Sie das Fenster wirklich schließen?"))
    meinFenster.close();
}
</SCRIPT>
<BODY onLoad="positionierung()">
<A href="JavaScript:fenster_auf()">
Neues Fenster auf
</A>
<BR>
<A href="JavaScript:fenster_zu()">
Neues Fenster zu
</A>
```

```
<BR>
</BODY>
</HTML>
```

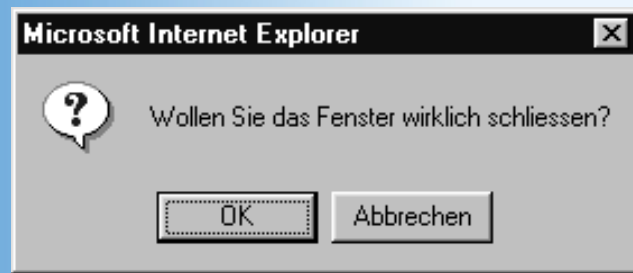


Abbildung 16.9: Rückfrage

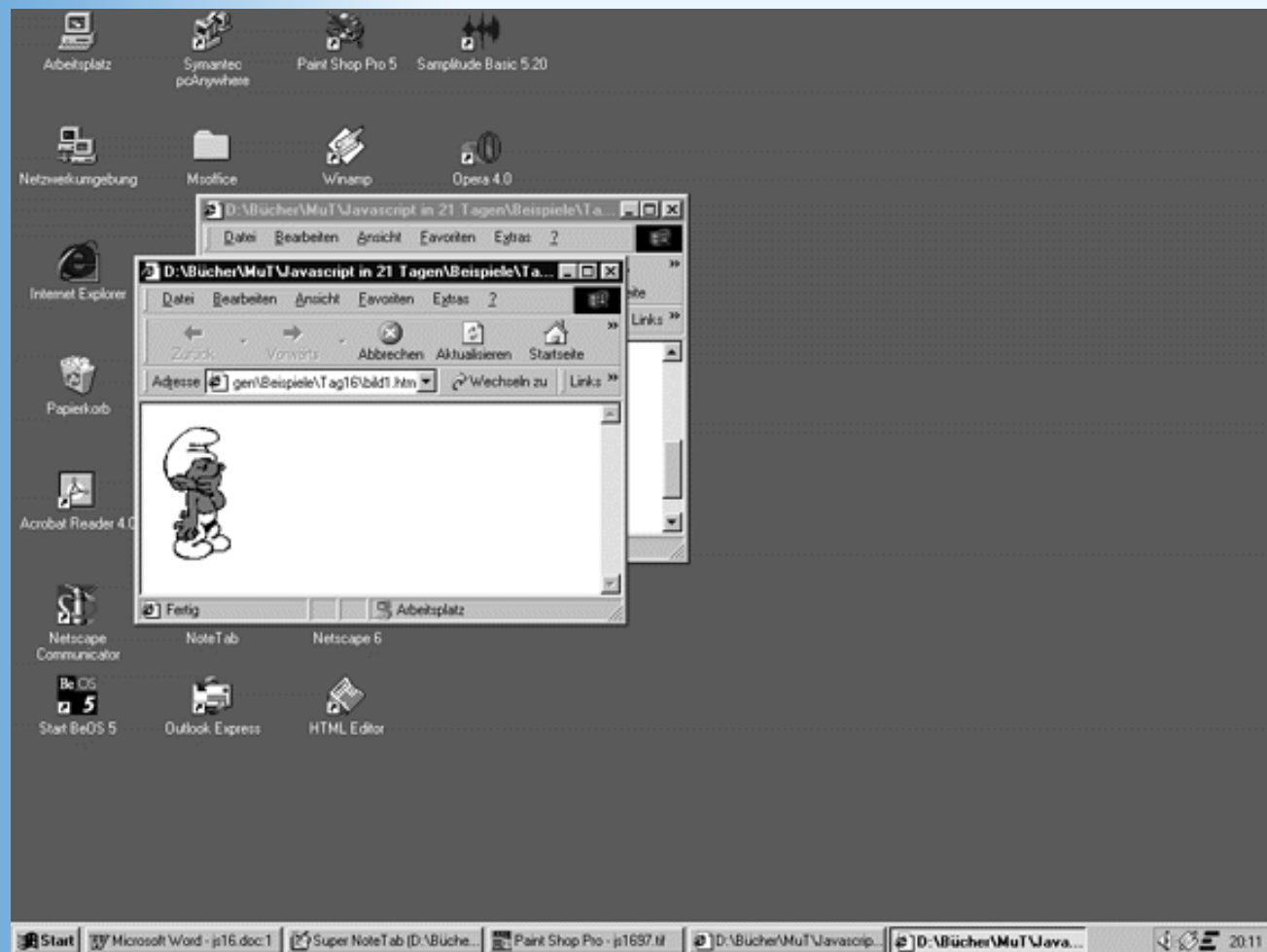


Abbildung 16.10: Das Folgefenster in Originalgröße und -positionierung im IE

Wenn Sie das Folgefenster über den Link wieder schließen wollen, wird Ihnen ein Dialogfenster angezeigt. Nur, wenn Sie **Ok** drücken, wird das Fenster geschlossen.

Frames

Frames haben wir bisher einmal in einem Beispiel knapp angerissen, aber im Wesentlichen ist das Thema für uns an dieser Stelle noch Neuland. Frames bedeuten die Aufteilung des Anzeigebereichs eines Browsers in einzelne Segmente, die unabhängig von einander mit Inhalt gefüllt werden können. Ursprünglich von der Firma Netscape entwickelt, gehört die Frame-Technologie zum festen Bestandteil von HTML 4.0. Dementsprechend werden wir uns Frames auch wieder von HTML-Seite aus nähern.

Die Frame-Technologie beinhaltet zwei Fassetten. Es gibt einmal die Frames selbst, welche den Anzeigebereich des Browsers gliedern und dennoch eigenständige Fenster mit allen HTML-Darstellungsmöglichkeiten bilden. Diese einzelnen Frames sind zusammen in einer umgebenden Struktur - dem so genannten Frameset - enthalten. Diese Frameset-Datei ist diejenige, welche in den Browser geladen wird. Sie enthält nur Informationen über Namen, Größe und Position der

einzelnen im Anzeigefenster enthaltenen Frames. Insbesondere ist der eigentliche Inhalt der Frames nicht in der Frameset-Datei notiert.

Eine HTML-Datei, welche als Frameset-Datei für eine Framestruktur verwendet werden soll, hat einen anderen Aufbau wie eine gewöhnliche HTML-Datei. Es wird vor allen Dingen keinen `<BODY>`-Tag mehr benötigt. Dessen Funktion wird von dem `<FRAMESET>`-Container übernommen.



Eine Frameset-Datei kann zu dem `<FRAMESET>`-Container einen zusätzlichen `<BODY>`-Container haben. Dieser wird dann außerhalb des `<FRAMESET>`-Bereichs notiert (normalerweise in einem `<NOFRAMES>`-Container) und deckt den Fall ab, dass ein Browser keine Frames versteht. Immerhin gehören Frames wie gesagt zu HTML 4.0, und diesen Standard können noch nicht alle Browser. Zudem kann die Framedarstellung in einigen Browsern deaktiviert werden. In dem `<BODY>`-Container werden alle Anweisungen notiert, die notwendig sind, um ohne Frames sinnvolle Informationen anzuzeigen (und wenn es nur der Hinweis ist, dass ein Projekt Frames verwendet). Wir werden das gleich beim Grundgerüst sehen.

Bei der Definition der Frameset-Datei müssen genaue Angaben darüber gemacht werden, wie das Anzeigefenster aufgeteilt werden soll und welche Dateien konkret in ein einzelnes Frame zu laden sind.

Es müssen bei der Angabe der Aufteilung je nach Konzept entweder Reihen oder Spalten (Anzahl und Größe) oder auch beides definiert werden. Die Definition der Reihen und Spalten erfolgt als Erweiterung des `<FRAMESET>`-Tag. Über den Parameter

`rows= "[Angabe in Prozent]"`

legen Sie die Anzahl von Reihen und deren anfänglichen, prozentualen Anteil an der Fenstergröße fest. So viele durch Kommata getrennte Prozentwerte dort notiert werden, so viele Zeilen hat der Frameset. Analog werden die Spalten im einleitenden `<FRAMESET>`-Tag festgelegt, falls man eine Aufteilung nach Spalten wünscht. Dazu dient der Parameter `cols= "[Angabe in Prozent]"`.



Die Prozentangaben hinter dem Gleichheitszeichen sollten in Anführungszeichen stehen und definieren Verhältnisse der einzelnen Frames zueinander - nicht einen Faktor von der Größe des Gesamtanzeigefensters vom Browser! Insbesondere müssen die Summen der Prozentangaben nicht 100% ergeben. Die Angabe `cols="50%,50%"` definiert bei identischer Gesamtgröße des Browser-Anzeigefensters die gleiche Fenstergröße der Frames wie `cols="17%,17%"` oder `cols="12345%,12345%"`.

Wenn also über den `<FRAMESET>`-Tag und seine Parameter ein äußeres Gerüst für eine Framestruktur aufgebaut ist, wird im Inneren für jede dort definierte Zeile oder Spalte ein einzelnes Frame angegeben.

Der Inhalt der Frames wird mit der Syntax

`<FRAME src= "[URL]" name="[Name]">`

spezifiziert. Über die `src`-Angabe wird der URL des Inhalts von dem jeweiligen Frame gesetzt, `name` definiert einen internen Namen des Frames, über den es in Verweisen, aber auch aus JavaScript heraus angesprochen werden kann. Beide Angaben, sowohl der URL als auch der Name, sollten in Anführungszeichen stehen.



Der Name eines Frames ist relativ frei wählbar. Es gelten nur die üblichen Regeln für HTML-Anweisungen (keine Leerzeichen, so gut wie keine Sonderzeichen - außer dem Unterstrich `_`) und die Namen sollten recht sprechend sein. Gewisse Namen für Frames sind verboten, da diese reservierten Fensteramen bei Verweisen eine spezielle Bedeutung haben. Dies sind die folgenden Token:

Reserviert	Bedeutung
<code>_self</code>	Referenz auf das aktuelle Frame, wo eine Seite gerade dargestellt wird (Standardsituation).

<code>_parent</code>	Referenz auf das übergeordneten Frame bei verschachtelten Framestrukturen (darauf kommen wir gleich). Falls es kein übergeordnetes Fenster gibt, bedeutet dies eine Referenz auf ein neues Fenster (ohne Framestruktur).
<code>_blank</code>	Referenz auf ein neues Browserfenster ohne Framestruktur.
<code>_top</code>	Referenz auf das volle Browserfenster. Die Framestruktur verschwindet.

Tabelle 16.4: Reservierte Fensteramen.

Ein typisches Grundgerüst einer Framestruktur sieht von der Theorie her so aus:

```
<HTML>
<FRAMESET>
  ...Frames
</FRAMESET>
<NOFRAMES>
  <BODY>
    ... Alternativinformationen
  </BODY>
</NOFRAMES>
</HTML>
```



Beachten Sie bitte, dass in der <FRAMESET>-Anweisung noch keine Parameter vorkommen. Diese müssen für eine endgültige Frameset-Datei noch sinnvoll ergänzt werden.

Wenn ein Browser keine Frames darstellen kann, wird er nach dem Prinzip der Fehlertoleranz den vollständigen <FRAMESET>-Bereich samt darin enthaltener <FRAME>-Tags ignorieren. Er wird in der Regel auch die Anweisung <NOFRAMES> nicht kennen und auch diese ignorieren, was aber so gewollt ist. Diese Anweisung ist nur für Frame-fähige Browser gedacht. Ein Frame-fähiger Browser kennt die <NOFRAMES>-Anweisung und wird den vollständigen Container ignorieren. Nicht auf Grund des Prinzips der Fehlertoleranz, sondern weil er programmiert ist, diesen Container nicht anzuzeigen .

Ein nicht-Frame-fähiger Browser fängt hinter <NOFRAMES> mit dem Anzeigen der nachfolgenden Anweisungen und dem dort stehenden Klartext an. In dem <NOFRAMES>-Bereich können Sie eine vollständige HTML-Datei aufbauen. Auch wie in dem Schema mit <BODY>-Tag, es kann aber auch nur Klartext sein.



Die Notation von Alternativinformationen im <NOFRAMES>-Container kann schiefgehen (auch wenn fast alle HTML-Tools das Gerüst so generieren). So zeigt beispielsweise der Opera-Browser bei deaktivierten Frames¹ dort notierten Text nicht an (obwohl eventuell dort notierte <BODY>-Anweisungen wie Hintergrundfarben umgesetzt werden). Man kann aber gefahrlos auf dem <NOFRAMES>-Bereich verzichten und einfach den Body direkt hinter den Frameset notieren. Browser, die Frames anzeigen, werden den Body-Bereich ignorieren, alle anderen sich so verhalten wie oben beschrieben. Noch sicherer ist es, sowohl im <NOFRAMES>-Bereich als auch dahinter die Zusatzinfos zu notieren (also zweimal). Ob dies in der Praxis aber notwendig ist, muss jeder für sich entscheiden.

Ein Frame muss nicht von vornherein etwas anzeigen. Es kann leer bleiben, bis dort Inhalt benötigt wird. Verzichtet man auf die Angabe eines Inhaltes über den <FRAME>-Tag, wird der Raum vom Browser neutral ausgefüllt. Sofern dies für alle Frames geschieht, erkennen Sie unter Umständen überhaupt keine Framestruktur im Browserfenster (bei entsprechender Konfiguration des Browsers - meist sind aber die Trennlinien zu sehen).

Wenn Sie Frames mit Inhalt angeben, spielt die Reihenfolge der Frame- Notierungen eine wichtige Rolle. Die erste Angabe wird automatisch im ersten Frame, die zweite Datei im zweiten Frame und so fort angezeigt.

Beispiel 8:

Erstellen Sie bitte nachfolgende HTML-Datei (Name der Datei Li.htm):

```
<HTML>
<BODY>
<H1>
<div align="center">
Links
</div>
</H1>
</BODY>
</HTML>
```

Erstellen Sie bitte auch die zweite HTML-Datei Re.htm:

```
<HTML>
<BODY>
<H1>
<div align="center">
Rechts
</div>
</H1>
</BODY>
</HTML>
```

Jetzt folgt unser eigentlicher Frameset (FS1.htm). Dabei werden wir ein vollständiges Alternativkonzept mit zweifachem Body-Bereich verwenden:

```
<HTML>
<FRAMESET cols="25%,75%">
<FRAME src= "Li.htm" name="links" >
<FRAME src= "Re.htm" name="rechts" >
</FRAMESET>
<NOFRAMES>
<BODY bgcolor=cyan>
<H1 align=center>
Sorry, hier werden Frames verwendet
</H1>
</BODY>
</NOFRAMES>
<BODY bgcolor=cyan>
<H1 align=center>
Sorry, hier werden Frames verwendet
</H1>
</BODY>
</HTML>
```

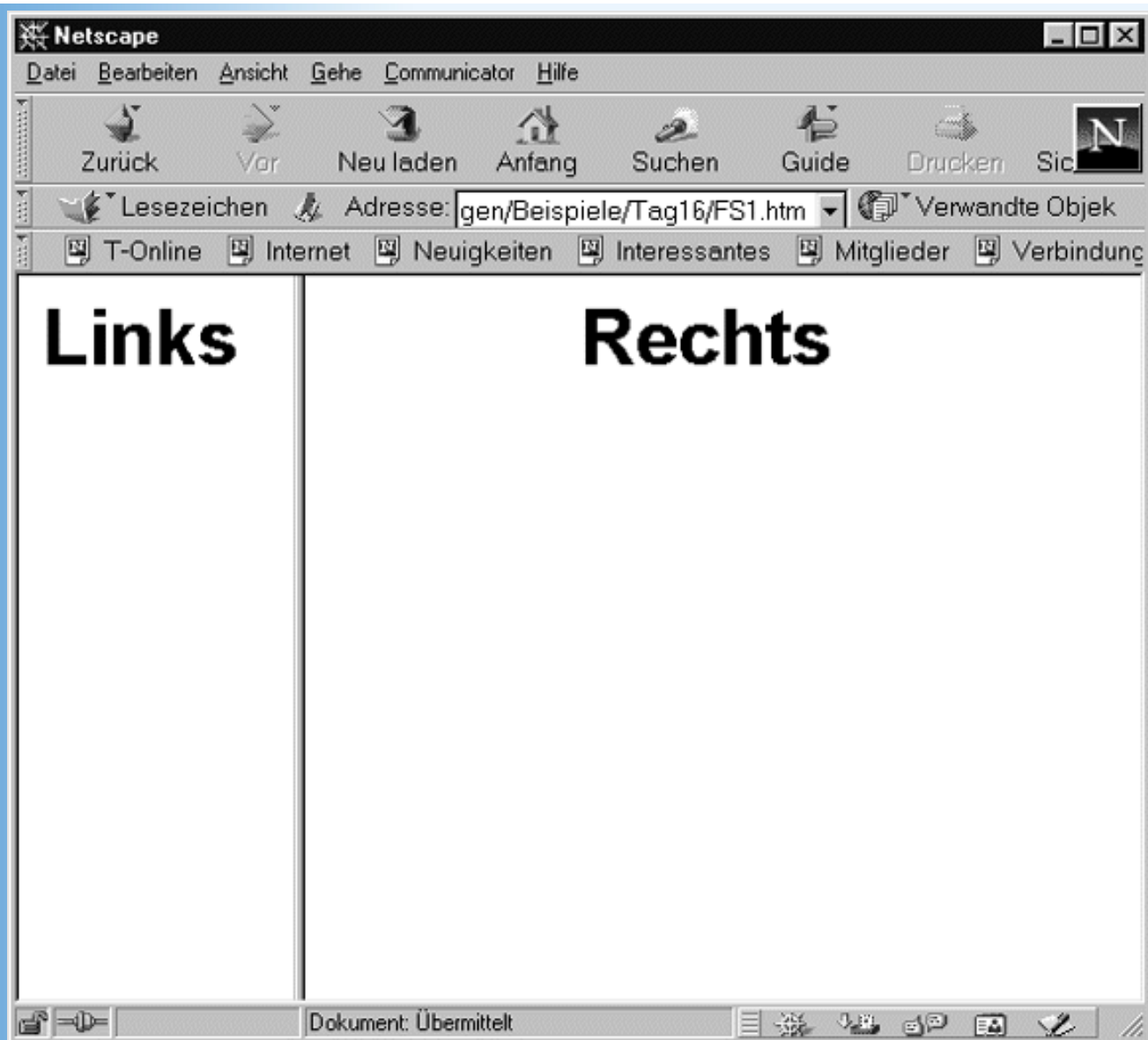


Abbildung 16.11: Eine Frameaufteilung 1 zu 3

Im linken Frame wird die zuerst angegebene Datei Li.htm angezeigt, im rechten Frame die Datei Re.htm .



Abbildung 16.12: Durch das Alternativkonzept werden die Alternativinfos angezeigt



Sie können auch andere Ressourcen als HTML-Dateien direkt referenzieren. Etwa Bilder (ohne umgebendes HTML-Gerüst). Dann müssen Sie nur den URL der jeweiligen Datei beim src-Attribut angeben. Sie können ebenso natürlich mehr als zwei Spalten oder Zeilen angeben. Beispiel für vier Reihen:

```
<FRAMESET rows="2%, 2%, 3%, 5%">
```

Anstelle von Prozentangaben für die Größe von Frames können Sie Zahlenwerte benutzen, mit denen Sie tatsächliche Größen der Frames in Pixeln definieren. Dies erfolgt über die Angabe

```
rows="[Angabe in Pixel]",
```

bzw.

```
cols= "[Angabe in Pixel]").
```

Diese Größenangaben werden allerdings durch die Größe des Browser- Anzeigefensters wieder relativiert, d.h. die Frames werden wieder im angegebenen Verhältnis an das Anzeigefenster des Browsers angepasst. Sinn machen absolute Angaben nur, wenn sie mit so genannten Jokerzeichen, variable Angaben, verwendet werden. In Verbindung mit Jokerzeichen (der Stern, wie er auch sonst als Jokerzeichen üblich ist) bewirkt zum Beispiel

```
cols="300, *, 200"
```

- eine Spalte, die 300 Pixel breit ist (die erste),
- eine Spalte, die 200 Pixel breit ist (die dritte) und
- eine Spalte, die variabel ist und den jeweiligen Rest vom Anzeigefenster des Browsers einnimmt (die mittlere Spalte).

Beispiel 9:

Erstellen Sie die Datei FS2.htm:

```
<HTML>
```

```
<FRAMESET rows="200, *, 100">
```

```
<FRAME src= "Li.htm" name="links" >
<FRAME src= "FS1.htm" name="mitte" >
<FRAME src= "Re.htm" name="rechts" >
</FRAMESET>
<NOFRAMES>
<BODY bgcolor=cyan>
<H1 align=center>
Sorry, hier werden Frames verwendet
</H1>
</BODY>
</NOFRAMES>
<BODY bgcolor=cyan>
<H1 align=center>
Sorry, hier werden Frames verwendet
</H1>
</BODY>
</HTML>
```



Abbildung 16.13: Eine Frameaufteilung in Zeilen in der ersten Fenstergröße

Verändern Sie die Größe des Browsers.

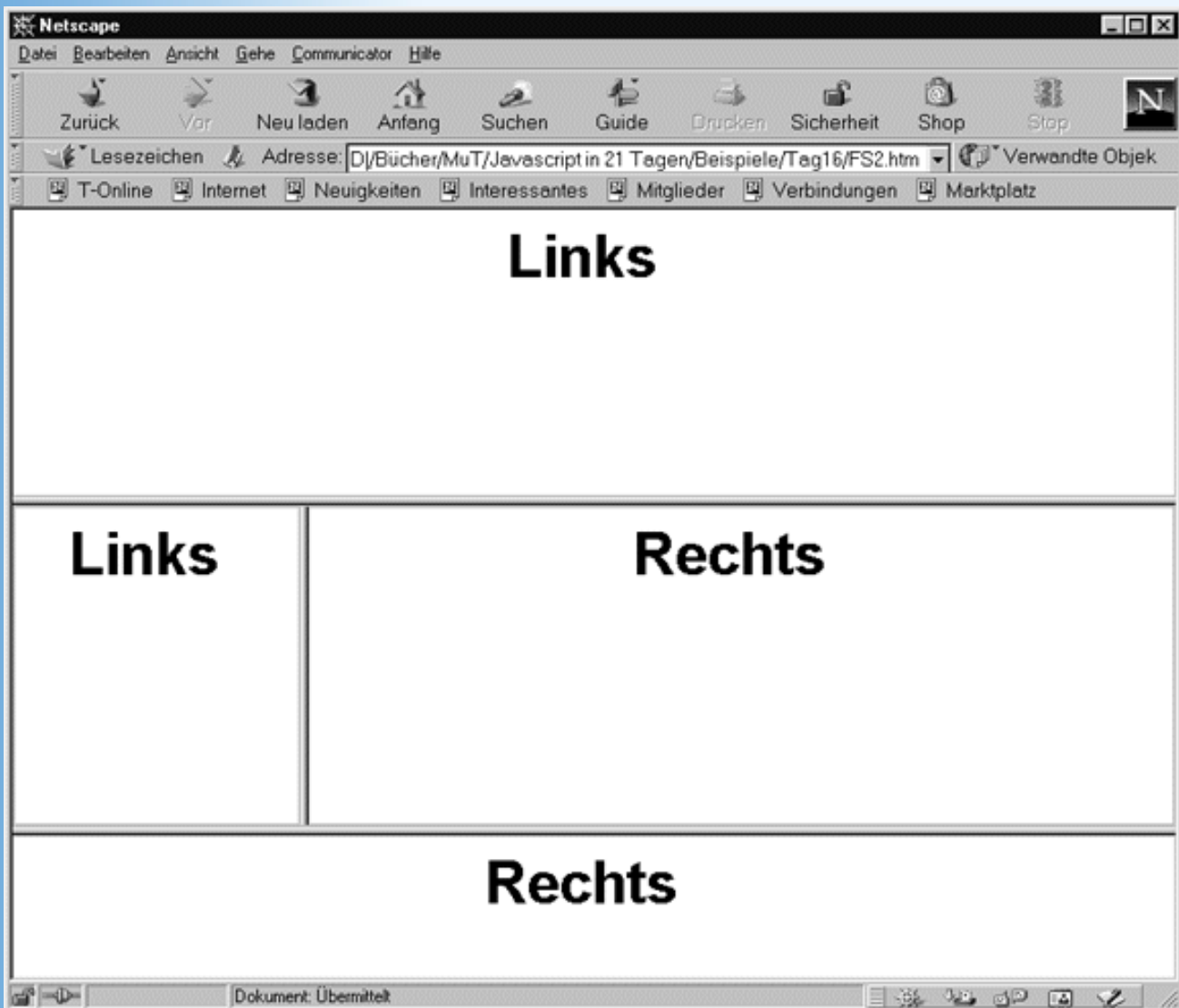


Abbildung 16.14: Die Frameaufteilung in der zweiten Fenstergröße

Im oberen Frame wird die zuerst angegebene Datei Li.htm angezeigt, im mittleren Frame die Datei FS1.htm und im unteren Frame die Datei Re.htm. Wenn Sie die Größe des Browserfensters verändern, wird die Größe der oberen und unteren Zeile fest bleiben, die mittlere Zeile füllt den restlichen Raum aus. Erst wenn die Gesamtgröße des Anzeigefensters zu klein wird, werden sich die beiden über Pixelwerte angegebenen Fenster auch verkleinern.



Im Zusammenhang mit prozentualen Angaben kann man das Jokerzeichen gleichfalls verwenden. Dort ist aber Vorsicht geboten, da sich dann die Relativbeziehungen von Prozentangaben durch das Jokerzeichen verändern. Es ist eigentlich unsinnig, hier mit Jokern zu arbeiten.

In dem letzten Beispiel habe ich Ihnen so ganz nebenbei eine Verschachtelung von Frames »untergejubelt«. In der mittleren Zeile ist das erste Frame-Beispiel mit einer spaltenweisen Framestruktur geladen worden, was eine Unterteilung dieses Frames in weitere zwei Frames bewirkt hatte. Dies kann beliebig fortgesetzt werden und ist theoretisch nur durch Plattformbeschränkungen begrenzt. Es macht aber jenseits von drei Ebenen selten Sinn.

Im Allgemeinen erfolgt die Verschachtelung von Frames nicht auf dem Weg, wie es in dem letzten Beispiel quasi nebenbei erfolgt ist. Es wird normalerweise innerhalb eines Framset-Containers ein darin verschachtelter Frameset-Container definiert werden. Die Verschachtelung erfolgt hierarchisch. Dabei ist offensichtlich, dass nur eine wechselweise Verschachtelung von Reihen und Spalten Sinn macht.

Beispiel 10:

Erstellen Sie die Datei FS3.htm:

```
<HTML>
```



```
<FRAMESET cols="30%,70%">
  <FRAME src="li.htm">
  <FRAMESET rows="30%,70%">
    <FRAME src="131.gif">
    <FRAME src="132.gif">
  </FRAMESET>
</FRAMESET>
<NOFRAMES>
Ihr Browser unterstützt keine Frames!
</NOFRAMES>
</HTML>
```

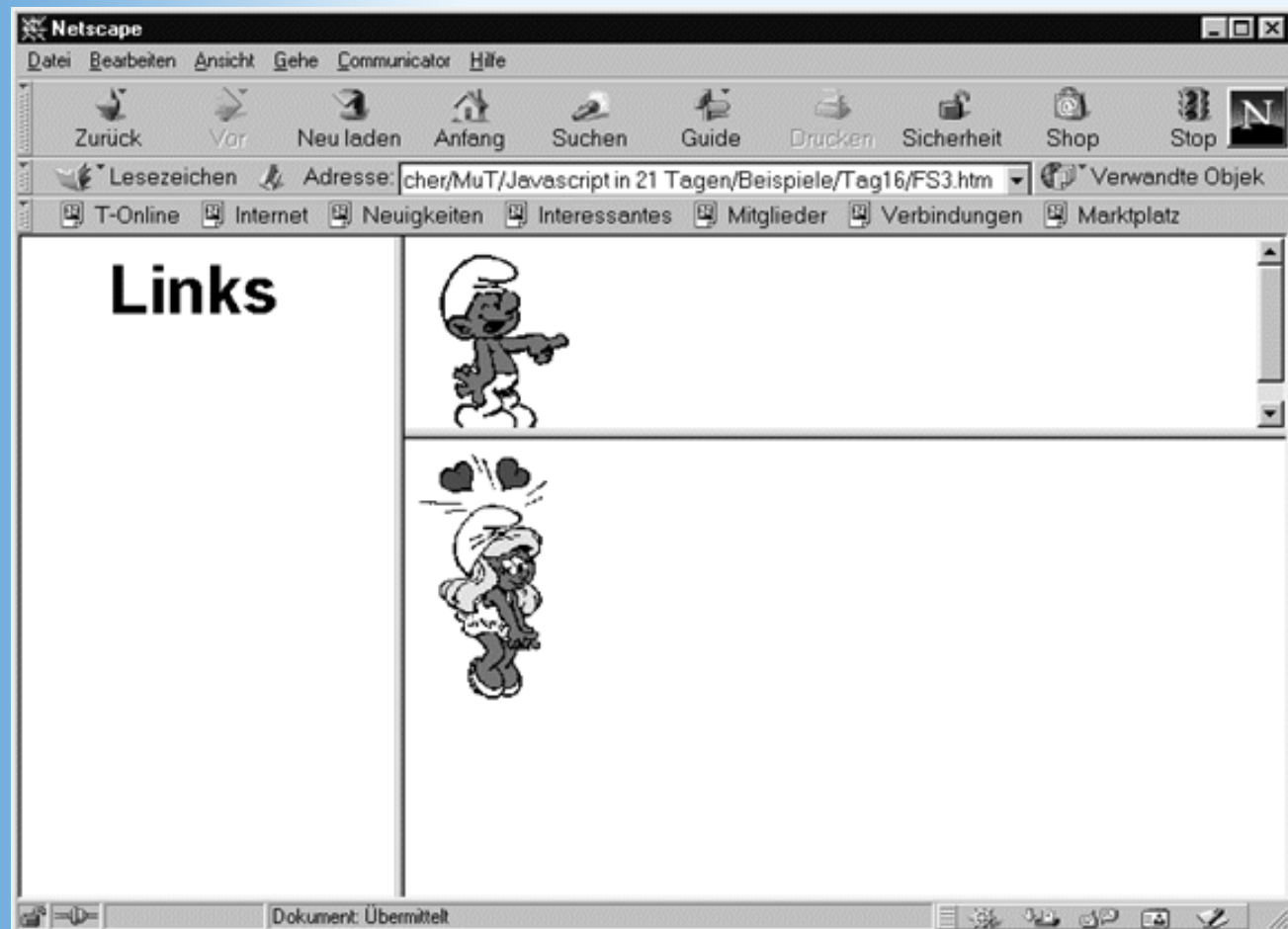


Abbildung 16.15: Eine verschachtelte Frame-Aufteilung

Das zweite Frame ist durch ein neues Frameset in zwei Zeilen unterteilt. Hier haben wir die Verschachtelung von Frameset-Containern. Die Einrückung des Quelltextes ist nicht willkürlich, sondern soll die Struktur der Verschachtelung deutlich machen. Auf die Darstellung hat die Einrückung keinen Einfluss, der Quelltext ist nur besser lesbar. In den beiden rechten Frames wird jeweils eine Grafik direkt referenziert.

Eigenschaften von Frames

Jedes Frame hat bestimmte Eigenschaften, die teilweise unabhängig von den anderen Frames des Framesets verändert werden können. Diese Festlegung der Eigenschaften erfolgt in der Regel im `<FRAME>`-Tag (wenn es ein einzelnes Frame betrifft) oder aber auch gelegentlich im `<FRAMESET>`-Tag (wenn es den gesamten Frameset betrifft) über optionale Attribute. Obwohl dies kein HTML-Buch ist, sollen ein paar Eigenschaften hier doch behandelt werden.

Jedes Frame zeigt standardmäßig unabhängig von den anderen Frames Bildlaufleisten an, wenn ein Inhalt nicht vollständig in ihm angezeigt werden kann. Diese Automatik lässt sich abschalten. Über den Parameter `scrolling` im `<FRAME>`-Tag legen Sie ausdrücklich fest, ob ein Frame automatisch mit Bildlaufleisten versehen werden soll oder nicht. Die Angabe `scrolling=no`

unterdrückt Bildlaufleisten, auch wenn ein Inhalt nicht vollständig in das Frame passt. Die Angabe `scrolling=yes`

erzwingt in jedem Fall Bildlaufleisten, sogar wenn es von der Größe des anzuzeigenden Inhaltes nicht notwendig wäre. Die Angabe `scrolling=auto`

entspricht der Standardeinstellung (Bildlaufleisten wenn nötig, sonst nicht).

Falls Sie die Größenveränderung eines Frames durch den Anwender unterbinden wollen, steht Ihnen ein weiterer Parameter des `<FRAMES>`-Tag zur Verfügung - `noresize`.

Der Parameter bedarf keiner weiteren Wertzuweisung und beeinflusst weitere Frames in derselben Reihe beziehungsweise Spalte, die dann an der Schnittstelle zu diesem Frame genauso wenig bewegt werden können. An der Verbindung von zwei Frames ohne den Parameter sind aber immer noch Verschiebungen der Framegröße möglich.

Frames werden standardmäßig mit einer Gitterstruktur angezeigt (die sichtbaren Ränder zwischen den einzelnen Frames). Dies stört oft das Layout, und Sie können diese Gitterlinien unterdrücken. Der Parameter `frameborder` im `<FRAMESET>`-Tag erlaubt die Unterdrückung. Mit der Angabe `frameborder=0`

können Sie die Gitterlinien ausschalten.

Beispiel 11:

Erstellen Sie die Datei FS4.htm:

```
<HTML>
<FRAMESET cols="30%,70%">
  <FRAME src="li.htm">
  <FRAMESET rows="30%,70%">
    <FRAME src="131.gif">
  <FRAMESET cols="50%,50%,50%" frameborder=0>
    <FRAME src= "li.htm" noresize>
    <FRAMESET rows="50%,50%">
      <FRAME src= "132.gif">
      <FRAME src= "132.gif">
    </FRAMESET>
    <FRAME src= "131.gif" scrolling=no>
  </FRAMESET>
  <FRAME src="132.gif">
</FRAMESET>
<NOFRAMES>
Ihr Browser unterstützt keine Frames!
</NOFRAMES>
</HTML>
```

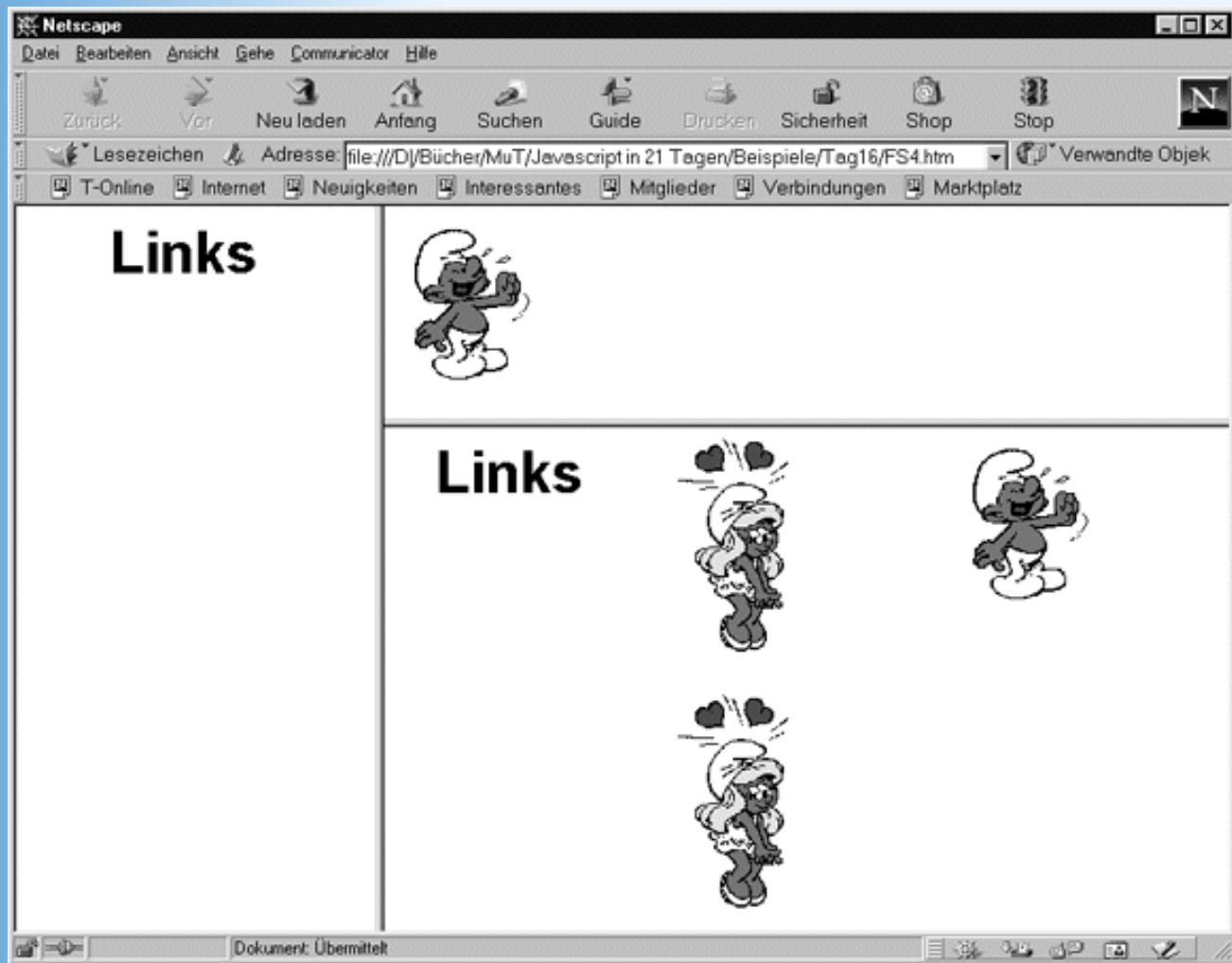


Abbildung 16.16: Eine mehrfach verschachtelte Frame-Aufteilung - teilweise ohne Linien

Was fällt Ihnen auf? Erst Mal ist die Framestruktur in dem Beispiel mehrfach verschachtelt und es werden einige Dateien gleich mehrfach im mehreren Frames geladen. Das ist selbstverständlich erlaubt. Aber es gibt mehr zu beachten. Versuchen Sie die Größe der Frames zu verändern. Die Frames lassen sich nicht in der Größe verändern. Dies bewirkt die Angabe `noresize`. Zwischen den Frames im Bereich links unten werden Sie keine Gitterlinie finden. Dies bewirkt die Angabe `frameborder=0`. Damit können Sie auch die Höhe der beiden Zeilen nicht mehr verändern, denn es steht kein Bereich zur Verfügung, wo der Mauszeiger zum Ziehen platziert werden kann.

Wenn Sie das Fenster ganz klein machen, wird das ganz rechts unten angeordnete Frame selbst dann keine Bildlaufleisten anzeigen, wenn der Inhalt nicht ganz zu sehen ist (`scrolling=no`).



Abbildung 16.17: Rechts unten fehlen die Bildlaufleisten

Verweise zu anderen Frames

Wir haben bisher noch kein einziges Mal den Parameter name verwendet. Mal war er da, mal nicht. Er schien überhaupt keinen Zweck zu haben. Jetzt aber werden wir ihn brauchen. Er hilft bei Verweisen zu anderen Frame-Inhalten. Normalerweise werden innerhalb einer Framestruktur Folgeseiten von Links in dem gleichen Frame angezeigt, von dem aus sie vom Benutzer aufgerufen werden. Sie können aber auch direkt bestimmte Frames adressieren und damit explizit festlegen, wo dieser neue Inhalt angezeigt werden soll. Über den mit dem Parameter name definierten Namen. Dieser Name wird im Link selbst über einen weiteren Parameter angegeben - target. Über eine Zuweisung wird in Anführungszeichen der Name angegeben.



Es können sowohl im Frameset vergebene Namen als auch die reservierten Fensternamen adressiert werden. Sofern der Framename in einem fremden Projekt bekannt ist, kann natürlich ebenso dieser notiert werden.

Beispiel 12:

Erstellen Sie zuerst die folgenden zwei HTML-Dateien (die Datei Bild1.htm haben wir noch von dem ersten Beispiel) - Datei Bild2.htm:

```
<HTML>
<BODY>
<IMG src="132.gif">
</BODY>
</HTML>
```

Datei Bild3.htm:

```
<HTML>
<BODY>
<IMG src="133.gif">
</BODY>
</HTML>
```

Erstellen Sie als Nächstes die Frameset-Datei FS5.htm:

```
<HTML>
<FRAMESET cols="30%,70%">
  <FRAME src="Hyper.htm">
  <FRAMESET rows="1%,1%,1%">
    <FRAME src="bild1.htm" name="eins">
    <FRAME src="bild2.htm" name="zwo">
    <FRAME src="bild3.htm" name="drei">
  </FRAMESET>
</FRAMESET>
<NOFRAMES>
Ihr Browser unterstützt keine Frames!
</NOFRAMES>
</HTML>
```

Die nächste Datei Hyper.htm wird diejenige, wo die Hyperlinks notiert werden. Diese verwenden die in der Frameset-Datei angegebenen Namen. Die Datei wird im Frameset in der linken Spalte geladen:

```
<HTML>
<BODY>
<A href="bild1.htm" target="zwo">
Bild 1 in die Mitte
</A>
<BR>
<A href="bild1.htm" target="eins">
```

```
Bild 1 nach oben
</A>
<BR>
<A href="bild1.htm" target="drei">
Bild 1 nach unten
</A>
<BR>
<A href="bild2.htm" target="zwo">
Bild 2 in die Mitte
</A>
<BR>
<A href="bild2.htm" target="eins">
Bild 2 nach oben
</A>
<BR>
<A href="bild2.htm" target="drei">
Bild 2 nach unten
</A>
<BR>
<A href="bild3.htm" target="zwo">
Bild 3 in die Mitte
</A>
<BR>
<A href="bild3.htm" target="eins">
Bild 3 nach oben
</A>
<BR>
<A href="bild3.htm" target="drei">
Bild 3 nach unten
</A>
<BR>
</BODY>
</HTML>
```

Die Datei mit den Hyperlinks erlaubt das Laden jeder unserer drei Bilddateien in jedem der Frames auf der rechten Seite. Laden Sie die Datei FS5.htm und testen Sie die Links.

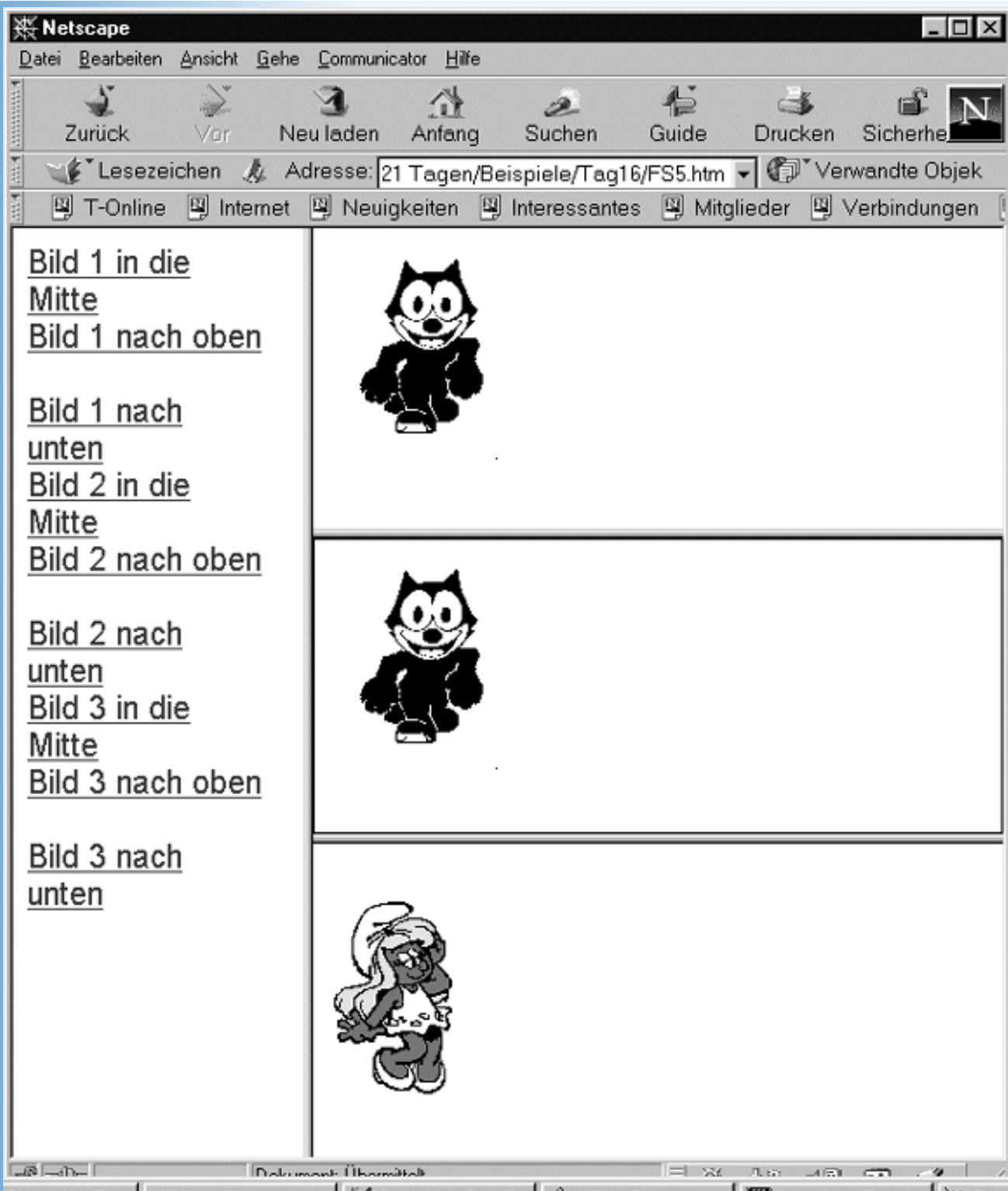


Abbildung 16.18: Umsortierung

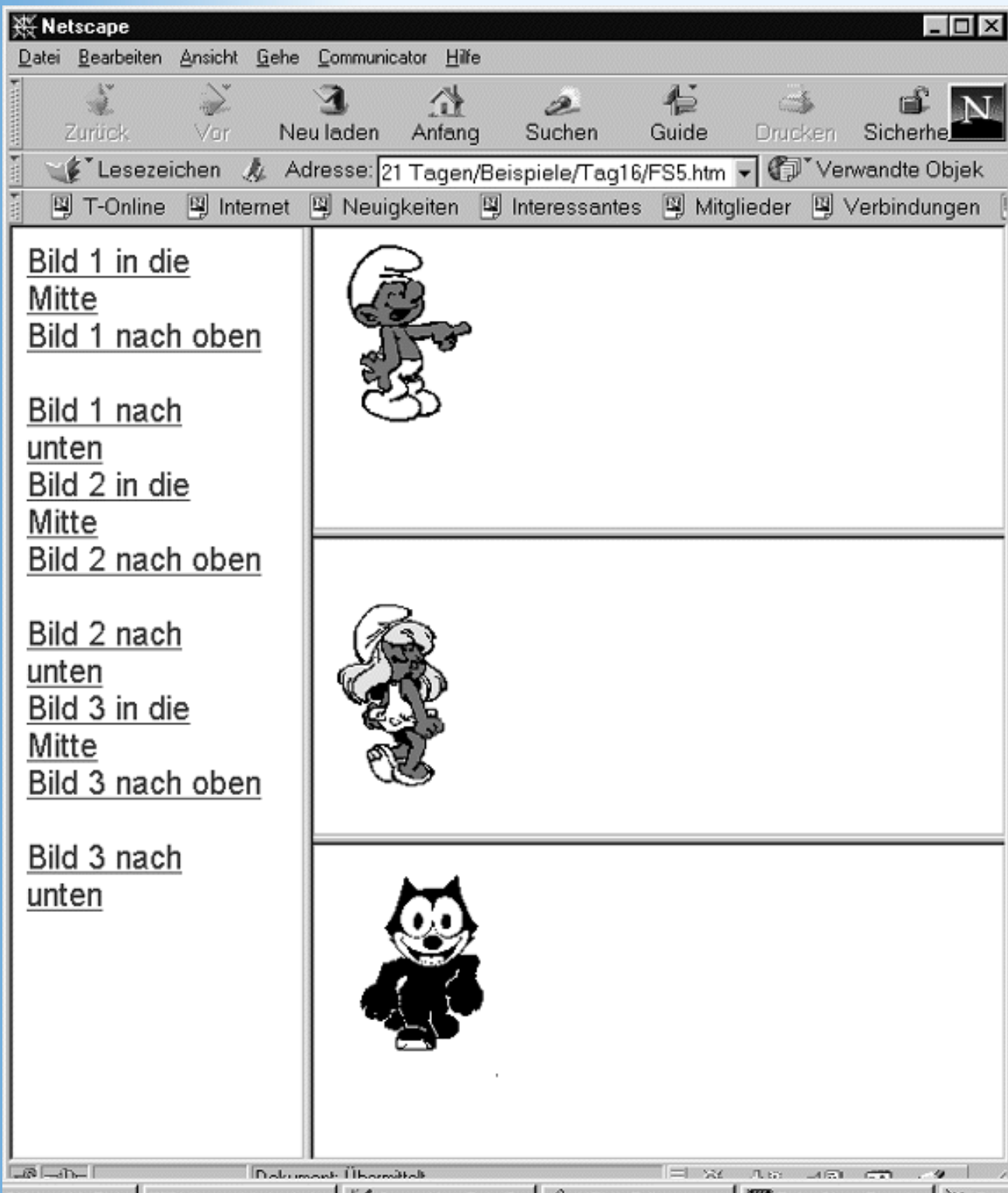


Abbildung 16.19: Die Anordnung beim Laden des Framesets



Die target-Angabe `_top` (einer der reservierten Fensternamen) sollte immer dann zu einem Verweis gehören, wenn ein Link zu Seiten erfolgt, die nicht zu Ihrem Projekt gehören. Ansonsten erscheinen bei einem Besucher Ihrer Seiten unter Umständen auch die Folgeseiten von anderen Projekten in der von Ihnen definierten Framestruktur.

JavaScript auf Frames anwenden

Natürlich können Sie mit JavaScript auf Frames zugreifen und diese beeinflussen. Dazu steht Ihnen das Objekt `frame` zur Verfügung. Über das Objekt `frame` - einem Unterobjekt von `window` - haben Sie vollen Zugriff auf Frames, die in einer

HTML-Datei definiert sind.

Eine Objektinstanz von `frame` wird automatisch erzeugt, wenn der Browser ein Frameset in einer HTML-Datei vorfindet. Die Instanz wird in einem Objektfeld mit Namen `frames` (beachten Sie wieder das `s`) gespeichert. Das erste Frame ist also über das Objekt `frames[0]`, das zweite über `frames[1]` usw. aus JavaScript heraus erreichbar. Die Frame-Objekte stehen in der Datei zur Verfügung, in der das Frameset definiert wird sowie in allen Dateien, die in einem Frame-Fenster des Framesets angezeigt werden.

Eine Instanz eines `frame`-Objekts können Sie nicht nur über den Index des zugeordneten Objektfeldes ansprechen, sondern ebenso über den Namen des Frame-Fensters, wie er in der HTML-Frameset-Seite angegeben wurde.

Da `frames` `window` untergeordnet ist, kann ein jedes Frame samt seiner Eigenschaften und Methoden direkt über die Punktnotation angesprochen werden. Entweder über

```
frames[n].[Eigenschaft/Methode()]
```

oder

```
[FrameName].[Eigenschaft/Methode()].
```



Wie beim Objekt `window` schon erwähnt gibt es für Frames reservierte Schlüsselwörter für Fensterreferenzen, über die Sie vorhandene Frames ansprechen können. Über `parent` können Sie innerhalb eines Frame-Projekts den Namen des obersten Fensters, in dem das Frameset definiert wurde, ansprechen. Einen ähnlichen Zweck hat die Angabe `top`. Sie bezeichnet im Falle von Frames in den meisten Fällen dasselbe Frame wie `parent`. Bei Verschachtelungseffekten können Sie mit `top` das oberste Frame direkt ansprechen. Letzteres ist meistens der Fall, denn wenn Sie aus einem Frame heraus agieren, müssen Sie in der Regel über die oberste Ebene den Zugriff steuern.

Die Eigenschaften und Methoden von `frame`

Das `frame`-Objekt wird in JavaScript nur als (besonderes) `window`-Objekt gesehen und hat dementsprechend alle Methoden und Eigenschaften eines solchen. Dennoch, es gibt kleine Unterschiede, die sich aus dem Kontext eines Frames ergeben und die Funktionalität von Eigenschaften und Methoden einschränken (innerhalb von einem Frameset und nicht unabhängig von anderen Frames dort). Das `frame`-Objekt besitzt zusätzlich die nachfolgenden Eigenschaften:

Eigenschaft	Beschreibung
<code>length</code>	Die Angabe der Anzahl der Frames in einem Frameset
<code>name</code>	Bei einem Frame der Name eines Frame-Fensters

Tabelle 16.5: Eigenschaften von `frame`

Das Objekt `frame` erbt als Unterobjekt von `window` dessen Methoden und stellt keine weiteren zur Verfügung.

Beispiel 13:

Eine der interessantesten Anwendungen von JavaScript in Zusammenhang mit Frames ist das gleichzeitige Aktualisieren von mehreren Frames mit einem Mausklick. Das wollen wir im nachfolgenden Beispiel machen. Dazu überarbeiten wir unser Beispiel 11 entsprechend. Zuerst brauchen wir eine etwas veränderte Frameset-Datei (`FSJS1.htm`):

```
<HTML>
<FRAMESET cols="30%, 70%">
  <FRAME src="HyperJS.htm">
  <FRAMESET rows="1%, 1%, 1%">
    <FRAME src="bild1.htm" name="eins">
    <FRAME src="bild2.htm" name="zwo">
    <FRAME src="bild3.htm" name="drei">
  </FRAMESET>
</FRAMESET>
<NOFRAMES>
```

Ihr Browser unterstützt keine Frames!

</NOFRAMES>

</HTML>

Dann überarbeiten wir die Datei mit den Links (HyperJS.htm). Dabei sprechen wir in den open()-Anweisungen die einzelnen Frames über den im Frameset definierten Namen an:

<HTML>

<HTML>

<SCRIPT language="JavaScript">

function a()

```
{
open("bild1.htm","eins");
open("bild2.htm","zwo");
open("bild3.htm","drei");
}
```

function b()

```
{
open("bild3.htm","eins");
open("bild1.htm","zwo");
open("bild2.htm","drei");
}
```

function c()

```
{
open("bild2.htm","eins");
open("bild3.htm","zwo");
open("bild1.htm","drei");
}
```

</SCRIPT>

<BODY>

Permutation 1

Permutation 2

Permutation 3

</BODY>

</HTML>

Laden Sie die Frameset-Datei und testen Sie die Links. Es werden immer alle drei Frames gleichzeitig aktualisiert.

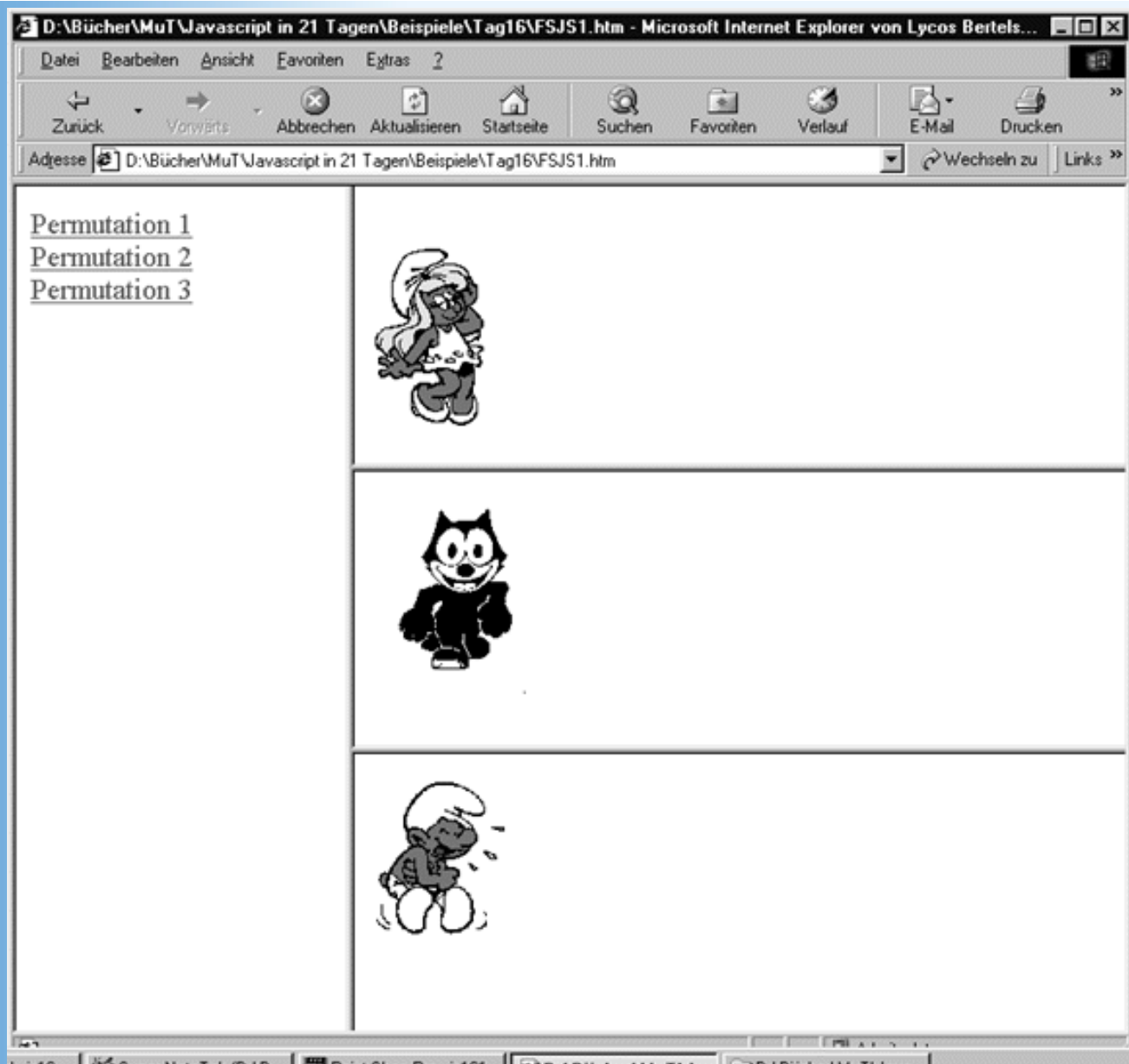


Abbildung 16.20: Umsortierung nach einem Klick

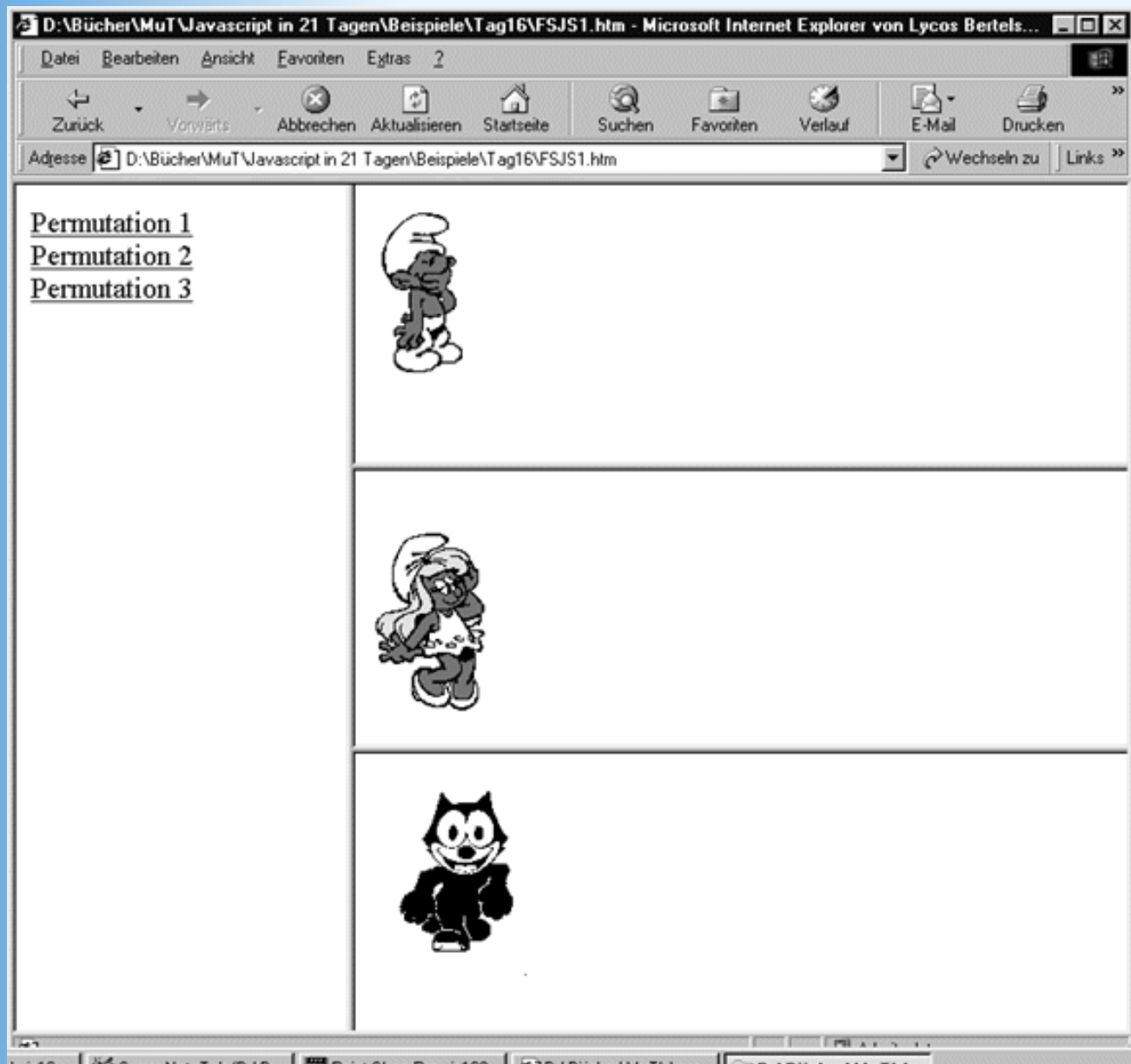


Abbildung 16.21: Die Anordnung beim Laden des Framesets



Vielleicht denken Sie, dass man auch so etwas machen kann, wie eine Seite per HTML-Hyperlink aufrufen, und diese lädt dann automatisch per <META>-Tag eine weitere Seite. Ich muss Sie enttäuschen. Das ist nicht möglich. Der <META>-Tag arbeitet nicht mit der target-Angabe zusammen.

Das zweite Beispiel zum Einsatz von JavaScript und Frames verwendet die Eigenschaft name, indem diese in einer Datei ausgegeben wird. Die Abfrage erfolgt diesmal explizit über die Objektfelder frames[], wobei beachtet werden sollte, dass man von top aus agieren muss.

Beispiel 14:

Zuerst brauchen wir wieder eine neue Frameset-Datei (FSJS2.htm):

```
<HTML>
<FRAMESET cols="30%,70%">
  <FRAME src="info.htm" name="info">
  <FRAMESET rows="1%,1%,1%">
    <FRAME src="bild1.htm" name="eins">
    <FRAME src="bild2.htm" name="zwo">
    <FRAME src="bild3.htm" name="drei">
  </FRAMESET>
</FRAMESET>
```

```
</FRAMESET>
<NOFRAMES>
Ihr Browser unterstützt keine Frames!
</NOFRAMES>
</HTML>
```

Beachten Sie, dass jetzt alle Frames das HTML-Attribut name gesetzt haben. Als neue Datei erstellen wir die Datei Info.htm:

```
<HTML>
<HTML>
<SCRIPT language="JavaScript">
function welcheFrames()
{
document.write("<H2>Die geladenen Framenamen</H2>");
document.write(top.frames[0].name);
document.write("<BR>");
document.write(top.frames[1].name);
document.write("<BR>");
document.write(top.frames[2].name);
document.write("<BR>");
document.write(top.frames[3].name);
}
</SCRIPT>
<BODY onLoad="welcheFrames()">
</BODY>
</HTML>
```

Laden Sie die Frameset-Datei. Die in der Frameset-Datei angegebenen Namen werden im linken Frame ausgegeben.

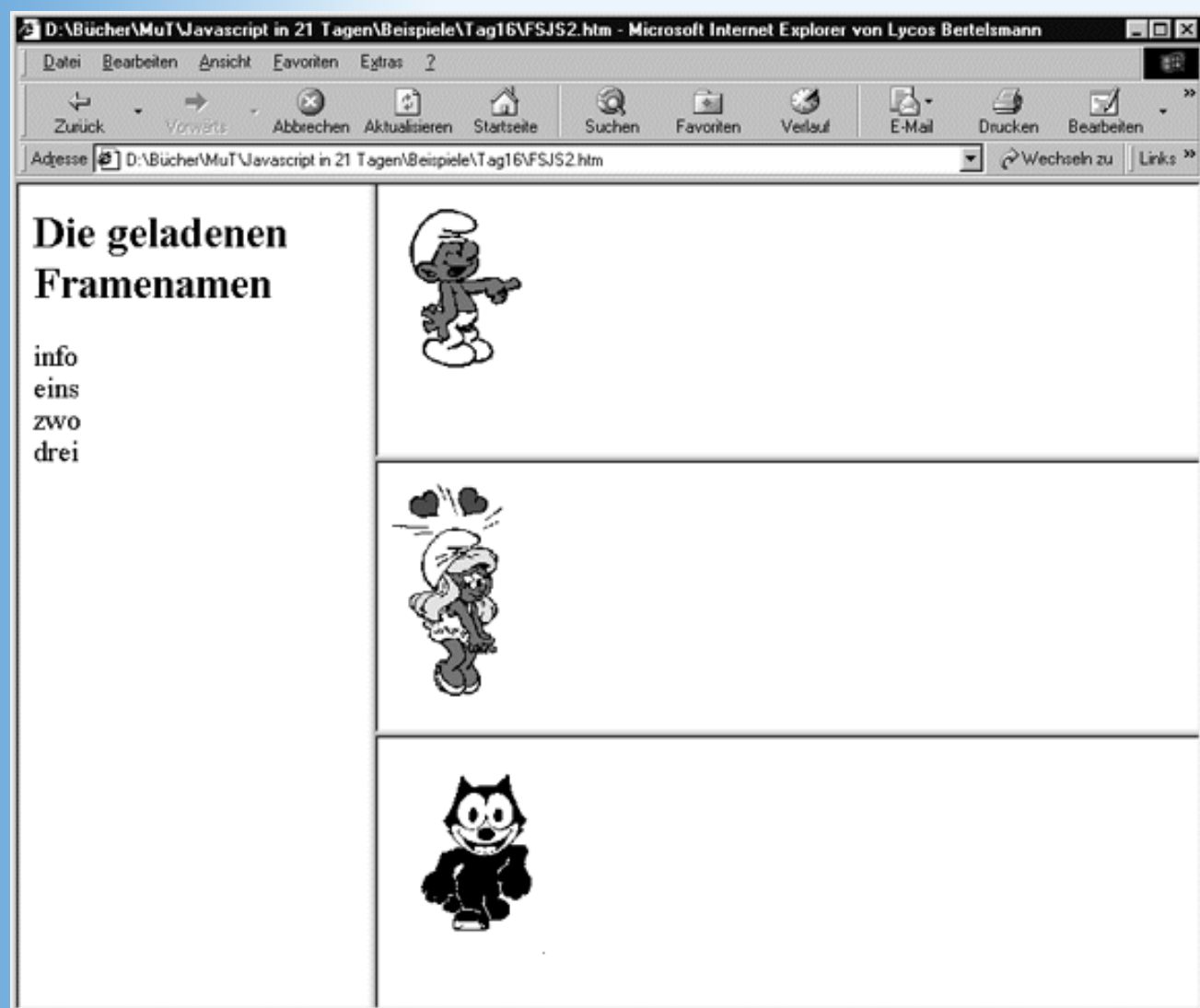


Abbildung 16.22: Die Anordnung beim Laden des Framesets



Morgen werden wir noch ein weiteres Beispiel mit Frames durchspielen. Dort wird direkt aus einer in einem Frame geladenen Datei in ein anderes Frame geschrieben.

Zusammenfassung

Am heutigen Tag haben wir mit dem window-Objekt und seinem eng verwandten Unterobjekt frame wichtige Objekte durchgesprochen, die in nahezu allen JavaScripten eine Rolle spielen. Das Objekt window ist auf der obersten Objektebene in der Objekthierarchie angesiedelt und beinhaltet alle diejenigen Objekte als Unterobjekte, die mit der Anzeige von Informationen im Browser irgendetwas zu tun haben. Das Objekt window steht über das Hauptanzeigefenster eines Browsers immer zur Verfügung. Zusätzliche Instanzen des Objekts window werden in der Regel mit der window-Methode open() erzeugt. Das Objekt stellt viele der Methoden bereit, die wir in den vorherigen Tagen einfach verwendet haben. Von alert() über prompt() hin zu confirm().

Über das Objekt frame haben Sie Kontrolle über Frames, einer Technik zur Aufteilung des Anzeigebereichs eines Browsers in einzelne Segmente, die unabhängig von einander mit Inhalt gefüllt werden können. JavaScript erlaubt den gezielten Zugriff auf jedes einzelne Frame. Die Frame- Technologie besteht einmal aus den Frames selbst, welchen den Anzeigebereich des Browsers gliedern und dennoch eigenständige Fenster mit allen HTML-Darstellungsmöglichkeiten bilden. Diese einzelnen Frames sind zusammen in einer umgebenden Struktur - dem so genannten Frameset - enthalten. Diese Frameset-Datei ist diejenige, welche in den Browser geladen wird. Sie enthält nur Informationen über Namen, Größe und Position der einzelnen im Anzeigefenster enthaltenen Frames. Insbesondere ist der eigentliche Inhalt der Frames nicht in der Frameset- Datei notiert. Eine HTML-Datei, welche als Frameset-Datei für eine Frame- struktur verwendet werden soll, hat einen anderen Aufbau als eine gewöhnliche HTML-Datei. Es wird vor allen Dingen kein <BODY>-Tag mehr benötigt. Dessen Funktion wird von dem <FRAMESET>-Container übernommen. Der Zugriff auf Frames aus JavaScript auf Frames erfolgt über das Objekt frame. Eine Objektinstanz von frame wird automatisch erzeugt, wenn der Browser ein Frameset in einer HTML-Datei vorfindet. Die Instanz wird in einem Objektfeld mit Namen frames gespeichert. Eine Instanz eines frame-Objekts können Sie nicht nur über den Index des zugeordneten Objektfeldes ansprechen, sondern ebenso über den Namen des Frame-Fensters, wie er in der HTML-Frameset-Seite angegeben wurde.

Fragen und Antworten

Frage:
Das window-Objekt hat neben frame noch zahlreiche Unterobjekte. Was ist mit diesen?

Antwort:
Wir werden die anderen Objekte wie location, history oder document die nächsten Tage genauer behandeln.

Frage:
Die Browser verhalten sich bei der Festlegung von Größe oder Position von Fenstern nicht immer gleich. Wie kann man dem begegnen?

Antwort:
Trennen der Browser-Welten. Es ist aus zahlreichen Gründen unabdingbar, die unterschiedlichen Browser mit verschiedenen Webseiten, Scripten und Style Sheets zu bedienen, wenn man etwas über einfaches HTML hinaus geht. Wir werden auch das in den nächsten Tagen behandeln.

Frage:
Gibt es zu der open()-Methode noch eine Alternative?

Antwort:
Ja. Sie können mit dem location-Objekt - einem Unterobjekt von window - arbeiten. Bitte im Chor ;-) - darauf gehen wir noch in den nächsten Tagen ein.

Workshop

Experimentieren Sie mit den Beispielen in dem Kapitel. Lassen Sie sich beispielsweise den Namen des Folgefensters in Beispiel 1 per JavaScript ausgeben. Oder verändern Sie Werte von Variablen, Dateien usw.

Nutzen Sie bei der open()-Methode die in den Beispielen nicht verwendeten Optionen. Testen Sie ebenso die in den Beispielen nicht verwendeten Methoden und Eigenschaften.

Versuchen Sie in den Beispielen der vorherigen Tage die Methoden zu finden, welche zu window gehören. Wie wurden sie eingesetzt?

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was ist der Unterschied zwischen frame und frames[]?
2. Geht das?
`open( "", "" );`
3. Welcher Browser reagiert bei self.close(); mit einer Warnmeldung - der Navigator oder der Internet Explorer?
4. Stimmt das?

```
<FRAMESET rows="55%, 1%, 1%, 5%">
```

5. Was stimmt hier nicht?

```
open("bild1.htm",  
"DiesistkeinleeresFenster",  
"width=300;  
height=250;  
screenX=100;  
screenY=100" );
```

6. Für was steht self?
7. Was macht die Methode home()?
8. Stimmt das?

```
<FRAMESET cols="25%, 75%">  
<FRAME src= "Li.htm" name="links" >  
<FRAME src= "Re.htm" name="rechts" >  
</FRAMESET>
```

9. Versuchen Sie den Fehler zu finden:

```
<FRAMESET cols="30%, 70%">  
  <FRAME src="li.htm">  
  <FRAMESET rows="30%, 70%">  
    <FRAME src="131.gif">  
  <FRAMESET cols="50%, 50%, 50%" frameborder=0>  
    <FRAME src= "li.htm" noresize>  
    <FRAMESET cols="50%, 50%">  
      <FRAME src= "132.gif">  
      <FRAME src= "132.gif">  
    </FRAMESET>  
    <FRAME src= "131.gif" scrolling=no>  
  </FRAMESET>  
</FRAMESET>  
  <FRAME src="132.gif">
```

</FRAMESET>
</FRAMESET>

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

1

Selbst die kann man in Opera deaktivieren.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 3

Tag 17

Globale Ereignisbehandlung

Nachdem der gestrige Tag ziemlich anstrengend und umfangreich war, wird der heutige Tag ein kleines, aber feines Thema untersuchen. Der Tag 17 soll abermalig im Zeichen der Ereignisbehandlung unter JavaScript stehen. Bisher haben wir JavaScripte nur über das Laden einer Webseite oder die Reaktion auf Ereignisse über Event-Handler ausgelöst. An verschiedenen Stellen wurde jedoch schon darauf hingewiesen, dass es auch anders geht. Es gibt im Konzept der Ereignisbehandlung ein spezielles Objekt, das dafür eigens bereitsteht. Wir müssen also das Objektkonzept mit dem Ereigniskonzept zusammenbringen.

Wenn man das tut, hat das einige weitreichende Konsequenzen. Die Ereignisüberwachung wird beispielsweise direkt in JavaScript programmiert. Die Verbindung von HTML (über die dort zugehörnden Event-Handler) und JavaScript in einer Zweckehe wird aufgelöst. Statt dessen wird von speziellen JavaScript-Routinen das Auftreten eines Ereignisobjekts überwacht. Je nach Art des Ereignisses kann der JavaScript-Interpreter darauf reagieren.

Das event-Objekt in JavaScript

Wenn in einer Webseite ein Ereignis auftritt, wird dabei seit JavaScript 1.1 ein eigenes Objekt generiert - das event-Objekt, welches ein Unterobjekt von window ist. Das event-Objekt beinhaltet verschiedenste Informationen über ein aufgetretenes Ereignis in einer Webseite samt der Randbedingungen, welche dabei eine Rolle gespielt haben.

Sie müssen sich vergegenwärtigen, dass solche event-Objekte während der Anzeige einer Webseite im Browser permanent im Hintergrund erzeugt werden. Unabhängig davon, ob Sie diese in Ihrem Script zur Kenntnis nehmen oder nicht. Sie sind einfach da.

Das wollen wir erst einmal mit einem Fitnessvideo vergleichen, wo ein Vorturner bestimmte Übungen demonstriert. Wenn das Ereignis »Liegestütze machen« auftritt, werden Sie es sehen, falls Sie gerade auf den Bildschirm schauen. Ob Sie darauf reagieren oder nicht, das ist Ihre Sache und beeinflusst den Ablauf des Videos nicht. Es kümmert den Vorturner nicht weiter - er hat keine Ahnung davon, ob und wie Sie darauf reagieren. (Wie sollte er auch - das Video ist ja schon lange vor der Sendung gedreht worden.) Er wird sein Programm runterspielen und immer neue Ereignisse abfeuern¹.

Wechseln wir wieder zu JavaScript. In einer Webseite ist eine Situation, welche ein solches event-Objekt erzeugt, beispielsweise der Klick mit der Maus auf irgendeinen Bestandteil der Webseite.

Ein Mausklick erzeugt ein event-Objekt, welches folgende Informationen enthält:

- Die verwendete Taste,
- eventuell gedrückte Zusatz Tasten (Strg), (Alt), (Alt-Gr), (^a) und
- die Koordinaten, wo der Klick erfolgte.

Andere event-Objekte, die bei anderen Ereignissen erzeugt werden, beinhalten natürlich andere Informationen, welche dem Ereignis angepasst sind. So könnte etwa bei einem Tastendruck die gedrückte Taste als abzufragende Information bereitstehen.



Das Entstehen eines Ereignisses wird in der englischsprachigen Literatur bei vielen Ereigniskonzepten als »to fire an event« beschrieben - abfeuern. Den Knall nimmt jedermann zur Kenntnis, aber inwieweit und wie sich wer darum kümmert, ist nicht festgelegt.

Nun ist es sicher nicht Sinn und Zweck eines Ereignismodells, dass permanent Ereignisse abgefeuert und diese nie beachtet werden (sprich darauf reagiert wird). Bei bestimmten Situationen will man ja explizit eine Reaktion haben, d.h. es soll ein Script aufgerufen werden.

Überarbeiten wir unseren Vergleich mit dem Fitnessvideo ein wenig. Es sei nun kein Video mehr, sondern ein Kurs in einem Fitnessstudio. Der »Fitness-Instructor« wird sehr wohl eine Rückmeldung durch die Teilnehmer zur Kenntnis nehmen können. Wenn es dem Menschen nun ziemlich egal ist, ob alle Teilnehmer mitmachen oder nicht und er nur dann reagiert, wenn Teilnehmer sich durch eine Botschaft bemerkbar machen², sind wir der Idee eines Ereignismodells schon recht nahe.

Ein event-Objekt wird, wenn man unter JavaScript darauf reagieren will, an einen Mechanismus zum Behandeln von event-Objekten in Form einer Message (Botschaft) weitergereicht.

Auf welche Art und Weise auf welches erzeugte Ereignisobjekt dann reagiert werden soll, wird durch ein abstraktes Modell geregelt - dem Ereignismodell. Darin sind sowohl die Verhaltensweisen auf Ereignisse festgelegt, die konkrete Syntax zur Umsetzung, aber auch erst recht diejenigen Ereignisse, welche überhaupt zu Kenntnis genommen werden³.



Um gleich die größte Problematik anzuschneiden: Können Sie sich vorstellen, dass man sich bei Microsoft und Netscape auf ein gemeinsames, kompatibles Konzept geeinigt hat? Ja? Und wo haben Sie den Osterhasen genau gesehen? Selbstverständlich⁴ sind die Konzepte nicht vollkommen kompatibel. Im Prinzip ist bei beiden Varianten vom Konzept her fast alles gleich, aber von der Syntax so gemacht, dass auf keinen Fall der jeweilig gegnerische Browser damit klarkommt.

Prinzipiell unterscheidet man im Ereignismodell von Netscape zwischen fünf verschiedenen Kategorien von Ereignissen, die im Laufe des Lebens einer Webseite auftreten können:

- Dokumentenereignisse. Darunter versteht man die Ereignisse, welche beim Laden und Entfernen einer Seite aus dem Browser entstehen. Dies entspricht den Ereignissen, die mit den Event-Handlern onLoad und onUnload abgefragt werden (wenn man über Event-Handler

arbeitet).

- Formular-basierende Ereignisse. Diese Ereignisse werden erzeugt, wenn ein Anwender mit der Seite über ein Formular interagiert, d.h. den Fokus in einer Formularmaske verschiebt, eine Auswahl ändert oder ein Formular abschickt. Das entspricht den vier Event-Handlern onFocus zum Fokussieren und onBlur zum Fokus an ein anderes Element weiterreichen bzw. onSelect und onSubmit.
- Referenz-basierende Ereignisse. Sie werden im Netscape-Modell ausgelöst, wenn der Anwender auf einen Hyperlink in einer Webseite klickt oder den Mauszeiger darüber bewegt. Das entspricht den Event- Handlern onClick und onMouseOver.
- Elementbezogene Ereignisse. Sie geben den Status eines Elements in einer Webseite an.
- Fensterbezogenen Ereignisse. Diese treten auf, wenn mehr als eine Fensterinstanz des Navigator angezeigt wird. Für jede Instanz stehen dann verschiedene Ereignisse zur Verfügung.



Für die konkrete Anwendung ist es nicht von Interesse, zu welcher Kategorie ein Ereignis gehört. Sie müssen es nur korrekt anwenden.

Auswerten eines event-Objekts und die Reaktion

Da Ereignisobjekte während der Lebenszeit einer Webseite permanent im Hintergrund entstehen, stehen sie auch überall im Code zur Verfügung. Es gibt verschiedene Konzepte, auf welche Weise so ein event-Objekt angesprochen werden kann. Sie können es einfach über seinen Namen (event) ansprechen. Dabei müssen Sie noch angeben, von welchem Objekt aus das event-Objekt erzeugt wurde. Die Syntax dazu ist die in der objektorientierten Programmierung übliche Punktnotation: `[Objekt].event`

Beispiel (Webseite):
`document.event`

Sie wollen bei einer Auswertung in der Regel dann natürlich eine ganz bestimmte Information eines Ereignisses verwenden. Dies sind bei einer objektorientierten Betrachtungsweise einfach die Eigenschaften des Objektes. So können Sie etwa einen Mausklick wie folgt ansprechen:
`document.event.CLICK`

Dies versteht vom Prinzip her sowohl der Internet Explorer als auch der Netscape Navigator. Dummerweise setzen sie aber die konkrete Implementierung völlig unterschiedlich um. Auch in der Syntax zur Überwachung von Ereignissen (wie schon bei der Unterstützung der Event- Handler) unterscheiden sich beide Browser.



Die Auswertung eines event-Objektes wird Auffangen genannt.

Globales Auffangen der Ereignisse

Sowohl das Netscape-Objektmodell als auch das Microsoft-Objektmodell erlauben das globale Auffangen von event-Objekten. Sie können damit an zentraler Stelle in einer Webseite oder gar einem ausgelagerten Script behandelt werden. Dies ist sehr sinnvoll, wenn eine konsequente strukturelle Trennung einer Seite erreicht werden soll. Globales Eventhandling ist damit eine logische Konsequenz der Bemühungen, Layout, Inhalt und Scriptfunktionalitäten in einer Webseite zu entflechten.

Leider sind jedoch - wie mehrfach erwähnt - die von den beiden großen Internet-Protagonisten angewandten Techniken nicht kompatibel⁵. Der Internet Explorer wird beim Laden vom Javascript-Code, der dem Netscape- Objektmodell entsprechend programmiert wurde, in der Regel eine Fehlermeldung bringen oder es ignorieren. Analog ist die Situation im Navigator beim Laden von JavaScript-Code nach der Microsoft-Variante - Ignoranz oder Fehler.

Das Netscape-Ereignismodell

Unter dem Netscape-Modell können Sie über eine zum window-Objekt gehörende Methode bewirken, dass alle dort als Parameter spezifizierten Ereignisse eines spezifizierten Typs innerhalb eines bestimmten Objektes (etwa einer ganzen Webseite) zentral behandelt werden. Dies ist die Methode `captureEvents()`, die wie folgt angewandt wird:

```
[Objekt].captureEvents([Ereignistyp])
```

Wenn so eine Zeile im JavaScript-Code notiert wird, werden alle spezifizierten Ereignisse, die bei dem vorangestellten Objekt auftreten, aufgefangen und an eine zentrale Stelle weitergeleitet - an die zentrale Ereignisbehandlungsroutine. Dies ist in diesem Fall einfach eine Funktion, welche als Argument das erzeugte event-Objekt mit der gewünschten Eigenschaft verwendet. Als Ereignistyp können die bei einem Objekt von Netscape unterstützen Event-Handler verwendet werden.



Das Ereignismodell von Netscape ist in Zusammenhang mit einem allgemeineren Konzept zu sehen, mit dem Netscape dynamische Webseiten realisiert - dem Layer-Konzept (Schichten), das Netscape mit der Version 4.0 seines Browsers einführte und als Sprachelement von HTML versteht. Mit Hilfe des <LAYER>-Containers ist es möglich, beliebige Bereiche einer HTML-Datei exakt zu kontrollieren und Elemente zu positionieren. Neben der auf HTML beschränkten Funktion von Layern verwendet Netscape diese als Zielobjekt für Manipulationen unter JavaScript. Wir werden diese Technik jedoch nicht weiter verfolgen, da sie ausschließlich auf Netscape-Browser ab dem Navigator 4.0 beschränkt ist und vor allen Dingen vom W3-Konsortium nicht gestützt wird. Wahrscheinlich wird Netscape über kurz oder lang die Technik einstellen müssen.

Das Netscape-Ereignismodell unterstützt die folgenden Events bzw. Event- Handler in den beschriebenen Situationen:

Ereignis	Zugehörige Elemente	Abgefeuert bei	Event-Handler
Abort	Images	Abbruch des Ladens eines Bildes	onAbort

Blur	Browserfenster und alle Formularelemente	Entfernen des Eingabefokus durch den User	onBlur
Change	Textfelder, Textareas, Select-Listen	Wertveränderung des Elements durch den User	onChange
Click	Button, Radiobutton, Checkboxen, Submit-Button, Reset-Button, Links	User-Klick auf ein Element oder Link	onClick
DragDrop	Das Browserfenster	Der User zieht ein Objekt in das Browserfenster und lässt es los	onDragDrop
Error	Images und Browserfenster	Fehler beim Laden eines Dokuments oder Bilds	onError
Focus	Browserfenster und alle Formularelemente	Setzen des Eingabefokus durch den User auf ein Element	onFocus
KeyDown	Dokumente, Bilder, Links, Textbereiche	Der User drückt eine Taste herunter	onKeyDown
KeyPress	Dokumente, Bilder, Links, Textbereiche	Der User hält eine Taste gedrückt	onKeyPress
KeyUp	Dokumente, Bilder, Links, Textbereiche	Der User lässt eine Taste los	onKeyUp
Load	Dokumentenbody	Laden einer Seite	onLoad
MouseDown	Dokumente, Button, Links	Drücken einer Maustaste	onMouseDown
MouseMove	Keine Default-Situation	Mausbewegung	onMouseMove
MouseOut	Areas, Links	Mauszeiger bewegt sich aus dem Bereich der client-side-Imagemap oder dem Link	onMouseOut
MouseOver	Links	Bewegung des Mauszeigers auf dem Link	onMouseOver
MouseUp	Dokumente, Button, Links	Loslassen einer Maustaste	onMouseUp
Move	Browserfenster	Der User oder ein Script bewegt ein Fenster	onMove
Reset	Formulare	Der User klickt auf einen Resetbutton	onReset
Resize	Browserfenster	Der User oder ein Script verändert eine Fenstergröße	onResize
Select	Textfelder oder Textbereiche	Der User selektiert in einem Formular ein Eingabefeld	onSelect
Submit	Formulare	Der User schickt ein Formular ab	onSubmit
Unload	Dokumentenbody	Der User verlässt eine Seite	onUnload

Tabelle 17.1: Events bzw. Event-Handler des Netscape-Ereignismodells

Grundsätzlich wird der Bezeichner eines Event-Handler aus dem Namen des Events selbst gebildet. Es wird einfach an vorangestellt.

Das Verfahren zum Auffangen und Behandeln von Ereignisobjekten beinhaltet im Wesentlichen vier Grundmethoden, die (bis auf eine Ausnahme) beim window-, document- und layer-Objekt spezifiziert sind. Es ist einmal die bereits angesprochene Methode

`captureEvents()`

sowie

`releaseEvents()`

womit das Ignorieren eines aufgefangenen Ereignisobjekts vom angegebenen Typ definiert werden kann,

`routeEvent()`

zum Weiter- bzw. Umleiten eines aufgefangenen Ereignisobjekts zu einem angegebenen Zielobjekt und

`handleEvent()`

womit das aufgefangene Ereignisobjekt explizit behandelt werden kann (diese Methode ist beim layer-Objekt nicht vorhanden).

Grundsätzliche Vorgehensweise bei dieser Form der Ereignisbehandlung ist folgende:

1. Das Auffangen eines Events wird ermöglicht.
2. Die Definition des Event-Handlers erfolgt.
3. Die Registrierung des Event-Handlers erfolgt.

Um das Auffangen eines Events zu ermöglichen, wird bei dem entsprechenden Objekt (etwa window) eine Anweisung der folgenden Art notiert:

```
window.captureEvents(Event.CLICK);
```

Das Argument der Methode `captureEvents()` ist eine Eigenschaft von dem event-Objekt, die den Typ des Ereignisses spezifiziert. Der Aufbau ist einfach das vollständig groß geschriebene Ereignis, das per Punktnotation Event nachgestellt wird.

Um mehrere Ereignisse aufzufangen, kann als Argument eine Liste von solchen Eigenschaften angegeben werden. Die Eigenschaften werden mit dem Zeichen `|` separiert. Um beispielsweise die Ereignisse Click, MouseDown und MouseUp aufzufangen, muss Folgendes notiert werden:

```
window.captureEvents(  
Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP)
```

Die Definition der Event-Handler bedeutet, man definiert eine Funktion, die das Ereignis behandelt. Das Argument `e` ist das Ereignisobjekt für das Ereignis. Beispielsweise so:

```
function BehandleEreignis(e)  
{  
  ...  
}
```

Sinnvollerweise sollte die Funktion einen boolschen Rückgabewert liefern, damit auf eine erfolgreiche Behandlung oder aber auch einen Fehler bei der Behandlung reagiert werden kann. Ansonsten kann

man hier sehr differenzierte Vorgänge angeben, worauf wir aber verzichten wollen.

Das abschließende Registrieren des Event-Handlers läuft so ab:

```
window.onClick = BehandleEreignis;
```

Das Ereignis, für das eine bestimmte Funktion als Event-Handler agieren soll, wird dem zugehörigen Objekt nachgestellt notiert. Auf der rechten Seite der Anweisung wird der Bezeichner der Event-Handler-Funktion zugewiesen. Dabei sollte wieder beachtet werden, dass dort keine Klammern auftauchen. Event-Handler sind Funktionsreferenzen. Deshalb dürfen bei der Zuordnung keine (!) Klammern angegeben werden. Ansonsten würde die Funktion unmittelbar bei der Zuordnung ausgeführt und das Ergebnis undefined zugewiesen (siehe dazu auch Tag 11, Beispiel 11).

Beispiel 1:

Erstellen wir einfach einmal einen vollständigen Code, der eine solche zentrale Ereignisbehandlung unter dem Netscape-Objektmodell bewirkt (Name der Datei: Event1.htm).

```
<HTML>
<SCRIPT language="JavaScript">
document.captureEvents(Event.MOUSEOVER);
function BehandleEreignis(e)
{
meldung =
"An der Position " + e.layerX +
", " + e.layerY + " trat das Ereignis "
+ e.type + " auf.";
alert(meldung);
return true;
}
document.onMouseOver = BehandleEreignis;
</SCRIPT>
<BODY>
<A href="Event1.htm">
Dies ist ein Hyperlink
</A>
<P>
<A href="Event1.htm">
Dies ist ein zweiter Hyperlink
</A>
</BODY>
</HTML>
```



Abbildung 17.1: Das beim Überstreichen ausgelöste Ereignis

Wenn Sie die Datei in den Navigator laden und den Mauszeiger über einen der beiden Links bewegen, wird ein Mitteilungsfenster aufgehen (dies bewirkt die Methode `alert(meldung);`), in der die aktuelle Position des Mauszeigers und der Ereignistyp angezeigt werden. In der Zeile

```
meldung = "An der Position " + e.layerX + ", " +  
e.layerY + " trat das Ereignis " + e.type + " auf.";
```

werden dazu diverse Eigenschaften des event-Objektes (X- und Y- Koordinaten sowie der Typ des Ereignisses) abgefragt und zu einer Meldung zusammengesetzt. Beachten Sie die veränderten Positionsangaben, wenn der Mauszeiger an einer anderen Stelle das Ereignis auslöst. Wenn Sie sich diese Datei im Internet Explorer anschauen, werden Sie eine Fehlermeldung bekommen.

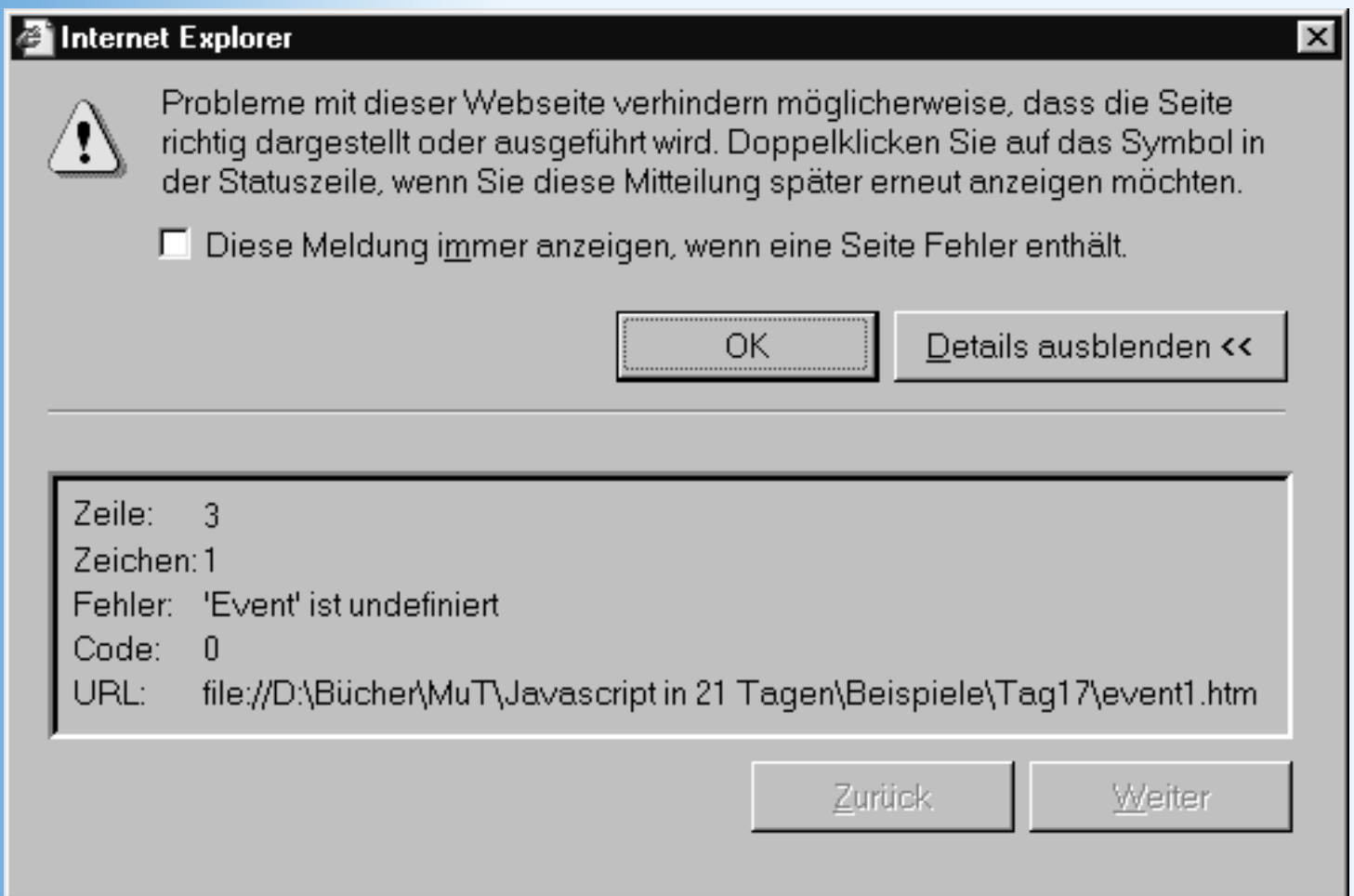


Abbildung 17.2: Die Fehlermeldung im IE.



Es kann durchaus mehr als eine Behandlungsfunktion definiert werden, wenn entsprechend in der `captureEvents()`-Methode zugehörige Ereignisse aufgefangen werden. Das werden wir im nächsten Beispiel machen.

Die Eigenschaft des event-Objekts

Betrachten wir hier der Vollständigkeit halber die spezifischen Eigenschaften des Netscape-Ereignismodells in Zusammenhang mit dem event-Objekt.

- Die Eigenschaft `data` tritt in Zusammenhang mit dem `DragDrop`-Event auf und gibt ein Array von Strings zurück, welche die URLs der gezogenen Objekte beinhaltet.
- Die Eigenschaften `height` und `width` beinhalten die Höhe und Breite von dem Fenster oder Frame.
- Die Eigenschaften `layerX` und `layerY` beinhalten die aktuelle Breite (`layerX`) und Höhe (`layerY`) eines Objekts, wenn das Ereignis `onresize` überwacht wird, oder die horizontalen Pixel (`layerX`) und die vertikalen Pixel (`layerY`) der Cursorposition relativ zur oberen linken Ecke eines Layer-Bereichs, wenn z.B. Mausereignisse überwacht werden. Synonym dafür stehen die Eigenschaften `X` und `Y`.
- Zur Abfrage der Zusatz Tasten gibt es die Eigenschaft `modifiers`. Darin wird gespeichert, ob Zusatz Tasten wie (`ALT`), (`ALTGR`), (`CRTL`) oder (`^`) gemeinsam mit einer anderen Taste oder einem Mausklick gedrückt wurden. Dabei werden folgende Variablen des event-Objektes

verwendet:

Event.ALT_MASK wird bei der (ALT)-Taste ausgelöst

Event.CONTROL_MASK wird bei der (CTRL) -Taste ausgelöst

Event.SHIFT_MASK wird bei der (a) -Taste ausgelöst

Event.META_MASK wird bei der (ALTGR)-Taste ausgelöst

- Zur Auswertung der Cursorkoordinaten stellt Netscape zwei Varianten zur Verfügung:
- 1. eine relative Auswertungsmöglichkeit
- 2. eine absolute Auswertungsmöglichkeit

Über die Eigenschaften pageX und pageY kann auf die horizontalen Pixel (pageX) und die vertikalen Pixel (pageY) der Cursorposition relativ zur oberen linken Ecke der Seite zugegriffen werden.

Die Eigenschaften screenX und screenY enthalten die horizontalen Pixel (screenX) und die vertikalen Pixel (screenY) der Cursorposition absolut gesehen zum Bildschirm.

- Die Eigenschaft target ist eine Stringrepräsentation des Objekts, um dem ein Ereignisobjekt ursprünglich gesendet wurde (eine sinnvolle Information, wenn ein Ereignisobjekt umgeleitet wurde, was man mit der Methode routeEvent() ja machen kann).
- In der Eigenschaft which steckt nach Tastaturereignissen der dezimale Code (ASCII-/ANSI-Wert) der gedrückten Taste und bei Mausereignissen die Information, welche Maustaste gedrückt wurde.
- In der Eigenschaft type wird dokumentiert, welches Ereignis eingetreten ist. Der gespeicherte Wert ist der Name des Ereignisses ohne das vorangestellte on.

Beispiel 2:

Erstellen wir noch ein weiteres Beispiel, das diesmal mit Frames arbeitet und das Ereignis Click sowie MouseOut überwacht. In dem einen Frame befindet sich eine Datei mit einem Link, der überwacht wird. Die Ausgabe der interessanten Informationen erfolgt bei einem Klick darauf im zweiten Frame (Name der Datei: Event2.htm). Wenn Sie den Bereich des Links wieder verlassen, wird eine Mitteilung erscheinen, da auch MouseOut überwacht wird.

Sie benötigen als Erstes einen Frameset, der FS.htm heißen soll:

```
<HTML>
<FRAMESET cols="25%,75%">
<FRAME src= "Event2.htm" name="links" >
<FRAME src= "blank.htm" name="rechts" >
</FRAMESET>
<NOFRAMES>
<BODY bgcolor=cyan>
<H1 align=center>
Sorry, hier werden Frames verwendet
</H1>
</BODY>
</NOFRAMES>
<BODY bgcolor=cyan>
<H1 align=center>
Sorry, hier werden Frames verwendet
```



```
</H1>
</BODY>
</HTML>
```

Es wird im rechten Frame eine leere Datei benötigt. Das kann einfach eine Datei ohne irgendwelchen Inhalt sein, die nur unter dem Namen Blank.htm gespeichert sein muss. Erstellen Sie nun die Datei Event2.htm:

```
<HTML>
<SCRIPT language="JavaScript">
window.captureEvents(Event.CLICK | Event.MOUSEOUT);
function BehandleEreignis1(e)
{
meldung = "Hoehe: " + e.height +
  " Breite: " + e.width +
  " Modifiers: " + e.modifiers +
  " pageX: " + e.pageX +
  " pageY: " + e.pageY +
  " which: " + e.which +
  " screenX " + e.screenX +
  " screenY " + e.screenY + " !<BR> ";
top.frames[1].document.write(meldung);
return true;
}
function BehandleEreignis2(e) {
  alert("Und raus");
  return true;
}
window.onClick = BehandleEreignis1;
window.onMouseOut = BehandleEreignis2;
</SCRIPT>
<BODY>
<A href="Event2.htm">
Dies ist ein Hyperlink
</A>
</BODY>
</HTML>
```

Sie finden in dem Beispiel zwei Überwachungsfunktionen und zwei angemeldete Ereignisse. Testen Sie den Link, indem Sie darauf klicken (es kann sein, dass die Reaktion etwas dauert, da die Ereignisbehandlung nicht allzu flott und etwas ungenau erfolgt - klicken Sie bei Bedarf mehrmals). Im rechten Frame sollten Werte von diversen Eigenschaften der Ereignisobjekts angezeigt werden. Beachten Sie die Zeile

```
top.frames[1].document.write(meldung);
```

Damit können Sie via dem Frame-Objektfeld direkt in der dort geladenen Datei schreiben.

Wenn Sie den Bereich des Links mit dem Mauszeiger verlassen, erscheint das Mitteilungsfenster.

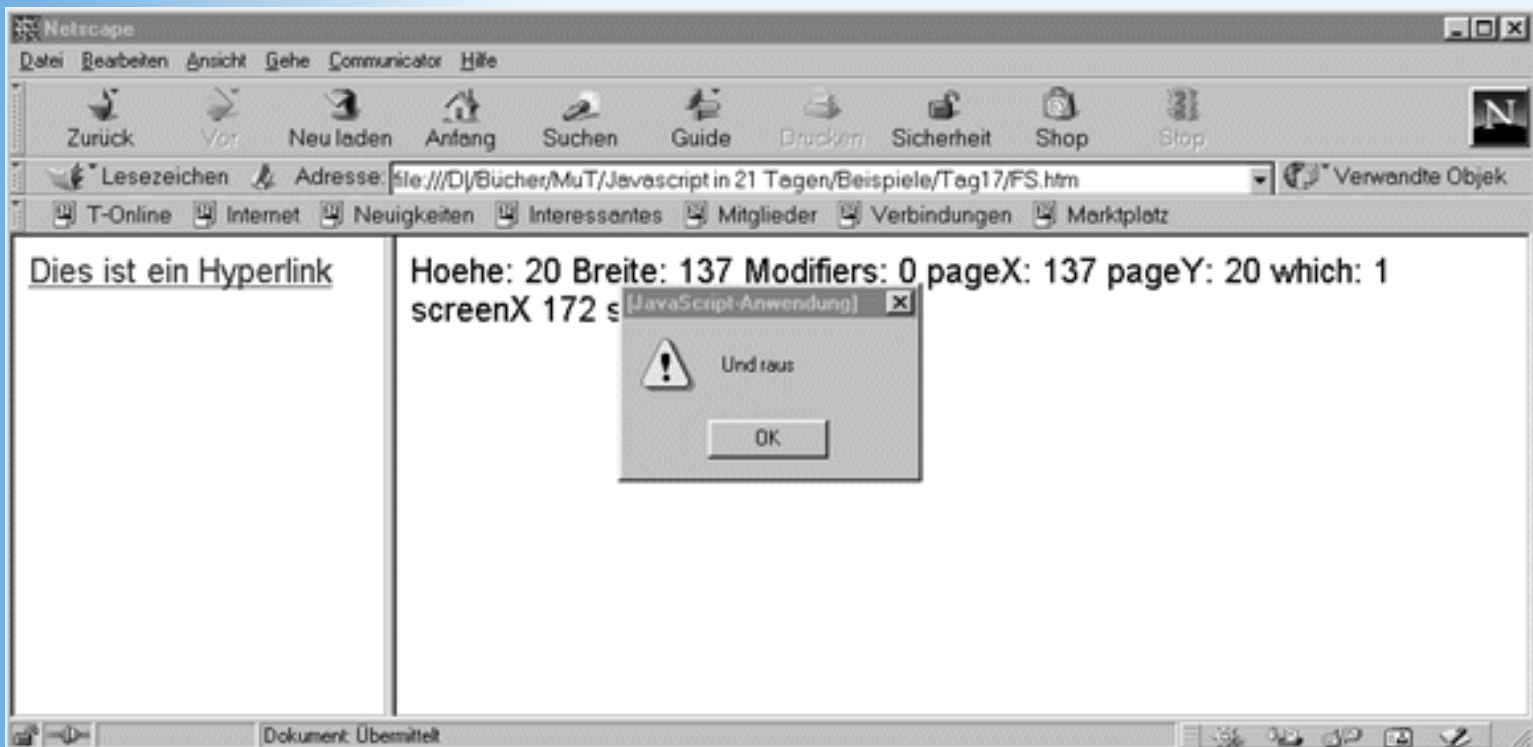


Abbildung 17.3: Die Reaktion auf beide Ereignisse

Die Microsoft-Variante

Das Microsoft-Objektmodell erlaubt die zentrale Behandlung von Ereignissen mittels der folgenden Syntax:

```
<SCRIPT FOR=[Objekt] EVENT=[Ereignis]
language="JavaScript">
...[Konkrete Anweisungen]...
</SCRIPT>
```

Als [Ereignis] geben Sie einfach die im vorherigen Abschnitt erklärten Event-Handler an (allerdings auf jeden Fall vollkommen klein geschrieben!). Mehr ist nicht notwendig, um alle Ereignisse des spezifizierten Typs von dem angegebenen Objekt zentral zu verarbeiten. Innerhalb des Scripts können Sie dann auf alle Eigenschaften des event-Objekts zugreifen, die der Internet Explorer kennt. Das gesamte Prozedere ist bedeutend unkomplizierter als bei der Netscape-Variante. Gefahren lauern aber dennoch. Schauen wir uns Beispiele an.

Beispiel 3:

Erstellen Sie die Datei Event3.htm:

```
<HTML>
<SCRIPT FOR=document EVENT=onclick
language="JavaScript">
alert("Das Klickt");
</SCRIPT>
<BODY>
<H2>
Klicken Sie hier
</H2>
```

```
<H3>
Klicken Sie da
</H3>
</BODY>
</HTML>
```

Laden Sie sie in den Internet Explorer und klicken Sie auf die Überschriften.



Abbildung 17.4: Der IE reagiert bei Klicks auf die einzelnen Elemente

Laden Sie die Datei nun in den Netscape Navigator. Es wird schnell deutlich, welche Probleme auftauchen.

1. Die Anweisung wird unerwünschterweise beim Laden der Seite abgearbeitet.
2. Dafür funktionieren die Klicks auf die Überschriften (oder auch andere Elemente) nicht.

Was ist los? Die Angabe FOR=document legt für den Internet Explorer fest, dass die zentrale Behandlung für die gesamte Webseite gilt, EVENT=onclick spezifiziert das Ereignis Mausklick. Beachten Sie bei der Spezifizierung des Ereignisses unbedingt die Groß- und Kleinschreibung! Für den Netscape Navigator sind diese Angabe unbekannt und er ignoriert sie. Dementsprechend wird er nicht den Container als verschieden von einem normalen Script-Container erkennen und die darin enthaltenen Anweisungen abarbeiten, also würde dort ein gewöhnlicher Script-Container stehen. Der Internet Explorer unterscheidet die Container - deshalb führt er die im Container notierten Anweisungen nicht beim Laden der Seite aus.



Wenn man globale Ereignisbehandlung für beide Browser-Welten sicher machen möchte, kommt man um eine Trennung nicht herum. Wir werden das morgen durchsprechen.

Betrachten wir hier als Zusammenfassung, wie beim Netscape-Modell, die spezifischen Eigenschaften des Microsoft-Ereignismodells in Zusammenhang mit dem event-Objekt.

Zur Abfrage der Zusatz Tasten stehen die boolschen Eigenschaften

altKey,
ctrlKey

und

shiftKey

bereit. Sie stehen für die Zusatz Tasten (ALT), (CTRL) oder (^a). Wenn sie gemeinsam mit einer anderen Taste oder einem Mausklick gedrückt wurden, werden sie den Wert true enthalten.

Beispiel 4:

Erstellen Sie die Datei Event4.htm:

```
<HTML>
<SCRIPT
  FOR=document
  EVENT=onclick
  language="JavaScript">
alert("Shift: " + event.shiftKey +
  " Alt: " + event.altKey +
  " Ctrl: " + event.ctrlKey);
</SCRIPT>
<BODY>
<H2>
Klicken Sie hier
</H2>
</BODY>
</HTML>
```

Laden Sie sie in den Internet Explorer und klicken Sie auf die Überschrift unter Verwendung der Zusatz Tasten.



Abbildung 17.5: Alt war gedrückt

Die Eigenschaften

`clientX`

und

`clientY`

beinhalten die Information über die horizontalen Pixel (`clientX`) und die vertikalen Pixel (`clientY`) der Cursorposition relativ zur oberen linken Ecke des Anzeigefensters, wenn ein Koordinaten-abhängiges Ereignis (etwa eine Mausektion) ausgelöst wurde.

Beispiel 5:

Erstellen Sie die Datei `event5.htm`:

```
<HTML>
```

```
<SCRIPT
```

```
  FOR=document
```

```
  EVENT=onmouseover
```

```
  language="JavaScript">
```

```
window.status = "X-Position: " + event.clientX +  
  ", Y-Position: " + event.clientY;
```

```
</SCRIPT>
```

```
<BODY>
```



```
<H2>
Klicken Sie hier
</H2>
<IMG src="100.gif">
</BODY>
</HTML>
```

Laden Sie sie in den Internet Explorer und bewegen Sie die Maus auf die Überschrift oder die Grafik. In der Statuszeile wird jeweils die Mausposition angezeigt.

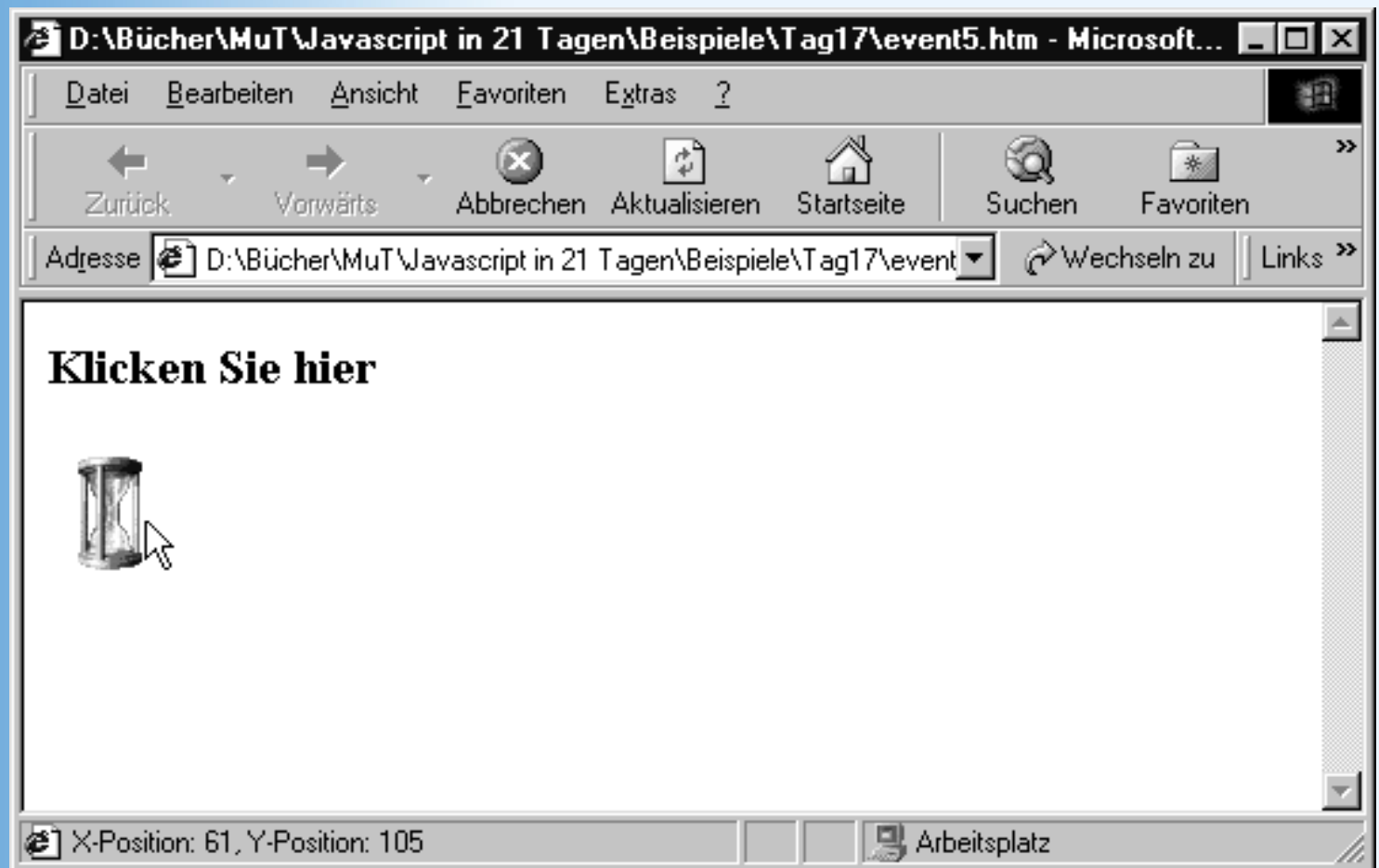


Abbildung 17.6: Die Mausposition

Die Eigenschaft

`keyCode`

speichert bei Tastaturereignissen den dezimalen Code (ASCII-/ANSI-Wert) der gedrückten Taste.

Über die Eigenschaften

`offsetX`

und

`offsetY`

kann auf die horizontalen Pixel (`offsetX`) und die vertikalen Pixel (`offsetY`) der Cursorposition relativ zur oberen linken Ecke des Elements, das ein Ereignis ausgelöst hat, zugegriffen werden.

Es gibt im Microsoft-Modell noch ein weiteres Koordinatenpaar, welche die Cursorposition speichert. Die Variablen

x

und

y

speichern die horizontalen Pixel (x) und die vertikalen Pixel (y) der Cursorposition relativ zur oberen linken Ecke des Eltern-Elements von dem Element, das ein Ereignis ausgelöst hat. Wenn ein absolut positionierter Bereich das Eltern-Element ist, ist dessen obere linke Ecke der Bezugspunkt. Wenn das auslösende Element sonst kein Eltern-Element hat, gilt die linke obere Ecke des Dokuments als Koordinatenursprung.

Zusammenfassung

Die zentrale Ereignisbehandlung unter JavaScript ist über das event-Objekt möglich. Das event-Objekt beinhaltet Informationen über ein aufgetretenes Ereignis in einer Webseite samt der Randbedingungen, welche dabei eine Rolle gespielt haben. Allerdings ist die Anwendung des event-Objekts jeweils auf das zugrunde liegende Ereignismodell beschränkt und man sollte bei globaler Ereignisbehandlung sicherstellen, dass nur der richtige Browser das jeweilige globale Ereignismodell zu sehen bekommt.

Fragen und Antworten

Frage:

Soll man nun Ereignisse lokal über Event-Handler realisieren oder ein globales Management vorziehen?

Antwort:

Das kommt auf die konkrete Situation an. Bei nur wenigen Ereignissen und kleinen Seiten lohnt sich ein globales Management kaum. Wenn aber konsequent die logische Trennung der Bestandteile einer Webseite erfolgen soll, ist das sinnvoll. Allerdings muss dabei die Inkompatibilität der Modelle beachtet werden. Zwei getrennte Konzepte sollten realisiert werden.

Frage:

Warum behandeln wir nicht schon heute die Trennung der Browser- Welten?

Antwort:

Die Trennung an sich erfolgt über das Objekt navigator, und das wollen wir morgen etwas ausführlicher behandeln.

Frage:

Warum ist das Ereignismodell von Netscape so viel komplexer und dennoch in der Leistung eingeschränkter als das Microsoft-Modell?

Antwort:

Das Netscape-Modell ist stark von Ereignismodellen aus anderen Techniken beeinflusst. Es gibt einige Parallelen zu der Ereignisbehandlung unter Java. Das Microsoft-Modell ist einfacher und dennoch leistungsfähiger, da der Browser stärker Betriebssystembestandteile verwenden kann.

Workshop

Einige Eigenschaften des event-Objekts im Rahmen des Microsoft-Modells wurden nicht mit Beispielen getestet. Holen Sie das nach. Erstellen Sie Beispiele, die diese Eigenschaften ausgeben oder sonst wie verwenden. Benutzen Sie dabei auch andere Ereignistypen.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was stimmt hier nicht?

```
window.captureEvents(Event.Click);
```

2. Stimmt das?

```
window.captureEvents(  
Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP)
```

3. Ist das korrekt oder müssen for und event groß geschrieben werden?

4. Was macht releaseEvents()?

5. Was macht routeEvent()?

6. Stimmt das?

```
window.captureEvents(Event.ONCLICK);
```

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

1

Feuern ist hier nicht willkürlich als Wortspiel gewählt - das wird gleich noch eine Rolle spielen.

2

Übermäßiges Hecheln, Stöhnen, leuchtend rote Birne, Zitate wie »Ich kann nicht mehr« oder »Ich will nicht mehr« usw.

3

Der Fitnesstrainer kann nicht jeden individuell anfeuern - nur halt in Ausnahmesituationen oder wenn er explizit eine Botschaft (s.o.) bekommt.

4

: -)

5

Viele Quellen nennen die gesamte Thematik nur noch ein großes Chaos.

Woche 3

Tag 18

Top-Level-Objekte und das Objekt navigator

Am heutigen Tag wollen wir uns um das navigator-Objekt und zwei wichtige Unterobjekte von window, location und history sowie die wichtigsten derjenigen Objekte kümmern, die man als Top-Level-Objekte (Objekte, die direkt an jeder Stelle in einem JavaScript zur Verfügung stehen) bezeichnet. Dies sind die Objekte

- Array
- Boolean
- Date
- Function
- Math
- Number
- RegExp
- Screen
- String

Ebenso soll heute auf dem Plan stehen, wie Browser und Plattformen beim Laden einer Webseite automatisch erkannt werden und wie man dann darauf reagieren kann. Etwa zu verschiedenen Webseiten weiterleiten.

Array

Um Arrays haben wir uns ja schon an Tag 15 gekümmert. Hier soll nur noch die Auflistung der wichtigsten Eigenschaften und Methoden angegeben werden. Das Array-Objekt wurde in JavaScript 1.1 eingeführt und JavaScript 1.3 fügte die Methode toSource() hinzu und überarbeitete die length-Eigenschaft sowie die Methoden push() und splice().

Eigenschaften

Eigenschaften	Description
constructor	Spezifiziert die Funktion, die den Prototyp eines Objekts kreiert
length	Anzahl der Elemente in einem Array

prototype

Erlaubt das nachträgliche Hinzufügen von Eigenschaften zu einem Array-Objekt

Tabelle 18.1: Eigenschaften von Array

Die Eigenschaften constructor und prototype werden bei fast allen Objekten bereitstehen und nachfolgend nur noch erwähnt.

Methoden

Methode	Beschreibung
concat()	Verbindet zwei Arrays und gibt ein neues Array zurück.
join([Trennzeichen])	Liefert eine Zeichenkette der Elemente eines Arrays mit den angegebenen Trennzeichen.
pop()	Entfernt das letzte Element aus einem Array und gibt es zurück.
push(Element1, ..., ElementN)	Fügt ein oder mehr Elemente an das Ende von einem Array ein und gibt die neue Länge des Arrays zurück.
reverse()	Dreht die Reihenfolge der Elemente in einem Array um.
shift()	Entfernt das erste Element aus einem Array und gibt es zurück.
slice(Beginn [,Ende])	Extrahiert eine Auswahl von Elementen ab einem Beginn bis zu einer optionalen Endangabe.
splice(Index, Anzahl, [Element1], [, ..., ElementN])	Eine Methode zum gezielten Entfernen oder Hinzufügen von Elementen ab dem angegebenen Index. Wenn bei Anzahl 0 angegeben wird, wird kein Element entfernt; die nachfolgend angegebenen optionalen Elemente werden in jedem Fall hinzugefügt.
sort([Vergleichsfunktion])	Sortiert die Elemente von einem Array nach ihrem Wert und der angegebenen Vergleichsfunktion.
toSource()	Rückgabe der String-Repräsentation des Codes.
toString()	Gibt das Array als String-Repräsentation zurück.
unshift(Element1, ..., ElementN)	Fügt ein oder mehr Elemente an den Anfang von einem Array ein und gibt die neue Länge des Arrays zurück.
valueOf()	Gibt den primitiven Wert von einem Array zurück.

Tabelle 18.2: Methoden von Array



Die Methoden `toSource()`, `toString()` und `valueOf()` werden bei fast allen Objekten bereitstehen und nachfolgend nur noch erwähnt.

Boolean

Das Objekt Boolean dient dazu, Boolesche Objekte zu erstellen. Diese können in JavaScript wie einfache boolesche Werte verwendet werden, denn JavaScript ruft bei der Angabe von Boolean-Objekten in entsprechenden Situationen automatisch die zugehörigen Methoden `valueOf()` auf, um den enthaltenen Wert auszulesen. Das Objekt hat nur wenige Methoden (`toSource()`, `toString()` und `valueOf()`) und Eigenschaften (constructor und prototype) und wegen der ähnlichen Anwendung wie die booleschen Werte eine nicht so große Bedeutung. Der Nutzen von Boolean zeigt sich, wenn eine explizite Umwandlung eines Ausdrucks in einen Wahrheitswert gewünscht ist.

Date

Das Objekt Date ist in JavaScript für Aktionen rund um Datum und Zeit zuständig. Der 1. Januar 1970, 0.00 Uhr ist der interne Zeitnullpunkt, der als Speicherungs- und Berechnungsbasis für alle Operationen dient. Ein Datumsobjekt kann mit folgenden Konstruktoren erzeugt werden:

```
new Date()  
new Date(millisekunden)  
new Date(DatumsString)  
new Date(Jahr, Monat, Tag,  
[, Stunde, Minute, Sekunde, Millisekunde])  
new Date(Jahr, Monat, Tag,  
[, Stunde, Minute, Sekunde])
```

Wir wollen uns die nachfolgenden drei Varianten anschauen:

```
var [Objektname] = new Date()  
var [Objektname] = new Date(Jahr, Monat, Tag)  
var [Objektname]  
= new Date(Jahr, Monat, Tag, Stunden, Minuten, Sekunden)
```

Variante (1) erzeugt eine Objektinstanz, in der das zum Zeitpunkt der Programmausführung aktuelle Datum und die aktuelle Uhrzeit (Systemzeit des Rechners) gespeichert wird. Die Varianten (2) und (3) initialisieren das neue Date-Objekt mit bestimmten Werten (also einem bestimmten Datum und einer bestimmten Uhrzeit). Alle Initialisierungswerte bei Variante 2 und 3 (Monat, Stunden oder Minuten) müssen in Zahlenform angegeben werden.

Das Objekt Date besitzt nur die Eigenschaften constructor und prototype, aber eine ganze Reihe von Methoden, von denen eine Auswahl der wichtigsten hier folgt.

Methoden

Methoden	Beschreibung
<code>getDate()</code>	Der Monatstag des Objekts als Zahl, beispielsweise 3, wenn der 3.12.1991, 23:59:00, gespeichert ist.

<code>getDay()</code>	Der Wochentag des Objekts als Zahl, etwa 3, wenn implizit ein Mittwoch gespeichert ist. Die möglichen Rückgabewerte sind 0 (Sonntag) bis 6 (Samstag).
<code>getHours()</code>	Die Stunden der im Objekt gespeicherten Uhrzeit.
<code>getMinutes()</code>	Die Minuten der im Objekt gespeicherten Uhrzeit.
<code>getMonth()</code>	Der Monat des im Objekt gespeicherten Datums.
<code>getSeconds()</code>	Die Sekunden der im Objekt gespeicherten Uhrzeit.
<code>getTime()</code>	Die Methode gibt die Anzahl Millisekunden als Zahl zurück, die seit dem 1. Januar 1970, 0:00:00, bis zu dem im Objekt gespeicherten Zeitpunkt vergangen sind.
<code>getTimeZoneoffset()</code>	Die Methode gibt den Unterschied zwischen lokaler Zeit und Greenwich Meridian Time (GMT) in Anzahl Minuten zurück.
<code>getYear()</code>	Das Jahr des im Objekt gespeicherten Datums.
<code>parse(Zeitpunkt)</code>	Diese Methode ermittelt aus einer zu übergebenden Zeichenkette Zeitpunkt die Anzahl Millisekunden, die zwischen dem 1.1.1970 0:00:00, und dem übergebenen Zeitpunkt verstrichen sind. Die Zeichenkette Zeitpunkt muss dem IETF-Standard folgen. Beispiele: "Wed, 16Oct 1991 23:59:00 GMT", oder »Mon, 11Dec 1995 15:30:00 GMT+0430« .
<code>setDate(Tag)</code>	Hiermit ändert man den in dem Objekt gespeicherten Monatstag auf den als Zahl zu übergebenden Monatstag.
<code>setHours(Stunden)</code>	Veränderung der in dem Objekt gespeicherten Stunden der Uhrzeit auf die als Zahl zu übergebenden Stunden. Die übergebenen Werte können zwischen 0 und 23 liegen.
<code>setMinutes(Minuten)</code>	Ändert die in dem Objekt gespeicherten Minuten der Uhrzeit auf die als Zahl zu übergebende Minutenzahl. Werte zwischen 0 und 59 sind erlaubt.

<code>setMonth(Monat)</code>	Ändert den in dem Objekt gespeicherten Monat auf den als Zahl zu übergebenden Monat. Es sind Werte zwischen 1 und 12 als Übergabeparameter erlaubt.
<code>setSeconds(Sek)</code>	Verändert die in dem Objekt gespeicherten Sekunden der Uhrzeit auf die als Zahl zu übergebenden Sekunden. Es gibt Übergabewerte zwischen 0 und 59.
<code>setTime(Millisek)</code>	Diese Methode ändert den kompletten Inhalt von dem Objekt auf einmal durch Übergeben einer Zahl. Diese Zahl stellt die Anzahl der Millisekunden seit dem 10.01.1970 - 0:00:00 Uhr dar.
<code>setYear(Jahr)</code>	Hier wird das in dem Objekt gespeicherte Jahr auf das als Zahl zu übergebende Jahr geändert.
<code>toGMTString()</code>	Diese Methode wandelt die in dem Objekt gespeicherten Daten in eine Zeichenkette nach dem IETF-Standard um.
<code>toLocaleString()</code>	Diese Methode wandelt die in dem Objekt gespeicherten Daten in eine Zeichenkette um. Der Unterschied zur Methode <code>toGMTString()</code> ist der, dass die zurückgegebene Zeichenkette die lokale Uhrzeit berücksichtigt.
<code>UTC(Jahr, Monat, Tag [, Stunden] [, Minuten] [, Sekunden])</code>	Die Methode gibt die Anzahl Millisekunden zurück, die zwischen dem 1.1.1970 - 0:00:00 Uhr und dem übergebenen Zeitpunkt verstrichen sind. Alle Parameter sind als Zahlenwerte zu übergeben. Die Angabe der Datumswerte ist zwingend, die der Uhrzeitwerte optional.

Tabelle 18.3: Methoden von Date

Function

Das Objekt `Function` ist ein selten verwendetes Objekt. Es dient dazu, einen alternativen Weg zur Definition einer Funktion beschreiten zu können. Es wird genau genommen ein `Function`-Objekt erzeugt, was es ermöglicht, in JavaScript objektorientierter zu arbeiten. Wir wollen nicht näher darauf eingehen, sondern nur ein kleines Beispiel demonstrieren.

Beispiel 1:

Erstellen Sie die Datei `Funktion.htm`:

```
<HTML>
<SCRIPT language="JavaScript">
```

```
berechne = new Function("a","b","return a + b");  
alert(berechne(1,2));  
</SCRIPT>  
<BODY>  
</BODY>  
</HTML>
```

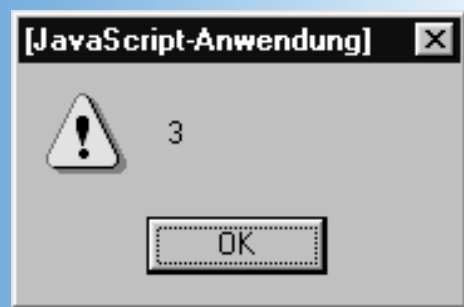


Abbildung 18.1: Einsatz von Function

Math

Mit dem Objekt Math stehen Ihnen diverse Funktionen, Methoden und Eigenschaften für Berechnungen zur Verfügung. Eine Instanz des Objekts brauchen Sie nicht eigens erzeugen, da es sich um ein so genanntes »built in«-Objekt handelt. Sie haben automatisch Zugriff auf Eigenschaften und Methoden von Math über die folgende Syntax:

Math.[Eigenschaft/Methode]

Eigenschaften

Das Objekt Math besitzt eine ganze Menge von Konstanten und Eigenschaften. Hier nur eine kleine Auswahl.

Eigenschaft	Beschreibung
E	Die Eulersche Konstante
LN10	Der natürliche Logarithmus von 10
LOG2E	Der Logarithmus von 2
PI	Die berühmte Konstante Pi
SQRT2	Eine Konstante für Quadratwurzel aus 2

Tabelle 18.4: Eigenschaften von Math

Methoden

Das Objekt Math besitzt eine ganze Reihe von Methoden, welche Sie sehr wahrscheinlich zum großen Teil nicht benötigen werden. Hier folgt aber eine kleine Auswahl.

Methode	Beschreibung
abs(Zahl)	Der absolute Wert des übergebenen Parameters. Dies ist immer ein positiver Wert, selbst wenn er vorher negativ war.
cos(Zahl)	Die Methode berechnet den Cosinus des übergebenen Parameters, einer beliebigen Zahl.

<code>exp(Zahl)</code>	Die Methode berechnet die Eulersche Zahl hoch dem übergebenen Parameter, einer beliebigen Zahl.
<code>log(Zahl)</code>	Die Methode berechnet den Logarithmus zur Basis 10 von dem übergebenen Parameter, einer beliebigen Zahl.
<code>max(Zahl1, Zahl2)</code>	Die Methode berechnet das Maximum von zwei beliebigen als Parameter übergebenen Zahlen.
<code>min(Zahl1, Zahl2)</code>	Die Methode berechnet das Minimum von zwei beliebigen als Parameter übergebenen Zahlen.
<code>pow(Zahl, Exponent)</code>	Die Methode berechnet den Wert von Zahl hoch Exponent, wobei Zahl und Exponent als Parameter übergeben werden.
<code>random()</code>	Erzeugt eine Zufallszahl zwischen 0 und 1.
<code>round(Zahl)</code>	Die Methode berechnet die nächste Ganzzahl des übergebenen Parameters, einer beliebigen Zahl. Dabei wird die kaufmännische Rundung angewendet. Falls die Zahl selbst ganzzahlig ist, wird sie nicht verändert.
<code>sin(Zahl)</code>	Die Methode berechnet den Sinus des übergebenen Parameters, einer beliebigen Zahl.
<code>sqrt(Zahl)</code>	Die Methode berechnet die Quadratwurzel des übergebenen Parameters, einer beliebigen Zahl.
<code>tan(Zahl)</code>	Der Tangens einer als Parameter übergebenen Zahl.

Tabelle 18.5: Methoden von Math

Number

Das Objekt `Number` dient zur Unterstützung der Arbeit mit numerischen Werten. Bei vielen Berechnungen ist es sehr hilfreich. Besonders, wenn Bedingungen eintreten wie Unendlichkeit oder irgendwelche Grenzwerte. Es fungiert als ein Objekt-Wrapper für primitive numerische Werte.



Ein Wrapper dient dazu, mit Hilfe des `new`-Schlüsselworts jeweils ein Objekt-Gegenstück zu einem primitiven Datentyp zu erstellen.

Der `Number`-Konstruktor sieht so aus:

```
new Number([numerischer Wert])
```

Man setzt dieses Objekt ein, um Zugriff auf spezielle darüber verfügbare Methoden zu haben oder numerische Objekte zu erstellen, denen man bestimmte Eigenschaften noch hinzufügen möchte.

Eigenschaften

Eigenschaft	Beschreibung
<code>MAX_VALUE</code>	Größte darstellbare Zahl.
<code>MIN_VALUE</code>	Kleinste darstellbare Zahl
<code>NaN</code>	Not a Number.

NEGATIVE_INFINITY	Repräsentation von »Negativ-unendlich«; wird bei einer Berechnung bei einem Überlauf zurückgegeben.
POSITIVE_INFINITY	Repräsentation von »Positiv-unendlich«; wird bei einer Berechnung bei einem Überlauf zurückgegeben.

Tabelle 18.6: Eigenschaften von Number.

Daneben gibt es noch die allgemeinen Eigenschaften prototype und constructor.

Methoden

Das Objekt stellt nur die allgemeinen Methoden toSource(), toString() und valueOf() bereit.

RegExp

Das Objekt RegExp ist eines der leistungsfähigsten Objekte, auf das man unter JavaScript zugreifen kann. Es ist aber auch eines der komplexesten, weshalb es hier nur sehr komprimiert vorgestellt wird. Es handelt sich um ein Objekt, mit dem sehr detailliert Informationen zum Suchen von Ausdrücken spezifiziert werden können. Dazu gibt es eine umfangreiche Liste von Eigenschaften und Methoden, aber vor allem eine ganze Familie von Token mit spezieller Bedeutung. Da das Thema extrem umfangreich ist¹ und andererseits ein nur sehr spezialisiertes Anwendungsgebiet abdeckt, soll explizit auf den Anhang und weiterführende Literatur verwiesen werden.

screen

Ein sehr spannendes Objekt ist screen. Das Objekt beinhaltet keine Methoden, aber einige Eigenschaften. Darüber haben Sie Zugriff auf diverse plattformspezifische Informationen beim Anwender, genau genommen im Wesentlichen die Auflösung und die beim Anwender verfügbaren Farben.



Wir wollen diese screen-Informationen in Zusammenhang mit der Abfrage der Browserversion am Ende des Kapitels in einem umfangreichen Beispiel einsetzen, um eine Schablone zu schaffen, die verschiedene Plattformen trennen und damit optimal bedienen kann.

Eigenschaften	Beschreibung
availHeight	Höhe des verfügbaren Bildschirms in Pixeln, d.h. minus Elementen, die den Bereich einschränken, wie die Taskbar in Windows.
availLeft	Angabe der X-Koordinate von dem ersten verfügbaren Pixel.
availTop	Angabe der Y-Koordinate von dem ersten verfügbaren Pixel.
availWidth	Breite des verfügbaren Bildschirms in Pixeln.
colorDepth	Die Bit-Tiefe der Farbpalette.
height	Bildschirmhöhe.
pixelDepth	Bildschirmfarbauflösung (Bits per Pixel).
width	Bildschirmbreite.

Tabelle 18.7: Eigenschaften von screen

String

String-Manipulationen gehören zu den wichtigsten Aktionen bei komplexeren Scripten. Das Objekt String ist deshalb eines der wichtigsten Objekte. Damit können Sie mit verschiedensten Methoden Zeichenketten manipulieren. Eine Instanz von String brauchen Sie nicht eigens erzeugen, da es sich um ein so genanntes »built in«-Objekt handelt. Eine Zuweisung der Form

```
var text="irgend ein Text";
```

erzeugt automatisch ein String-Objekt. Eine Verwendung des new- Schlüsselwortes in der Form

```
var text = new String("irgend ein Text");
```

funktioniert aber auch.

Eigenschaften hat das Objekt nur wenige (length zur Speicherung der Anzahl von Zeichen einer Zeichenkette sowie prototype und constructor). Dafür gibt es aber umso mehr Methoden.

Methoden

Methode	Beschreibung
anchor (Ankname)	Hiermit wird ein Verweisziel innerhalb einer HTML-Datei mit dem übergebenen Namen für den Anker erzeugt.
big()	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <big> und bewirkt eine größere Schrift.
blink()	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <blink> und bewirkt eine blinkende Schrift.
bold()	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container und bewirkt eine fette Schrift.
charAt (Nummer)	Die Methode liefert das Zeichen zurück, das in der Zeichenkette an der Stelle steht, die im Parameter Nummer übergeben wird.
charCodeAt (Index)	Gibt ein Zahl zurück, die dem Unicode-Wert des Zeichens an dem angegebenen Index entspricht.
concat(String1, String2 [, ..., StringN])	Verbindet zwei oder mehr Strings zu einem neuen.
fixed()	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <tt> und bewirkt eine dichtengleiche Schrift.
fontcolor (Farbe)	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container ... und bewirkt eine andersfarbige Schrift. Die gewünschte Farbe wird der Funktion als Parameter übergeben, und zwar entweder in hexadezimaler Schreibweise oder als Farbname.

<code>fontSize(Zahl)</code>	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <code> ... </code> und bewirkt eine größere oder kleinere Schrift. Die gewünschte Größe wird der Funktion als Parameter übergeben. Es muss eine Zahl von 1 bis 7 sein. 1 ist sehr kleine Schrift, 3 ist Normalschrift, 7 ist sehr große Schrift.
<code>fromCharCode(num1, ..., numN)</code>	Gibt einen String zurück, der aus der angegebenen Sequenz aus Unicode-Werten erstellt wurde.
<code>indexOf(Zeichen, [Nummer])</code>	Hier wird das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb der mit String spezifizierten Zeichenkette ermittelt. Die Rückgabe ist die Stelle, an der das Zeichen in der Zeichenkette steht. Die Zählung beginnt bei 0. Optional ist es möglich, die Funktion in einem zweiten Parameter [Nummer] anzuweisen, ab der wie vielten Stelle in der Zeichenkette sie mit der Suche beginnen soll. Wenn das gesuchte Zeichen nicht gefunden wurde, wird -1 zurückgegeben.
<code>italics()</code>	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <code><i></code> und bewirkt eine kursive Schrift.
<code>lastIndexOf(Zeichen, [Nummer])</code>	Hier wird das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb der mit String spezifizierten Zeichenkette ermittelt. Die Rückgabe ist die Stelle, an der das Zeichen in der Zeichenkette steht. Die Zählung beginnt bei 0. Optional ist es möglich, die Funktion in einem zweiten Parameter [Nummer] anzuweisen, ab der wie vielten Stelle in der Zeichenkette sie mit der Suche beginnen soll.
<code>link(Verweis)</code>	Die Methode erzeugt einen Verweis. Das Verweisziel, z.B. ein Dateiname oder eine URL-Adresse, wird als Parameter übergeben. Den Verweistext müssen Sie als Variable definieren.
<code>match(regExp)</code>	Die Methode wird dazu verwendet, um einen regulären Ausdruck mit einem String zu vergleichen.
<code>replace(was, womit)</code>	Die Methode ersetzt den zuerst angegebenen Ausdruck durch den zweiten Ausdruck.
<code>search(was)</code>	Die Methode durchsucht den String nach der als Parameter angegebenen Zeichenkette. Falls die Zeichenkette nicht gefunden wird, wird der Wert -1 zurückgeliefert, sonst ein davon verschiedener Wert.
<code>slice(Beginn[, Ende])</code>	Extrahiert einen Teilstring von dem angegebenen Index bis zum Ende oder einem optionalen Ende-Index. Dieser kann auch ein negativer Wert sein, wenn ab dem Ende des Strings rückwärts gezählt werden soll.
<code>small()</code>	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <code><small></code> und bewirkt dadurch eine kleinere Schrift.

<code>split([Separator][, Limit])</code>	Die Methode trennt einen String in ein Array von Teilstrings, indem an dem angegebenen Separator geteilt wird. Wird der Separator nicht vorgefunden oder er fehlt, wird ein Array mit einem Element und dem vollständigen String zurückgegeben. Das optionale Limit gibt eine maximale Anzahl von Trennungen an.
<code>strike()</code>	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <code><s></code> und bewirkt dadurch einen durchgestrichenen Text.
<code>sub()</code>	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <code><sub></code> und bewirkt eine tiefgestellte Schrift.
<code>substr(Start[, length])</code>	Gibt die Zeichen in einem String zurück, wobei bei dem angegebenen Index begonnen und bis zum Ende oder der optionalen Anzahl von Zeichen extrahiert wird.
<code>substring(Von, Bis)</code>	Liefert eine Zeichenkette zurück, die einen Teil innerhalb einer über String definierten Zeichenkette darstellt. Als Parameter werden der Funktion der Beginn (Von) und das Ende (Bis) der zurückzugebenden Zeichenkette übergeben. Das erste Zeichen hat den Wert 0.
<code>sup()</code>	Diese Methode setzt eine über String definierte Zeichenkette in den HTML-Container <code><sup></code> und bewirkt dadurch eine kleinere Schrift.
<code>toLowerCase()</code>	Die Methode wandelt alle in der über String spezifizierten Zeichenkette enthaltenen Großbuchstaben in Kleinbuchstaben um.
<code>toUpperCase()</code>	Die Methode wandelt alle in der über String spezifizierten Zeichenkette enthaltenen Kleinbuchstaben in Großbuchstaben um.

Tabelle 18.8: Methoden von String

Daneben gibt es noch die allgemeinen Methoden `toString()`, `toSource()` und `valueOf()`.

Wir wollen den Einsatz von String-Methoden mit mehreren Beispielen üben.

Beispiel 2:

Das Beispiel zeigt den Einsatz von den Methoden, welche HTML-Befehle nachbilden sowie die Konvertierung in Groß- und Kleinbuchstaben. Erstellen Sie die Datei `String1.htm`:

```
<HTML>
<SCRIPT language="JavaScript">
var meinText = "Das ist der Text aller Texte";
var vglText = "Dat is nur zum Vergleich da. ";
document.write(vglText + meinText.big());
document.write("<BR>");
document.write(vglText + meinText.blink());
document.write("<BR>");
document.write(vglText + meinText.bold());
```



```
document.write("<BR>");
document.write(vglText + meinText.fixed());
document.write("<BR>");
document.write(vglText + meinText.fontcolor("red"));
document.write("<BR>");
document.write(vglText + meinText.fontcolor("blue"));
document.write("<BR>");
document.write(vglText + meinText.fontcolor("green"));
document.write("<BR>");
document.write(vglText + meinText.fontSize(1));
document.write("<BR>");
document.write(vglText + meinText.fontSize(7));
document.write("<BR>");
document.write(vglText + meinText.fontSize(5));
document.write("<BR>");
document.write(vglText + meinText.italics());
document.write("<BR>");
document.write(vglText + meinText.small());
document.write("<BR>");
document.write(vglText + meinText.strike());
document.write("<BR>");
document.write(vglText + meinText.sub());
document.write("<BR>");
document.write(vglText + meinText.sup());
document.write("<BR>");
document.write(vglText + meinText.toLowerCase());
document.write("<BR>");
document.write(vglText + meinText.toUpperCase());
document.write("<BR>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```



Abbildung 18.2: Verschiedene Textformatierungen

Beispiel 3:

Das Beispiel zeigt den Einsatz von den Methoden, welche in Texten verschiedene Operationen ausführen. Erstellen Sie die Datei String2.htm:

```
<HTML>
<SCRIPT language="JavaScript">
var meinText = "Das ist der Text aller Texte";
var text1 = " Die ";
var text2 = "Antwort ";
var text3 = "ist ";
var text4 = "42";
document.write(meinText.charAt(2));
document.write(" ");
document.write(meinText.charAt(6));
document.write(" ");
document.write(meinText.charAt(8));
document.write(" ");
document.write(meinText.charAt(9));
document.write("<BR>");
document.write(meinText.charCodeAt(2));
document.write(" ");
document.write(meinText.charCodeAt(6));
document.write(" ");
document.write(meinText.charCodeAt(8));
```

```
document.write(" ");
document.write(meinText.charCodeAt(9));
document.write("<BR>");
document.write(
meinText.concat(text1,text2,text3,text4));
document.write("<BR>");
document.write(meinText.indexOf("i"));
document.write("<BR>");
document.write(meinText.search("aller"));
document.write("<BR>");
document.write(meinText.replace("aller", "welcher"));
document.write("<BR>");
document.write(meinText.lastIndexOf("e"));
document.write("<BR>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

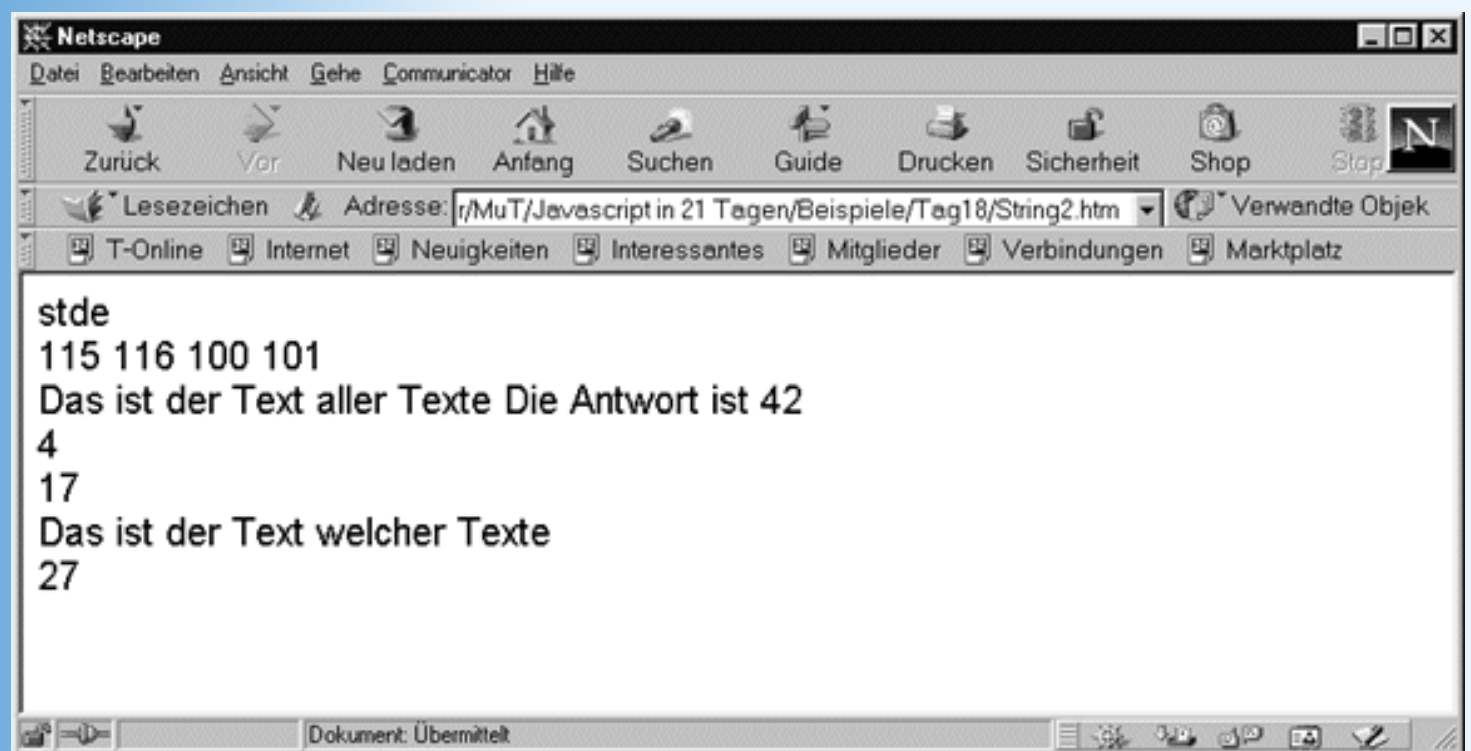


Abbildung 18.3: Verschiedene Operationen in Texten

Beispiel 4:

Das abschließende Beispiel zu diesem Thema zeigt wie das zweite Beispiel den Einsatz von den Methoden, welche in Texten verschiedene Operationen ausführen. Erstellen Sie die Datei String3.htm:

```
<HTML>
<SCRIPT language="JavaScript">
var meinText = "Der Wert der Zahl PI: 3.14.";
neuArray = meinText.match(/d/gi);
document.write(neuArray);
document.write("<BR>");
str=meinText.slice(3,-5);
document.write(str);
```

```
document.write("<BR>");
document.write("(1,2): ", meinText.substr(1,2))
document.write("<BR>");
document.write("(-2,2): ", meinText.substr(-2,2))
document.write("<BR>");
document.write("(1): ", meinText.substr(1))
document.write("<BR>");
document.write("(-10, 2): ", meinText.substr(1,10))
document.write("<BR>");
document.write("(10, 2): ", meinText.substr(10,2))
document.write("<BR>");
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

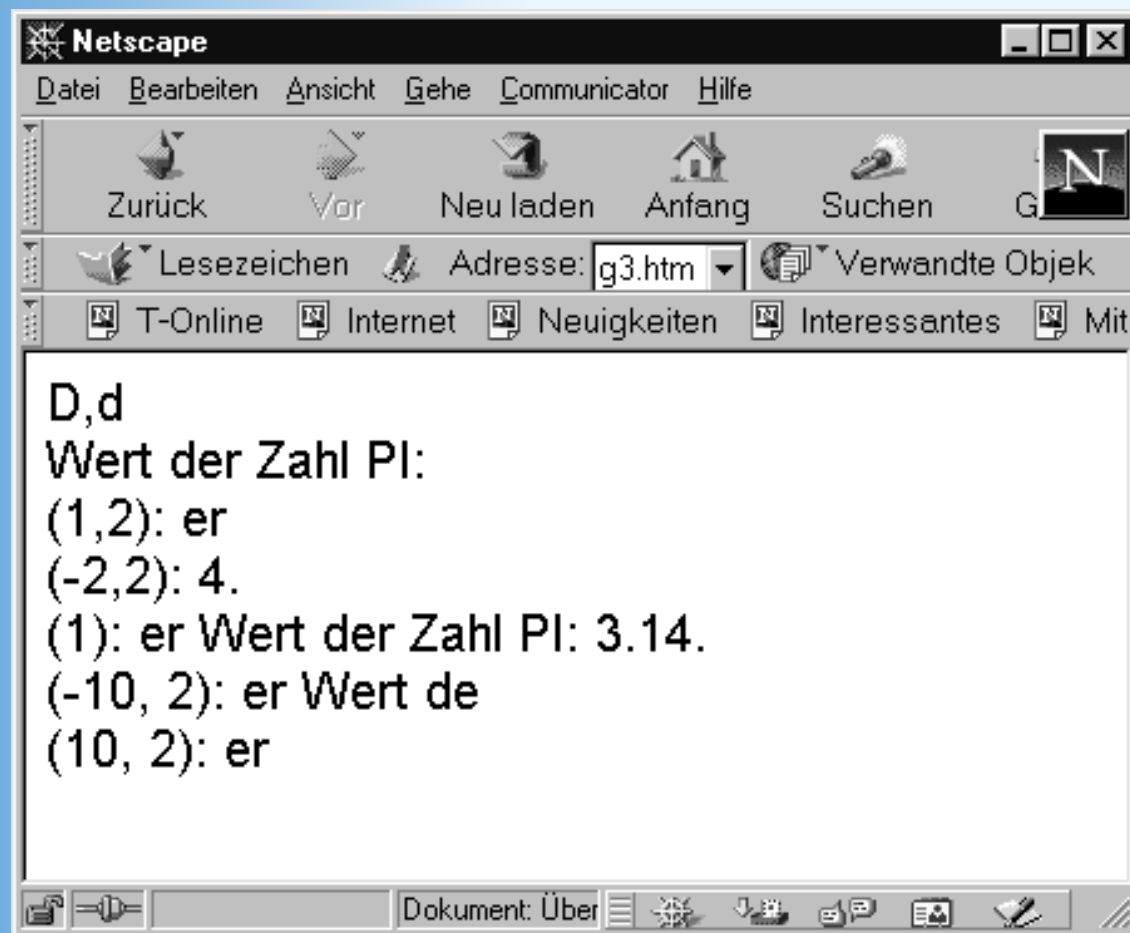


Abbildung 18.4: Verschiedene Operationen in Texten

Das navigator-Objekt

Über das Top-Level-Objekt navigator hat man Zugriff auf den verwendeten Browser des Anwenders. Dies ist besonders dann wichtig, wenn Browser-spezifische Anweisungen verwendet werden (damit sind sowohl Script-Anweisungen - insbesondere die Verwendung von Mischkonstellationen - und sogar HTML oder Style Sheets gemeint). Eine Instanz von navigator braucht nie extra erzeugt werden, da es sich um ein »built in«-Objekt handelt.

Eigenschaften

Das Objekt hat die folgenden Eigenschaften:

Eigenschaft	Beschreibung
<code>appName</code>	Codename eines Browsers.
<code>appVersion</code>	Der offizielle Anwendungsname eines Browsers. Der Unterschied zu <code>appName</code> ist beispielsweise bei Netscape, dass dort Mozilla und hier der Name des Browsers - Netscape - zurückgegeben wird.
<code>language</code>	Die Version und Betriebssystemplattform des Browsers.
<code>mimeTypes</code>	Die vom Anwender eingestellte Sprache des Browsers in international üblichen Abkürzungen (etwa <code>de</code> für deutsch).
<code>platform</code>	Ein Array von allen MIME-Typen, die auf dem Client unterstützt werden.
<code>plugins</code>	Betriebssystem des Anwenders.
<code>userAgent</code>	Ein Array von allen PlugIns, die beim Client installiert sind.
	Die Daten des verwendeten Browsers, und zwar genau so, wie sie im HTTP-Protokoll an den aufgerufenen WWW-Server übermittelt werden (eine Kombination von den restlichen Infos).

Tabelle 18.9: Eigenschaften von navigator

Das Objekt `navigator` hat nur wenige Methoden. Bis auf `javaEnabled()`, mit der die Java-Fähigkeit (Java - nicht JavaScript) eines Browsers getestet werden kann (`true`) und `plugins.refresh()`, um neu installierte PlugIns verfügbar zu machen, sind sie auch nicht so wichtig.



Praktische Beispiele zum Einsatz von dem navigator-Objekt werden wir im letzten Abschnitt zur Trennung von Browser- und Plattformwelten sehen.

Das history-Objekt

Über das `history`-Objekt - ein direktes Unterobjekt von `window` - hat man Zugriff auf die zuvor geladenen Seiten im Browser des Anwenders. Maßgeblich ist dabei die Liste, wie sie in der History-Liste des Browsers gespeichert ist. Eine Instanz des Objektes `history` steht automatisch bereit und sämtliche Eigenschaften und Methoden lassen sich für nachfolgende Syntax direkt verwenden:

`history.[Eigenschaft/Methode]`

Das Objekt besitzt im Wesentlichen die Eigenschaft `length`, worüber die Anzahl der Einträge in der aktuellen History-Liste dokumentiert wird. Die anderen Eigenschaften

`current`,
`next`

und

`previous`

(aktuelle Seite in der Historie, nächste und vorherige) funktionieren nicht auf allen Browsern. An Methoden steht die folgende Liste zur Verfügung:

Methode	Beschreibung
<code>back()</code>	Die Methode lädt die zuletzt besuchte WWW-Seite.
<code>forward()</code>	Die Methode macht die <code>back()</code> -Methode rückgängig, lädt also die gerade mit der <code>back()</code> -Methode verlassene Seite wieder.
<code>go(Anzahl)</code>	Die Methode lädt eine beliebige Seite der History-Liste. Bezugspunkt ist dabei die aktuelle Seite. Sie können die Anzahl mit Vorzeichen verwenden und damit sowohl zurückgehen als auch vorwärts, wenn Sie vorher zurückgegangen sind.

Tabelle 18.10: Methoden von history

Das location-Objekt

Das location-Objekt - ein weiteres direktes Unterobjekt von window - repräsentiert den kompletten URL, der mit einem window-Objekt verknüpft ist. Jede der Eigenschaften von dem location-Objekt steht für einen anderen Bestandteil des URL. Wenn man die allgemeine Struktur eines URL zugrunde legt, sieht er so aus:

```
protocol//host:port/pathname#hash?search
```

Etwa derart:

```
http://www.rjs.de:84/mainframe.htm#Anfang?x=7&y=2
```

Jeder der einzelnen Bestandteile des URL steht als Bezeichner einer Eigenschaft des location-Objekts zur Verfügung und kann direkt verwendet werden.

Eigenschaften

Eigenschaft	Beschreibung
<code>protocol</code>	Das den URL beginnende Protokoll inklusive des ersten Doppelpunkts.
<code>host</code>	Host- und Domainname, oder IP-Adresse.
<code>port</code>	Der Port.
<code>pathname</code>	Der Pfad inklusive Datei.
<code>hash</code>	Der Anker in der Webseite inklusive des Trennzeichens # (nur bei HTTP-URLs).
<code>search</code>	Suchabfragen inklusive dem Trennzeichen ? (nur bei HTTP-URLs). Der Suchstring besteht aus Variablen und Wertepaaren, jedes Paar durch & getrennt.
<code>href</code>	Die vollständige Repräsentation eines URL.
<code>hostname</code>	Die Repräsentation des kontaktierten Hosts samt Port.

Tabelle 18.11: Eigenschaften von location

Methoden

Methoden	Beschreibung
<code>reload()</code>	Erzwingt ein Reload von dem aktuell im Fenster angezeigten Dokument.
<code>replace()</code>	Lädt den angegebenen URL über den aktuellsten History-Eintrag.

Tabelle 18.12: Methoden von location.

Am wichtigsten an dem location-Objekt dürfte sein, dass man damit aus JavaScript heraus eine neue Seite in den Browser laden kann. Mit dem location-Objekt kann man auf folgende zwei Weise eine neue Webseite in den Browser laden:

```
window.location.href="http://www.rjs.de/"  
window.location="http://www.rjs.de /"
```



Bei der Arbeit mit Event-Handlern sollte window.location dem einfachen Verwenden von location vorgezogen werden. Das Objekt location ist keine Eigenschaft des document-Objekts; dort gibt es ein Äquivalent document.URL.



Den praktischen Einsatz des location-Objekts werden wir bei der Trennung von Browsern sehen.

Trennung von Browser- und Plattformwelten

Wir haben nun alles an notwendigem Rüstzeug zusammen, um Browser und Plattformen per JavaScript zu identifizieren und entsprechend darauf zu reagieren. Reagieren soll bei uns bedeuten, es werden verschiedene Seiten je nach Clientvoraussetzung geladen oder ähnlich differenzierende Schritte unternommen (die Ausarbeitung sei hier außer Acht gelassen - das sind dann nur noch Details). Dabei wollen wir verschiedene Stufen durchspielen - von der Entscheidung, ob überhaupt JavaScript bereitsteht, über das einfache Aufteilen nach Internet Explorer und Netscape Navigator bis zum differenzierten Aufteilen nach Versionsnummer, Bildschirmgrößen und Betriebssystem.

JavaScript-Test

Eine der grundlegenden Unterteilungen in Webprojekten ist die, dass erst Mal unterschieden wird, ob bei einem Client JavaScripte ausgeführt werden können oder nicht. JavaScript ist eine ideale Technik zum Trennen von Browser-Welten. Aber was ist, wenn JavaScript deaktiviert ist? Eine Abfrage per JavaScript, ob JavaScript aktiviert ist, ist ja nicht möglich. Oder doch? Es geht wirklich. Indirekt. Man nutzt die Tatsache, dass eine JavaScript-Aktion bei deaktiviertem JavaScript überhaupt nicht funktioniert. Wenn man auf einer Webseite eine automatische Weiterleitung per JavaScript durchführt (wir wollen dies in der Folge mit dem gerade beschriebenen Unterobjekt von window machen - location), wird diese bei deaktiviertem JavaScript nicht ausgeführt und die Seite bleibt stehen. Wenn ein zusätzlicher HTML-<META>-Tag, der nach einem Zeitintervall (unbedingt länger als das Weiterleitungsintervall des JavaScripts) automatisch auf ein Teilprojekt ohne JavaScript weiterleitet, trennt die Welten.

Beispiel 5:

Erstellen Sie die nachfolgende Datei Trennung1.htm und nachfolgende die beiden Dateien OhneJS.htm und MitJS.htm, die einfach nur eine kurze Begrüßung oder so enthalten sollen (das ist nicht wesentlich):

```
<HTML>  
<HEAD>  
<META http-equiv="refresh" content="8;URL=OhneJS.htm">  
</HEAD>
```

```
<SCRIPT LANGUAGE="JavaScript">
function JSweiter() {
location.href="MitJS1.htm";
}
function JSoderNicht() {
window.setTimeout('JSweiter()',1000);
}
</SCRIPT>
<BODY onLoad="JSoderNicht()">
</BODY>
</HTML>
```



Abbildung 18.5: Schnelle Weiterleitung per JavaScript

Automatische Trennung von Netscape- und Microsoft- Browsern

Um die beiden Browser-Welten bzw. ihr globales Eventhandling zu trennen, kann man bei aktiviertem JavaScript im Browser auf verschiedene Varianten zurückgreifen. Es gibt einmal die einfachste Version, in der ein ähnliches Prinzip wie in dem Fall von deaktiviertem JavaScript verwendet wird. Man bedient sich einer Anweisung, die jeweils von der nicht gewünschten Browservariante nicht verstanden wird, und wertet die Reaktion aus. Etwa wie im Beispiel mit einer if-Struktur. Der Zugriff auf das jeweilige Objekt liefert nur dann den Wert wahr, wenn es auch unterstützt wird. Dabei benutzt man zur Identifizierung des Internet Explorer das Objekt `all`, das nur von diesem unterstützt wird, und zur Identifizierung des Navigator das Objekt `layers`, das nur dieser versteht.

Beispiel 6:

Erstellen Sie die nachfolgende Datei `Trennung2.htm` und danach die beiden Dateien `IE1.htm` und `NC1.htm`, die wieder nur eine kurze Grußbotschaft oder so zu enthalten brauchen:

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
function BrowserTrennen()
{
if (document.all)
{
location.href="IE1.htm";
}
if (document.layers)
{
location.href="NS1.htm";
}
}
</SCRIPT>
<BODY onload="BrowserTrennen()">
</BODY>
</HTML>
```



Abbildung 18.6: In die Netscape-Welt weitergeleitet

So elegant und zuverlässig dieser Trick ist - er nutzt jeweils einen Mangel des unerwünschten Browsers aus. Wenn irgendwann dieser jeweilige Mangel beseitigt wird, wird die Überprüfung nicht mehr greifen. Eine andere Variante trennt die beiden Browser einfach über explizite Abfrage des jeweiligen Browsers mittels des navigator-Objekts und dessen Eigenschaft appName.

Beispiel 7:

Erstellen Sie die nachfolgende Datei Trennung3.htm:

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
function BrowserTrennen()
{
if(navigator.appName == "Netscape")
{
location.href="NC1.htm";
}
if(navigator.appName == "Microsoft Internet Explorer")
{
location.href="IE1.htm";
}
}
</SCRIPT>
<BODY onload="BrowserTrennen()">
</BODY>
</HTML>
```



Abbildung 18.7: Willkommen in der Microsoftwelt

Die explizite Abfrage des jeweiligen Browsers mittels des navigator- Objektes und dessen Eigenschaft appName auf diese Art ist die meistgenutzte Variante, um einfach nur beide Browser-Welten zu trennen - selbst von vielen professionellen Seiten. Leider wird die Eigenschaft appName von einigen Partnerfirmen der Browserhersteller gelegentlich für Werbezwecke missbraucht. So liefert beispielsweise T-Online den Netscape Navigator mit einer veränderten appName-Eigenschaft aus, weshalb eine Abprüfung auf Gleichheit mit "Netscape" durch Scripte false liefern kann, obwohl der Browser ursprünglich diesen Wert in der Eigenschaft gesetzt hatte.

Um diesem Mangel zu begegnen, darf man nicht einfach nur auf Gleichheit prüfen, sondern darauf, ob eine bestimmte Zeichenkette im Wert der Eigenschaft enthalten ist (das ist selbst bei veränderter appName-Eigenschaft so gut wie immer noch der Fall). Dabei helfen die Methoden des String- Objekts. Etwa indexOf(), die wir gleich noch einsetzen, oder search(), womit in dem String nach dem Vorkommen der Zeichenkette gesucht wird und nicht unbedingte Gleichheit notwendig ist. Aber auch einige andere der Methoden des String-Objekts sind da ganz nützlich.

Beispiel 8:

Erstellen Sie die nachfolgende Datei Trennung4.htm:

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
var browser=navigator.appName;
function BrowserTrennen()
{
if(browser.search("Netscape")==-1)
{
location.href="IE1.htm";
}
else
{
location.href="NS1.htm";
}
}
</SCRIPT>
<BODY onload="BrowserTrennen()">
</BODY>
</HTML>
```

Im Vergleich wird der Wert -1 zurückgegeben, wenn die Zeichenkette nicht vorkommt. So funktioniert das Verfahren viel zuverlässiger. Wenn die Zeichenkette Netscape vorkommt, wird der else-Zweig ausgewählt, falls die Bedingung erfüllt ist, der if-Zweig.

Beispiel 9:

Das nächste Beispiel trennt mehr Bedingungen. Dabei sollen zwei Betriebssysteme (Mac und PC), Browsertypen, Browserversionen und die Bildschirmauflösung eine Rolle spielen. Das Beispiel ist nur für den Fall des Navigator auf PC-Ebene zu Ende notiert, aber die restlichen Fälle sind vollkommen identisch. Das Beispiel verwendet den ?:-Operator und noch ein paar JavaScript-Feinheiten, die Sie aber kennen sollten (Trennen5.htm):

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
function BrowserTrennen()
{
```

```
var plattform = navigator.appVersion.indexOf('Mac') != -1 ? "mac" : "pc";
var browser = navigator.appName.indexOf('Netscape');
var browsver = parseInt(navigator.appVersion);
var bildschirmHoehe = screen.height;
if(plattform=="pc")
{
    if(browser!=-1) // Netscape
    {
        switch(browsver)
        {
            case 2:
            switch(bildschirmHoehe)
            {
                case 480:
                alert("Hallo PC NC 2 mit 480 Hoehe");
                break;
                case 600:
                alert("Hallo PC NC 2 mit 600 Hoehe");
                break;
                case 768:
                alert("Hallo PC NC 2 mit 768 Hoehe");
                break;
                default:
                alert("Hallo PC NC 2 mit grosser Hoehe");
                break;
            }
            break;
            case 3:
            switch(bildschirmHoehe)
            {
                case 480:
                alert("Hallo PC NC 3 mit 480 Hoehe");
                break;
                case 600:
                alert("Hallo PC NC 3 mit 600 Hoehe");
                break;
                case 768:
                alert("Hallo PC NC 3 mit 768 Hoehe");
                break;
                default:
                alert("Hallo PC NC 3 mit grosser Hoehe");
                break;
            }
            break;
            case 4:
            switch(bildschirmHoehe)
            {
                case 480:
                alert("Hallo PC NC 4 mit 480 Hoehe");
                break;
                case 600:
                alert("Hallo PC NC 4 mit 600 Hoehe");
```

```
break;
case 768:
alert("Hallo PC NC 4 mit 768 Hoehe");
break;
default:
alert("Hallo PC NC 4 mit grosser Hoehe");
break;
}
break;
```

```
default:
switch(bildschirmHoehe)
{
case 480:
alert("Hallo PC NC 2 mit 480 Hoehe");
break;
case 600:
alert("Hallo PC NC 2 mit 600 Hoehe");
break;
case 768:
alert("Hallo PC NC 2 mit 768 Hoehe");
break;
default:
alert("Hallo PC NC 2 mit grosser Hoehe");
break;
}
break;
```

```
}
}
else // IE
{
// analog
}
}
else
{
// analog
}
}
</SCRIPT>
<BODY onLoad="BrowserTrennen()">
</BODY>
</HTML>
```



Abbildung 18.8: Differenzierte Trennung

Beispiel 10:

Beide bisher genannten Varianten trennen die beiden Browser-Welten strikt. Oft ist das jedoch nicht unbedingt notwendig. Teile der beiden Welten sind gut kompatibel und man möchte vielleicht diese gemeinsam behandeln. Was jedoch unbedingt getrennt werden muss ist das globale Eventhandling über die jeweils inkompatiblen event-Objekte. So ganz trivial ist dies nicht, da bei der Microsoft-Variante des globalen Eventhandlings bereits im <SCRIPT>-Tag Attribute gesetzt werden müssen. Die Trennung von den Ereignisbehandlungsmodellen kann mit einem kleinen Trick erfolgen. Eine dritte Variante zur automatischen Trennung des Eventhandlings von Browsern (ohne den Rest der Seite auseinander zu dividieren) beruht darauf, dass man zwei getrennte <SCRIPT>-Container benutzt. Der erste <SCRIPT>-Container wird von jedem Browser geladen, verstanden und ausgeführt. Es handelt sich um einfachste JavaScript- Anweisungen. Wenn Sie an dieser Stelle bereits ältere Browserversionen abhalten wollen, können Sie auf language="JavaScript1.2" oder höher abfragen (sinnvoll). Im ersten <SCRIPT>-Container wird dann über eine if- Schleife abgefragt, ob explizit ein Netscape-Browser verwendet wird. Andere Browser werden den in der Schleife enthaltenen Code ignorieren und deshalb keine Probleme bereiten.

Die zweite <SCRIPT>-Container verwendet die spezielle Microsoft- Terminologie für das JavaScript-Eventhandling, aber dies ist auch für Netscape-Browser kein Problem. Die unbekannten Attribute werden einfach ignoriert, und es kommt normalerweise zu keinem Fehler. Die das zweite Script vollkommen ausfüllende if-Schleife hält den Netscape- Browser von den Anweisungen des Explorer ab und so werden Netscape- Browser damit keine Probleme bekommen. Erstellen Sie die Datei Trennen6.htm:

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
<!--
var browser=navigator.appName;
if(browser.search("Netscape")!=-1)
{
alert("Netscape");
}
//-->
</SCRIPT>
<SCRIPT FOR=document EVENT=onclick language="JavaScript">
<!--
if(browser.search("Netscape")==-1)
{
alert("Microsoft");
}
-->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Laden Sie die Datei. Im Internet Explorer tut sich nur etwas, wenn Sie in die Webseite klicken.

Zusammenfassung

Wir haben heute eine ganze Anzahl von Objekten kennen gelernt, auf die man mit JavaScript zugreifen kann. An erster Stelle sind dabei navigator und String zu nennen, aber auch Array, Date und Math sind sehr wichtig. Das Objekt Number wird Ihnen wahrscheinlich ebenso gelegentlich über den Weg laufen, während die restlichen Objekte sehr spezialisierte Anwendungen erlauben und in »Otto-Normal-Scripten« nicht so oft vorkommen werden.

Insbesondere haben wir mittlerweile genügend JavaScript-Techniken parat, um Browser und Plattformen per JavaScript zu identifizieren und entsprechend darauf reagieren zu können.

Fragen und Antworten

Frage:
Wie weit muss die Trennung von Browser- und Plattformwelten in der Praxis gehen?

Antwort:
Grundsätzlich sollte man die Browsertypen trennen, wenn man Style Sheets einsetzt oder globales Event-Handling betreiben möchte. Aber auch andere Indikatoren sprechen für eine Trennung der Browsertypen. Dazu ist die Abfrage, ob JavaScript aktiviert ist oder nicht, eigentlich eine Selbstverständlichkeit, wenn wesentliche Aspekte des Webprojekts davon abhängen. Wenn kein JavaScript aktiviert ist, kann man auch viele Rückschlüsse über die Verwendungsmöglichkeit von anderen Techniken ziehen. Entweder liegt ein ganz alter Browser vor - dann gehen weder Tabellen noch Frames, Java-Applets oder viele neue Multimedia-Erweiterungen. Oder der Anwender hat JavaScript nur deaktiviert. Dann kann man aber beim Anwender eine gewisse Feindlichkeit gegen so einen »Firlefanz« annehmen. Es ist in jedem Fall nicht unwahrscheinlich, dass dann auch Grafiken und alles, was sich sonst noch ausschalten lässt, deaktiviert ist. Ob man noch weiter differenzieren muss, kann bezweifelt werden. Besser ist es, wenn man auf Techniken verzichtet, die eine größere Zielgruppe ausschließen.

Frage:
Ist so eine differenzierte Aufteilung wie in Beispiel 9 überhaupt in der Praxis zu finden?

Antwort:
Ja. Sogar noch abgestuft. Etwa ob Flash-Formate verstanden werden, welche Farbtiefe vorhanden ist, welche PlugIns usw. Man findet so etwas oft dann, wenn Animationen auf der Webseite vorhanden sind, die auf Style Sheets, Bildgrößen und Positionierungen abgestimmt sind.

Workshop

Testen Sie das Math- und das Date-Objekt mit verschiedenen Beispielen. Dies ist ja in dem Kapitel nicht explizit gemacht worden. Nehmen Sie sich auch nochmals Arrays vor und wenden Sie einige der besprochenen Methoden an.

Erstellen Sie ein Script, das die Java-Fähigkeit eines Browsers abfragt.

Wenn Sie viel Zeit und Lust haben - vervollständigen Sie Beispiel 9.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Für was steht NaN?

2. Was ist das Problem an folgendem Vergleich?

```
if(navigator.appName == "Netscape")
```

3. Was macht getDate()?

4. Geht das?

```
berechne = new Function("a","b","return a * b + 5");
```

5. Was macht random() und zu welchem Objekt gehört die Methode?

6. Für was steht NEGATIVE_INFINITY?

7. Für was steht availLeft und zu welchem Objekt gehört es?

8. Was macht slice() und zu welchem Objekt gehört die Methode?

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

1

Viele Objekte können im Rahmen dieses JavaScript-Lehrgangs nur angerissen werden, aber wenn ein Objekt den Rahmen wirklich sprengt, dann dieses.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 3

Tag 19

Die Webseite als Objekt

Heute wollen wir die Webseite im Browser unter dem Objekt-Gesichtspunkt und dem JavaScript-Zugriff darauf betrachten. Zentraler Angelpunkt ist dabei das Objekt `document`, welches die Webseite repräsentiert, mit seinen zahlreichen Eigenschaften (von denen viele selbst wiederum Objekte sind) und Methoden. Eine wesentliche Anwendung für den umfangreichen Praxisteil am Ende ist der Einsatz von dem `document`-Unterobjekt `Image` bzw. dessen zugeordnetem Objektfeld `images`. Das letzte Beispiel in dem Kapitel geht als weitere Anwendung von `document` auf Cookies ein.

Um die Hintergründe dieser Verwaltung einer Webseite zu verstehen, wollen wir den Tag mit einem kleinen Exkurs zu dem DOM-Konzept und zu Dynamic HTML beginnen.

Dynamic HTML und DOM

Wesentliche Grundlagen für die Behandlung einer Webseite als Objekt und Zugriff darauf mittels JavaScript sind zwei Begriffe:

- DOM
- Dynamic HTML

DOM

Dem Zugriff auf eine Webseite unter dem Objekt-Gesichtspunkt liegt ein Konzept zugrunde, das DOM (Document Objekt Model) heißt. Darin wird eine HTML-Seite nicht als statisch aufgebaute, fertige und nicht unterscheidbare Einheit, sondern als differenzierbare Struktur betrachtet. Dies ermöglicht die Behandlung von Bestandteilen der Webseite auch dann, wenn die Webseite bereits in den Browser geladen ist. Und zwar eine Behandlung, die weit über die einfache Interpretation durch den Browser von oben nach unten hinausgeht.

Das Konzept beinhaltet verschiedene Teilaspekte. Es veranlasst beispielsweise einen Browser, eine HTML-Seite zwar wie eine gewöhnliche Textdatei zu lesen und entsprechende HTML-Anweisungen auszuführen. Darüber hinaus wird der Browser jedoch beim Laden der Webseite alle ihm im Rahmen des Konzepts bekannten und einzeln identifizierbaren Elemente einer Webseite bezüglich ihres Typs, ihrer relevanten Eigenschaften und ihrer Position innerhalb der Webseite indizieren. Dies ist eine Art dynamische Datenbank im Hauptspeicher des Rechners, die beim Laden der Webseite aufgebaut und beim Verlassen der Seite wieder gelöscht wird. Ähnliche Elemente werden dabei bei der Indizierung auf gleiche dynamische Datenstapel (Stacks) für die Seite abgelegt, d.h. in der Denkweise einer relationalen Datenbank in den gleichen Datenbanktabellen eingetragen. Auf diese Weise hat der Browser nach dem Laden der

Webseite genaue Kenntnis über alle relevanten Daten sämtlicher eigenständig für ihn ansprechbarer Elemente in der Webseite. Welche das jedoch sind und was er damit anstellen kann, das kann sich je nach Browser erheblich unterscheiden.

Jedes Element (etwa ein bestimmter HTML-Tag) ist in einem eigenen Datensatz abgespeichert, der bei Bedarf auch während der Lebenszeit der Webseite upgedatet wird. Etwa wenn mittels eines Scripts die Position eines Elementes in der Webseite verändert oder über Style Sheets nach dem vollständigen Laden der Webseite das Layout eines Elementes dynamisch verändert wird.

Das DOM-Konzept geht auf die Firma Netscape zurück und wurde mit dem Navigator 2.0 erstmals 1995 in Verbindung mit JavaScript der Öffentlichkeit präsentiert. Microsoft integrierte die hinter dem DOM-Konzept liegende Idee ab dem Internet Explorer 3.0 in Verbindung mit VBScript (wenngleich vollkommen unterschiedlich implementiert).

Ursprünglicher Entstehungsgrund von dem DOM-Konzept war, dass man die Beziehung von einigen Standardobjekten des Browsers zueinander über eine Hierarchie beschreiben wollte (siehe dazu Tag 15). Über das DOM-Konzept wurde, um ein Beispiel zu nennen, das die Webseite beschreibende Dokumentenobjekt mit Namen document in der Hierarchie in einem Fensterobjekt mit Namen window (dem Anzeigebereich des Browsers) angeordnet.

Das anfangs eingesetzte DOM-Konzept erlaubte noch keinen Zugriff auf einzelne Elemente innerhalb einer Webseite oder sonstwelchen Komponenten innerhalb eines dieser Standardobjekte. Jedoch kamen über die einzelnen Phasen der DOM-Versionen immer mehr Komponenten (auch Webseiten-Elemente) hinzu, auf die gezielt zugegriffen werden konnte. Leider verfolgten bei der Umsetzung der DOM-Idee die beiden großen Browserkontrahenten verschiedene Strategien¹, weshalb es mittlerweile zu zahlreichen Inkompatibilitäten zwischen den Browsern kommt.

In dem DOM-Konzept der Version 3.0 des Navigator kann bereits auf PlugIns, Applets und Bilder zugegriffen werden. Andere Elemente wie Überschriften oder Blöcke in Webseiten sind noch nicht erreichbar. Erst ab der Version 4.0 des Navigator (Communicators) kann zudem auf Formulare, Stilinformationen, Anker, Links oder Frames zugegriffen werden. Damit sind zwar die wichtigsten Elemente einer Webseite erreichbar, jedoch beileibe noch nicht alle Elemente. Aber auch die allgemeine Ereignisbehandlung wird im DOM-Konzept geregelt.

Insbesondere war Netscape in Hinsicht auf diese objektorientierte Sichtweise von Webseite und Browser der Zeit (des W3C - nicht der Technik) ein wenig zu weit voraus. Viele gut durchdachte und intelligente Ansätze wurden in das Netscape-Konzept integriert und auch in die gerade aktuellen Navigator-Versionen implementiert. Nur - nicht immer straft das Leben den, der zu spät kommt. Manchmal wird auch der bestraft, der zu früh kommt. Den straft das Internet über dessen konservativen »Reichsverweser« - das W3C. Obwohl wir - wie schon mehrfach angedeutet - diese Sackgasse² der DOM-Geschichte nicht intensiv verfolgen wollen, soll das Konzept zumindest kurz angerissen werden. Das DOM-Modell von Netscape stellt zum Zugriff auf die Ingredienzien einer Webseite noch eine weitere Struktur zur Verfügung - Layer bzw. Schichten, welche auf Seiten von HTML über den <LAYER>-Tag realisiert werden. Die Idee kann man sich vorstellen, als ob jedes einzelne Element einer Webseite auf eine Klarsichtfolie gezeichnet wird. Jeder der Folien wird gekennzeichnet und dann werden alle Folien in einer festgelegten Reihenfolge in Schichten übereinander gelegt.

Zusätzlich wurde in dem Navigator 4.0 eine besondere Art von Style Sheets eingeführt - die JavaScript-Assisted Style Sheets (JSSS)³. In Verbindung damit ergänzt der Navigator 4.0 seine bisherige Objekt-Hierarchie und führt für Webseiten ein neues Objekt ein, das - was leider verwirrend ist - jedoch den gleichen Namen wie das bisher die Webseite repräsentierende bekommt - document. Das neue Objekt kommt immer dann zum Tragen, wenn in Verbindung mit der Webseite diese Style Sheets zum Einsatz kommen (es befindet sich in der Objekthierarchie auch an tieferer Ebene wie das namensgleiche Objekt, das die Webseite repräsentiert). JSSS benutzt JavaScript, um darüber auf die Elemente auf der Webseite

zuzugreifen. Innerhalb eines Style-Sheet-Containers existiert über das neue document-Objekt für jedes Element der Webseite ein eigenes Objekt mit verschiedenen Formatinformationen und Kennzeichnungen. Außerhalb des Style-Sheet-Containers wirkt das alte document-Objekt weiter.

Die Konzeption von Microsoft hingegen stellt zur Unterstützung der DOM- Philosophie ab dem Internet Explorer 4.0 ein eigenes Objekt zur Verfügung (das Objekt all), das bereits überall (nicht nur für die Bereiche der Style Sheets) in der Webseite den Zugang auf sämtliche Inhalte der Seite erlaubt. Das DOM-Modell von Microsoft kann mittlerweile sowohl über VBScript als auch JavaScript angesprochen werden und beinhaltet zudem - im Gegensatz zu dem Netscape-Modell - eine konkrete und weitgehend vollständige Ereignisbehandlung für sämtliche (!) eigenständigen Elemente einer Webseite. Netscape beschränkt die Ereignisbehandlung auf DOM- Ebene wie angedeutet auf wenige ausgewählte Objekte.

Zusammenfassend kann man sich unter DOM die eigenständige Behandlung von Elementen einer Webseite in Verbindung mit Scripten und Style Sheets im Rahmen einer HTML-Seite vorstellen.

Dynamic-HTML

Dynamic-HTML oder DHTML ist ein Begriff, der eng mit dem direkt zuvor beschriebenen DOM zusammenhängt. Teilweise werden DOM und Dynamic-HTML synonym verwendet. Der Begriff ist jedoch weder eindeutig, noch standardisiert oder sonstwie geschützt. Insbesondere legen verschiedene Interessengruppen den Begriff unterschiedlich aus. Es gibt überdies verschiedene Schreibweisen, die aber alle im Wesentlichen das Gleiche meinen. Microsoft schreibt seine Variante Dynamic HTML (mit großem D), Netscape dagegen dynamic HTML (mit kleinem d). Bei aller Differenz in der konkreten Auslegung - die wichtigsten Protagonisten im Internet verstehen unter dynamischem HTML die Veränderungen einer Webseite, nachdem die Seite bereits beim Client (also im Browser) angelangt ist.

Wenn nun unter dem DOM-Konzept (welches ja sowohl Netscape als auch Microsoft - wenn auch technisch unterschiedlich - unterstützen) temporär alle notwendigen Daten gespeichert sind, um Elemente einer im Browser geladenen Webseite wiederzufinden, können diese Komponenten angesprochen, ausgewertet oder verändert werden.

Das document-Objekt

Das document-Objekt ist sicher eines der in dem Buch am meisten explizit verwendeten Objekte. Es ist die Repräsentation der verschiedenen ansprechbaren Elemente einer Webseite und besonders deshalb für uns wichtig, weil darüber die Möglichkeit besteht, HTML-formatierten Text in das angezeigte Dokument über Scripte zu schreiben. Es ist also die Basis für dynamisch generierten Text. In der Objekthierarchie von JavaScript liegt das Objekt unterhalb des window-Objekts und enthält die Unterobjekte all (nur Internet Explorer), anchors, applets, embeds, forms, images, layers (nur Netscape Navigator), links und plugins (dies ist jedoch teilweise Browser-abhängig). Eine Objektinstanz von document wird automatisch erzeugt, wenn der Browser einen einleitenden <BODY>-Tag in der HTML- Datei vorfindet.



Sowohl das Objektfeld forms als auch das von einigen document-Eigenschaften und -Methoden generierte Style-Objekt behandeln wir gesondert an den nächsten beiden Tagen.

Das Objekt document besitzt über die Unterobjekte hinaus diverse Eigenschaften, die sowohl abzufragen als auch teilweise zu ändern sind. In der nachfolgenden Tabelle finden Sie die Eigenschaften, die aber nicht konsequent in allen Browsern unterstützt werden (da hilft oft nur testen, was ja allgemein eine Notwendigkeit ist).

Eigenschaften

Eigenschaft	Beschreibung
alinkColor	Die Farbe für Verweise zu bereits besuchten Verweiszielen, wie sie bei der Zusatzangabe alink im <BODY>-Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
all	Über dieses Unterobjekt hat man Zugriff auf sämtliche Inhalte der Webseite (das Objekt gehört allerdings nicht zur offiziell unter JavaScript unterstützten Objektmenge und wird nur beim Microsoft Internet Explorer unterstützt). Das all-Objekt verfügt über Methoden, mit denen Sie beispielsweise sowohl HTML-Tags in eine Webseite dynamisch einfügen als auch entfernen können. Das Objekt besitzt insbesondere selbst ein weiteres Unterobjekt (style), worüber alle Stilinformationen von sämtlichen Elementen in der Webseite abgefragt und meist auch geändert werden können. Dies erfolgt über die Syntax <pre>document.all.tags["(Tagname)"].[index].style.[Eigenschaft/Methode]</pre> Beispiel: <pre>document.all.tags("H2")[0].style.color</pre> In dem Beispiel sprechen Sie die erste Überschrift der Ordnung 2 in dem Dokument an. Neben dem Namen können Sie auch wieder über eine ID arbeiten (darauf kommen wir nochmals am letzten Tag bei Style Sheets zu sprechen). Dies funktioniert über die Syntax <pre>document.all.[id].style.[Eigenschaft/Methode]</pre> Die Stil-Eigenschaften vom all-Objekt unterscheiden sich in der Schreibweise ein wenig von der Schreibweise bei den CSS1-Attributen. Dies führt oft zu Fehlern, da es nur geringe Unterschiede sind. Im Allgemeinen gilt, dass der Bindestrich bei CSS1-Attributen bei den Eigenschaften des all-Objekts entfällt und der nachfolgende Buchstabe groß zu schreiben ist (Groß- und Kleinschreibung ist zu beachten) Beispiele: aus font-size wird fontSize, aus background-color wird backgroundColor aus text-align wird textAlign. Wenn man diese Regel einhält, werden in den meisten Fällen die Eigenschaften des all-Objektes direkt aus den CSS1-Attributen herzuleiten sein. Beispiel 5 zeigt das Objekt in einem vollständigen Quelltext.

anchors	Ein Verweisanker-Objektfeld. Mit document.anchors[0] greift man auf den Namen des ersten Verweisankers in der Datei, mit document.anchors[1] auf den Namen des zweiten Verweisankers zu usw. Die Eigenschaft length (die document-Eigenschaft anchors als Objektfeld gesehen hat wiederum selbst Eigenschaften) beinhaltet die Anzahl der in der Datei definierten Verweisanker. Über name und text können Sie den Namen und Text eines Ankers bestimmen, die Eigenschaften x und y sind die Position des Ankers in Pixeln - relativ zur linken oberen Ecke des Dokuments.
applets	Ein Objektfeld für den Zugriff auf Java-Applets in einer HTML-Datei. Ein Objekt hat alle in Java als public deklarierten Eigenschaften als Eigenschaften. Über applets kann u.a. auch die Anzahl der Java-Applets in einer Webseite bestimmt werden.
bgColor	Angabe der Hintergrundfarbe der HTML-Datei, wie sie bei der Zusatzangabe bgcolor im <BODY>-Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
classes	Die Eigenschaft kreiert ein Style-Objekt, worüber die Stilangaben eines HTML-Tags mit Hilfe des dort angegebenen class-Attributs spezifiziert werden können. Syntax: <pre>document.classes.[className].[tagName]</pre> Die Angabe className ist der Wert von dem class-Attribut und tagName der Name von einem beliebigen HTML-Tag, beispielsweise H5 oder P. Wenn der Wert von tagName als all gesetzt wird, werden darüber sämtliche HTML-Tags referenziert.
cookie	Eine Eigenschaft für die Cookie-Verwaltung.
embeds	Ein Array, das als Einträge jedes PlugIn in dem Dokument enthält.
fgColor	Die Textvordergrundfarbe der HTML-Datei, wie sie bei der Zusatzangabe text im <BODY>-Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
formName	Für jedes benannte Formular in einem Dokument wird eine eigene Eigenschaft erstellt. Der Name von dieser Eigenschaft ist der Wert von dem HTML-name-Attribut.
forms	Ein Objektfeld für den Zugriff auf die Formulare, welche innerhalb der HTML-Datei definiert wurden. Mit document.forms[0] erfolgt der Zugriff auf das erste Formular usw. Die Eigenschaft length (die document-Eigenschaft forms als Objektfeld gesehen hat wiederum selbst Eigenschaften) beinhaltet die Anzahl der in der Datei definierten Formulare.
height	Höhe des Dokuments in Pixeln.
ids	Die Eigenschaft kreiert ein Style-Objekt, worüber die Stilangaben eines HTML-Tags mit Hilfe des dort angegebenen id-Attributs spezifiziert werden können. Syntax: <pre>document.ids.[idWert]</pre>

images	Ein Array, das die Bilder der Webseite als Elemente enthält. Das Objektfeld hat selbst die Eigenschaft <code>complete</code>, ein Boolean-Wert der anzeigt, ob ein Bild komplett geladen ist. Dazu gibt es die Eigenschaften <code>border</code> <code>height</code> <code>hspace</code> <code>lowsrc</code> <code>name</code> <code>src</code> <code>vspace</code> <code>width</code> was jeweils den gleichnamigen HTML-Attributen bei dem Tag entspricht. Wichtig für viele Einsätze bei der Arbeit mit Bildern ist auch die direkte Verwendung des <code>Image()</code>-Konstruktors zur Erstellung von Bildobjekten. Darüber kann explizit ein Bild geladen werden, das nicht sofort in der Webseite angezeigt wird. Dazu finden Sie mehr in Beispiel 9 und 10.
lastModified	Datum und Uhrzeit der letzten Änderung der Datei im internationalen Format nach GMT (Greenwich-Zeit).
layers	Ein Array, das jeden Layer der Webseite als Elemente enthält (nur Netscape).
linkColor	Angabe der Farbe für Verweise zu noch nicht besuchten Verweiszeilen, wie sie bei der Zusatzangabe <code>link</code> im <code><BODY></code> -Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
links	Ein Objektfeld für den Zugriff auf Verweise, die innerhalb der HTML-Datei definiert wurden. Mit <code>document.links[0]</code> erfolgt der Zugriff auf den ersten Link usw. Die Eigenschaft <code>length</code> (die <code>document</code>-Eigenschaft <code>links</code> als Objektfeld gesehen hat wiederum selbst Eigenschaften) beinhaltet die Anzahl der in der Datei definierten Verweise. Das Objektfeld hat selbst die Eigenschaften <code>hash</code> <code>host</code> <code>hostname</code> <code>href</code>

	pathname port protocol search die identisch mit den gleichnamigen Eigenschaften des location-Objekts sind sowie die Eigenschaften target für die gleichnamige HTML-Angabe, text für den Inhalt des korrespondierenden <A>-Tags und die Eigenschaften x und y mit der Position des Links in Pixeln - relativ zur linken oberen Ecke des Dokuments.
location	Der vollständige URL der HTML-Datei. Die Eigenschaft gilt als deprecated und wurde durch document.URL ersetzt.
plugins	Ein Array, das jedes PlugIn der Webseite als Elemente enthält.
referrer	Der URL der HTML-Datei, von der aus die aktuelle HTML-Datei aufgerufen wurde, als String.
tags	Die Eigenschaft kreiert ein Style-Objekt, worüber die Stilangaben eines HTML-Tags spezifiziert werden können. Syntax: document.tags.[tagName] Dabei ist tagName der Name eines beliebigen HTML-Tags, beispielsweise H1 oder BLOCKQUOTE.
title	Der Titel der HTML-Datei, wie er bei <TITLE> angegeben ist.
URL	Der vollständige URL der HTML-Datei (die Ablösung von location).
vlinkColor	Farbe für Verweise zu bereits besuchten Verweiszeilen, wie sie bei der Zusatzangabe vlink im <BODY>-Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
width	Die Breite des Dokuments.

Tabelle 19.1: Eigenschaften von document

Methoden

Die wichtigsten Methoden von dem Objekt document dürften die zum Schreiben in die Seite und dem Behandeln von Ereignissen sein. Das Objekt document besitzt aber noch mehr interessante Methoden:

Methoden	Beschreibung

<code>captureEvents()</code>	Überwacht im Rahmen des globalen Ereignisbehandlungsmodells Anwenderereignisse innerhalb des Dokumentfensters. Ereignisse außerhalb (beispielsweise Titelleiste, Menüleisten) werden nicht überwacht (wird nur von Netscape verstanden).
<code>close()</code>	Hiermit wird ein Dokument geschlossen. Dies erzwingt die sofortige Anzeige der bis dahin vorhandenen Daten. Man verwendet diese Methode normalerweise in Verbindung mit <code>document.open()</code> .
<code>contextual(context1, ...[contextN,] affectedStyle)</code>	Die Methode spezifiziert den Stil eines einzelnen HTML-Tags, wobei man verschiedene Tags schachteln kann, sodass nur die so verschachtelten Tags betroffen sind. Die Parameter sind über die oben genannten Eigenschaften spezifizierte HTML-Tags, denen eine bestimmte Stilinformation zugewiesen wird. Etwa wird so jedem EM-Tag innerhalb einer H1-Überschrift die Farbe Blau zugewiesen: <pre><STYLE TYPE="text/JavaScript"> contextual(document.tags.H1, document.tags.EM).color="blue"; </STYLE></pre> Auch ohne den <code><STYLE></code>-Container kann man die Methode nutzen. Dann verwendet man wie den <code><SCRIPT></code>-Container: <pre><SCRIPT LANGUAGE="JavaScript1.2"> document.contextual(document.tags.H1, document.tags.EM).color="blue"; </SCRIPT></pre> Mehr dazu finden Sie in Beispiel 3 und 4.
<code>getSelection()</code>	Ermittelt, welchen Text ein Anwender im Dokument selektiert hat, und gibt den Inhalt als String zurück.
<code>handleEvent()</code>	Im Rahmen des globalen Ereignisbehandlungsmodells Übergabe eines Ereignisses an ein Element, das in der Lage ist, auf das Ereignis zu reagieren (wird nur von Netscape verstanden)
<code>open()</code>	Die Methode öffnet ein Dokument zum Schreiben. Optional kann der Dateityp als Mime-Typ-Parameter übergeben werden. Erlaubt sind die Angaben <code>text/html</code> , <code>text/plain</code> , <code>image/gif</code> , <code>image/jpeg</code> , <code>image/x-bitmap</code> oder diverse PlugIn-Typen.
<code>releaseEvents()</code>	Veranlasst das Fenster oder das Dokument, aufgefangene Events des angegebenen Typs entlang der Objekthierarchie weiterzureichen.
<code>routeEvent()</code>	Weiterleitung eines Events.
<code>write(), writeln()</code>	Schreiben von HTML-formatiertem Text in das Dokumentfenster. Im Unterschied zu <code>write()</code> wird bei <code>writeln()</code> noch ein Zeilenumbruchzeichen am Ende eingefügt. Bitte beachten Sie, dass dies in einem HTML-Dokument keinen Zeilenumbruch bewirkt.

Tabelle 19.2: Methoden von document

Praktischer Einsatz von document

Da das document-Objekt so grundlegend für die Verbindung von den Elementen der Webseite und JavaScript ist, soll in diesem Abschnitt eine größere Anzahl an praktischen Beispielen mit dem document-Objekt und seinen Unterobjekten, Eigenschaften und Methoden durchexerziert werden.

Beispiel 1:

Das erste Beispiel testet diverse Eigenschaften vom document-Objekt, die Eigenschaften von HTML-Elementen aus der Webseite abfragen. Beachten Sie, dass wir diese Informationen über alert() ausgeben. Damit dort nur definierte Zeilenumbrüche entstehen, wird das Sonderzeichen \n für einen Umbruch im Mitteilungsfenster verwendet. Außerdem wird das Script im Bodybereich ganz am Ende notiert. Das ist ein sinnvoller Trick, wenn man auf Elemente in einer Webseite zugreifen möchte und diese vor einer Ausführung des Scripts vorhanden sein müssen (Doc1.htm):

```
<HTML>
<HEAD>
<TITLE>Meine Homepage</TITLE>
</HEAD>
<BODY TEXT="#001100" BGCOLOR="#EEFFFF"
LINK="#FF0000"
ALINK="#FFF000" VLINK="#FFAA00">
<A NAME="Anfang">
<H1>
Willkommen
</H1>
</A>
<applet
    code=UhrApplet.class
    width=100
    height=80 >
</applet>
<A NAME="Ende">
<H4>
Hier ist das Ende erreicht
</H4>
</A>
<SCRIPT LANGUAGE="JavaScript">
var text =
"alinkColor: " + document.alinkColor+
"\nanchors[0].name: " + document.anchors[0].name+
"\nanchors[0].text: " + document.anchors[0].text+
"\nanchors[0].x: " + document.anchors[0].x+
"\nanchors[0].y: " + document.anchors[0].y+
"\nanchors[1].name: " + document.anchors[1].name+
"\nanchors[1].text: " + document.anchors[1].text+
"\nanchors[1].x: " + document.anchors[1].x+
"\nanchors[1].y: " + document.anchors[1].y+
"\napplets: " + document.applets+
"\nbgColor: " + document.bgColor+
"\nfgColor: " + document.fgColor+
"\nheight: " + document.height+
```



```
"\nlastModified: " + document.lastModified+
"\nlinkColor: " + document.linkColor+
"\ntitle: " + document.title+
"\nvlinkColor: " + document.vlinkColor+
"\nwidth: " + document.width;
alert(text);
</SCRIPT>
</BODY>
</HTML>
```



Abbildung 19.1: Details der Webseite beim IE

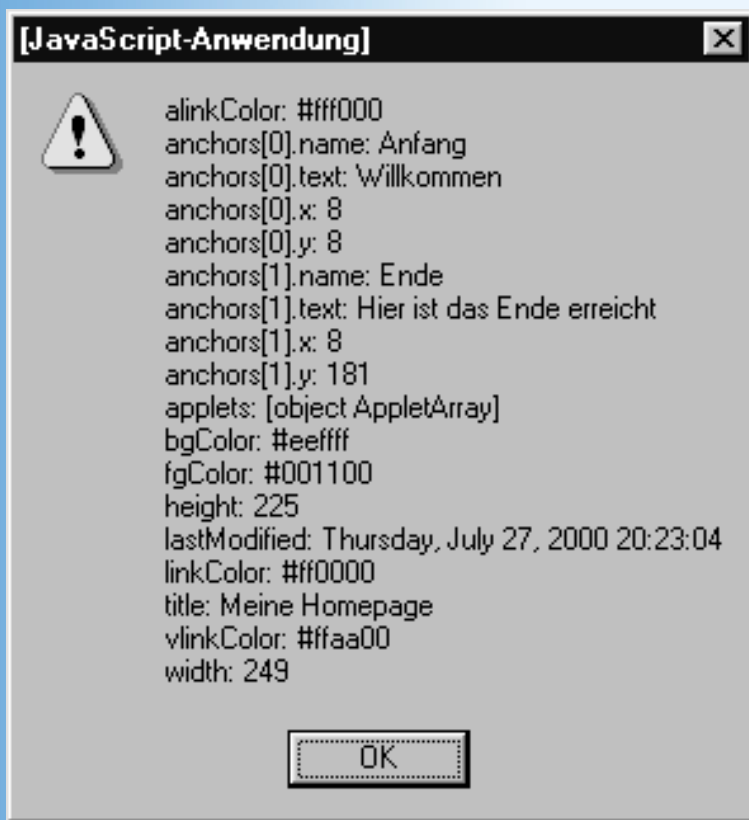


Abbildung 19.2: Details der Webseite beim Navi-gator

Laden Sie die Webseite in verschiedene Browser. Sie werden sehen, dass nicht alle Eigenschaften unterstützt werden.

Beispiel 2:

Das zweite Beispiel geht nun den umgekehrten Weg wie Beispiel 1 und setzt über das document-Objekt einige Eigenschaften der Webseite. Hier funktioniert im Navigator nicht alles (doc2.htm):

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
function lade()
{
document.bgColor = "red";
document.fgColor = "white";
document.title = "Meine Homepage";
}
</SCRIPT>
<BODY onLoad="lade()">
<H1>
Willkommen
</H1>
<H4>
Hier ist das Ende erreicht
</H4>
</BODY>
</HTML>
```



Abbildung 19.3: Der Navigator setzt weder Titel noch Vordergrundfarbe



Abbildung 19.4: Der Internet Explorer kann es

Beispiel 3:

Das dritte Beispiel soll den Einsatz der Methode `contextual()` in Verbindung mit dem `<STYLE>`-Container zeigen (`doc3.htm`):

```
<HTML>
<STYLE TYPE="text/JavaScript">
```

```
contextual(document.tags.H1,  
document.tags.EM).color="yellow";  
</STYLE>  
<BODY>  
<H1>Willkommen <EM>AUF MEINER</EM> Homepage  
</H1>  
<EM>Dat is normales em  
</EM>  
<H1>Hier ist das Ende erreicht  
</H1>  
</BODY>  
</HTML>
```

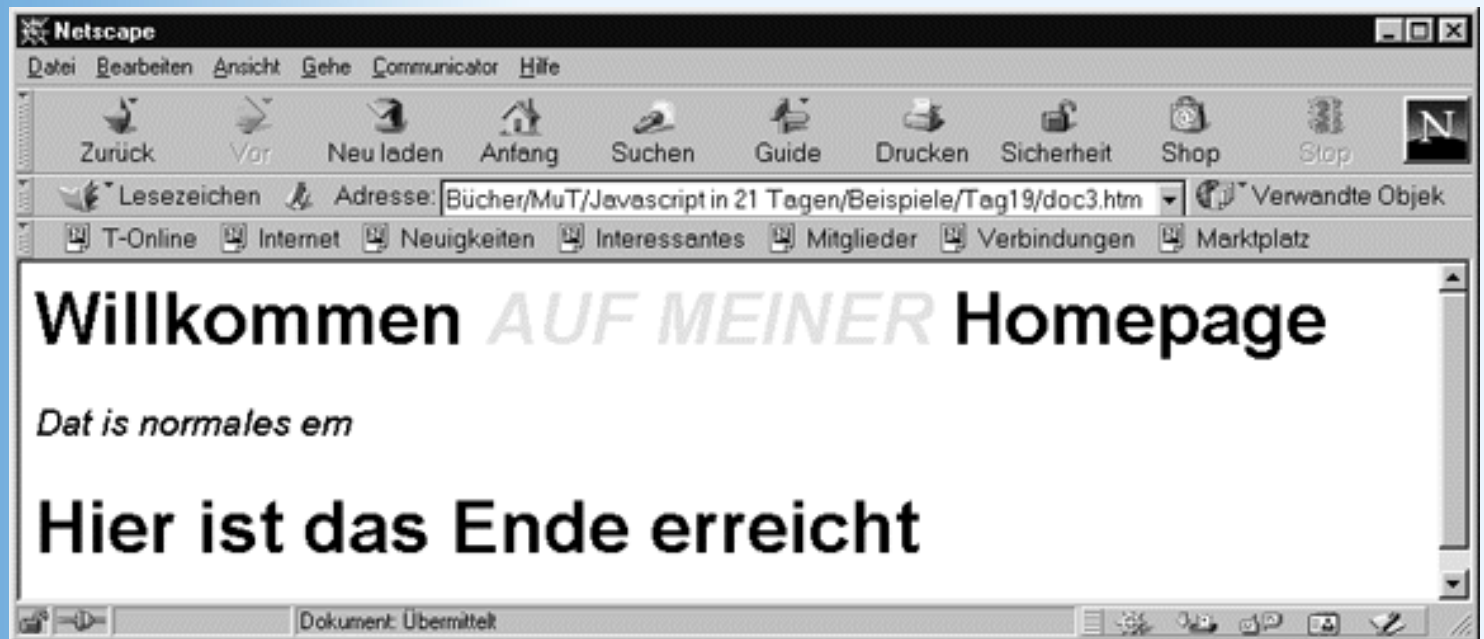


Abbildung 19.5: In der Überschrift ist der Tag verändert

Beispiel 4:

Beispiel vier demonstriert `contextual()` in Verbindung mit dem `<SCRIPT>`- Container und dem Einsatz der `tags`-Eigenschaft (doc4.htm):

```
<HTML>  
<SCRIPT LANGUAGE="JavaScript1.2">  
document.contextual(document.tags.DIV, document.tags.I).color="yellow";  
</SCRIPT>  
<BODY>  
<H1>Willkommen <I>AUF MEINER</I> Homepage  
</H1>  
<DIV>  
DIV oder nicht DIV, <I>das ist hier die Farbe</I>.  
</DIV>  
</BODY>  
</HTML>
```

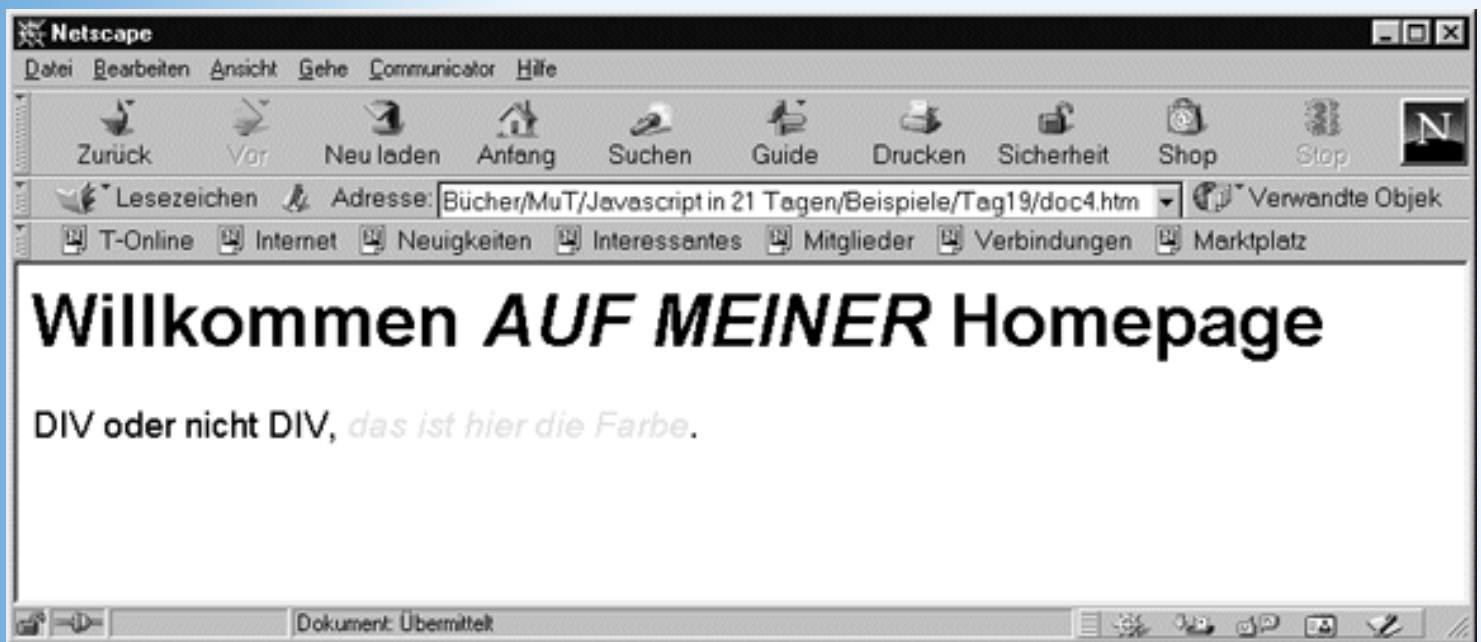


Abbildung 19.6: In DIV-Bereich ist der Tag verändert

Beispiel 5:

Wir wollen nun das Microsoft-spezifische Objekt all in der Praxis testen. Erstellen Sie dazu die Datei Dok5.htm:

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
function lade()
{
document.all.tags("h1")[0].style.color="red";
document.all.tags("h1")[1].style.color="blue";
document.all.tags("h3")[0].style.color="yellow";
document.all.tags("BODY")[0].style.backgroundColor=
"cyan";
document.all.tags("SPAN")[0].style.fontStyle="italic";
document.all.tags("DIV")[0].style.fontSize="44";
}
</SCRIPT>
<BODY onLoad="lade()">
<CENTER>
<H1>Willkommen
</H1>
<H3>auf
</H3>
<H1>meiner
</H1>
<H1>Homepage
</H1>
Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla
<DIV>DIVDIDIV DIVDIDIV
</DIV>
<SPAN>SPANSPAN SPAN SPANSPAN
```



```
</SPAN>
<BR>
Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla
<H3>Hier ist das Ende erreicht
</H3>
</CENTER>
</BODY>
</HTML>
```

Laden Sie die Datei in verschiedene Browser. Der Navigator wird sich sehr pikiert zeigen, einen Fehler melden und nur HTML-formatierten Text anzeigen.

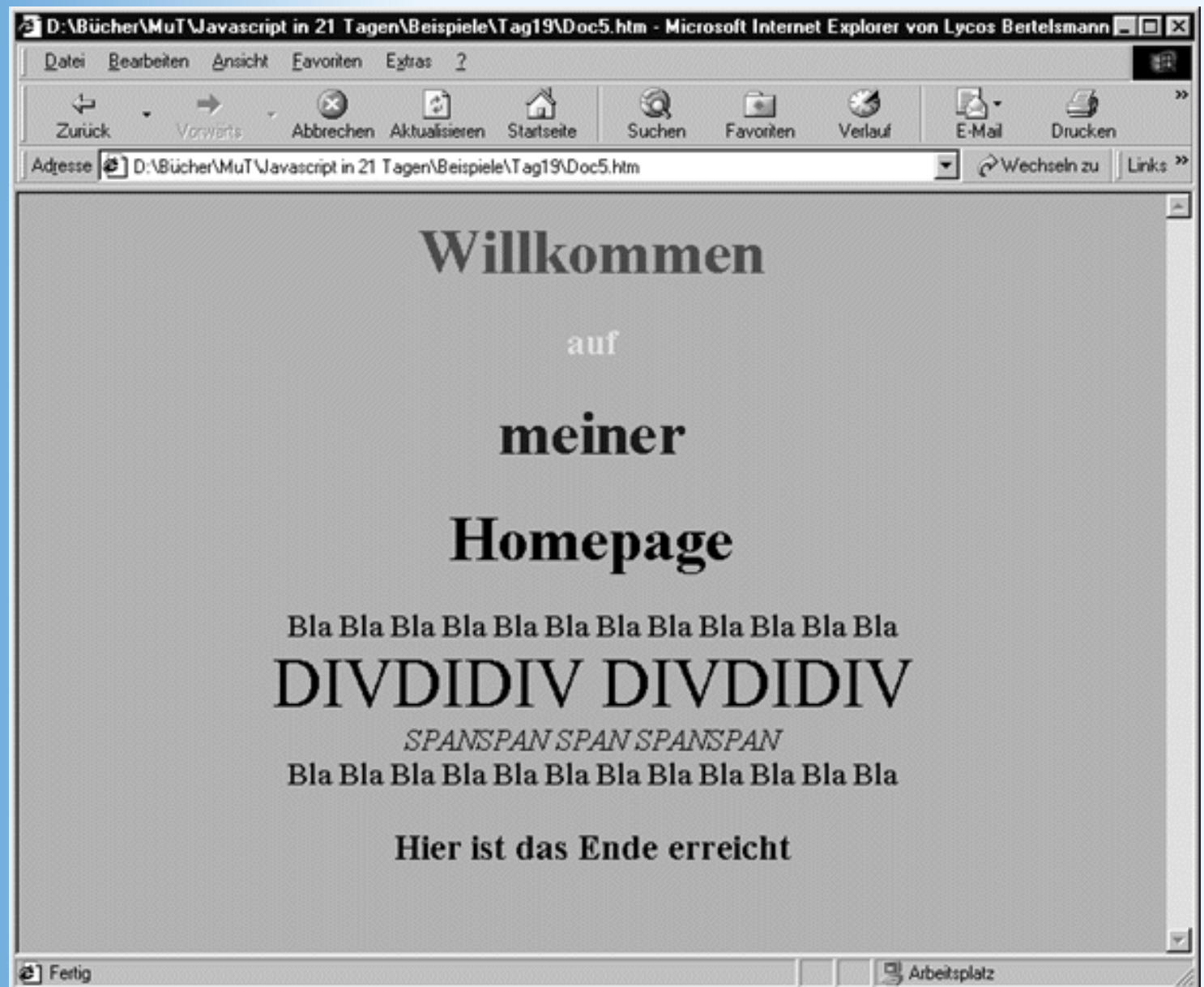


Abbildung 19.7: Wer kann, der kann

Der Internet Explorer wird in Formatierungen schwelgen.

In diesem Beispiel wird die Verwendung des `images[]`-Arrays demonstriert. Es wird dazu verwendet, nach dem Laden von verschiedenen Bildern einige ihrer Eigenschaften abzufragen und auszugeben ([doc6.htm](#)):

file:///C:/temp/pdf/JavaScript in 21 Tagen/kap19.htm (16 von 30) [14.04.2003 18:39:35]

```
" " + document.images[0].src +  
"\nBild 2: " + document.images[1].height +  
" " + document.images[1].width +  
" " + document.images[1].src +  
"\nBild 3: " + document.images[2].height +  
" " + document.images[2].width +  
" " + document.images[2].src +  
"\nBild 4: " + document.images[3].height +  
" " + document.images[3].width +  
" " + document.images[3].src +  
"\nBild 5: " + document.images[4].height +  
" " + document.images[4].width +  
" " + document.images[4].src +  
"\nBild 6: " + document.images[5].height +  
" " + document.images[5].width +  
" " + document.images[5].src;  
alert(text);  
</SCRIPT>  
</BODY>  
</HTML>
```

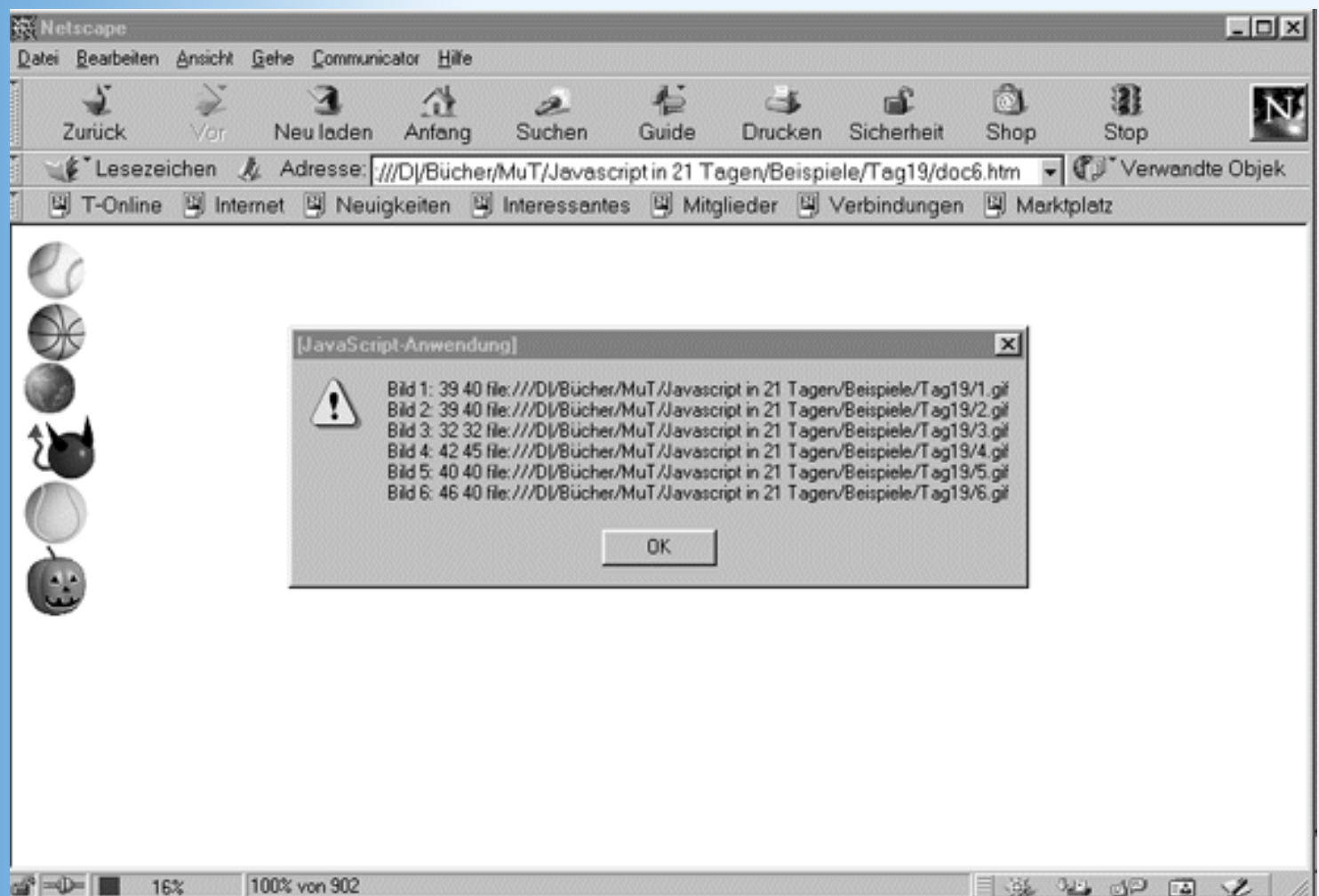


Abbildung 19.9: Informationen zu den Bildern

Beispiel 7:

Beispiel 7 soll das Setzen der Eigenschaften von Bildern in der Webseite über die Verwendung des

images[]-Array veranschaulichen (doc7.htm):

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
function lade()
{
for(i=0;i<6;i++)
{
document.images[i].border = "5";
document.images[i].height = "50";
}
}
</SCRIPT>
<BODY>
<IMG src="1.gif">
<br>
<IMG src="2.gif">
<br>
<IMG src="3.gif">
<br>
<IMG src="4.gif">
<br>
<IMG src="5.gif">
<br>
<IMG src="6.gif">
<br>
<FORM>
<INPUT type="Button" value="Klick" onClick="lade()">
</FORM>
</BODY>
</HTML>
```

Beachten Sie den Zugriff auf die Array-Elemente mit der for-Schleife (siehe Abbildung 19.10).

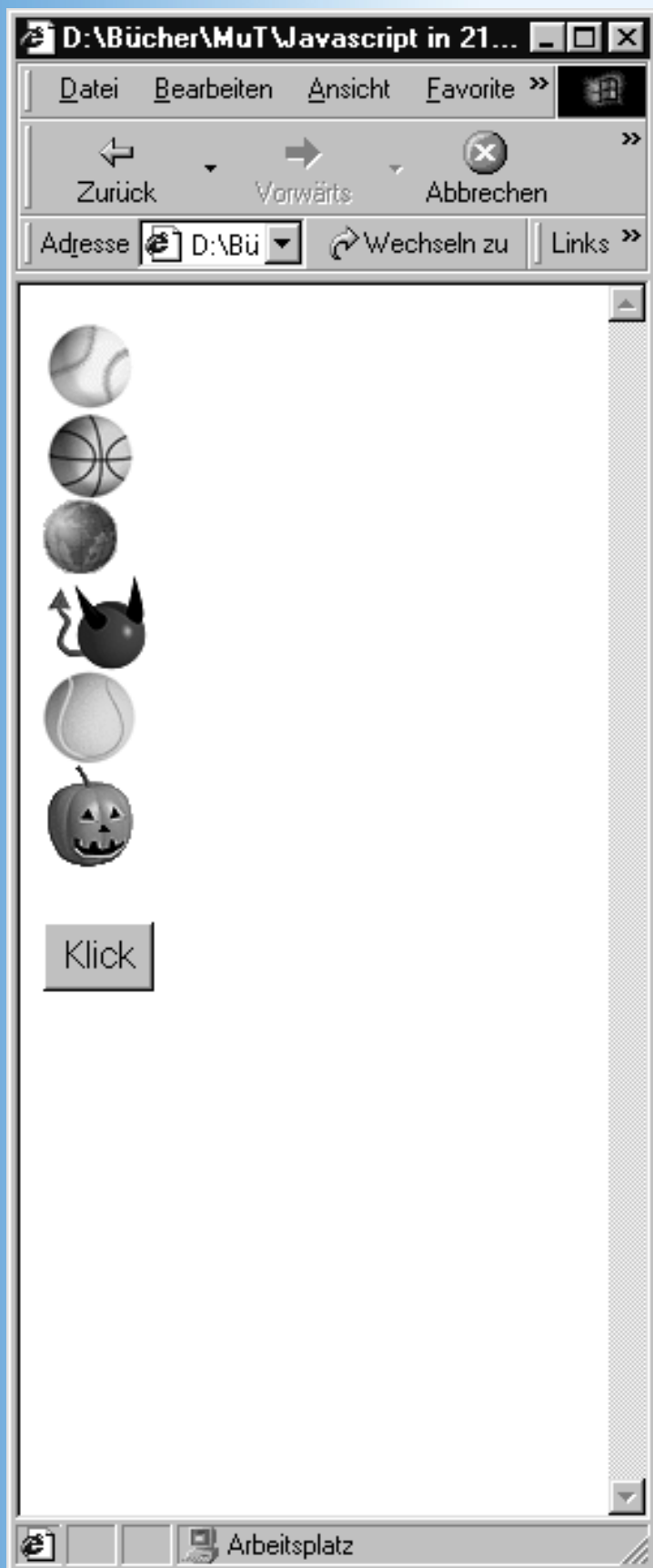


Abbildung 19.10: Vor dem Klick im IE

Laden Sie die Datei in den Internet Explorer und klicken Sie auf die Schaltfläche im Formular (siehe Abbildung 19.11).

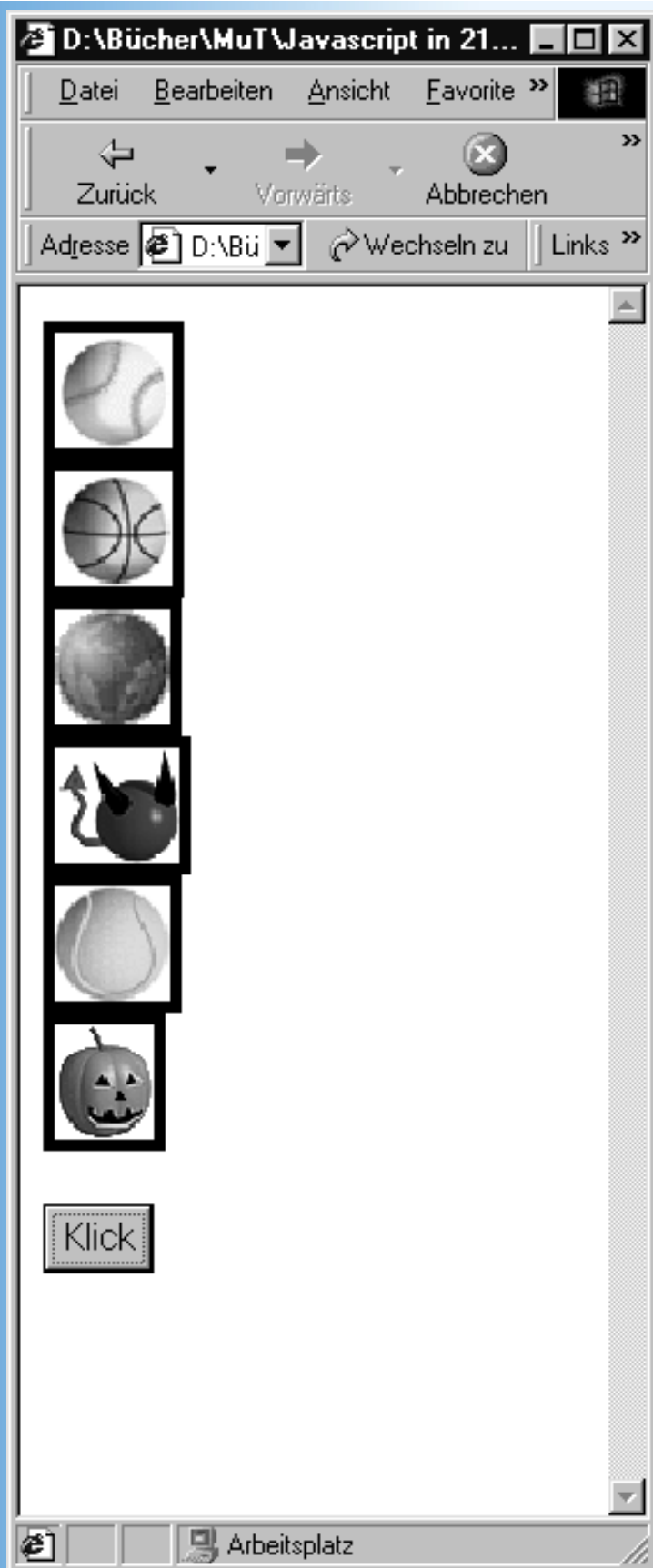


Abbildung 19.11: Nach dem Klick im IE

Der Navigator wird mit dem Beispiel Probleme haben und die Grafiken nicht verändern.



Bei der Arbeit mit Bildern und Stilinformatiionen wird man mit document schnell an Grenzen

stoßen und mit fehlender Unterstützung in Browsern zu kämpfen haben. Grundsätzlich bietet sich dann an, auf Style Sheets auszuweichen. Das werden wir an Tag 21 machen.

Beispiel 8:

Beispiel 8 überschreibt über die src-Eigenschaft von images Bilder von links nach rechts in dem Array. Die Anzahl der Bilder wird über document.images.length aus JavaScript heraus bestimmt (Doc8.htm):

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
var meineBilder= new Array();
function lade()
{
for(i=0;i<document.images.length;i++)
{
document.images[i].height = "75";
}
}

function tausche()
{
for(i=0;i<document.images.length-1;i++)
{
document.images[i].src = document.images[i+1].src;
}
}
</SCRIPT>
<BODY onLoad="lade()">
<IMG src="1.gif">
<IMG src="2.gif">
<IMG src="3.gif">
<IMG src="4.gif">
<IMG src="5.gif">
<IMG src="6.gif">
<br>
<FORM>
<INPUT type="Button" value="Klick" onClick="tausche()">
</FORM>
</BODY>
</HTML>
```

Laden Sie die Datei in einen Browser und klicken Sie auf die Schaltfläche im Formular.

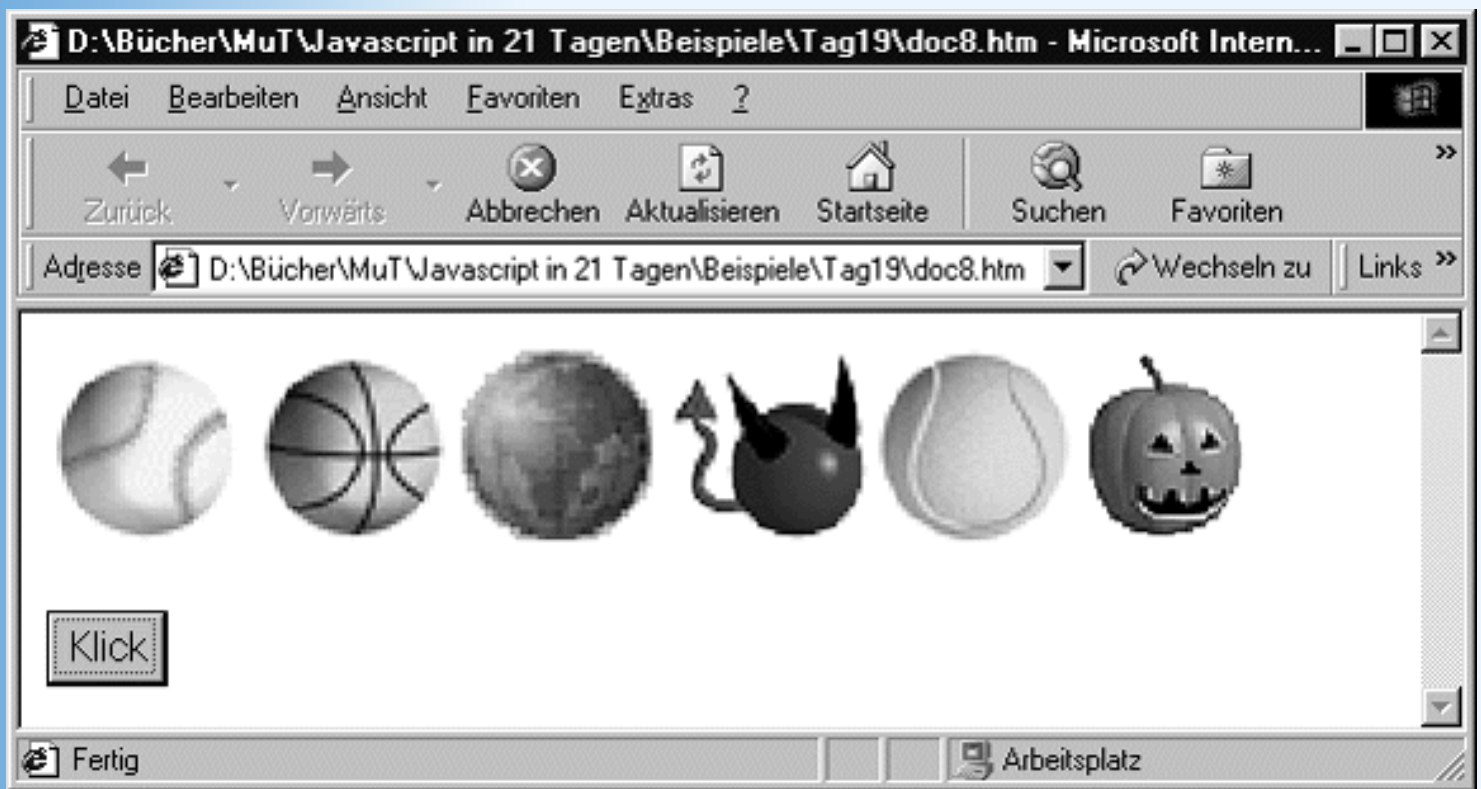


Abbildung 19.12: Vor dem Klick im IE

Eine Permutation, in der das erste Bild bei einem Klick auf die letzte Position geschoben wird, ist mit diesem Aufbau nicht möglich. Es werden nur Referenzen auf Array-Einträge kopiert und ein Array-Eintrag geht bei jeder Aktion als eigenständige Referenz auf ein Bild verloren. Für so was wie eine Permutation werden wir aber auch noch eine Lösung sehen. Und zwar mit dem Ansatz, den das nächste Beispiel verfolgt - unter der Verwendung von Image selbst.



Abbildung 19.13: Der Kürbis frisst sie alle

Beispiel 9:

In der nachfolgende Webseite wird erstmals nur ein einziges Bild angezeigt. Dieses steht dann in `document.images[0].src` zur Verfügung. Das Array- Element wird über das Script dynamisch mit verschiedenen Inhalten gefüllt. Hier verwenden wir explizit `Image` selbst, um über dessen Konstruktor `Bild-` Objekte zu erstellen und diesen Bilder zuzuweisen. Diese werden einem Array zugeordnet, bereits im Hintergrund geladen und erst dann aufgerufen, wenn Bedarf besteht, d.h. unter JavaScript eine Situation geschaffen wird, die ein bestimmtes Bild `document.images[0].src` zuweist (`Doc9.htm`):

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
var meineBilder = new Array();
var zaehler = 0;
meineBilder[0] = new Image();
meineBilder[1] = new Image();
meineBilder[2] = new Image();
meineBilder[3] = new Image();
meineBilder[4] = new Image();
meineBilder[5] = new Image();

function lade()
{
document.images[0].height = "75";
meineBilder[0].src="1.gif";
meineBilder[1].src="2.gif";
meineBilder[2].src="3.gif";
meineBilder[3].src="4.gif";
meineBilder[4].src="5.gif";
meineBilder[5].src="6.gif";
zaehler=1;
}
function tausche()
{
document.images[0].src = meineBilder[zaehler].src;
if(zaehler<5)
{
zaehler=zaehler + 1;
}
else
{
zaehler=0;
}
}
</SCRIPT>
<BODY onLoad="lade()">
<IMG src="1.gif" height=75>
<br>
<FORM>
<INPUT type="Button" value="Klick" onClick="tausche()">
</FORM>
</BODY>
</HTML>
```

Laden Sie die Datei in einen Browser und klicken Sie mehrfach auf die Schaltfläche im Formular.

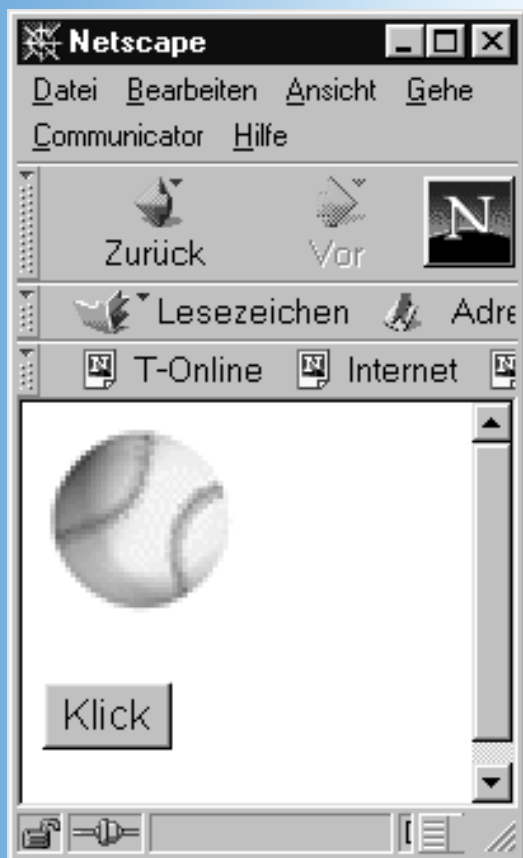


Abbildung 19.14: Vor dem Klick

Hier haben wir eine Permutation auf einer Bildposition, in der der Reihe nach die geladenen Bilder angezeigt werden. Vom letzten Bild geht es wieder zum ersten.



Abbildung 19.15: Und noch ein paar Klicks weiter

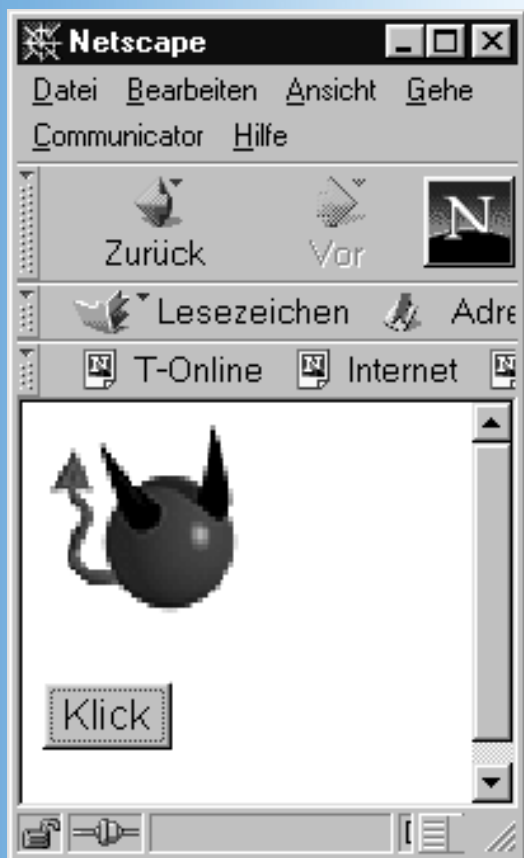


Abbildung 19.16: Ein paar Klicks weiter

Beispiel 10:

Abschließend zu diesem Bilder-Thema wollen wir noch eine Animation auf dieser Basis und die Veränderung eines Bilds beim Überstreichen mit dem Mauszeiger demonstrieren. Letzteres ist die für die Anwendung von JavaScript schon fast berühmte Technik, beispielsweise einen Hyperlink - wie in dem Beispiel - beim Überstreichen mit der Maus mit einem anderen Bild zu versehen und damit eine Zusatzinformation zu verbinden. Bei dem Beispiel greifen wir für die Animation auf die Rekursion über `window.setTimeout()` und `onMouseOver` bzw. `onMouseOut` bei der Veränderung des grafischen Hyperlinks zurück (Doc10.htm):

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
var meineBilder = new Array();
var zaehler = 0;
meineBilder[0] = new Image();
meineBilder[1] = new Image();
meineBilder[2] = new Image();
meineBilder[3] = new Image();
meineBilder[4] = new Image();
meineBilder[5] = new Image();

function lade()
{
document.images[0].height = "75";
meineBilder[0].src="1.gif";
meineBilder[1].src="2.gif";
meineBilder[2].src="3.gif";
meineBilder[3].src="4.gif";
```

```
meineBilder[4].src="5.gif";
meineBilder[5].src="6.gif";
zaehler=1;
}

function tausche()
{
document.images[0].src = meineBilder[zaehler].src;
if(zaehler<5)
{
zaehler=zaehler + 1;
}
else
{
zaehler=0;
}
ani();
}

function rein()
{
document.images[1].src = meineBilder[4].src;
}
function raus()
{
document.images[1].src = meineBilder[2].src;
}

function ani()
{
setTimeout('tausche()',250);
}

</SCRIPT>
<BODY onLoad="lade()">
<CENTER>
<IMG src="1.gif" height=75>
<P>
<A href="JavaScript:ani()"
onMouseOver="rein()" onMouseOut="raus()">
<IMG src="3.gif" height=50 border=0>
</A>
</CENTER>
</BODY>
</HTML>
```

Laden Sie die Datei in einen Browser und bewegen Sie den Mauszeiger auf die untere Grafik. Eine zweite Grafik wird an ihrer Stelle angezeigt. Klicken Sie darauf, und die Animation wird gestartet.

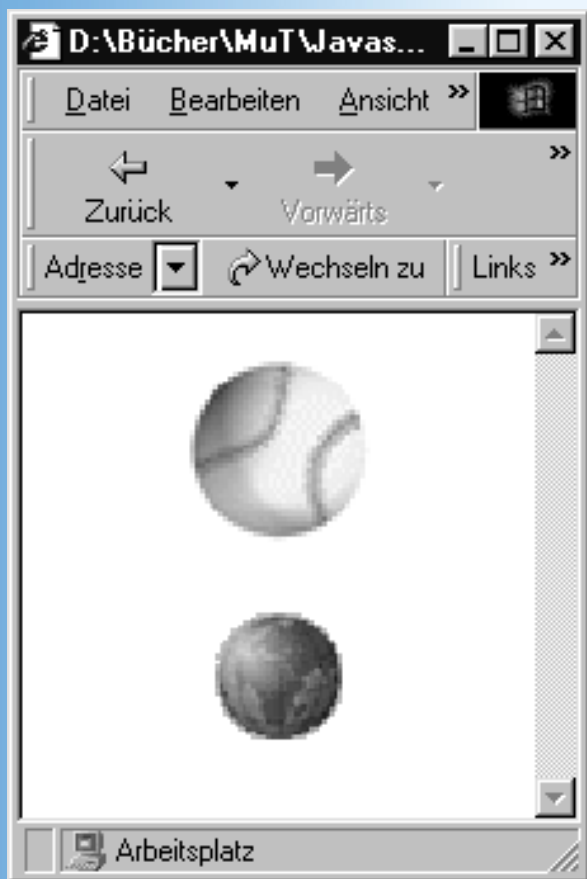


Abbildung 19.17: Mauszeiger außerhalb des Links

Hier haben wir eine Permutation auf einer Bildposition, in der der Reihe nach die geladenen Bilder angezeigt werden. Vom letzten Bild geht es wieder zum ersten.

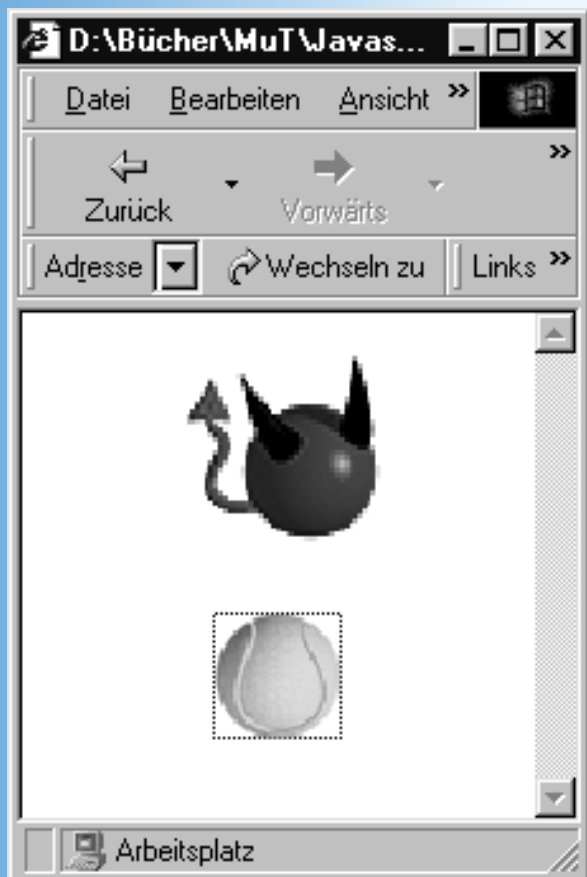


Abbildung 19.18: Mauszeiger innerhalb des Links. Zudem läuft die Animation

Cookies

Dieses abschließende Thema soll das Cookie-Handling über das document- Objekt skizzieren. Ein Cookie ist immer eine kleine Textinformation, welche von einem Browser in ein im Browser vorgegebenes Verzeichnis geschrieben und von dort wieder ausgelesen werden kann. Internet Explorer und Netscape Navigator gehen dabei zwar einen etwas unterschiedlichen Weg bezüglich der konkreten Dateien. Der Navigator speichert normalerweise alle Einträge in einer einzigen Datei namens cookies.txt in dem im Browser dafür vorgesehenen Verzeichnis. Der Internet Explorer erstellt dagegen gewöhnlich für jedes Cookie eine eigene Datei im Cookies-Verzeichnis von Windows. Dabei können in der Regel maximal 20 davon verwaltet werden. Für deren Verwaltung ist die dort zu findende Datei index.dat zuständig.

Ausnahmsweise spielt aber diese unterschiedliche Handhabung auf Dateiebene für die Vorgehensweise unter JavaScript keine Rolle. Grundsätzlich kann ein Cookie mit document.cookie gesetzt werden. Ergänzend muss man für verschiedene Operationen auf String-Methoden wie substring(), charAt(), indexOf() oder lastIndexOf() zurückgreifen.

Generell besteht ein Cookie mindestens aus der Angabe des Namens und einer Wertzuweisung. Über den Bezeichner wird beim Auslesen ein Cookie wieder identifiziert. Der zugewiesene Wert ist die Information, welche man eigentlich speichern möchte. Eine wichtige optionale Eigenschaft ist die expires-Eigenschaft, mit der ein Gültigkeitsdatum eines Cookies gesetzt werden kann. Das Format:

Wdy, DD-Mon-YY HH:MM:SS GMT

Wdy ist eine String-Repräsentation des Tags der Woche (englisch), DD zweistellig der Monatstag, Mon eine aus drei Buchstaben bestehende String- Repräsentation des Monats (englisch) und YY das zweistellige Jahr. HH, MM, und SS sind zweistellige Angaben für die Stunden, Minuten und Sekunden (optional).

Beispiel:

expires=Wednesday, 09-Nov-01 20:15:40 GMT

Weitere optionale Eigenschaften sind domain, worüber der URL der Domain als String spezifiziert werden kann, für welche ein Cookie gültig ist, path zur Angabe von Verzeichnissen innerhalb dieser Domain (fehlt die Angabe, wird der Pfad des aktuellen Dokumentes angenommen) und secure als Boolean-Wert zur Festlegung, dass ein Cookie nur über einen sicheren (d.h. verschlüsselten) Pfad übertragen werden darf.

Die an die cookie-Eigenschaft von document zu übergebende Zeichenkette muss die gewünschten Angaben als durch Parameter getrennte Werte enthalten. Soweit ist es also ganz einfach, ein Cookie zu setzen. Etwa über

```
document.cookie = "meinCookie=123";
```

Oder wenn ein Haltbarkeitsdatum gesetzt werden soll:

```
document.cookie =  
"meinCookie=123; expires=Wednesday, 09-Nov-01 20:15:40 GMT";
```

Leider ist es nicht ganz so einfach, wenn man komplexere Informationen in einem Cookie speichern und wieder auslesen möchte. JavaScript bietet da erst einmal keine große Unterstützung. Cookie-Werte dürfen beispielsweise keine Leerzeichen und keine Sonderzeichen enthalten. Deshalb müssen solche Zeichen mit der Funktion escape() beim Schreiben maskiert und mit unescape() beim Lesen wieder entschlüsselt werden. Des Weiteren müssen verschiedene Aktionen wie das Bestimmen der Länge eines Wertes oder das Trennen an bestimmten Zeichen im String durchgeführt werden. Gott sei Dank muss das unter JavaScript nicht alles neu programmiert werden - man greift auf einen Satz von freien Funktionen für das Cookie- Management zurück, die ein Programmierer namens Bill Dortch entwickelt hat und unter <http://www.hidaho.com/cookies> kostenlos bereitgestellt werden. Mit dessen Funktionen

SetCookie(name,value,expires,path,domain,secure)

und

`GetCookie(name)`

zum Setzen und Lesen von beliebigen Angaben ist das Cookie- Management dann wirklich eine Kleinigkeit. Die zusätzliche Funktion

`DeleteCookie(name, path, domain)`

erlaubt das Löschen eines Cookies, indem das Verfalldatum auf den 1. Januar 1970 gesetzt wird. (Ein Trick, der eine Eigenschaft von Cookies ausnutzt.)

Zusammenfassung

Der Tag hat mit einem kleinen Exkurs zu dem DOM-Konzept und zu Dynamic HTML begonnen, welcher für das Verständnis von vielen Vorgängen im Zusammenhang mit dem Objekt `document`, seinen zahlreichen Eigenschaften (von denen viele selbst wiederum Objekte sind) und Methoden grundlegend ist. Grundsätzlich werden die Elemente einer Webseite über Objekt `document` und seine Unterobjekte repräsentiert. Von einfachen HTML-Formatierungsanweisungen über Bilder, Plugins, Applets, Formulare hin zu Cookies.

Grundlage für den Zugriff auf eine Webseite unter dem Objekt- Gesichtspunkt ist das Document Object Model, in dem eine HTML-Seite nicht als statisch aufgebaute, fertige und nicht unterscheidbare Einheit, sondern als differenzierbare Struktur betrachtet wird. Dies ermöglicht die Behandlung von Bestandteilen der Webseite auch dann, wenn die Webseite bereits in den Browser geladen ist. Es ist auch die Basis dessen, was unter dynamischem HTML zu verstehen ist - die Veränderungen einer Webseite, nachdem die Seite bereits beim Client (also im Browser) angekommen ist.

Fragen und Antworten

Frage:
Formulare werden doch auch über das Objekt `document` angesprochen. Was ist damit?

Antwort:
Dazu kommen wir morgen in einem eigenständigen Kapitel.

Frage:
Wenn man mit dem Objekt `document` bereits so viele Dinge gestalten und auch dynamisch nach dem Laden einer Webseite verändern kann, wofür spricht dann noch der Einsatz von Style Sheets?

Antwort:
Style Sheets bieten einige entscheidende Vorteile. Da ist einmal die Trennung der Logik. Stilinformationen werden vollkommen separiert. Außerdem kann man mit Style Sheets noch viel mehr machen, als nur mit JavaScript und dem `document`-Objekt. Zu guter Letzt werden Style Sheets auch einheitlicher in den Browsern gehandhabt (wenngleich noch immer nicht perfekt).

Workshop

Erweitern Sie Beispiele um die Eigenschaften, welche nicht explizit durchgespielt wurden.

Nehmen Sie sich die Beispiele vor, in denen Eigenschaften beim Laden einer Webseite über JavaScript und `document`-Eigenschaften gesetzt wurden, und erweitern Sie diese Beispiele so, dass über Anwenderaktionen (Klick auf Schaltflächen, Hyperlinks, Mausbewegungen usw.) dynamische

Veränderungen ausgelöst werden.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Stimmt das?

```
var i = new image();
```

2. Für was ist DOM die Abkürzung?

3. Für was ist JSSS die Abkürzung?

4. Was passiert im Netscape Navigator, wenn er das Objekt all vorfindet?

5. Was ist referrer?

❖ Kapitel Inhalt Index **SAMS** Top Kapitel ❖

1
Was wirklich niemanden überraschen sollte.

2
Das muss leider so klar und deutlich festgehalten werden.

3
Wobei auch diese sich so wie es aussieht nicht im Web durchsetzen - an CSS führt kein Weg vorbei.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 3

Tag 20

Formulare

Formulare sind eine der wichtigsten Möglichkeiten der Interaktion zwischen dem Betrachter einer Webseite und dem Ersteller der Seite. Als Konkurrenztechnik gibt es nur noch den Fall einer einfachen Email, wobei in der Webseite einfach an irgendeiner Stelle eine entsprechende Referenz der Form

```
<A href="mailto:[Email-Adresse]">[anzuzeigender Text]
</A>
```

notiert wird, oder die Verwendung eines Gästebuches, was aber entweder recht aufwendig ist und Serverzugriff voraussetzt oder auf eine fremde Dienstleistung zugreift.

Beides ist jedoch in vielen Fällen nur eine Notlösung. HTML stellt für die Erstellung von Formularen Anweisungen bereit, womit sich insbesondere standardisierte Kommunikationswege besser und komfortabler realisieren lassen. Ein Formular hat gegenüber einer konventionellen Email dabei einige wesentliche Vorteile. So können von einem Formular komfortabel gleichartige, datensatzorientierte Informationen erfasst werden, die sich beispielsweise über Scripte und Datenbankprogramme gut auswerten und speichern lassen. Aber auch für den Anwender ergeben sich Vorteile. Bei intelligent gemachten Formularen kann sich die Eingabe oft auf ein paar Mausklicks und wenige Texteingaben reduzieren. HTML stellt Anweisungen zur Generierung von Eingabefeldern, mehrzeiligen Textfeldern, Listen und Aufzählungen, Schaltflächen und beschreibendem Text bereit.

Das große Problem an Formularen unter reinem HTML ist jedoch, dass damit keinerlei Auswertung möglich ist. An dieser Stelle ist man auf Server- seitige Techniken wie Server-seitige Scripte/Programme und/oder CGI angewiesen. Aber auch auf Seiten des Clients kann man bereits einiges machen. Da greift nun JavaScript ein, das über document und dessen Unterobjekt form auf Formulare zugreifen kann. In wieweit man dann Aufgaben bereits beim Client erledigen kann, hängt massiv von der Zielsetzung des Formulars ab. Während Datenbankaktionen eher die Serverseite beanspruchen, kann das Abfangen von fehlerhaften Benutzereingaben bereits oft auf Clientseite realisiert werden.

Die Grundlagen für den Zugriff auf Formulare unter JavaScript und die entsprechenden Objekte haben wir bereits zusammen. Schwerpunkt dessen, was uns noch zu Formularen fehlt, ist der HTML-Aspekt.

HTML-Grundlagen für Formulare

Die rein optische Seite eines Formulars im Rahmen der Webseite (inklusive der Reaktionsmöglichkeiten auf Anwenderaktionen - etwa Selektion oder Deselektion) wird in reinem HTML erstellt. Deshalb müssen wir uns zuerst mit diesen HTML-Grundlagen beschäftigen.

Bei Formularen gibt es einen äußeren

<FORM>

-Container und darin eingeschlossene Container und einzelne Steueranweisungen, welche die einzelnen Formularelemente definieren. Den äußeren Container ergänzen in der Regel zwei bis drei weitere, durch ein Leerzeichen getrennte Angaben. Diese sind im Prinzip optional. In der Webseite wird das Formular ohne Einschränkungen zu sehen sein und auch als reine Benutzeroberfläche einwandfrei funktionieren, wenn innerhalb der <FORM>-Containers korrekte Formular-Elemente notiert werden. Sinnvoll kann das Formular dann aber nur zum optischen Test oder zur Aktivierung von Script-Aufrufen verwendet werden.

Eine in der Regel notwendige Angabe ist gewöhnlich eine Adresse, wo das Formular ausgewertet werden soll, wenn es fertig ausgefüllt und abgeschickt wurde. Wenn die Auswertung nicht lokal erfolgt, ist dies der URL eines Servers und dort das für CGI eingerichtete Verzeichnis samt dort platziertem CGI-Script. Zu dieser Spezifikation dient die Angabe:

`action=[URL]`

Der URL bei der Angabe action setzt sich wie üblich zusammen aus:

- dem Protokolltyp,
- dem Namen des Rechners,
- dem Pfadnamen zu dem CGI-Verzeichnis,
- dem Namen des CGI-Scripts, auf das verwiesen wird.

Die gesamte Angabe muss in Anführungszeichen eingeschlossen werden.

Die zweite Standardangabe ist eine Methode, wie das Formular zur Auswertungsstelle gelangt und dort zu behandeln ist. Die Erweiterung

`method=[Methode]`

legt die Art der Übertragung bzw. Behandlung fest. Dabei wird gewöhnlicherweise einer von den nachfolgenden zwei Werten verwendet.

GET
POST

Beide Angaben machen nur in Kombination Sinn. Es ist sogar so, dass normalerweise zusätzlich - gerade in Verbindung mit Scripten - ein weiteres sehr wichtiges Attribut gesetzt wird - name. Darüber kann man z.B. bei der Kombination von Scripten und Formularen auf das Formular und darüber auf die einzelnen Elemente in dem Formular zugreifen (bei sämtlichen Formularelementen kann das optionale Attribut genauso gesetzt werden). Der schematische Aufbau eines äußeren (fast vollständigen - es gibt noch den Inhaltstyp, die Inhaltstypliste und die Zeichensatzliste - siehe dazu weiterführende HTML-Literatur) Formular-Containers sieht also so aus:

```
<FORM
  action="[URL]"
  method=[GET/POST]
  name=[Bezeichner]>
...    [Formular-Elemente]
</FORM>
```

Ein Gerüst für einen Formular-Container, der das Formular mit der GET- Methode an das CGI-Script meinScript.cgi im Verzeichnis cgi-bin auf dem Webserver www.abc.de schickt, sähe so aus:

```
<FORM
action="http://www.abc.de/cgi-bin/meinScript.cgi"
method=GET>
...
```

</FORM>

Die Verwendung von Formularen ist wie bereits mehrfach angedeutet nicht auf CGI mit Server-Kontakt beschränkt. So kann man aus den Eingabedaten eines Formulars eine beliebige neue HTML-Seite erzeugen, wie wir es an Tag 8 und 14 in einigen Beispielen gemacht haben.

Eine weitere Variante beim Abschicken des Formulars ergibt sich daraus, dass viele Provider keine Server-seitigen Aktionen wie die Ausführung von CGI-Scripten durch Anwender erlauben. Deshalb bleibt für die Auswertung von Benutzereingaben nur die Übermittlung dieser Daten per Email. Darunter ist dann aber nicht die gewöhnliche Email gemeint, die an irgendeiner Stelle in der Webseite per Verweis notiert wird und wo der Anwender im Email-Client sämtliche Angaben außer der Email-Adresse ergänzen muss. Eine Email kann leicht mit einem Formular so gekoppelt werden, dass sie im Wesentlichen aus den Eingabewerten des Anwenders besteht. Dies ist dann identisch zu den anderen Übermittlungssituationen - nur werden die vom Anwender eingegebenen Daten per Email an ein Postfach und nicht an einen Webserver verschickt.

Dies funktioniert so: Der Protokolltyp ist normalerweise bei Verweisen auf WWW-Servern `http://`. Jeder URL, der ein CGI-Script oder ein Programm auf dem Server starten soll, verwendet dieses Protokoll in der `action`-Angabe. Wenn ein Formular als Email versendet werden soll, muss in dem `<FORM>`-Tag nur das Protokoll im `action`-Attribut auf `mailto:` geändert bzw. insgesamt ein Mail-URL notiert werden (bitte beachten, dass hinter dem Attribut auf `mailto` nur ein Doppelpunkt folgt und die beiden Slashes unterbleiben). Das `method`-Attribut sollte auf `POST` gesetzt werden. Dies sieht dann schematisch so aus:

```
<FORM action="mailto:[Email-Adresse]" method=POST>
```

Das nachfolgende skizzierte Gerüst wäre ein Formular-Container, der das Formular per Email abschickt:

```
<FORM action="mailto:rjst@gmx.de" method=POST>
```

```
...
```

```
</FORM>
```



Ein grundsätzliches Problem stellen bei Formularen, die per `mailto` ihre Daten wegschicken, diejenigen Browser dar, die nicht korrekt dafür konfiguriert oder erst gar nicht dazu in der Lage sind (die `mailto`-Variante ist für Formulare irgendwo nur ein Notnagel). Oft kommt es vor, dass beim Abschicken eines solchen Formulars einfach das dem Browser assoziierte Mailprogramm geöffnet wird, ohne die Werte aus dem Formular zu übernehmen oder gar zu versenden. Anwender solcher Plattformen bekommen beim Abschicken eines `mailto`-Formulars Probleme. Um die gesamte Problematik zu umgehen, kann man einen öffentlichen CGI-Service (etwa CWAk Mailservice - <http://mailservice.cwak.com/>) verwenden, der - für Privatanwender meist kostenlos - Formulardaten von einem CGI-Programm verarbeiten lässt und dann als E-Mail weiterschickt.

Einige Hintergründe zu Formularen

Nahezu jedes (sinnvolle) Formular besitzt eine Schaltfläche zum Absenden des Formulars. Wenn nun die entsprechende Schaltfläche betätigt wird, findet normalerweise die Übertragung an den Webserver statt. Bevor die Daten an den Server gesendet werden, werden sie zunächst vom Browser zu einer einzigen Zeichenkette kodiert. Die Anwendereingaben werden, durch standardisierte Trennzeichen separiert, einfach hintereinander notiert. Die Art und Weise der Datenübertragung wird durch den Parameter `method` gesteuert. Dabei wird grundsätzlich zwischen den beiden Methoden `GET` und `POST` unterschieden. Der Unterschied zwischen den beiden Methoden besteht hauptsächlich darin, wie der WWW-Server das zurückgeschickte Formular speichert und verarbeitet.

Bei der GET-Methode wird das zurückgeschickte Formular in einer Standard-Umgebungsvariablen gespeichert, welche auf dem Server ein CGI-Programm auswertet. Anschließend verarbeitet das CGI-Programm den Inhalt weiter.

Die Methode POST hingegen veranlasst ein CGI-Programm, das Formular wie eine auf der Kommandozeilenebene erfolgte Benutzereingabe zu behandeln. Es gibt daher kein EndOfFile-Signal (EOF) und die Länge der übermittelten Daten muss vom CGI-Programm aus einer weiteren Standard-Umgebungsvariablen ausgelesen werden.

Wenn die Formularabsendung den Server erreicht, erkennt dieser, dass es sich nicht um eine Webseitenanforderung handelt, sondern ein CGI- Programm zur Auswertung eines Formular gestartet werden soll. Sobald dies geschehen ist und das Programm die gewünschten Aktionen durchgeführt hat, wird die Antwort an den Browser zurückgesendet.

Die im <FORM>-Container enthaltenen Elemente

Ein Formular kann folgende Elemente enthalten:

- einzeilige Eingabefelder,
- Schaltflächen,
- mehrzeilige Eingabefelder,
- Auswahllisten,
- Radiobuttons,
- Checkbuttons.

Für die Eingabe von freien Texten gibt es bei Formularen sowohl einzeilige als auch mehrzeilige Eingabefelder. Diese stellen die unter fast jeder grafischen Oberfläche üblichen Funktionalitäten bereit:

- Einzeilige Eingabefelder dienen zur Eingabe von Zahlen und/oder wenigen Wörtern. Also Text im Allgemeinen Sinn, der nur in einer Zeile eingegeben werden kann. HTML versteht darunter aber auch die Eingabe über Schaltflächen. Diese werden mit dem gleichen Tag realisiert.
- Mehrzeilige Eingabefelder dienen zur Eingabe von längeren Texten.

Die Beschriftung der Eingabefelder erfolgt bei Feldern zur Eingabe von allgemeinem Text einfach durch die den Eingabefeldern vorangestellten Texte (reiner HTML-Klartext mit allen Formatierungsmöglichkeiten). Bei Buttons ist es etwas anders - das sehen wir gleich.

Einzeilige Eingabefelder, Schaltflächen, Radiobuttons und Checkbuttons

Einzeilige Eingabefelder werden über den <INPUT>-Tag realisiert. Dieser hat u.a. folgende Attribute:

- Einen (optionalen) formularweit eindeutigen Namen über name="[Bezeichner]". Der Bezeichner für den Feldnamen darf wie üblich keine Leerzeichen und keine deutschen Umlaute enthalten und muss in Anführungszeichen gesetzt werden.
- Die (optionale) Angabe der maximal anzeigbaren Zeichen wird über das Attribut size=[numerischer Wert] gesetzt. Diese Angabe beschränkt in keiner Weise die Anzahl der Zeichen, die ein Anwender eingeben darf. Wenn mehr Zeichen als anzeigbar eingegeben werden, wird der Inhalt seitwärts gescrollt.
- Die (optionale) Angabe der maximal einzugebenden Zeichen wird über das Attribut maxlength=[numerischer Wert] gesetzt. Wenn diese Angabe die maximal anzeigbaren Zeichen überschreitet, wird bei Bedarf automatisch seitwärts gescrollt.
- Die (optionale) Angabe des Typs wird über das Attribut type=[Typ] gesetzt.
- Die (in der Regel optionale) Angabe des Vorbelegungswerts wird über das Attribut value=[Wert]

gesetzt. Im Fall von Schaltflächen ist das die Beschriftung des Buttons.

Die Syntax für eine vollständige Anweisung eines einzeiligen Eingabefelds sieht so aus:

```
<INPUT
  name="[Feldname]"
  size=[Wert]
  maxlength=[Wert]
  type=[Typ]
  value=[Wert]>
```

Die Angabe der maximal anzeigbaren Zeichen und der maximal einzugebenden Zeichen ist optional. Wenn diese Angaben nicht gemacht werden, bekommt das Eingabefeld eine voreingestellte Größe (in der Regel 20 Zeichen) zugewiesen und es kann eine beliebige Anzahl von Zeichen eingegeben werden (bis auf Plattformbeschränkungen). Wenn die Anzahl der eingegebenen Zeichen die maximal anzeigbaren Zeichen überschreitet, wird automatisch seitwärts gescrollt (die Größe eines Eingabefeldes verändert sich nach dem Laden nicht mehr). Fehlt die Typ-Angabe, wird ein einfaches Eingabefeld generiert. Es gibt aber verschiedene Typen von einzeiligen Eingabefeldern. Das optionale Attribut type kann den genauen Typ des Eingabefelds festlegen. Das Attribut kann folgende Ausprägungen annehmen:

Wert	Beschreibung
button	Eine Schaltfläche ohne besondere Funktion. Meist nur sinnvoll in Kombination mit einem Scriptaufruf und der Belegung der Beschriftung über value.
checkbox	Die Typangabe spezifiziert einen Checkbutton. Checkbuttons sind Kontrollfelder, die einzeln und in einer Gruppe auftreten und im Fall einer Gruppe unabhängig von einander selektiert werden können.
date	Nur die Eingabe eines Kalenderdatums ist erlaubt.
file	Ein Spezialfall der Befehlsschaltflächen ist die Typausprägung file. Damit kann ein Standard-Dateiauswahlfenster des Betriebssystems geöffnet werden, welches die dort durch den Anwender ausgewählte Datei in das dort ebenfalls spezifizierte Eingabefeld übernimmt. Dies kann beispielsweise für die Angabe der Datei angewendet werden, die per Upload zu einem Server übertragen werden soll.
float	Nur die Eingabe von Dezimalkommazahlen ist erlaubt.
int	Nur die Eingabe von Ganzzahlen ist erlaubt.
password	Die Feldeingaben erfolgen verdeckt. Bei der Dateneingabe erscheinen dann nur Sternchen im Eingabefeld.
radio	Die Typangabe spezifiziert einen Radiobutton. Radiobuttons sind Optionsfelder, die eine Gruppe bilden und von denen immer genau eins selektiert ist ¹ .
reset	Eine Schaltfläche zum Leeren der Formulareingaben. Meist nur sinnvoll in Kombination mit der Belegung der Beschriftung über value.
submit	Eine Schaltfläche zum Abschicken des Formulars. Meist nur sinnvoll in Kombination mit der Beschriftung über value.
url	Nur die Eingabe eines URL.

Tabelle 20.1: Werte von type

Schauen wir uns ein paar Muster an.^{[1](#)}

Ein einzeiliges Eingabefeld mit zehn sichtbaren Zeichen und maximal 15 erlaubten Eingabezeichen würde so aussehen:

Bitte geben Sie Ihren Namen ein:

```
<INPUT name="Name" size=10 maxlength=15>
```

Ein einzeliges Eingabefeld mit theoretisch 30 sichtbaren Zeichen, aber nur maximal zehn erlaubten Eingabezeichen:

Bitte geben Sie Ihren Nachnamen ein:

```
<INPUT name="Nachname" size=30 maxlength=10>
```

Ein einzeliges Eingabefeld ohne Angaben für die theoretisch sichtbaren und die maximal erlaubten Eingabezeichen oder sonst eine Spezifikation. Die Größe wird nach Defaultwerten gesetzt:

Bitte geben Sie was ein: `<INPUT>`

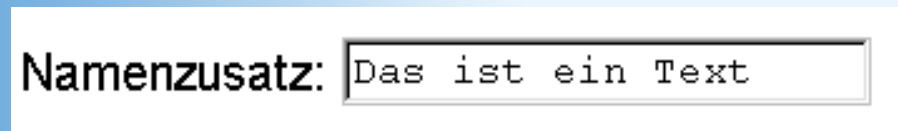


Abbildung 20.1: Ein einzeliges Eingabefeld

Eine Befehlsschaltfläche in Verbindung mit einem Event-Handler und dem Aufruf einer JavaScript-Funktion.

```
<INPUT type=button value ="Ende"
onClick='JavaScript:alert("Hallo")'>
```

Eine Befehlsschaltfläche zur Bereitstellung eines Eingabefeldes, welches über das Standard-Dateiauswahlfenster des Betriebssystems gefüllt werden kann:

```
<INPUT type=file size=60 name=uploadDatei>
```

Eine Befehlsschaltfläche zum Abschicken eines Formulars:

```
<INPUT type="submit" name="Ok" value="OK">
```

Eine Befehlsschaltfläche zum Zurücksetzen eines Formulars:

```
<INPUT type="reset" name="leer" value="Abbruch">
```

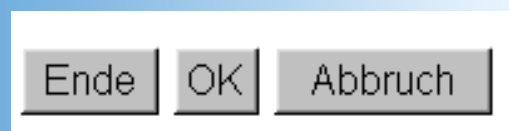


Abbildung 20.2: Verschiedene Buttons

Ein einzeliges Passwort-Eingabefeld mit gleichen Angaben für die theoretisch sichtbaren und die maximal erlaubten Eingabezeichen:

Das Passwort:

```
<INPUT name="PW" size=8 maxlength=8 type=password>
```



Abbildung 20.3: Einzeiliges Passwortfeld

Die meisten Browser ignorieren bedauerlicherweise - bis auf die Ausprägung password - Typ-Angaben für den Fall von Text-Eingabefeldern (die Buttontypen werden allgemein unterstützt). Hier bietet JavaScript die ideale Ergänzung von HTML. Das gilt auch für die so gut wie nie beachteten optionalen Attribute `min=[numerischer Wert]`

und

max=[numerischer Wert]

mit denen man eine untere und/oder obere Grenze für erlaubte Werte bei numerischen Eingaben festlegen kann.

Radiobuttons und Checkbuttons unter HTML sind wie üblich ausgeprägt. Da Optionsfelder ausschließlich in Gruppenform Sinn machen (mindesten zwei müssen zusammengehören, damit eine Option an- und ausgeschaltet werden kann), müssen sie einer Gruppe zugeordnet werden. Dabei gehen Optionsfelder (und Kontrollkästchen) einen anderen Weg als die meisten HTML-Strukturen, die einer Oberstruktur zugeordnet werden müssen. Meist werden diese dann von einem HTML-Container umschlossen (etwa Tabelleneinträge). Hier erfolgt die Zuordnung über das Attribut

value="[Bezeichner]"

mit einem internen Namen, der ein Element einer Gruppe zuordnet (das bedeutet, alle zu einer Gruppe gehörenden Radiobuttons müssen den gleichen Wert zugewiesen bekommen). Es gibt explizit keinen umschließenden Container. Wenn abwechselnd Optionsfelder angeklickt werden, werden immer alle anderen zu der Gruppe gehörigen Optionsfelder deselektiert.

Abschließend muss ein Optionsfeld noch beschriftet werden (hier sind auch deutsche Umlaute erlaubt), damit der Anwender sieht, was er auswählt. Die Beschriftung eines Optionsfeldes ist in keiner Weise an die internen Namen oder die Feldnamen gekoppelt und besteht aus gewöhnlichem HTML-Text. Damit sieht die Syntax so aus (wenn die Beschriftung nachgestellt wird):

```
<INPUT
  type=radio
  name="[Bezeichner]"
  value="[Bezeichner]"> [Beschriftung]
```



Beachten Sie, dass die grundsätzlich einzelnen Einträge nicht automatisch durch Zeilenumbrüche getrennt werden. Diese müssen bei Bedarf extra notiert werden. Dies gilt auch für Auswahllisten.

Eine Gruppe von vier zusammengehörigen Optionsfeldern:

Weihnachtswunsch:

```
<INPUT type=radio name="eins"
value="Geschenk">Teddy
<INPUT type=radio name="zwei"
value="Geschenk">Lutscher
<INPUT type=radio name="drei"
value="Geschenk">Malstifte
<INPUT type=radio name="vier"
value="Geschenk">PC
```

Kontrollkästchen bestehen aus einem einzelnen Feld oder einer Gruppe von beschrifteten Schaltflächen, von denen ein Anwender unabhängig von anderen Schaltflächen (auch innerhalb einer Gruppe) eine oder mehrere Schaltflächen selektieren kann. Der einzige Syntaxunterschied zu Optionsfeldern ist, dass beim type-Attribut der Wert checkbox angegeben wird. Damit sieht die Syntax so aus:

```
<INPUT
  type=checkbox
  name="[Bezeichner]"
```

```
value="[Bezeichner]"> [Beschriftung]
```

Eine Gruppe von Kontrollfeldern:

Weihnachtswunsch:

```
<INPUT type=checkbox name="eins"
value="Geschenk">Teddy
<INPUT type=checkbox name="zwei"
value="Geschenk">Lutscher
<INPUT type=checkbox name="drei"
value="Geschenk">Malstifte
<INPUT type=checkbox name="vier"
value="Geschenk">PC
```

Weihnachtswunsch:

- ☐ Teddy
- ☐ Lutscher
- ☐ Malstifte
- ☐ PC

Abbildung 20.4: Kontrollkästchen

Sowohl Kontrollkästchen als auch Optionsfelder können über das optionale Attribut checked (ohne Wertzuweisung) für den `<INPUT>`-Tag vorselektiert werden. Es ist in der Regel sogar so, dass beim Laden einer Webseite mit einem Formular keines der Options- oder Kontrollfelder selektiert sein wird. Es kann die Situation auftreten, dass ohne den Einsatz dieses Attributs sogar in einer Gruppe von Optionsfeldern anfänglich keines davon selektiert ist (ein normalerweise nicht erlaubter Zustand).

Eine Gruppe von drei zusammengehörigen Optionsfeldern, wobei das zweite Optionsfeld vorselektiert ist:

Bevorzugte Eissorte:

```
<INPUT type=radio name="a"
value="Eis">Erdbeere
<INPUT type=radio name="b"
value="Eis" checked>Zitrone
<INPUT type=radio name="c"
value="Eis">Kirsch
```

Bevorzugte Eissorte:

- ☐ Erdbeere
- ☒ Zitrone
- ☐ Kirsch

Abbildung 20.5: Radiobutton mit Zeilenumbrüchen zwischen den Einträgen



Bei allgemeinen Befehlsschaltflächen bedeutet das Attribut value die Beschriftung des Buttons (in Anführungszeichen zu setzen). Dies unterscheidet Button in der Syntax von Optionsfeldern und Kontrollkästchen und ist dementsprechend nicht ganz ungefährlich.

Mehrzeilige Eingabefelder

Mehrzeilige Eingabefelder werden mit dem <TEXTAREA>-Container realisiert. Dieser hat folgende Attribute:

- Einen formularweit eindeutigen Namen über name="[Bezeichner]". Der Bezeichner für den Feldnamen darf wie üblich keine Leerzeichen und keine deutschen Umlaute enthalten und muss in Anführungszeichen gesetzt werden.
- Eine (optionale, aber in der Regel äußerst sinnvolle) Größenangabe in Form von Zeilen (rows=[numerischer Wert]) und Spalten (cols=[numerischer Wert]) - zu verstehen als erlaubte Anzahl der eingebbaren Zeichen.

Ein Beispiel für ein Eingabefeld mit 80 Zeichen Breite und sechs Zeilen:

Kommentar:

```
<TEXTAREA name="Kommentar" cols=80 rows=6></TEXTAREA>
```

Ein Beispiel für ein Eingabefeld mit Defaultgröße:

Kommentar: <TEXTAREA></TEXTAREA>

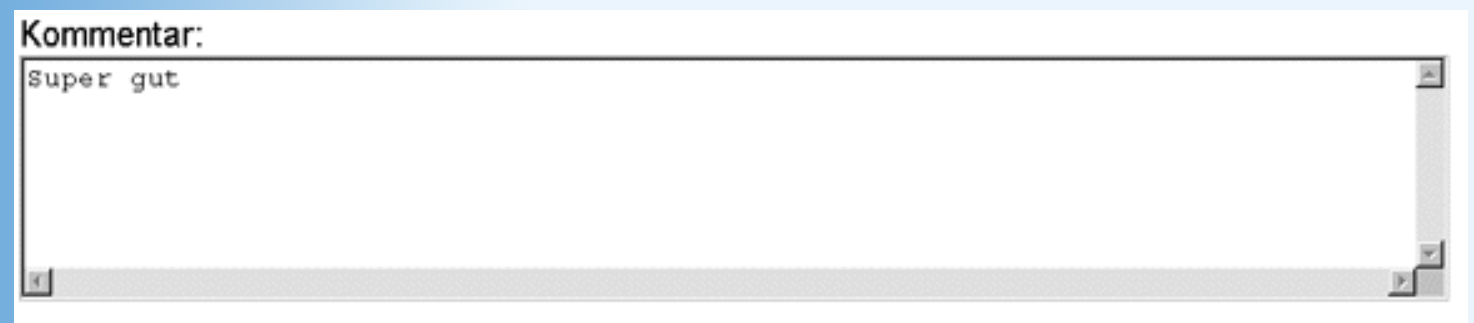


Abbildung 20.6: Ein mehrzeiliges Textfeld

Die Größenangaben bei einem mehrzeiligen Listenfild beschränken nur die sichtbaren Zeichen, nicht die Anzahl der Zeichen, welche eingegeben werden dürfen. Sowohl in Hinsicht auf Zeilen als auch in einer Zeile eingebbaren Zeichen beschränkt nur die Plattform (und natürlich der sinnvolle Umgang mit dem Eingabefeld) die Anzahl. Die Angabe der Zeilen und Spalten ist optional. Wenn diese Angaben nicht gemacht werden, bekommt das Eingabefeld eine voreingestellte Größe zugewiesen. Wenn die Anzahl der eingegebenen Zeichen die maximal anzeigbaren Zeichen überschreitet, wird entweder automatisch seitwärts oder horizontal gescrollt. Mehrzeilige Listenfelder werden von den meisten Browsern mit Bildlaufleisten angezeigt, die bei Überschreitung des Anzeigebereichs durch die Menge des Textes automatisch aktiviert, andernfalls deaktiviert werden. Einen automatischen Zeilenumbruch am Ende des sichtbaren Bereichs einer Zeile vollziehen nur wenige Browser. Dies muss in der Regel per Zeilenschaltung manuell durch den Anwender erfolgen.

Innerhalb des <TEXTAREA>-Containers kann ein Text als Vorbelegung definiert werden.

Ein Beispiel für ein vorbelegtes Eingabefeld mit 80 Zeichen Breite und zwei Zeilen:

Kommentar:

```
<TEXTAREA name="Kommentar" cols=80 rows=2>
```

Das ist vorbelegt!

```
</TEXTAREA>
```

Auswahllisten

Auswahllisten in Formularen stellen die bekannten Selektionsmöglichkeiten für Anwender zur Verfügung. Eine Auswahlliste besteht aus einem äußeren <SELECT>-Container für die Listenstruktur und jeweils einem inneren Container für jeden Listeneintrag.

Der <SELECT>-Tag besitzt das optionale Attribut name="[Bezeichner]", über das die Liste wie üblich einen formularweit eindeutigen Namen nach den üblichen Regeln bekommt, und size=[numerischer Wert] (damit wird die Anzahl der anzuzeigenden Einträge der Liste festgelegt). Falls die Liste mehr Einträge enthält, als angezeigt werden sollen/können, stellen die meisten Browser die Liste scrollbar dar. Wenn die Angabe fehlt, wird ein einzeliges Listenfeld angezeigt, das wie gewohnt beim Anklicken die vollständige Liste darstellt. Die Breite des Listenfelds ergibt sich automatisch aus der Länge der jeweiligen Listeneinträge. Die Syntax des <SELECT>-Containers sieht also so aus:

```
<SELECT
  name="[Listenname]"
  size=[numerischer Wert]>
  ...[Listeneinträge]
</SELECT>
```

Ein Listeneintrag wird über einen <OPTION>-Container definiert. Syntax:

```
<OPTION>
...[Listeneintrag]
</OPTION>
```

Eine beispielhafte Auswahlliste ohne Angabe der Größe der Liste sieht so aus (ein einzeliges Listenfeld mit mehreren Einträgen):

Bitte geben Sie das Geschlecht an:

```
<SELECT name="Geschl">
<OPTION>
Mann
</OPTION>
<OPTION>
Frau
</OPTION>
</SELECT>
```

Die folgende Auswahlliste gibt die Größe der Liste an (die Anzahl der Einträge ist kleiner als die maximal anzeigbaren Einträge):

Bitte wählen Sie das Land:

```
<SELECT name="Land" size=5>
<OPTION>
Deutschland
</OPTION>
<OPTION>
Schweiz
</OPTION>
<OPTION>
Österreich
</OPTION>
</SELECT>
```

Die folgende Auswahlliste gibt die Größe der Liste an, wobei die Anzahl der Einträge größer als die maximal anzeigbaren Einträge ist:

Bitte wählen Sie das Land:

```
<SELECT name="Land" size=2>
<OPTION>
Deutschland
</OPTION>
<OPTION>
Schweiz
</OPTION>
<OPTION>
Österreich
</OPTION>
</SELECT>
```



The image shows three multi-row list boxes. The first is labeled 'Familienstand:' and contains 'Verheiratet' and 'Glücklich'. The second is labeled 'Wohnsitz:' and contains 'Deutschland', 'Schweiz', and 'Österreich'. The third is labeled 'Urlaubsziel:' and also contains 'Deutschland', 'Schweiz', and 'Österreich'. Each list box has a small arrow icon on the right side of the selected item.

Abbildung 20.7: Verschiedene mehrzeilige Listenfelder



The image shows a single-row list box labeled 'Geschlecht:' with the value 'Mann' selected. It has a small arrow icon on the right side.

Abbildung 20.8: Ein einzeliges Listenfeld

Mehrfachauswahl in einer Liste ist möglich, aber nicht selbstverständlich. Dies hängt an den Einstellungen des Browsers, ob es standardmässig erlaubt oder verboten ist. Eine explizite Erlaubnis von Mehrfachauswahl erfolgt mit dem Attribut `multiple` im `<SELECT>`-Tag (ohne Wertzuweisung).

Eine Auswahlliste mit explizit erlaubter Mehrfachauswahl:

Haustiere:

```
<SELECT name="Tiere" size=3 multiple>
<OPTION>
Hund
</OPTION>
<OPTION>
Katze
</OPTION>
<OPTION>
Maus
</OPTION>
</SELECT>
```

Eine Vorselektion eines Listeneintrags ist mit dem Attribut selected (ohne Wertzuweisung) möglich.

Eine Auswahlliste mit expliziter Vorselektion eines Listeneintrags:

Familienstand:

```
<SELECT NAME="Famstand" SIZE=2>
```

```
<OPTION>
```

Verheiratet

```
</OPTION>
```

```
<OPTION selected>
```

Glücklich

```
</OPTION>
```

```
</SELECT>
```



Ein Listeneintrag kann im Prinzip über einen <OPTION>-Container als auch über einen einzelnen <OPTION>-Tag formuliert werden. Für eine strenge SGML-Konformität und um eventuell dennoch mögliche Fehler bei Browsern auszuschließen, ist am Ende des Listeneintrags ein abschließendes Tag </OPTION> trotz Mehraufwand sinnvoll.

Erstellen wir nun ein vollständiges Formular, das die möglichen Elemente enthält. Wir verzichten nur auf die Attribute im <FORM>-Tag, aber die sind für die optische Gestaltung hier nicht von Belang.

Beispiel 1:

Erstellen Sie die Datei Form1.htm:

```
<HTML>
```

```
<BODY>
```

```
<FORM>
```

Vorname:

```
<INPUT name="VName" size=10 maxlength=15>
```

Nachname:

```
<INPUT name="Nachname" size=30 maxlength=10>
```

```
<BR>
```

Namenzusatz: <INPUT>

```
<BR>
```

Passwort:

```
<INPUT name="PW" size=8 maxlength=8 type=password>
```

```
<BR>
```

Weihnachtswunsch:

```
<INPUT type=checkbox name="eins"
```

```
value="Geschenk">Teddy
```

```
<INPUT type=checkbox name="zwei"
```

```
value="Geschenk">Lutscher
```

```
<INPUT type=checkbox name="drei"
```

```
value="Geschenk">Malstifte
```

```
<INPUT type=checkbox name="vier"
```

```
value="Geschenk">PC
```

```
<BR>
```

Bevorzugte Eissorte:

```
<INPUT type=radio name="a"
```

```
value="Eis">Erdbeere
<INPUT type=radio name="b"
value="Eis" checked>Zitrone
<INPUT type=radio name="c"
value="Eis">Kirsch
<BR>
Kommentar:
<TEXTAREA name="Kommentar" cols=80 rows=6>
Super gut
</TEXTAREA>
<BR>
Geschlecht:
<SELECT name="Geschl">
<OPTION>Mann
</OPTION>
<OPTION>Frau
</OPTION>
</SELECT>
Familienstand:
<SELECT NAME="Famstand" SIZE=2>
<OPTION>Verheiratet
</OPTION>
<OPTION selected>Glücklich
</OPTION>
</SELECT>
<P>
Wohnsitz:
<SELECT name="Land" size=3>
<OPTION>Deutschland
</OPTION>
<OPTION>Schweiz
</OPTION>
<OPTION>Österreich
</OPTION>
<OPTION>Lichtenstein
</OPTION>
</SELECT>
Urlaubsziel:
<SELECT name="Land" size=3>
<OPTION>Deutschland
</OPTION>
<OPTION>Schweiz
</OPTION>
<OPTION>Österreich
</OPTION>
<OPTION>Schweden
</OPTION>
<OPTION>Norwegen
</OPTION>
</SELECT>
<P>
<INPUT type=file size=60 name=uploadDatei>
```

```
<BR>
<INPUT type=button value ="Ende"
onClick='JavaScript:alert("Nö - keine Chance")'>
<INPUT type="submit" name="Ok" value="OK">
<INPUT type="reset" name="leer" value="Abbruch">
</FORM>
</BODY>
</HTML>
```

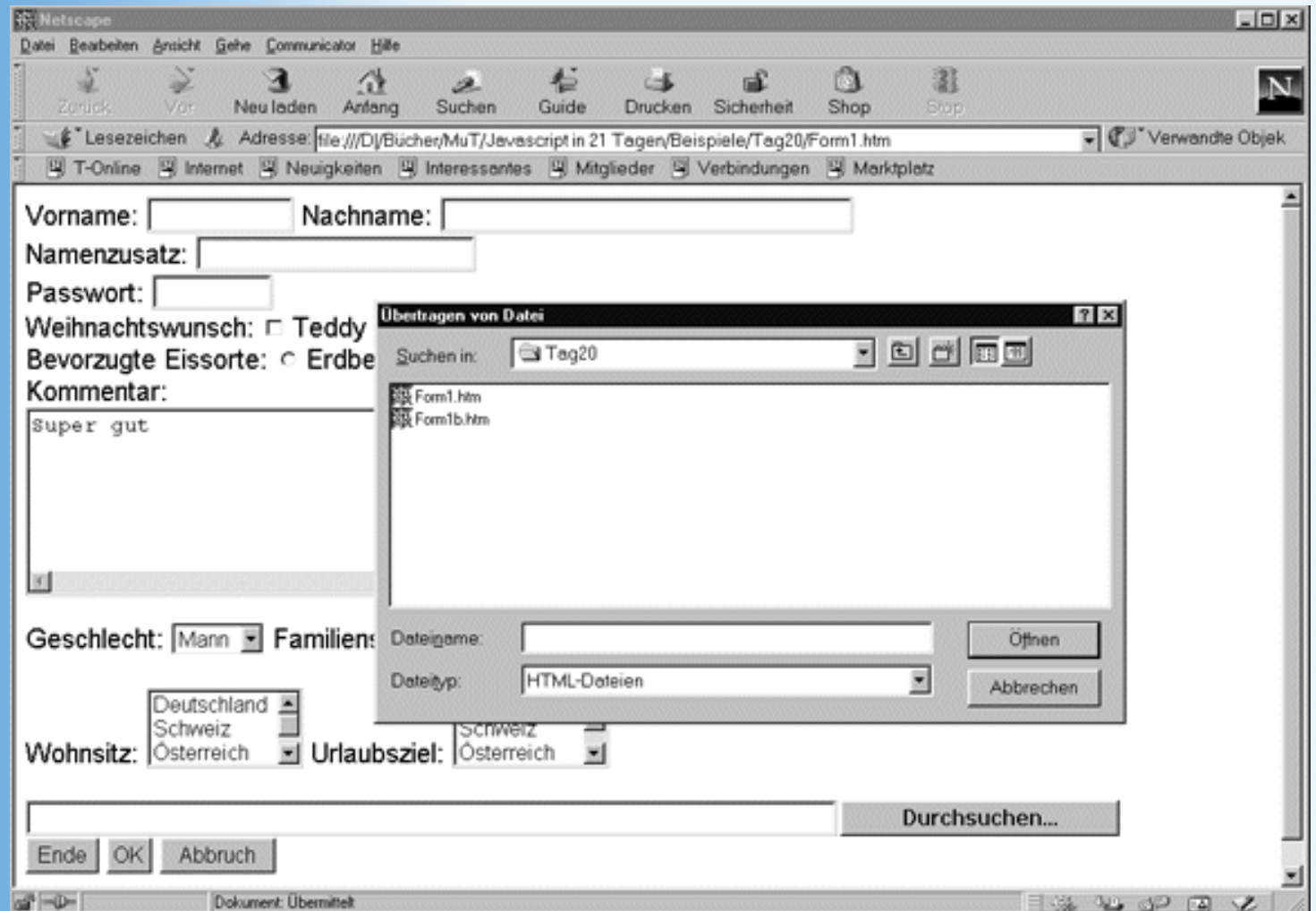


Abbildung 20.9: Das Formular mit gerade aufgerufener Upload-Funktion

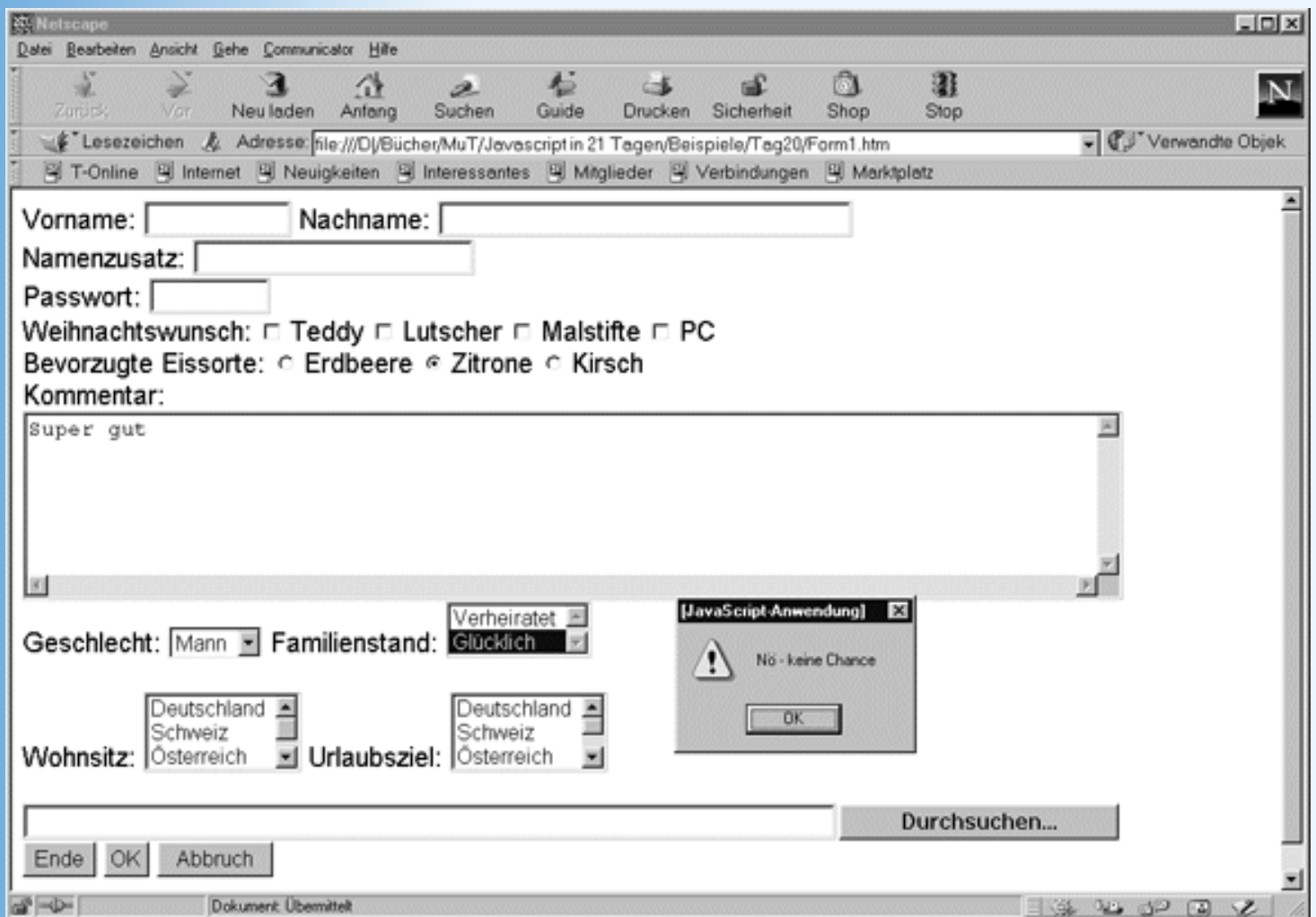


Abbildung 20.10: Das vollständige Formular mit gerade aufgerufener JavaScript-Funktion

Der JavaScript-Zugriff auf Formulare

In den vergangenen Kapiteln des Buches haben wir bereits mehrfach auf Formulare in Verbindung mit JavaScript zurückgegriffen. Fassen wir hier nur noch mal zusammen, was wir dazu schon parat haben, und ergänzen es mit einigen wichtigen Erweiterungen.

Grundsätzlich werden Formulare in Webseiten durch das Objekt `form` repräsentiert, das in der Objekthierarchie unter `document` angeordnet ist. Wenn eine Webseite mit Formularen geladen wird, wird für jedes Formular so ein `form`-Objekt erzeugt und in einem Objektfeld mit Namen `forms[]` gespeichert. Die Indexnummern entstehen automatisch, wenn der Browser das Objekt bei Abarbeitung der HTML-Seite erzeugt und in ein Element des Arrays einordnet. Das erste im Dokument auftretende Formular erhält den Index 0, das zweite den Index 1 und so fort.

Das `form`-Objekt hat weder viele Eigenschaften noch Methoden. Die Bedeutung erhält es darüber, dass man darüber jedes Formularelement in der Webseite auswerten und in der Regel auch setzen kann. Das wird hauptsächlich über eines seiner Unterobjekte erfolgen.

Eigenschaften

Eigenschaft	Beschreibung
<code>action</code>	Das <code>action</code> -Attribut im <code><FORM></code> -Tag.

elements	Ein Array, das alle Elemente als vollständige Repräsentation der HTML-Implementierung in dem Formular enthält.
encoding	Das enctype-Attribut. Das ist der Kodierungstyp des Formulars. Mehr Informationen dazu finden Sie im WWW in der Spezifikation RFC 1867.
length	Die Anzahl der Elemente im Formular.
method	Das method-Attribut im <FORM>-Tag.
name	Das name-Attribut im <FORM>-Tag.
target	Das target-Attribut im <FORM>-Tag. Das ist die Angabe des Wertes, der bei der Definition des Formulars in der Zusatzangabe target steht (bei CGI-Formularen).

Tabelle 20.2: Eigenschaften von form

Methoden

Methode	Beschreibung
reset()	Simuliert einen Mausklick auf einen Resetbutton.
submit()	Schickt das Formular ab.

Tabelle 20.3: Methoden von form

Daneben steht noch die übliche `handleEvent()`-Methode bereit.

Konkrete Anwendung von JavaScript auf Formulare

Die Anwendung von JavaScript auf Formulare erfolgt im Wesentlichen über die wohl wichtigste Eigenschaft von forms - das Objektfeld `elements`. Darüber hat man Zugriff auf alle einzelnen Elemente innerhalb von Formularen, welche in der HTML-Datei definiert sind. Das Objektfeld wird automatisch erzeugt, wenn der Browser mindestens ein Formular in der HTML-Datei mit Elementen vorfindet. Dabei wird für jedes Element in jedem Formular automatisch eine Instanz eines dem Element entsprechenden Objekts erzeugt.

Da das Objektfeld `elements` eine Eigenschaft von dem Objektfeld `forms` ist, gibt es verschiedene Möglichkeiten, um einzelne Formularelemente anzusprechen. Entweder über die Form-Instanz selbst und einen Index oder den Formularnamen und ein `elements`-Eintrag mit Index oder den Elementnamen (beliebig kombinierbar). Schematisch sehen die Zugriffe so aus:

```
document.forms[index1].elements[index2].[Eigenschaft]
document.forms[index1].[Elementname].[Eigenschaft]
document.[Formularnamen].elements[index].[Eigenschaft]
document.[Formularnamen].[Elementname].[Eigenschaft]
document.forms[index1].elements[index2].[Methode()]
document.forms[index1].[Elementname].[Methode()]
document.[Formularnamen].elements[index].[Methode()]
document.[Formularnamen].[Elementname].[Methode()]
```

Einige Eigenschaften des Objektsfeldes sind bereits verfügbar, wenn der Browser wenigstens ein Formular mit wenigstens einem Element in der HTML-Datei vorfindet. Andere Eigenschaften sind jedoch nur für bestimmte Formularelemente verfügbar (d.h. wenn ein Element des zugeordneten Typs überhaupt vorhanden ist und wenn diese Eigenschaft überhaupt Sinn macht²). Nachfolgend sind die Eigenschaften von `elements` angegeben:

Eigenschaft	Beschreibung
<code>checked</code>	Diese Eigenschaft steht zur Verfügung, wenn das Element ein Radio- oder Checkbutton ist. Sie fragt ab oder bestimmt, ob ein Radio- oder Checkbutton aktiviert ist oder nicht. Mögliche Werte sind <code>true</code> bzw. 1 oder <code>false</code> bzw. 0.
<code>defaultChecked</code>	Diese Eigenschaft steht zur Verfügung, wenn das Element ein Radio- oder Checkbutton ist. Sie fragt ab oder bestimmt, ob ein Radio- oder Checkbutton per Voreinstellung aktiviert ist oder nicht. Mögliche Werte sind <code>true</code> bzw. 1 oder <code>false</code> bzw. 0.
<code>defaultValue</code>	Diese Eigenschaft steht zur Verfügung, wenn das Element ein Eingabefeld ist. Sie beinhaltet den voreingestellten Text eines Eingabefeldes.
<code>form</code>	In der Eigenschaft <code>form</code> wird das zugehörige Formular als Objekt referenziert. Darüber besteht Zugriff auf alle Eigenschaften und Methoden des Form-Objekts aus einem Element eines Formulars heraus.
<code>name</code>	Der Name des Formularelements, wie er in HTML bei der Definition des Elements mit der Zusatzangabe <code>name</code> zugewiesen wurde.
<code>type</code>	Die Variable hat die Information über den Elementtyp eines Formularelements gespeichert. Sie kann in Zusammenhang mit allen Formularfeldern angewendet werden
<code>value</code>	Die wahrscheinlich wichtigste Eigenschaft. Dies ist der Wert, den der Anwender in einem Formularelement eingegeben bzw. diesem zugewiesen hat.

Tabelle 20.4: Eigenschaften von `elements`

Über das Objektfeld `options`, das in der Objekthierarchie unter `elements` angeordnet und damit eine weitere Eigenschaft davon ist, hat man vollständigen Zugriff auf Auswahllisten innerhalb eines Formulars. Auswahllisten können wie jedes andere Formularelement mit einer Indexnummer oder dem Namen angesprochen werden. Da es sich selbst um ein Objekt handelt, stellt es auch eigene Eigenschaften bereit, die nur zur Verfügung stehen, wenn das Element eine Auswahlliste ist:

Eigenschaft	Beschreibung
<code>length</code>	Ermittelt wird die Anzahl der Auswahlmöglichkeiten.
<code>options[n].defaultSelected</code>	Wird zum Abfragen oder Bestimmen einer Default-Einstellung in einer Auswahlliste genutzt, d.h. ob sie per Voreinstellung aktiviert ist oder nicht. Mögliche Werte sind <code>true</code> bzw. 1 oder <code>false</code> bzw. 0.
<code>selectedIndex</code>	Ermittelt oder bestimmt, welche Auswahlmöglichkeit in einer Auswahlliste aktiviert ist. Mögliche Werte sind <code>true</code> bzw. 1 oder <code>false</code> bzw. 0.
<code>options[n].selected</code>	Die Eigenschaft kann dazu verwendet werden, zu ermitteln oder zu bestimmen, welche Auswahlmöglichkeit in einer Auswahlliste aktiviert ist. Mögliche Werte sind <code>true</code> bzw. 1 oder <code>false</code> bzw. 0.
<code>options[n].value</code>	Beinhaltet den internen Wert des über den Index angegebenen Eintrags in einer Auswahlliste.
<code>options[n].text</code>	Der Text einer Auswahlmöglichkeit in einer Auswahlliste wird abgefragt oder bestimmt.

Tabelle 20.5: Eigenschaften von `options`

Das Objektfeld `elements` stellt neben seinen Eigenschaften samt Unterobjekten die folgenden Methoden

bereit:

Methoden	Beschreibung
<code>blur()</code>	Der Focus verlässt ein Element und es wird damit deselektiert.
<code>click()</code>	Die Methode bewirkt das Gleiche, als wenn der Anwender auf ein Element klickt.
<code>focus()</code>	Der Focus wird auf ein Element positioniert.
<code>handleEvent()</code>	Die Methode übergibt ein Ereignis an ein Element, das auf das Ereignis reagieren kann (nur Netscape).
<code>select()</code>	Die Methode selektiert den gesamten in einem Feld enthaltenen Text.

Tabelle 20.6: Methoden von elements

Reaktionen auf das Abschicken und Zurücksetzen eines Formulars

Bei der Behandlung der beiden Event-Handler `onReset` und `onSubmit` haben wir bereits gesehen, dass diese für die Versendung eines Formulars und das Leeren genutzt werden können, um JavaScripte bei der Betätigung aufzurufen.

Die Reaktion auf das Zurücksetzen eines Formulars über einen Event-Handler (`onReset`) erlaubt die Erweiterung der Funktionalität, welche über die durch HTML generierte **Abbrechen**-Schaltfläche standardmäßig erfolgt. Statt direkt beim Anklicken dieses Buttons alle Eingaben im Formular sofort zu löschen, kann man statt dessen, wenn im einleitenden Formular-Tag (!) der Event-Handler `onReset` notiert ist, die dort stehende JavaScript-Anweisung ausführen.

Beispiel:

```
<FORM Name="meinFormular" onReset="return wirklich()">
```

Von entscheidendem Anteil an der Unterdrückung des Leerungsvorgangs ist der Rückgabewert, welchen die aufgerufene Methode zurückgibt. Wenn beim Funktionsaufruf über `onSubmit` und `onReset` das Schlüsselwort `return` vorangestellt ist und man im Rahmen der aufgerufenen Funktion die Werte `true` oder `false` zurückgibt, kann man die Abarbeitung der ausgelösten Aktion noch einmal abbrechen (Rückgabe von `false` durch die aufgerufene Funktion). Wenn darauf verzichtet wird, wird zwar die aufgerufene Funktion ausgeführt, die eigentliche Aktion (also das Leeren oder Abschicken des Formulars) aber anschließend durchgeführt.

Vollkommen analog ist der Event-Handler `onSubmit` zu sehen. Er erweitert nur statt der **Abbrechen**-Schaltfläche den Submit-Button, mit dem ein Formular weggeschickt wird. Statt es einfach abzusenden, kann eine JavaScript-Funktion aufgerufen werden, die beispielsweise überprüft, ob alle Felder eines Formulars ausgefüllt sind.

Es wird Zeit für noch ein Beispiel. Testen wir die Kontrolle, ob ein Formular wirklich abgeschickt oder geleert werden soll. Dabei verändern wir das Beispiel von neulich ein wenig. Insbesondere soll per JavaScript geprüft werden, ob alle Felder im Formular gefüllt sind. Beachten Sie, dass wir über den Namen des Formulars und die beiden Eingabefelder arbeiten.

Beispiel 2:

Geben Sie folgenden Quelltext ein:

```
<HTML>
<SCRIPT language="JavaScript">
function kontrolle()
```

```
{
if((document.meinForm.eins.value=="")
||(document.meinForm.zwei.value==""))
{
alert("Das Formular muss vollständig gefüllt werden");
return false;
}
else
{
return true;
}
}
function sicher()
{
return window.confirm(
"Wollen Sie dieses Formular wirklich leeren?");
}
</SCRIPT>
<BODY>
<FORM
name="meinForm"
action="mailto:hotdampf@hotmail.com"
method=POST
onSubmit="return kontrolle()"
onReset="return sicher()">
<INPUT name="eins">
<INPUT name="zwei">
<BR>
<INPUT type="submit" value="Ok">
<INPUT type="reset" value="Abbrechen">
</FORM>
</BODY>
</HTML>
```

Speichern Sie das Script unter Form2.htm und laden Sie es in einen Browser. Lassen Sie die beiden Formularfelder oder eines davon leer und betätigen Sie die **Ok**- und die **Abbrechen**-Schaltfläche. Testen Sie dann in dem Dialogfenster jeweils beide Möglichkeiten.

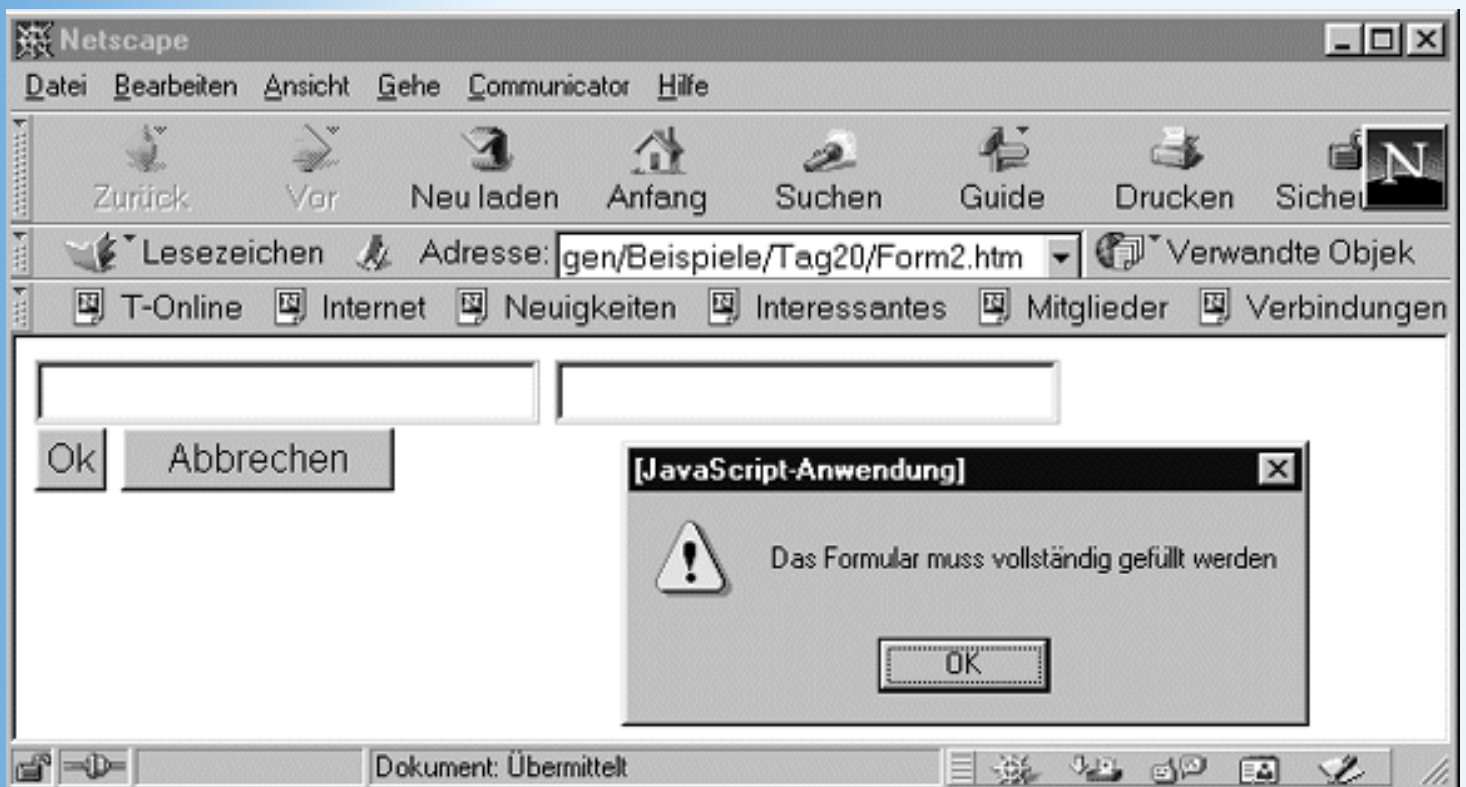


Abbildung 20.11: Rückfrage, ob das Formular vollständig ist

Zusammenfassung

Formulare sind eine der wichtigsten Möglichkeiten der Interaktion zwischen dem Betrachter einer Webseite und dem Ersteller der Seite. Mit einem Formular können komfortabel gleichartige, datensatzorientierte Informationen erfasst werden, die sich über Scripte und Datenbankprogramme gut auswerten und speichern lassen. Ein Formular wird erst mal in purem HTML erstellt. Dazu stellt HTML Anweisungen zur Generierung von Eingabefeldern, mehrzeiligen Textfeldern, Listen und Aufzählungen, Schaltflächen und beschreibendem Text bereit. Bei Formularen gibt es immer einen äußeren `<FORM>`-Container und darin eingeschlossene Container und einzelne Steueranweisungen, welche die einzelnen Formularelemente definieren. Den äußeren Container ergänzen in der Regel zwei bis drei weitere, durch ein Leerzeichen getrennte Angaben. Die eine in der Regel notwendige Angabe ist gewöhnlich eine Adresse, wo das Formular ausgewertet werden soll, wenn es fertig ausgefüllt und abgeschickt wurde. Die zweite Standardangabe ist eine Methode, wie das Formular zur Auswertungsstelle gelangt und dort zu behandeln ist. Beide Angaben machen nur in Kombination Sinn.

Um über JavaScript auf Formulare zugreifen zu können, nutzt man das Objekt `form`, das in der Objekthierarchie unter `document` angeordnet ist. Wenn eine Webseite mit Formularen geladen wird, wird für jedes Formular solch ein `form`-Objekt erzeugt und in einem Objektfeld mit Namen `forms[]` gespeichert. Die Indexnummern entstehen automatisch, wenn der Browser das Objekt bei Abarbeitung der HTML-Seite erzeugt und in einem Element des Arrays einordnet.

Das `form`-Objekt hat weder viele Eigenschaften noch Methoden. Die Bedeutung erhält es dadurch, dass man darüber jedes Formularelement in der Webseite auswerten und in der Regel auch setzen kann. Das wird hauptsächlich über eines seiner Unterobjekte erfolgen - `elements`.

Fragen und Antworten

Frage:

Ist der Zugriff über Formularnamen und Elementnamen sinnvoller oder der Zugriff über die

Objektfelder und Indizes?

Antwort:

Beides hat Vor- und Nachteile. Wenn Sie Namen verwenden, können Sie leichter neue Elemente in ein Formular einfügen, entfernen usw. Das Gleiche gilt für ganze Formulare in Webseiten. Außerdem sind Namen immer sprechender. Arrays sind immer dann besser, wenn man mit Schleifen oder sonstigen automatisierten Abläufen arbeiten möchte.

Frage:

Was ist mit dem <BUTTON>-Tag? Gehört der zur Formulartechnik?

Antwort:

Nein. Das ist ein Tag, der nicht dazu gehört. Er sieht zwar genauso aus wie eine Formular-Schaltfläche, wird aber ohne umgebenden Formular-Container notiert. Er wird auch nicht in jedem Browser unterstützt.

Workshop

Verbinden Sie die Formulare in dem Kapitel mit der Tabellentechnik, um das Layout sinnvoller zu gestalten.

Suchen Sie in den bisherigen Kapiteln die Beispiele, wo wir mit Formularen gearbeitet haben. Analysieren Sie diese aus dem Blickwinkel der hier neu hinzugekommenen Erkenntnisse.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Was ist <FORM>?
2. Was ist <OPTION>?
3. Was ist multiple?
4. Ist das richtig?

```
<SELECT name="Tiere" size=1>
<OPTION>
Hund
</OPTION>
<OPTION>
Katze
</OPTION>
<OPTION>
Maus
</OPTION>
</SELECT>
```

5. Mit welcher HTML-Anweisung wird ein mehrzeiliges Eingabefeld realisiert?
6. Wie unterscheiden sich Radiobuttons von Checkbuttons ?
7. Was ist options?

1 Unter HTML gibt es da eine Besonderheit zu beachten, die gleich noch geklärt wird.

2 Ein Textfeld kann nicht die Eigenschaft checked haben.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 3

Tag 21

Style Sheets und das Style-Objekt

Wir haben das Ende unseres JavaScript-Lehrgangs fast erreicht. Für den letzten Tag haben wir uns aber noch ein sehr spannendes und interessantes Thema aufgehoben: die Verbindung von Style Sheets und JavaScript bzw. Aufgabenstellungen, die sich darum ranken.

Wir werden also heute Animationen mit JavaScript und das, was man als Dynamic HTML bezeichnet, in den Vordergrund stellen. Neben der Verbindung von Style Sheets und JavaScript wird dies auch die Behandlung eines Objekts beinhalten, das von Aktionen erzeugt wird, die über document auftreten - das Style-Objekt.

Das Kapitel können Sie sich als Endspurt vorstellen. Dementsprechend werden wir heute nochmals eine längere Strecke zurücklegen, die aber sicher sehr spannend und unterhaltsam wird.

Das Style-Objekt

Das Style-Objekt spezifiziert seit JavaScript 1.2 die stilistischen Charakteristika von einem HTML-Element. Es wird von den folgenden Eigenschaften oder Methoden des document-Objekts erzeugt (siehe dazu Tag 19):

```
document.classes  
document.contextual  
document.ids  
document.tags
```

Das Style-Objekt eröffnet die Möglichkeit, in Webseiten Dynamic HTML zu realisieren und Style Sheets mit JavaScript zu verbinden. Die Methoden (nicht allzu viele) und umfangreichen Eigenschaften des Objekts implementieren die CSS-Stileigenschaften von HTML in JavaScript. Das wird vor allem deutlich, wenn man dann die nachfolgend beschriebenen CSS-Eigenschaften mit denen des Objekts vergleicht.

Eigenschaften

Eigenschaft	Beschreibung
align	Ausrichtung eines HTML-Elements in seinem Elternelement.
backgroundColor	Spezifiziert eine Hintergrundfarbe für ein Element.
backgroundImage	Spezifiziert ein Hintergrundimage für ein HTML-Element.
borderBottomWidth	Spezifiziert die Breite von dem unteren Rahmen des HTML-Elements.
borderColor	Spezifiziert die Farbe von dem Rahmen des HTML-Elements.
borderLeftWidth	Spezifiziert die Breite von dem linken Rahmen des HTML-Elements.
borderRightWidth	Spezifiziert die Breite von dem rechten Rahmen des HTML-Elements.
borderStyle	Spezifiziert den Stil vom Rahmen, wie einfach oder doppelt, um ein Blocksatz-HTML-Element.
borderTopWidth	Spezifiziert die Breite von dem oberen Rahmen des HTML-Elements.

clear	Spezifiziert die Seite des HTML-Elements, die umfließende Elemente erlaubt.
color	Spezifiziert die Farbe von dem Text eines HTML-Elements.
display	Überschreibt die normale Anzeige eines Elements und gibt an, ob ein Element in eine Zeile, als Blocksatzelement oder als Blocksatzlisteneintrag erscheinen soll.
fontFamily	Spezifiziert die Font-Familie, etwa Helvetica oder Arial, für ein HTML-Textelement.
fontSize	Spezifiziert die Fontgröße für ein HTML-Textelement.
fontStyle	Spezifiziert den Stil von der Schrift des HTML-Elements.
fontWeight	Spezifiziert die Dicke von der Schrift des HTML-Elements.
lineHeight	Spezifiziert den Abstand zwischen der Basislinie von zwei benachbarten Linien in einem Block.
listStyleType	Spezifiziert den Stil von dem Bullet in Listeneinträgen.
marginBottom	Spezifiziert die minimale Distanz zwischen dem Boden des HTML-Elements und der Spitze des darunter befindlichen Elements.
marginLeft	Spezifiziert die minimale Distanz zwischen der linken Seite des HTML-Elements und der rechten Seite von einem Nachbarelement.
marginRight	Spezifiziert die minimale Distanz zwischen der rechten Seite des HTML-Elements und der linken Seite von einem Nachbarelement.
marginTop	Spezifiziert die minimale Distanz zwischen der Spitze des HTML-Elements und dem Boden von einem Nachbarelement.
paddingBottom	Spezifiziert, wie viel Leerraum zum Einfügen zwischen dem Boden von einem Element und seinem Inhalt, etwa Text oder ein Bild, gelassen wird.
paddingLeft	Spezifiziert, wie viel Leerraum zum Einfügen zwischen der linken Seite von einem Element und seinem Inhalt, etwa Text oder ein Bild, gelassen wird.
paddingRight	Spezifiziert, wie viel Leerraum zum Einfügen zwischen der rechten Seite von einem Element und seinem Inhalt, etwa Text oder ein Bild, gelassen wird.
paddingTop	Spezifiziert, wie viel Leerraum zum Einfügen zwischen der Spitze von einem Element und seinem Inhalt, etwa Text oder ein Bild, gelassen wird.
textAlign	Spezifiziert die Fließrichtung von einem HTML-Textelement in einem Block.
textDecoration	Spezifiziert Spezialeffekte wie Blinken u.ä. für ein HTML-Textelement.
textIndent	Spezifiziert die Größe der Einrückung vor der ersten formatierten Zeile eines HTML-Textblocks.
textTransform	Spezifiziert die Art von einem HTML-Textelement.
whiteSpace	Spezifiziert, ob nicht sichtbare Zeichen (so genannter white space) innerhalb eines HTML-Elements wegoptimiert werden sollen oder nicht.
width	Spezifiziert die Breite von einem HTML-Blockelement.

Tabelle 21.1: Eigenschaften von Style

Methoden

Methode	Beschreibung
borderWidths()	Spezifiziert die Breite der Rändern von einem HTML-Element.
margins()	Spezifiziert die minimale Distanz zwischen den Seiten von einem HTML-Element und den Seiten der Nachbarelemente.
padding()	Spezifiziert, wie viel Leerraum zwischen dem Inhalt eines Elements und dem Rand gelassen wird.

Tabelle 21.2: Methoden von Style

Die praktische Anwendung des Objekts sehen wir gleich in Verbindung mit den Style Sheets bzw. haben wir bereits an Tag 19 genutzt.

Style Sheets

Style Sheets haben wir schon in früheren Passagen des Buches kennen gelernt (siehe dazu insbesondere Tag 4). Wie auch Scripte gehören sie nicht zu HTML, sondern sind eine eigenständige Technik, die seit HTML 4.0 in HTML verankert werden und - wie oben besprochen - via dem Style- Objekt mit JavaScript verbunden werden kann. Style Sheets kann man sich als eine Art Analogon zu Formatvorlagen im Rahmen von Textverarbeitungen vorstellen. Wesentliche Bedeutung erlangen sie durch den Umstand, dass sie eine Trennung von Layout und Inhalt einer Webseite ermöglichen, was mit anderen Techniken nur schwer oder gar nicht erreichbar wäre.

Es gibt nun verschiedene Versuche, Style Sheets zu standardisieren. Bedauerlicherweise muss man festhalten, dass man bis jetzt noch nicht zu einer vollständigen Einigung auf einen Standard gekommen ist oder sich eine Norm wirklich überzeugend durchsetzt. Dies gilt sogar für die Umsetzung von Cascading Style Sheets (CSS), die bereits im Januar 1997 vom W3C eingeführt wurden. Bezeichnend ist, dass, obwohl Cascading Style Sheets bereits seit 1998 in der 2. Version vorliegen und schon einige Jahre ins Land gegangen sind, der Internet Explorer 5.5 (Beta) nicht einmal den auf den Webseiten des W3C zu findenden Acid-Test zur korrekten Darstellung von CSS 1.0 (!!) besteht (<http://www.w3.org/>). Derzeit haben die neusten Browser also nicht einmal die Version CSS1 vollständig umgesetzt. Man kann sich nur darauf verlassen, dass die neuen Browser von Microsoft und Netscape die CSS1-Empfehlung zwar beide implementiert haben, aber sich nur in den Kernbereichen an die Vorschläge zur Umsetzung halten.

Die Verbindung von Style Sheets mit einer HTML-Seite kann darüber geschehen, dass man Style Sheets in eine Seite einbettet oder aus einer externen Datei importiert. Zur tatsächlichen Definition von Style Sheets stehen Ihnen mehrere Standardtechniken zur Verfügung (das haben wir an Tag 4 ausführlich behandelt):

- direkte Notation im <STYLE>-Container,
- Import externer Stilinformationen über den <STYLE>-Container,
- die <LINK>-Referenz,
- Inline-Referenz.

Die wohl meistgebrauchte Technik zur Einbindung eines Style Sheet in eine Webseite erfolgt über die Notation eines <STYLE>-Containers direkt in den HTML-Code. In dem Inneren des <STYLE>-Containers werden - meist in HTML-Kommentare eingeschlossen, damit für ältere Browser die Anweisungen verborgen werden - alle HTML-Elemente definiert, denen ein besonderes Layout zugewiesen werden soll. Das Style Sheet wird mit dem Tag <STYLE> wie folgt notiert:

```
<STYLE TYPE="text/css">
<!--... irgendwelche Anweisungen ... -->
</STYLE>
```

Grundregeln von CSS-Deklarationen.

Die Syntax einer CSS-Deklaration sieht immer gleich aus. Es gibt jedes Mal den folgenden Aufbau:

name : **wert**

Dabei gibt

name

das zu formatierende Element (den so genannten Selector) an und

wert

die konkrete Stilinformation. Bei Bedarf werden mehrere Deklarationen mit Semikolon getrennt. In vielen Situationen werden die Werte auch in geschweifte Klammern gesetzt. Der Selector ist bis auf wenige Ausnahmen einer der existierenden HTML-Tags.

Ein HTML-Tag kann, mit einem Punkt abgetrennt, um die optionale Angabe einer Klasse erweitert werden. Die Syntax sieht so aus:

[HTML - Tag] . [class]

Dies ist die Festlegung der Eigenschaften für eine Instanz eines Elements. Dabei legt man eine Klasse fest und verwendet diese bei der Stilzuweisung. Für class kann jeder beliebige Name genommen werden. Damit ist es möglich, verschiedene Stile für ein und dasselbe HTML-Element zu erstellen. Der ursprüngliche Selector, welcher vor dem Punkt steht, ist das Elternelement zu dieser Instanz.

Die Angabe eine Klasse kann auch alleine in der Stildeklaration angegeben werden. In diesem Fall muss einem Klassenbezeichner in der Deklaration ein Punkt vorangestellt werden und bei der konkreten Verwendung ein HTML-Element um den so benannten neuen Parameter erweitert werden.



Die Festlegung von Klassen bei Stilvereinbarungen ist besonders deshalb interessant, weil man mittels Scripten über diese Angabe auf diese Stilvereinbarungen zugreifen kann.

Beispiel1:

Das Beispiel zeigt die parallele Verwendung verschiedener Stilinformationen bei gleichen HTML-Elementen. Diese hier recht einfach gehaltenen Formatierungen werden von allen neueren Browsern unterstützt (CSS1.htm):

```
<HTML>
<HEAD>
<TITLE>Style Sheets</TITLE>
<STYLE TYPE="text/css">
  H1
  {
    color : white
  }
  H1.neuefarbe
  {
    color : blue
  }
  H1.nocheinefarbe
  {
    color : cyan
  }
  BODY
  {
    background : green
  }
  .meineKlasse1
  {
    background : red;
    color : yellow
  }
  .meineKlasse2
  {
    background : yellow;
    color : blue
  }
</STYLE>
</HEAD>
<BODY>
<H1>Überschrift H1 mit normalen Style-Sheets
</H1>
<H1 class=neuefarbe>
H1-Überschrift mit der Class-Information neuefarbe
</H1>
<P>Normaler Text und darin mit
```

```
<SPAN class=meineKlasse1>
meineKlasse1
</SPAN>
veränderte Farbe.
<H1 class=nocheinefarbe>
H1-Überschrift mit der Class-Information nocheinefarbe
</H1>
Nachfolgender Text <u class=meineKlasse1>unterstrichen, Style-Sheet-formatiert und
hier noch mit
<SPAN class=meineKlasse2>meinKlasse2</SPAN>
formatiert.</u>
</BODY>
</HTML>
```

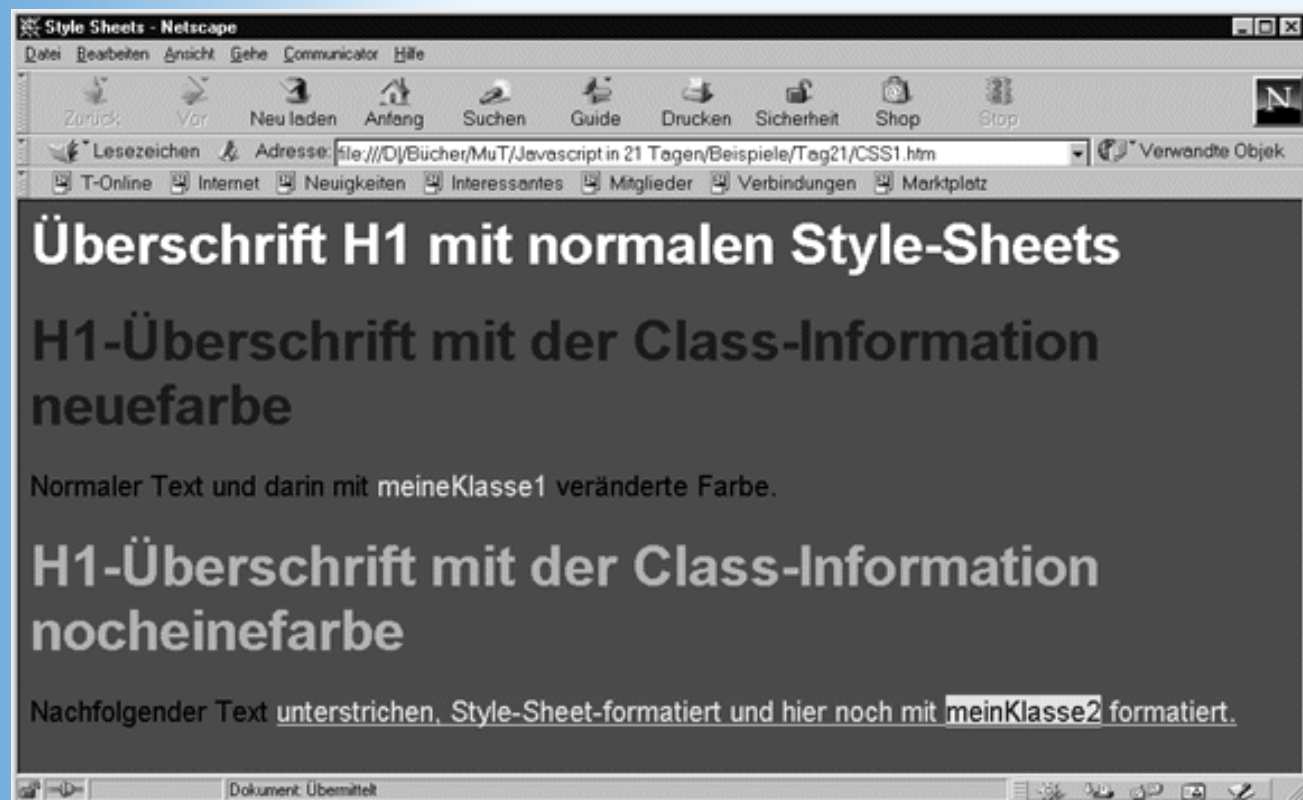


Abbildung 21.1: Verschiedene Textformatierungen über Style Sheets

Im <STYLE>-Container wird zuerst eine Stilinformation (Schriftfarbe Weiß) für alle Überschriften der Ordnung 1 vereinbart (H1 { color : white }). Danach wird über eine mit Punkt abgetrennte class-Erweiterung ein Unterstil (Schriftfarbe Blau) für die Überschriften der Ordnung 1 vereinbart (H1.neuefarbe { color : blue}). Die Angabe (H1.nocheinefarbe { color : cyan}) deklariert noch einen zweiten Unterstil (Schriftfarbe Grün) für die Überschriften der Ordnung 1. Die nächste Stildeklaration legt die Hintergrundfarbe der Webseite fest. Die beiden letzten Stildeklarationen deklarieren eine Stilinformation (rote Hintergrundfarbe und gelbe Textfarbe bzw. umgekehrt), die bei Bedarf an jedes HTML-Element als neu definierter Parameter angehängt werden kann.

Innerhalb des BODY-Bereichs werden die vereinbarten Stilinformationen angewandt. Einmal auf die Überschriften-Tags und über den - Container im normalen Textfluss. Dieser bietet sich geradezu für solche Stilvereinbarungen an, da er keinerlei eigene Wirkung hat (das ist genau genommen der Sinn, warum es den Container überhaupt gibt). Aber auch in Verbindung mit Elementen, die bereits eine Wirkung haben (wie in dem Beispiel <U>), lässt sich die Stilinformation einsetzen. Dabei gilt, dass Stilvereinbarungen über HTML so lange unverändert weiterwirken, so lange sie der Style-Sheet-Vereinbarung nicht in die Quere kommen. Ansonsten sollten Sie in dem Beispiel beachten, dass mit der Klasse meineKlasse2 eine Klasse geschaffen ist, die innerhalb eines Containers eingesetzt wird, der mit meineKlasse1 formatiert ist. Die innere Klasse überschreibt die Wirkung der äußeren Klasse in allen gleichen Eigenschaften. Hier sind das vollkommen deckungsgleiche, das kann aber auch nur eine teilweise Überdeckung der Eigenschaftenmenge sein. Dabei gilt wie bei HTML-Elementen, dass Stilvereinbarungen der äußeren Klasse so lange unverändert weiterwirken, so lange sie den Style-Sheet-Vereinbarung der inneren Klasse nicht widersprechen. Dies Verhalten ist kein Zufall - es gibt definierte Anwendungsregeln von CSS-Stil-Formatierungen. Das CSS-Formatierungsmodell teilt HTML-Elemente in

zwei Gruppen ein. Es gibt einmal die Elemente, die einen abgesetzten Textblock darstellen - dazu zählen im Wesentlichen Überschriften, Listen, Absätze usw. Der andere Typ von Elementen sind diejenigen, welche im Fliesstext stehen; dazu zählen u.a. Hervorhebungen und Links. Bei den Stilattributen gibt es feste Regeln, die angewendet werden, wenn es zu Konflikten zwischen widersprechenden Stilanweisungen auf einer Seite kommt oder Anweisungen auf Elemente angewendet werden, die diese nicht umsetzen können:

- Die vereinbarten Stildeklarationen werden angewendet, wenn der Selector vernünftige Werte hat. Kann die Deklaration nicht angewandt werden, wird die Deklaration des Elternelementes verwendet. Gibt es kein solches Elternelement, kommen die Standardeinstellungen des Browsers zum Tragen.
- Gibt es mehrere Deklarationen zu einem Element, hat der Stil Vorrang, der über einen gesonderten Modifier, `important`, deklariert worden ist.
- Stilvereinbarungen eines Autors einer Webseite haben höhere Priorität als Stilvereinbarungen des Besuchers einer Webseite. Diese wiederum überschreiben Standardeinstellungen des Browsers.
- Globale Stilinformationen werden von lokalen überschrieben.
- Gibt es mehrere Deklarationen mit gleicher Priorität zu einem Element, hat der Stil Vorrang, der als Letztes vereinbart wurde.

Style Sheets lassen sich auch über eine so genannte ID auswählen. Dies wird zwar vom W3C nicht empfohlen, aber von fast allen Browsern unterstützt. Man ordnet damit eine Formatierung einem bestimmten Element (keiner Elementklasse wie eben) zu. Das funktioniert, indem bei der Deklaration eine Raute `#` statt eines Punkts vorangestellt wird. Die Syntax sieht dann so aus:

`#[ID]`

In einem konkreten Beispiel:

```
#meineID {color : red}
```

Anwenden kann man diese ID, indem man statt `class` dem jeweiligen Element `id` als Parameter nachstellt (im Beispiel wäre das so: `<P id=meineID>`).

Beispiel2:

Das Beispiel zeigt die Verwendung von IDs (CSS2.htm):

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
#farbe1 {color:blue}
#farbe2 {color:green}
#farbe3 {color:red}
-->
</STYLE>
</HEAD>
<BODY>
<H1 id=farbe3>Willkommen
</H1>
<P id=farbe1>Es grünt so grün
</P>
<P id=farbe2>Was sind wir wieder blau
</P>
</BODY>
</HTML>
```

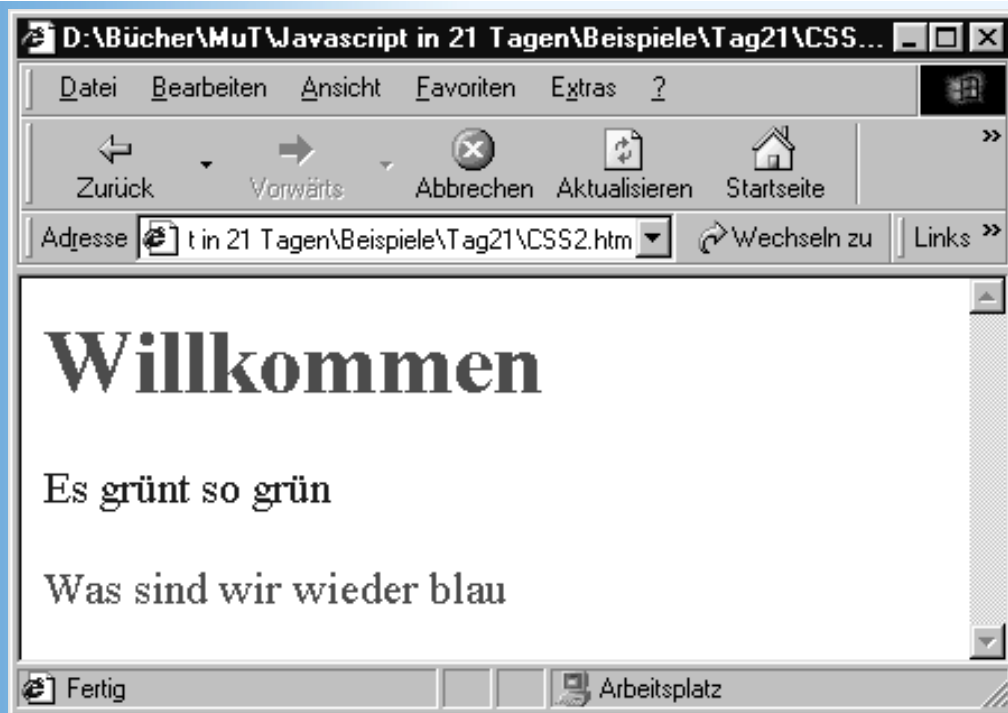


Abbildung 21.2: Verschiedene Textformatierungen über IDs

Wenn verschiedene Selectoren zusammen in einer Stilinformation deklariert werden, bilden sie eine Gruppe. Die einzelnen Selectoren werden durch Kommata voneinander getrennt und bekommen alle die jeweilig angegebenen Stile zugeordnet. Nicht explizit angegebene Attribute werden davon nicht beeinflusst.

Beispiel 3:

Das Beispiel zeigt die Verwendung einer Gruppe von Selektoren (CSS3.htm):

```
<HTML>
<HEAD>
<TITLE>Style Sheets</TITLE>
<STYLE TYPE="text/css">
  H1, H3, H4, H6
  {
    color : yellow
  }
</STYLE>
</HEAD>
<BODY>
<H1>Überschrift H1
</H1>
<H2>Überschrift H2
</H2>
<H3>Überschrift H3
</H3>
<H4>Überschrift H4
</H4>
<H5>Überschrift H5
</H5>
<H6>Überschrift H6
</H6>
</BODY>
</HTML>
```

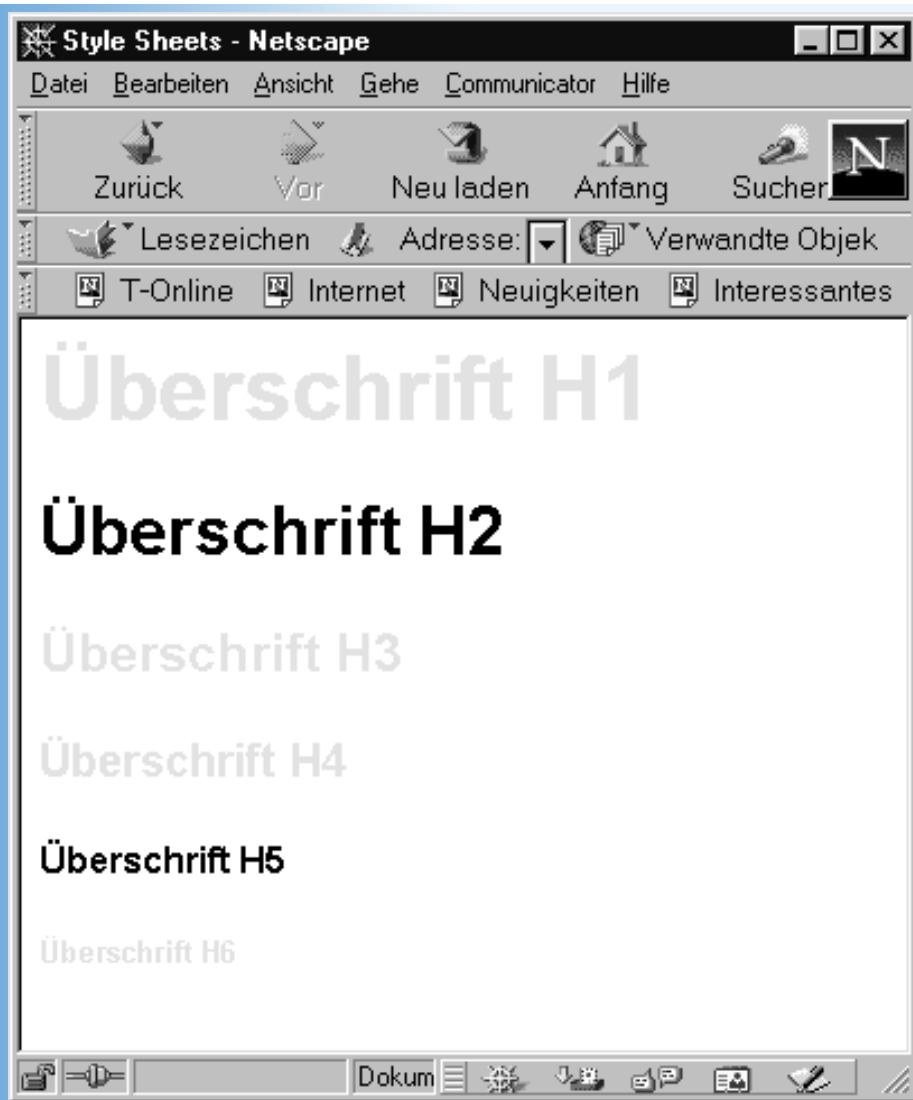



Abbildung 21.3: Eine Gruppe von Selektoren

Grundsätzlich werden CSS-Stilvereinbarungen in verschiedene Kategorien unterteilt, die wir uns nun ansehen wollen.

Font-Stilvereinbarungen

Für die Verwendung von Stilvereinbarungen zur Beeinflussung von Schriften gibt es die in der folgenden Tabelle aufgelisteten CSS- Schriftformatierungseigenschaften (mit Angabe der erlaubten Werte). Der Wertebereich umfasst sowohl Konstanten wie large, medium oder small samt ihrer Abarten für Schriftgrößen als auch Längenangaben als absolute Werte (Pixel, Zentimeter, Millimeter, Zoll, Punkt, Pica) oder relativ (prozentuale Angaben) zu einer Referenzgröße (meist die Größe des Standardzeichensatzes). Des Weiteren können Farben in der RGB-Notation oder durch Angabe eines Farbnamens festgelegt werden und URLs über grafische Angaben für Hintergrundgrafiken).

Eigenschaft	Beschreibung	Wertebereich	Beschreibung
font-size	Schriftgrößen	xx-small	relativ zu den anderen Größen der Familie kleinste Schriftgröße
		x-small	relativ zu den anderen Größen der Familie sehr kleine Schriftgröße
		small	relativ zu den anderen Größen der Familie kleine Schriftgröße
		medium	relativ zu den anderen Größen der Familie mittlere Schriftgröße
		large	relativ zu den anderen Größen der Familie große Schriftgröße
		x-large	relativ zu den anderen Größen der Familie sehr große Schriftgröße

		xx - large	relativ zu den anderen Größen der Familie größte Schriftgröße
		larger	relativ zum Elternstil größer
		smaller	relativ zum Elternstil kleiner
		Länge	Faktor, der mit der ursprünglichen Schriftgröße multipliziert wird. Einheit ist ein Em, was der Platz ist, der von dem Buchstaben M eingenommen wird.
		Prozentwert	Multiplikationsfaktor mit der Größe der Elternschrift (in Prozent)
		Absolutangabe	absoluter Zahlenwert in einer Einheit für die Textgröße (Pixel, Zentimeter, Millimeter, Zoll, Punkt, Pica)
font-family	Angabe einer ganzen Gruppe von Schrifttypen (Schriftfamilie). Mehrere Schriftfamilien werden durch Komma getrennt	serif	Schriften mit Häkchen an den Linien (z.B. Times)
		sans-serif	Schriften ohne Häkchen an den Linien (z.B. Helvetica)
		cursive	geneigte Schriften
		fantasy	dekorative Schriften
		monospace	nichtproportionale Schriften
		Angabe einer konkreten Schrift	(Times, Helvetica,...)
font-weight	Festlegung, wie fett eine Schrift dargestellt wird	normal	mittlere Gewichtung
		bold	fette Darstellung
		bolder	Verstärkung um eine Ebene
		lighter	Verringerung um eine Ebene
		Zahlenwert mit Abstufungen von 100 bis 900	Fett entspricht dem Wert 700.
font-style	Schriftstile	normal	Normaler Schriftstil. Entspricht gelegentlich Roman.
		italic	kursiver Schriftstil
		oblique	Kursiver Schriftstil. Die Neigung wird aber durch das System aus der normalen Schrift errechnet.
font-variant	Kapitälchen	normal	normal
		small-caps	klein
font	Abkürzung für alle oben genannten Eigenschaften	font-style, font-variant, font-weight, font-size, line-height, font-family	Die Angabe sämtlicher Attribute erfolgt in der angegebenen Reihenfolge ohne Kommata, es sei denn, es werden Schriftfamilien angegeben.

Tabelle 21.3: Font-Stilvereinbarungen



Die genannten Attribute mit ihren erlaubten Werten werden nicht alle von jedem Browser verstanden und auch die Darstellung kann teilweise erhebliche Unterschiede in verschiedenen Browsern aufweisen. Beachten Sie dazu auch das nachfolgende Beispiel.

Beispiel 4:

Das nachfolgende Beispiel zeigt die Verwendung von verschiedenen Stilinformationen in Zusammenhang mit Fonts (CSS4.htm):

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
.a
{
font-size: large;
font-family : sans-serif;
font-weight: bolder
}
.b
{
font-size: xx-large;
font-family : fantasy;
font-variant : small-caps ;
font-weight: bold
}
.c
{
font-size: xx-small;
font-family : serif;
font-weight: normal ;
line-height : 180%
}
.d
{
font-size: medium;
font-family : cursive;
font-variant : small-caps ;
font-weight: lighter;
line-height : 150%
}
.e
{
font-size: 42pt;
font-family : monospace;
font-variant : normal ;
font-weight: 888 ;
line-height : 50%
}
.f
{
font-size: small;
font-family : serif;
font-variant : small-caps ;
font-weight: 111 ;
line-height : 140%
```

```
}  
</STYLE>  
</HEAD>  
<body>  
<DIV class=a>Hund  
</DIV>  
<DIV class=b>Katze  
</DIV>  
<DIV class=c>Maus  
</DIV>  
<DIV class=d>Vogel  
</DIV>  
<DIV class=e>Pferd  
</DIV>  
<DIV class=f>Fisch  
</DIV>  
<DIV class=b>Schwein  
</DIV>  
<DIV class=d>Rind  
</DIV>  
</BODY>  
</HTML>
```



Abbildung 21.4: Die Darstellung im Internet Explorer



Abbildung 21.5: Die Darstellung im Navigator

Beachten Sie die Abbildungen, welche die unterschiedlichen Interpretationen zwischen dem Navigator und dem Explorer zeigen. Der Internet Explorer stuft den Fettwert besser ab und kann die Kapitälchen darstellen sowie verschiedene Schrifttypen (cursive und fantasy) richtig interpretieren (siehe Abbildung 21.4 und 21.5).

Stilvereinbarungen für Farbe und Hintergrund

Zur Beeinflussung von Farbe und Hintergrund in Webseite mittels Style Sheets steht eine Vielzahl von Eigenschaften bereit. Die nachfolgende Tabelle bietet dazu eine Übersicht. Die Farb- und Hintergrundangaben über CSS sind recht einfach anzuwenden, denn es gibt viele Übereinstimmungen zu gewöhnlichem HTML. Die Möglichkeiten sind jedoch erheblich erweitert und nicht vollständig identisch, weshalb es leicht Flüchtigkeitsfehler geben kann. Da dies keine Fehlermeldung im Browser auslöst, erkennt man nur, dass der Browser eine Stilinformation nicht wie gewünscht umsetzt. Dabei kann man aber nie ganz sicher sein, dass es nicht ein »normales« Problem eines Browsers ist, der mit den Angaben nicht vollständig zurechtkommt (siehe Tabelle 21.4).

Eigenschaft	Beschreibung	Wertebereich	Beschreibung
color	Textfarbe	Angabe der Farbe im RGB-Modell oder über Farbnamen	
background-color	Hintergrundfarbe	Angabe der Farbe im RGB-Modell oder über Farbnamen	

background-image	Hintergrundbild	url([URL des Bildes])	Die Funktion url() wird direkt hinter die Eigenschaft notiert und beinhaltet die Pfadangabe eines Bildes.
background-repeat	in Verbindung mit einem Hintergrundbild Art der Wiederholung	repeat	gekachelte Wiederholung über den gesamten Bereich
		repeat-x	Wiederholung über die horizontale Zeile
		repeat-y	Wiederholung über die vertikale Spalte
background-position	Startposition eines Hintergrundbildes. In Kombination von einer horizontalen und einer vertikalen Angabe kann diese festgelegt werden.	left	Horizontale Positionierung links
		center	Zentrierung, sowohl für die horizontale als auch die vertikale Positionierung
		right	horizontale Positionierung rechts
		top	vertikale Positionierung oben
		bottom	vertikale Positionierung unten
		Koordinaten-Tupel in %. 0% 0% steht für oben links, 100% 100% für unten rechts.	Positionierung über Koordinatenangaben
background-attachment	Verankerung eines Hintergrundbildes	scroll	Hintergrundbild mit Text verbunden
		fixed	Hintergrundbild mit dem Hintergrund verbunden
background	Abkürzung für alle oben genannten Eigenschaften zu Hintergrundfarben und Hintergrundgrafik	background-color, background-image, background-repeat, background-attachment, background-position,	Die Angabe sämtliche Attribute erfolgt in der angegebenen Reihenfolge ohne Kommata.

Tabelle 21.4: Stilvereinbarungen für Farbe und Hintergrund

Beispiel 5:

Es folgt nun eine Demonstration der verschiedenen Möglichkeiten zur Hintergrundgestaltung. Einige Elemente sind mit definierter Hintergrundfarbe und Textfarbe sowie individuellen Hintergrundbildern versehen. Was die Ausdehnung der Stilinformationen angeht, haben Navigator und Explorer verschiedene Ansichten¹, wie die dem Quelltext folgenden Screenshots zeigen. Vergleichen Sie die Unterschiede in den Details (CSS5.htm):

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
.a
{
  color: red;
  background-color : yellow
```

```
}
.b
{
color: blue;
background-color : cyan;
background-image:url(1.gif)
}
.c
{
color: green;
background-color : blue;
background-image:url(2.gif)
}
.d
{
background-image:url(1.jpg);
color = White
}
.bild
{
background-image:url(3.gif);
color: Blue
}
H1.bild2
{
background-image:url(2.gif);
background-repeat:repeat-x;
color: Blue
}
.bild3
{
background-image:url(3.gif);
background-repeat:repeat-y;
color: Red
}
</STYLE>
</HEAD>
<body class=bild3>
<DIV class=bild3>Die Würfel haben sich verformt
</DIV>
<SPAN class=bild>
<DIV class=a>Hund
</DIV>
<DIV class=b>Katze
</DIV>
<DIV class=c>Maus
</DIV>
<DIV>Vogel
</DIV>
<DIV class=a>Pferd
</DIV>
<DIV class=c>Fisch
</DIV>
<DIV>Schwein
</DIV>
<DIV class=a>Rind
</DIV>
</SPAN>
```

```
<p CLASS=d>
Das ist ein Text, der nur lang genug sein muss.
<BR>
Ob er irgendeinen Sinn hat oder nicht,
<BR>
das ist vollkommen egal
</p>
Jetzt ist der Absatz zu Ende.
<H1 class=bild2>
Big Brüter is cooking you
</H1>
</BODY>
</HTML>
```

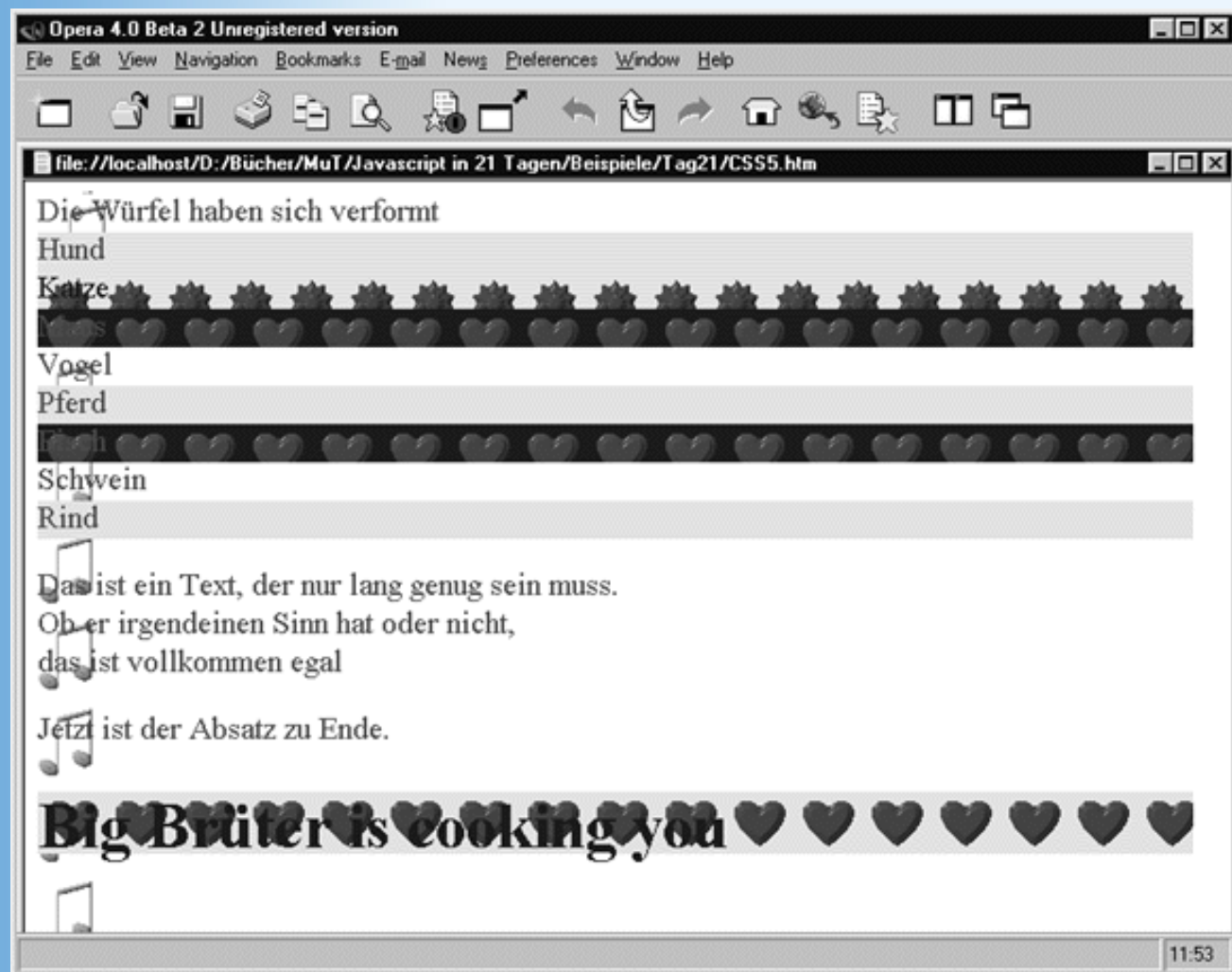


Abbildung 21.6: Das macht der Opera daraus



Abbildung 21.7: Darstellung im Internet Explorer



Abbildung 21.8: Die Darstellung im Navigator

Stilvereinbarungen zur Gestaltung von Textformatierungen

Das CSS-Konzept beinhaltet neben den allgemeinen Font- Beeinflussungsmöglichkeiten zahlreiche Eigenschaften, mit denen dann der daraus aufgebaute Text individuell gestaltet werden kann. Die nachfolgende Tabelle bietet einen Überblick.

Eigenschaft	Beschreibung	Wertebereich	Beschreibung
vertical-align	Vertikale Ausrichtung von Text, z.B. für tief- oder hochgestellten Text. Die als Basis dienende Grundlinie ist die Grundlinie des Elternelements, welches neu formatiert wird.	baseline	Ausrichtung an der Grundlinie
		sub	Tiefstellung
		super	Hochstellung

		top	Ausrichtung des oberen Randes eines Elementes an dem größten Zeichen der aktuellen Zeile
		text-top	Ausrichtung am oberen Rand
		middle	zentrierte Ausrichtung in der Mitte einer Zeile
		bottom	Ausrichtung des unteren Randes eines Elementes an dem am tiefsten stehenden Zeichen der aktuellen Zeile
		text-bottom	Ausrichtung am unteren Rand
		Angabe von Prozentwerten	Verschieben eines Elementes um einen positiven oder negativen Wert über oder unter die Grundlinie
text-transform	transformiert Zeichen, etwa in Klein- oder Großbuchstaben	capitalize	erster Buchstabe eines Wortes groß
		uppercase	alle Buchstaben groß
		lowercase	alle Buchstaben klein
		none	keine Transformation
text-align	die üblichen Angaben der horizontalen Ausrichtung	left	linksbündig
		right	rechtsbündig
		center	zentriert
		justify	Blocksatz
text-indent	Einrückung der ersten Zeile eines Textblocks	Angabe der direkten Länge oder einer Prozentangabe	Einheit in Ems oder Zoll
line-height	Abstand zwischen zwei benachbarten Grundlinien	Angabe als Faktor	ohne Einheit
word-spacing	Abstand zwischen einzelnen Worten	Einheit	Angabe in Ems
letter-spacing	Abstand zwischen einzelnen Buchstaben	Einheit	Angabe in Ems
text-decoration	weitergehende Textdekoration	none	keine
		underline	unterstrichen
		overline	Strich oben
		line-through	durchgestrichen
		blink	Blinken
line-height	Höhe einer Zeile	direkte Angabe der Länge oder in Prozent	

Tabelle 21.5: Stilvereinbarungen zur Gestaltung von Textformatierungen

Diese Style-Sheet-Formatierungen zur Textgestaltung werden nur sehr mangelhaft in vielen Browsern

unterstützt. Deshalb sollten Sie sich darauf nie verlassen. Beachten Sie das nachfolgende Beispiel (CSS6.htm).

Beispiel 6:

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
.hoch
{
  vertical-align:super
}
.tief
{
  vertical-align:sub
}
.oben
{
  vertical-align:text-top
}
-->
</STYLE>
</HEAD>
<body>
<DIV type="text/css"
style="text-transform:capitalize;
text-align:center">
Macht <SPAN class=hoch>hoch </SPAN> die
<SPAN class=tief>Tür</SPAN>.
</DIV>
<p type="text/css"
style="text-transform:lowercase;
word-spacing:5Em;
text-align:right">
Auf die Plätzchen <SPAN class=oben>fertig </SPAN>
<SPAN class=tief>raus</SPAN>.
</p>
<p type="text/css"
style="text-decoration:line-through;
text-transform:uppercase">
Hin und weg.
</p>
</BODY>
</HTML>
```

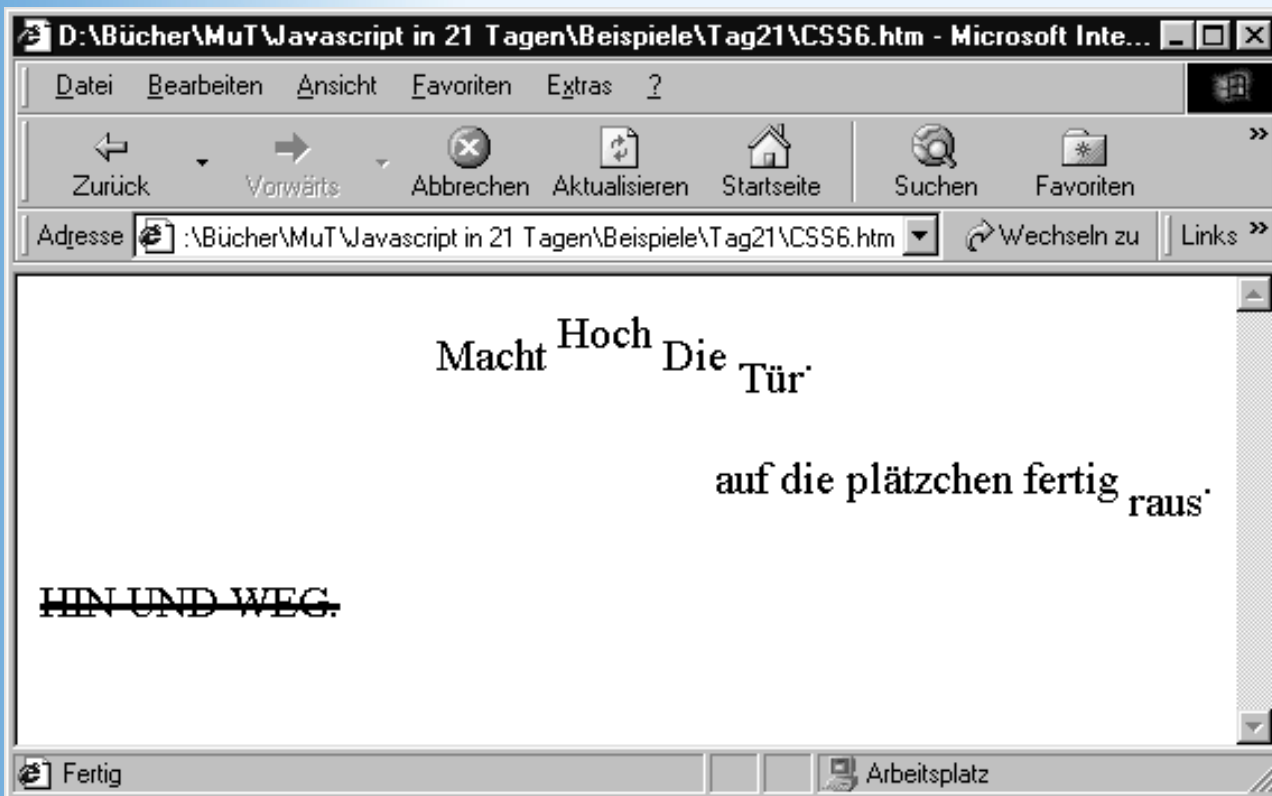


Abbildung 21.9: Der Internet Explorer gibt sich mehr Mühe

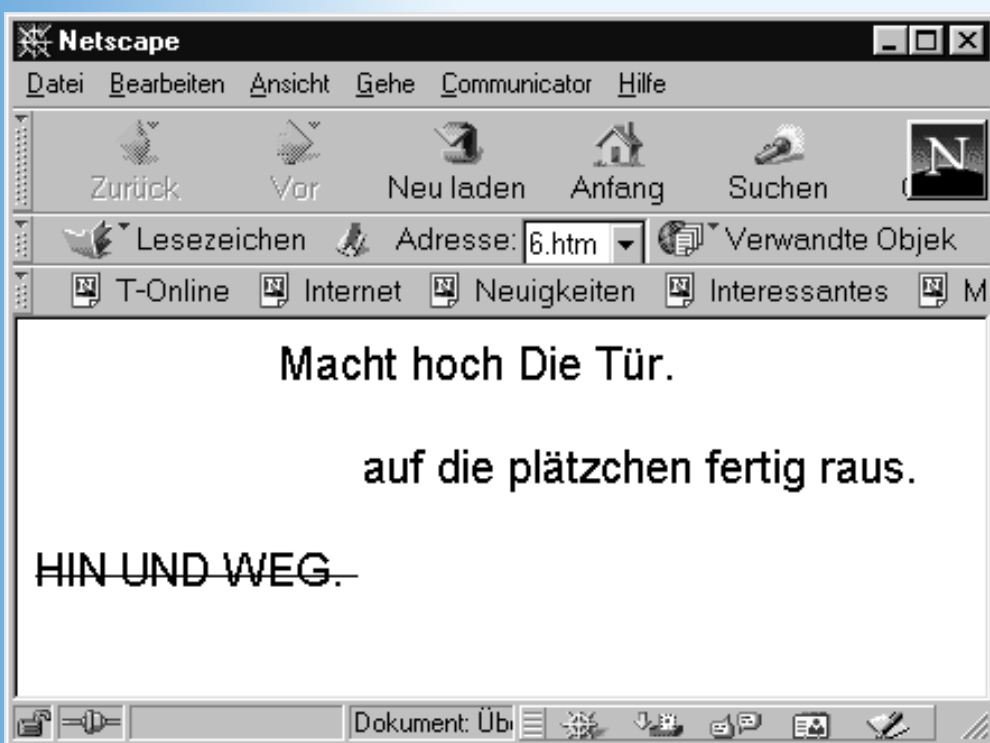


Abbildung 21.10: Der Navigator will nicht richtig ran

Stilvereinbarungen zur Gestaltung von Textblock- Eigenschaften

Stilinformationen können auf ganze Textblöcke angewandt werden. Etwa zum Definieren von Abständen zu Rändern oder anderen Elementen, zum Festlegen von Rahmen usw. Die nachfolgend in der Tabelle angegebenen Eigenschaften behandeln immer einen vollständigen Textblock:

Eigenschaft	Wertebereich	Beschreibung

padding-left padding-right padding-top padding-bottom padding	Angabe in Pixeln oder Prozent	Padding (Polsterung) definiert den Abstand, den der Inhalt eines Elements (etwa Text oder Grafik) vom Rand des Elements hält. Der Hintergrund scheint durch. Wenn die Eigenschaft ohne nachgestellte Richtung angegeben wird, werden alle Seiten gepolstert. Sie können die Eigenschaft ohne nachgestellte Richtung auch mit Werten - ohne Komma dazwischen - angeben. Dann werden die Werte in der Richtung oben, unten, rechts, links interpretiert. Fehlende Werte werden von gegenüberliegenden Seiten ergänzt.
margin-left margin-right margin-top margin-bottom margin	Angabe in Pixeln oder Prozent	Setzen von Rändern. Die Zusatzangaben -left, -right, -top, -bottom legen, wenn gebraucht, Platz zum benachbarten Element in der jeweiligen Richtung fest. Dabei können Sie auch negative Werte nehmen, um Abstände zu verkürzen.
border-color	Farbangabe	Rahmenfarbe
border-style	none, dotted, dashed, solid, double, groove, ridge, inset, outset	Erscheinungsbild eines Rahmens
border	Abkürzung für einen Rahmen	Ein Rahmen mit den gewünschten Ausprägungen. Die Breite, der Stil und die Farbe können nacheinander - ohne Komma - angegeben werden.
width	auto, Länge, Prozent	Breitenangabe. Es können Elemente mit Bindestrich angehängt werden (z.B. border-width= 5% oder border-top-width=10%)
height	auto, Länge	Höhenangabe (wie Breite)
float	left, right, none	Ausrichtung eines Blockelementes in Bezug auf umgebenden Text. Beispiel: float:left platziert ein Element, Text fließt rechts um das Element herum.
clear	left, right, both, none	Legt fest, ob floating Elemente auf der linken, rechten oder auf beiden Seiten erlaubt sind.

Tabelle 21.6: Stilvereinbarungen zur Gestaltung von Textblock-Eigenschaften**Beispiel 7:**

Das Beispiel CSS7.htm zeigt die Anwendung von Padding und Rahmen. Auch hier werden wieder gewaltige Divergenzen zwischen verschiedenen Browsern zu sehen sein (siehe Abb. auf Seite 681).

```

<HTML>
<HEAD>
<BODY>
<SPAN type="text/css" style="padding: 5% 25%">
Puffer hier, Puffer da.
</SPAN>
<SPAN type="text/css"
style="border : 10pt groove red; padding : 10%">
Rahme mich
</SPAN>
<p type="text/css"
style="border : 25pt ridge white; padding-left: 5%">
www
</p>
</BODY>
</HTML>

```

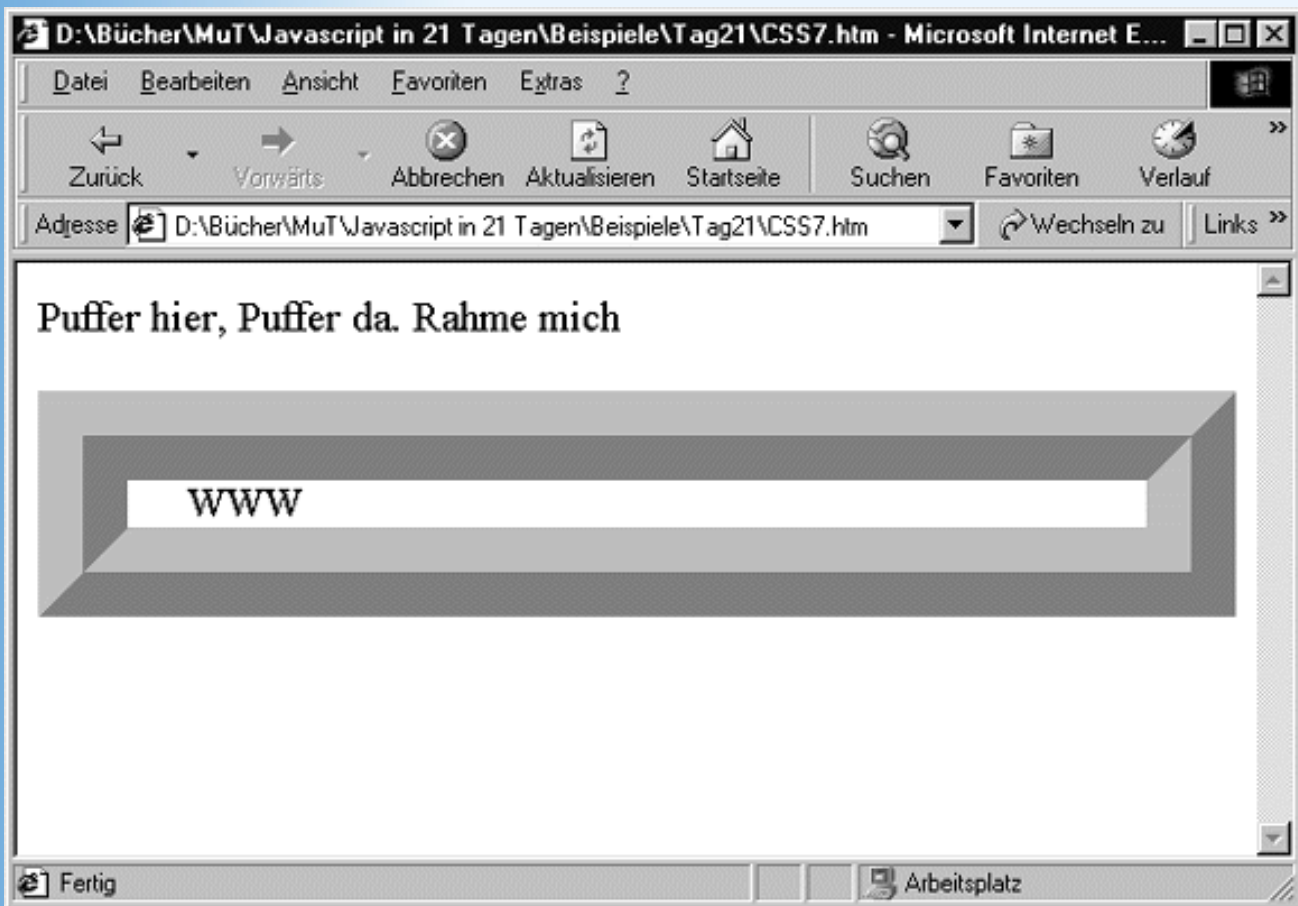


Abbildung 21.11: Der Internet Explorer



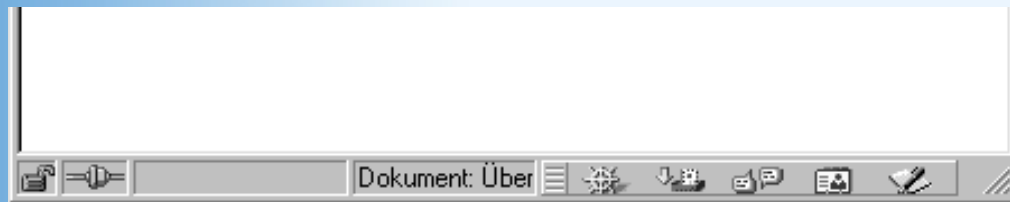


Abbildung 21.12: Darstellung im Navigator

Positionierung mit CSS

Eine der interessantesten Anwendungen von Style Sheets sind Positionierungsangaben von beliebigen Elementen. Dies kann entweder mit relativen Positionsangaben oder über absolute Positionsangaben erfolgen. Für Letzteres gibt es die gemeinsam zu verwendenden Angaben:

```
position:absolute  
left=[Position]  
top=[Position]
```

Damit können beliebige Elemente pixelgenau positioniert werden und sowohl der Navigator als auch der Explorer verstehen es.



Die beiden konkurrierenden Browser Navigator und Explorer kennen aber jeweils zwei zusätzliche Techniken zur Positionierung von Elementen, die der jeweilige Konkurrent nicht darstellen kann. Netscape stellt die Layer-Technik zur Verfügung. Diese versteht der Explorer nicht. Der Explorer hingegen kennt eine Abart der allgemeinen Positionierungstechnik, bei der der Positionierungsparameter mit STYLE als Attribut direkt in einen Tag geschrieben wird. Damit kann der Navigator nicht umgehen.

Beispiel 8

Das Beispiel CSS8.htm demonstriert mehrere Grafiken, die so positioniert werden, dass sie sich überlagern:

```
<HTML>  
<HEAD>  
<STYLE TYPE="text/css">  
<!--  
.pos1 {position:absolute; left:60px; top:40px}  
.pos2 {position:absolute; left:100px; top:60px}  
.pos3 {position:absolute; left:140px; top:80px}  
.pos4 {position:absolute; left:180px; top:70px}  
-->  
</STYLE>  
<TITLE>Absolute Positionierung</TITLE>  
</HEAD>  
<BODY>  
<DIV class=pos1><IMG src="2.jpg">  
</DIV>  
<DIV class=pos2><IMG src="1.gif">  
</DIV>  
<DIV class=pos3><IMG src="2.gif">  
</DIV>  
<DIV class=pos4><IMG src="3.gif">  
</DIV>  
</BODY>  
</HTML>
```

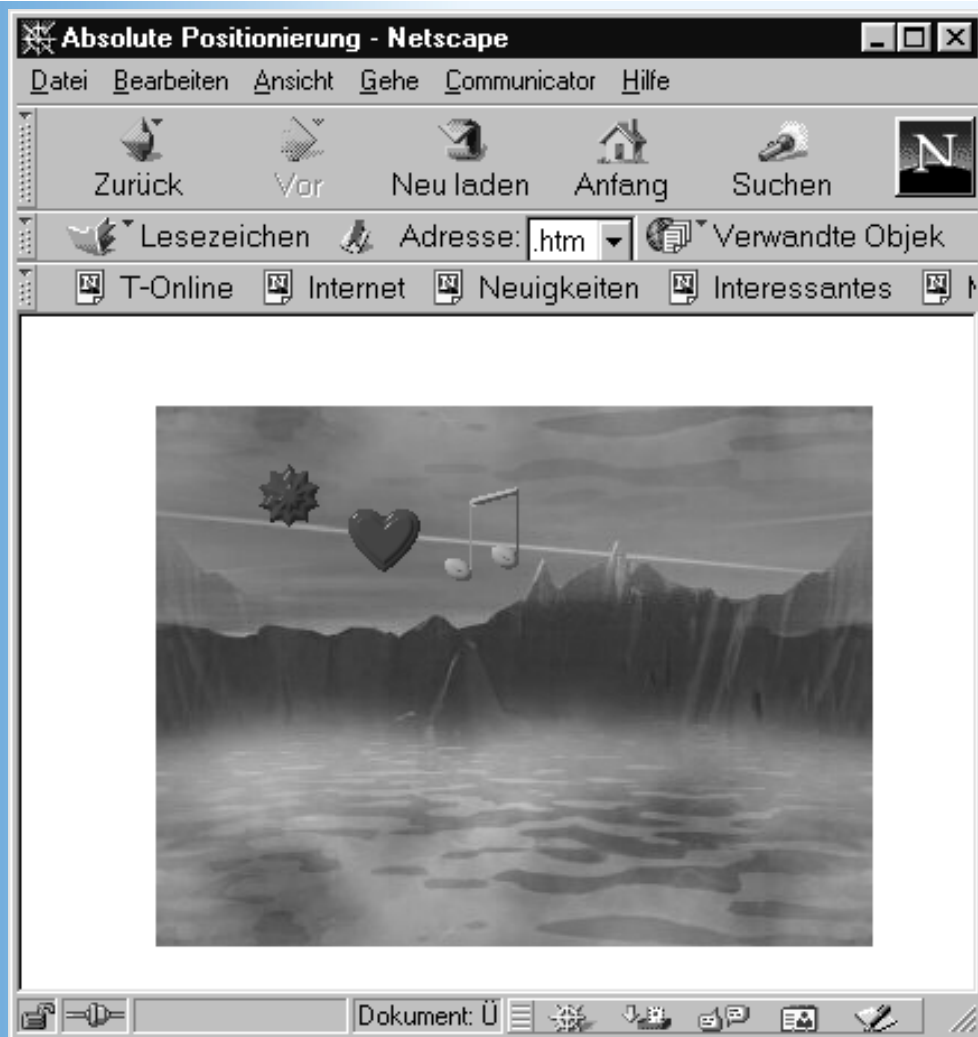


Abbildung 21.13: Überlagerte Bilder

Beispiel 9

Das zweite Beispiel, CSS9.htm zum Positionieren von Elementen legt Texte über Grafiken. Damit lassen sich sehr effektiv und ressourcenschonend Buttons beschriften (etwa bei der Verwendung als Links). Statt mehrerer Grafiken für verschiedene Buttons wird nur eine Grafik mit jeweils unterschiedlichen Texten überlagert:

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
.pos1 {position:absolute; left:50px; top:40px}
.pos2 {position:absolute; left:50px; top:70px}
.pos3
{
position:absolute; left:55px; top:42px;
color : yellow
}
.pos4
{
position:absolute; left:55px; top:72px;
color : yellow
}
-->
</STYLE>
<TITLE>Absolute Positionierung</TITLE>
</HEAD>
<BODY>
<DIV class=pos1><IMG src="button.gif">
```

```
</DIV>
<DIV class=pos2><IMG src="button.gif">
</DIV>
<DIV class=pos3>Kontakt
</DIV>
<DIV class=pos4>Weiter
</DIV>
</BODY>
</HTML>
```



Abbildung 21.14: Nur eine Grafik, aber zweimal verwendet und unterschiedlich beschriftet

Das letzte Beispiel funktioniert allerdings nicht, wenn die im Browser eingestellte Schrift nicht der entspricht, die auf der Entwicklerplattform eingestellt oder deren Größe verändert ist. Auch sonstige Einstellungen können dazwischenfunken (etwa Zoom der Schrift).



Abbildung 21.15: Plattformprobleme

Um die Technik sinnvoll einzusetzen, muss noch mehr festgelegt werden als bei dem letzten Beispiel. Das nächste

Beispiel CSS10.htm sollte soweit alle Randbedingungen festhalten, dass die Geschichte gegen solche unterschiedlichen Situationen gefeit ist. Allerdings kann das Zoomen von Text immer noch Probleme machen (zumindest beim Netscape Navigator - der Internet Explorer ist damit abgesichert²).

Beispiel 10:

```
<HTML>
  <HEAD>
    <STYLE TYPE="text/css">
      <!--
        .pos1 {position:absolute; left:50px; top:40px}
        .pos2 {position:absolute; left:50px; top:70px}
        .pos3
          {position:absolute; left:55px; top:42px;
            color : yellow;
            font-size: 12pt;
            font-family : Arial;
            font-weight : normal
          }
        .pos4
          {position:absolute; left:55px; top:72px;
            color : yellow;
            font-size: 12pt;
            font-family : Arial;
            font-weight : normal
          }
      -->
    </STYLE>
    <TITLE>Absolute Positionierung</TITLE>
  </HEAD>
  <BODY>
    <DIV class=pos1><IMG src="button.gif" height=25>
    </DIV>
    <DIV class=pos2><IMG src="button.gif" height=25>
    </DIV>
    <DIV class=pos3>Kontakt
    </DIV>
    <DIV class=pos4>Weiter
    </DIV>
  </BODY>
</HTML>
```

JavaScript-Zugriff auf CSS

Wenn eine HTML-Seite vom Browser geladen wird, können im Rahmen vom Document Object Model (DOM - siehe Tag 19) die eigenständigen Elemente als Objekte klassifiziert werden (abhängig von der genauen Umsetzung der Browserhersteller). Auf diese Weise kann der Browser jedes unterscheidbare Element einer Seite eindeutig wiederfinden. Als Elemente fasst man in diesem Zusammenhang auch die unter CSS erstellten Stilvorlagen auf, die über Scripte auch angesprochen und verändert werden können.

Dies kann etwa direkt über Veränderungen der Webseite durch den Anwender (etwa über die Tastatur oder die Maus) geschehen oder auf Grund von Eingaben bei irgendwelchen Elementen als eine automatische Folgereaktion.

Das große Problem wird hier nur wieder sein, dass sich die Inkompatibilitäten der unterschiedlichen Ereignisbehandlungsmodelle mit den verschiedenen Style-Sheet-Konzepten potenzieren. Wenn Sie JavaScript unter dem Netscape-Browser in Verbindung mit einem Style-Sheet-Konzept einsetzen wollen, ist das Layer-Konzept in Verbindung mit dem Netscape-eigenen JSSS-Standard für Style Sheets natürlich ideal. Dahingegen werden Microsoft-Browser bei JSSS und dem Layer-Konzept bzw. dem Ereignismodell von Netscape aussteigen. Es ist also unbedingt von Nöten, zwei getrennte Projekte aufzubauen, wenn Sie mit JavaScript Style Sheets unter Ausnutzung des vollständigen

Leistungsumfangs manipulieren wollen.



Für den Einsatz im Rahmen des Netscape-Modells verweisen wir auf Tag 19 und die Ausführungen zum Style- und document-Objekt. Hier wollen wir nach dem Microsoft-Modell vorgehen.

Im Mittelpunkt der dynamischen Veränderung von Webseiten über JavaScript steht das Attribut ID, mit dem Sie HTML-Tags erweitern können und über das ein Script ein Element in einer Webseite identifizieren kann.

Beispiel 11:

Das Beispiel CSSJS1.htm verwendet zwei Stilklassen (stil1 und stil2), die verschiedene Stilangaben definieren. Den beiden Absätzen in der Webseite wird jeweils eine der beiden Stilinformationen als Defaultwert zugewiesen. In dem nachfolgenden Formular wird über die Schaltfläche eine Funktion (farbwechsel()) aufgerufen, welche die beiden Stilvereinbarungen vertauscht. Gleichzeitig wird im Rahmen einer if-Abfrage kontrolliert, welche Stilvereinbarung aktuell zugewiesen ist, und immer die alternative ausgewählt (eine Schalterfunktionalität).

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
.stil1
{
color:red;
font-style:italic;
font-size=18;
background-color = yellow
}
.stil2
{
color:white;
font-style:normal;
font-size=42;
background-color = blue;
border : 15pt groove green;
}
-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function farbwechsel()
{
if (P1.className=="stil1")
{
P1.className="stil2";
P2.className="stil1";
}
else
{
P1.className="stil1";
P2.className="stil2";
}
}
}
</SCRIPT>
<BODY>
<P class=stil1 id=P1>Das ist der Anfang
</P>
<P class=stil2 id=P2>Das ist das Ende
```



```
</P>
<form name="meinFormular">
<input type="button" value="Farbwechsel"
onClick="farbwechsel()">
</form>
</BODY>
</HTML>
```



Abbildung 21.16: Die Absätze verändern dynamisch ihr Aussehen

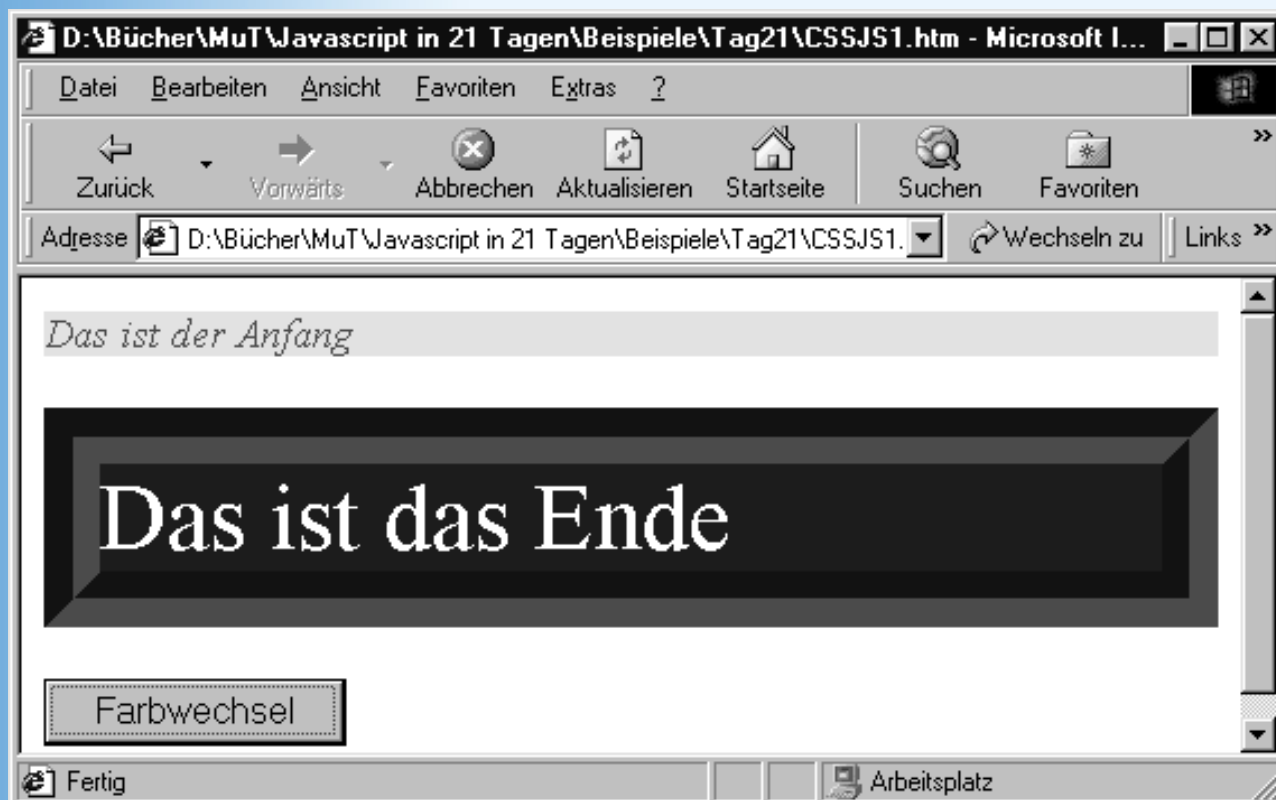


Abbildung 21.17: Originalaussehen

Beispiel 12:

Das nächste Beispiel (CSSJS2.htm) beeinflusst dynamisch die Position von Elementen über Style Sheets. Dabei wird der HTML-<DIV>-Tag verwendet, der mit der ID versehen wird, worüber die Scripte auf die Stilinformation zugreifen. Sobald der Mauszeiger auf das Bild bewegt wird, wird die Funktion poswechsel() ausgelöst. Diese verschiebt jeweils das Bild auf die Alternativposition und verändert dabei die Hintergrundfarbe.

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
  .pos1
  {
    position:absolute; left:100px; top:60px;
    background-color = yellow
  }
  .pos2
  {
    position:absolute; left:180px; top:100px;
    background-color = blue
  }
-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function poswechsel()
{
if (P1.className=="pos1")
{
P1.className="pos2";
document.bild.src='2.gif'
}
else
{
P1.className="pos1";
document.bild.src='1.gif'
}
}
-->
</SCRIPT>
</HEAD>
<BODY>
<DIV class=pos1 id=P1 onmouseover="poswechsel()">
<IMG src="1.gif" height=80 name="bild">
</DIV>
</BODY>
</HTML>
```

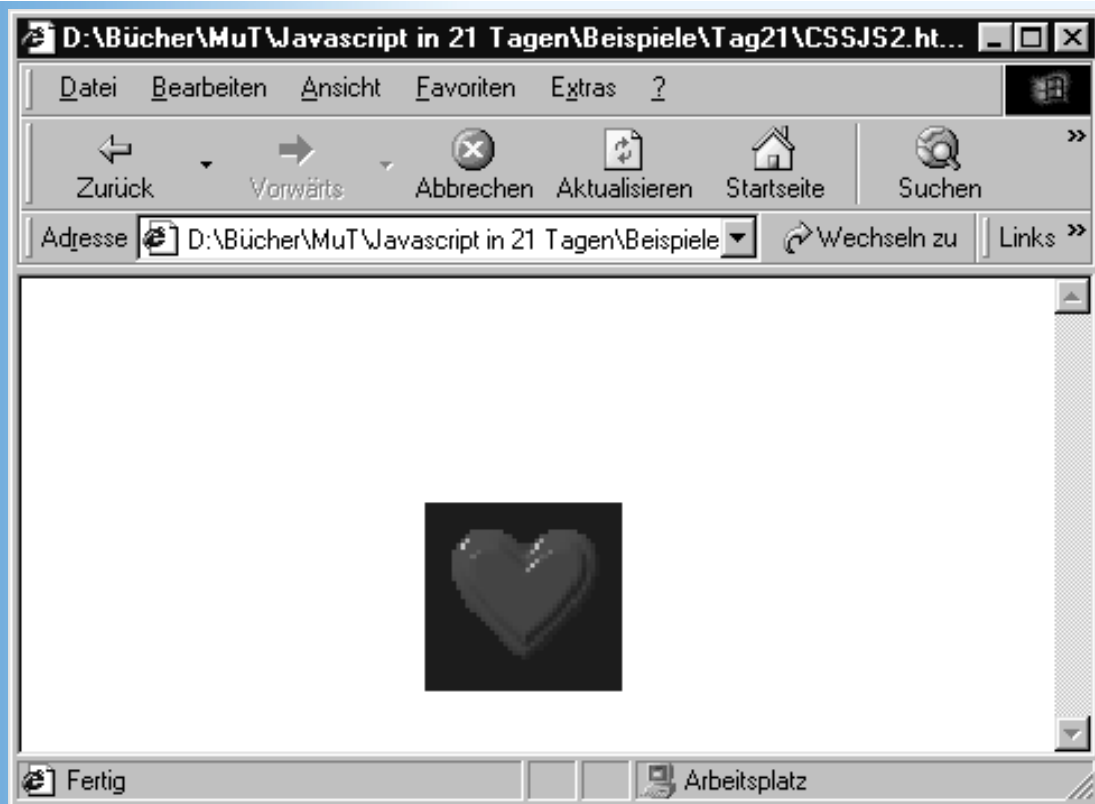


Abbildung 21.18: Hintergrund und Position sind verändert

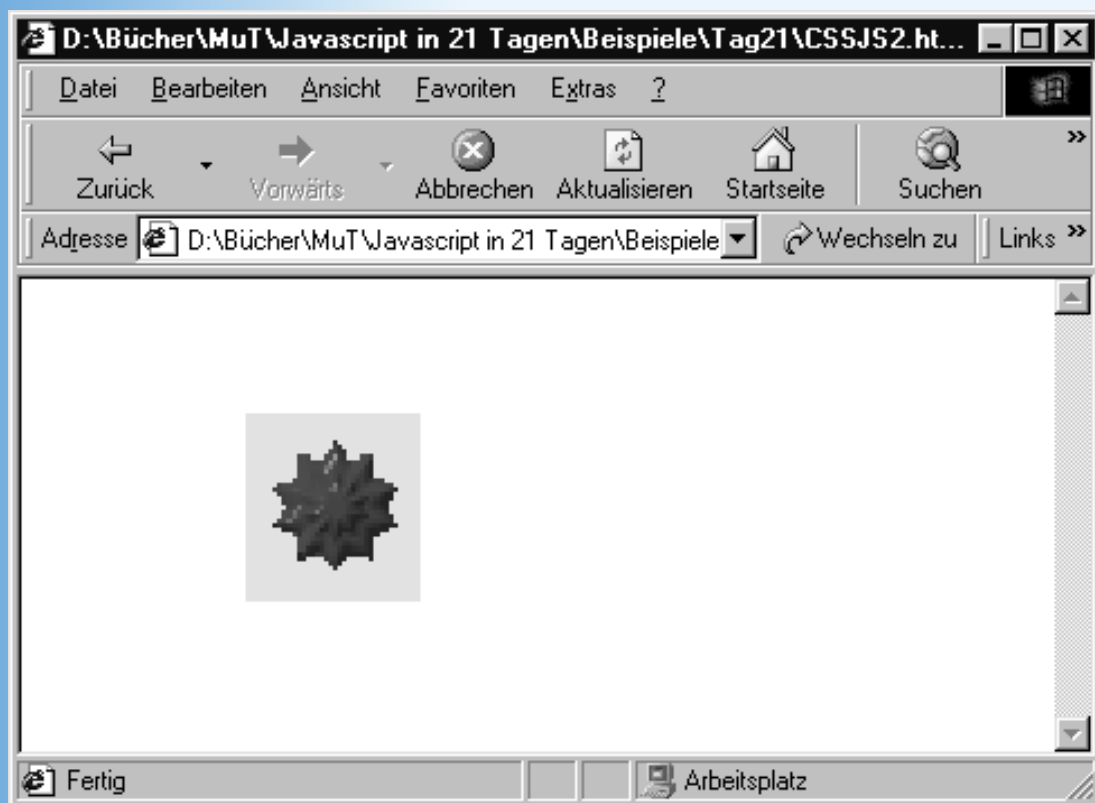


Abbildung 21.19: Originalaussehen

Beispiel 13:

Das nächste Beispiel (CSSJS3.htm) veranschaulicht, wie man einem Anwender bei Hyperlinks Hilfestellung geben kann, welchen Link er gerade mit dem Mauszeiger überstreicht. Es wird in dem Beispiel dem gerade überstrichenen Link eine andere Hintergrundfarbe zugewiesen (Abb. siehe Seite 693):

<HTML>

```
<HEAD>
<STYLE TYPE="text/css">
  .farbe1
  {
    color:blue;
    background-color:white;
    font-size:18
  }
  .farbe2
  {
    color:green;
    background-color:yellow;
    font-size:18
  }
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function weiter1()
{
window.open("css1.htm","top");
}

function weiter2()
{
window.open("css2.htm","top");
}
function weiter3()
{
window.open("css3.htm","top");
}
function weiter4()
{
window.open("css4.htm","top");
}
function farbwechsel(a)
{
if (a.className=="farbe1")
{
a.className="farbe2";
}
else
{
a.className="farbe1";
}
}
</SCRIPT>
</HEAD>
<BODY>
<A href="JavaScript:weiter1()"
  class=farbe1
  id=P1
  onMouseOver="farbwechsel(P1)"
  onMouseOut="farbwechsel(P1)">Link 1
</A><br>
<A href="JavaScript:weiter2()"
  class=farbe1
  id=P2
  onMouseOver="farbwechsel(P2)"
  onMouseOut="farbwechsel(P2)">Link 2
```

```
</A><br>
<A   href="JavaScript:weiter3()"
    class=farbe1
    id=P3
    onMouseOver="farbwechsel(P3)"
    onMouseOut="farbwechsel(P3)">Link 3
</A><br>
<A   href="JavaScript:weiter4()"
    class=farbe1
    id=P4
    onMouseOver="farbwechsel(P4)"
    onMouseOut="farbwechsel(P4)">Link 4
</A><br>
</BODY>
</HTML>
```



Abbildung 21.20: Mauszeiger auf Link 4



Abbildung 21.21: Mauszeiger auf Link 1

Beispiel 14:

Das Beispiel CSSJS4.htm ist eine kleine Animation, die einmal ein animiertes GIF verwendet, es aber zusätzlich in der

Größe verändert. Es entsteht der Eindruck, als würde sich die Fledermaus auf den Betrachter hin bewegen. Von zentraler Bedeutung ist bei dieser - wie allen folgenden - Animationen die Methode `setTimeout()`. Damit kann in definierten Zeitabständen ein Bild vergrößert oder - wie wir es später noch tun werden - verschoben werden. Die Animation startet, wenn der Mauszeiger auf das animierte GIF bewegt wird. Dazu wird dann die Funktion `gross()` aufgerufen. Wenn sich der Mauszeiger aus dem Bildbereich hinaus bewegt, wird die Funktion `klein()` aufgerufen, die das Bild auf die Originalgröße zurücksetzt und damit die Originalstartsituation wiederherstellt:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function gross()
{
setTimeout("document.bat.height=160",500);
setTimeout("document.bat.height=170",1000);
setTimeout("document.bat.height=180",1500);
setTimeout("document.bat.height=190",2000);
setTimeout("document.bat.height=200",2500);
setTimeout("document.bat.height=220",3000);
setTimeout("document.bat.height=250",3500);
setTimeout("document.bat.height=300",4000);
}
function klein()
{
document.bat.height=150;
}
</SCRIPT>
</HEAD>
<BODY>
<IMG src="bat1.gif" height=150 name="bat" onMouseOver="gross()"
onMouseOut="klein()">
</BODY>
</HTML>
```

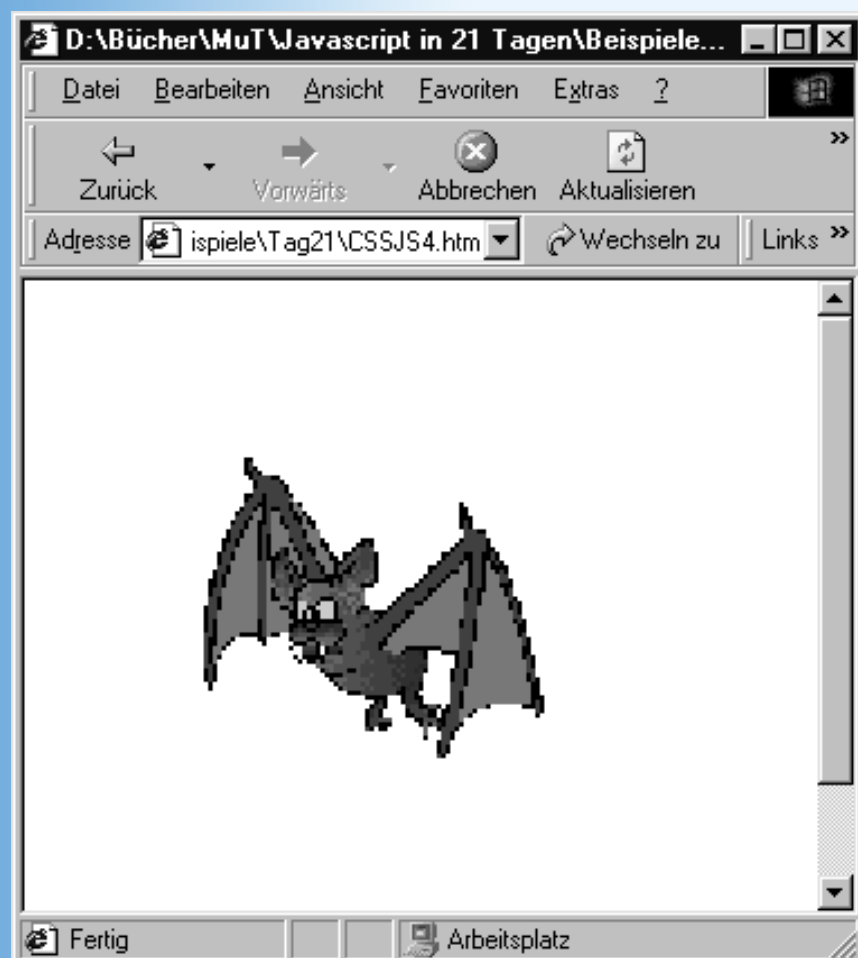


Abbildung 21.22: Aug in Aug mit der Maus

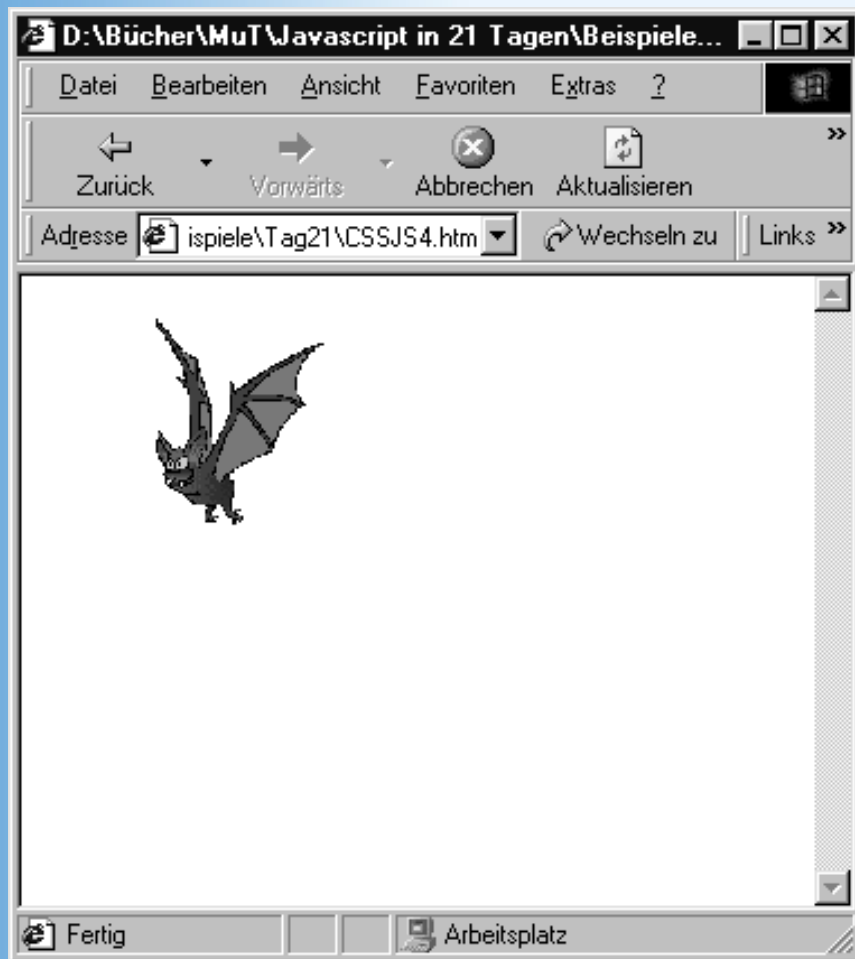


Abbildung 21.23: Noch ist die Fledermaus weit weg

Beispiel 15:

Wir erweitern das Beispiel noch ein wenig. Nach zwei Zeitintervallen wird das Bild zweimal ausgetauscht - die Fledermaus explodiert und geht in eine Rose über (CSSJS5.htm):

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function gross()
{
setTimeout("document.bat.height=60",500);
setTimeout("document.bat.height=70",1000);
setTimeout("document.bat.height=80",1500);
setTimeout("document.bat.height=90",2000);
setTimeout("document.bat.height=100",2500);
setTimeout("document.bat.height=120",3000);
setTimeout("document.bat.height=150",3500);
setTimeout("document.bat.height=200",4000);
setTimeout("document.bat.src='14.gif'",4200);
setTimeout("document.bat.src='219.gif'",5000);
}
function klein()
{
document.bat.height=150;
}
</SCRIPT>
</HEAD>
<BODY>
```

```
<IMG  
  src="bat1.gif"  
  height=60  
  name="bat"  
  onMouseOver="gross()"  
  onMouseOut="klein()">  
</BODY>  
</HTML>
```

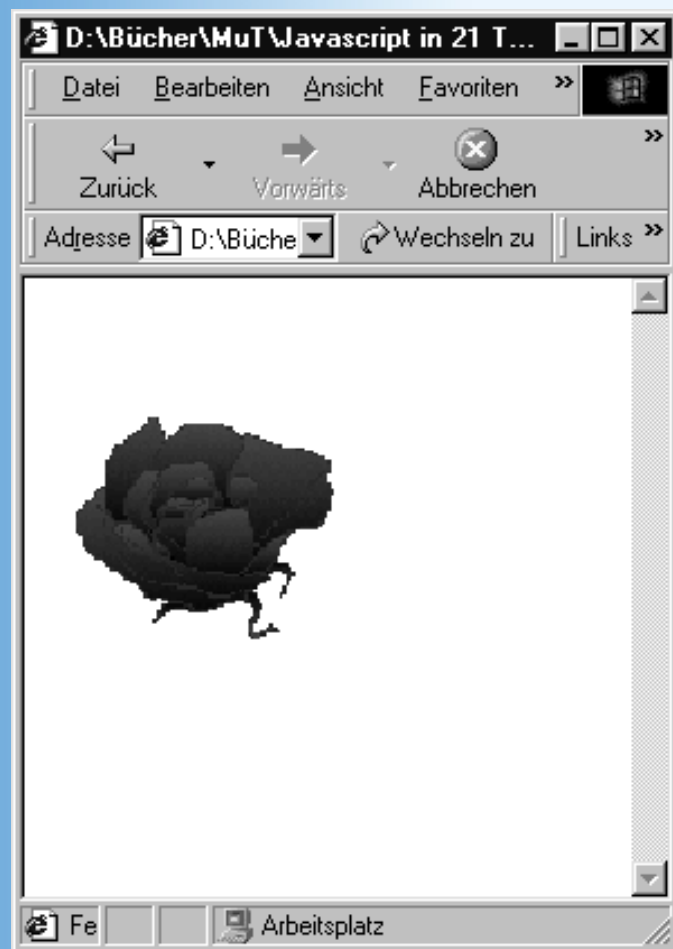


Abbildung 21.24: Es entsteht eine Rose



Abbildung 21.25: Die Fledermaus explodiert

Beispiel 16:

Das abschließende und recht umfangreiche Beispiel ist eine vollständige Animation (CSSJS6.htm). Dabei werden mehrere animierte GIFs verwendet. Eines davon - die Fledermaus - wird entlang einer vorgegebenen Route verschoben. Diese ist aus Style-Sheet- Positionsangaben aufgebaut. Die Animation entsteht, indem das GIF in Zeitintervallen vergrößert und anhand der verschiedenen Positionen verschoben wird.

Die Aktionen können synchronisiert werden oder aber - wie bei uns - asynchron laufen, um die Effekte noch in Hinsicht auf 3D zu steigern. Wenn dann noch Bilder ausgetauscht werden, entsteht ein richtiges kleines Video. Die Handlung dieser Animation ist, dass sich die Fledermaus dem drehenden Totenkopf nähert. Wenn die Positionen deckungsgleich sind, werden beide³ (!) Bilder zeitgleich durch eine Explosion (ein neues animiertes GIF) ersetzt, die dann nach einem weiteren Zeitintervall durch ein animiertes GIF ersetzt wird, das ein Gewitter darstellt.

```
<HTML>
<HEAD>
<STYLE type="text/css">
.pos1
{
  position:absolute;
  left:400px;
  top:100px
}
.pos2
{
  position:absolute;
  left:390px;
  top:110px
}
.pos3
{

```

```
    position:absolute;
    left:380px;
    top:120px
  }
.pos4
{
  position:absolute;
  left:360px;
  top:130px
}
.pos5
{
  position:absolute;
  left:350px;
  top:150px
}
.pos6
{
  position:absolute;
  left:340px;
  top:160px
}
.pos7
{
  position:absolute;
  left:330px;
  top:140px
}
.pos8
{
  position:absolute;
  left:320px;
  top:130px
}
.pos9
{
  position:absolute;
  left:310px;
  top:120px
}
.pos10
{
  position:absolute;
  left:300px;
  top:130px
}
.pos11
{
  position:absolute;
  left:295px;
  top:145px
}
.pos12
{
  position:absolute;
  left:290px;
  top:150px
}
```



```
.pos13
{
  position:absolute;
  left:280px;
  top:160px
}
.pos14
{
  position:absolute;
  left:290px;
  top:165px
}
.pos15
{
  position:absolute;
  left:295px;
  top:170px
}
.pos16
{
  position:absolute;
  left:296px;
  top:185px
}
.pos17
{
  position:absolute;
  left:297px;
  top:200px
}
.pos18
{
  position:absolute;
  left:298px;
  top:215px
}
.pos19
{
  position:absolute;
  left:299px;
  top:230px
}
.pos20
{
  position:absolute;
  left:300px;
  top:250px
}
.endpos
{
  position:absolute;
  left:300px;
  top:260px
}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function gross()
{
```

```
setTimeout("document.bat.height=60",200);
setTimeout("document.bat.height=65",400);
setTimeout("document.bat.height=70",600);
setTimeout("document.bat.height=75",800);
setTimeout("document.bat.height=80",1000);
setTimeout("document.bat.height=85",1200);
setTimeout("document.bat.height=90",1400);
setTimeout("document.bat.height=92",1600);
setTimeout("document.bat.height=94",1700);
setTimeout("document.bat.height=95",1800);
setTimeout("document.bat.height=96",1900);
setTimeout("document.bat.height=97",1950);
setTimeout("document.bat.height=98",2000);
setTimeout("document.bat.height=100",2200);
setTimeout("document.bat.height=105",2400);
setTimeout("document.bat.height=108",2600);
setTimeout("document.bat.height=110",2700);
setTimeout("document.bat.height=111",2800);
setTimeout("document.bat.height=112",2900);
setTimeout("document.bat.height=113",3000);
setTimeout("document.bat.height=114",3100);
setTimeout("document.bat.height=115",3200);
setTimeout("document.bat.height=116",3400);
setTimeout("document.bat.height=117",3600);
setTimeout("document.bat.height=118",3700);
setTimeout("document.bat.height=119",3800);
setTimeout("document.bat.height=120",3900);
setTimeout("document.bat.height=125",4000);
setTimeout("document.tk.height=125",4000);

setTimeout("P1.className='pos2'",300);
setTimeout("P1.className='pos3'",600);
setTimeout("P1.className='pos4'",900);
setTimeout("P1.className='pos5'",1200);
setTimeout("P1.className='pos6'",1600);
setTimeout("P1.className='pos7'",1900);
setTimeout("P1.className='pos8'",2100);
setTimeout("P1.className='pos9'",2200);
setTimeout("P1.className='pos10'",2300);
setTimeout("P1.className='pos11'",2500);
setTimeout("P1.className='pos12'",2800);
setTimeout("P1.className='pos13'",3100);
setTimeout("P1.className='pos14'",3400);
setTimeout("P1.className='pos15'",3600);
setTimeout("P1.className='pos16'",3700);
setTimeout("P1.className='pos17'",3900);
setTimeout("P1.className='pos18'",3950);
setTimeout("P1.className='pos19'",3980);
setTimeout("P1.className='pos20'",4000);

setTimeout("document.bat.src='14.gif'",4200);
setTimeout("document.tk.src='14.gif'",4200);
setTimeout("document.bat.src='233.gif'",5000);
setTimeout("document.tk.src='233.gif'",5000);
setTimeout("klein()",8000);
}
function klein()
{
```

```
document.bat.height=50;
P1.className='pos1';
document.bat.src='bat1.gif';
document.tk.height='75';
P2.className='endpos';
document.tk.src='30.gif';
}
</SCRIPT>
</HEAD>
<BODY bgcolor=black>
<SPAN class=pos1 id=P1>
<IMG
  src="bat1.gif"
  height=50
  name="bat"
  onClick="gross()">
</SPAN>
<SPAN class=endpos id=P2>
<IMG
  src="30.gif"
  height=75
  name="tk">
</SPAN>
</BODY>
</HTML>
```

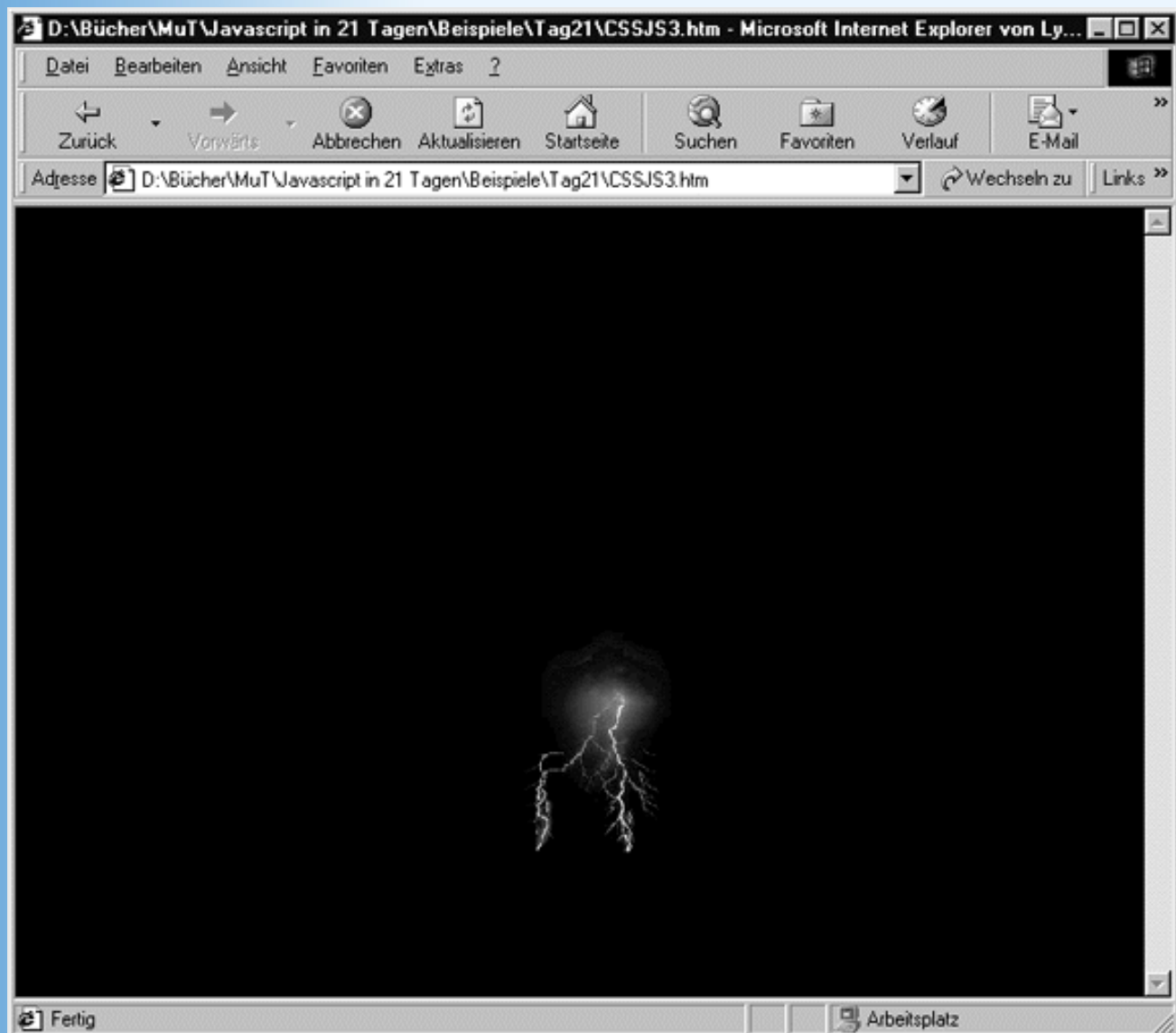


Abbildung 21.26: Die Animation endet im Gewitter

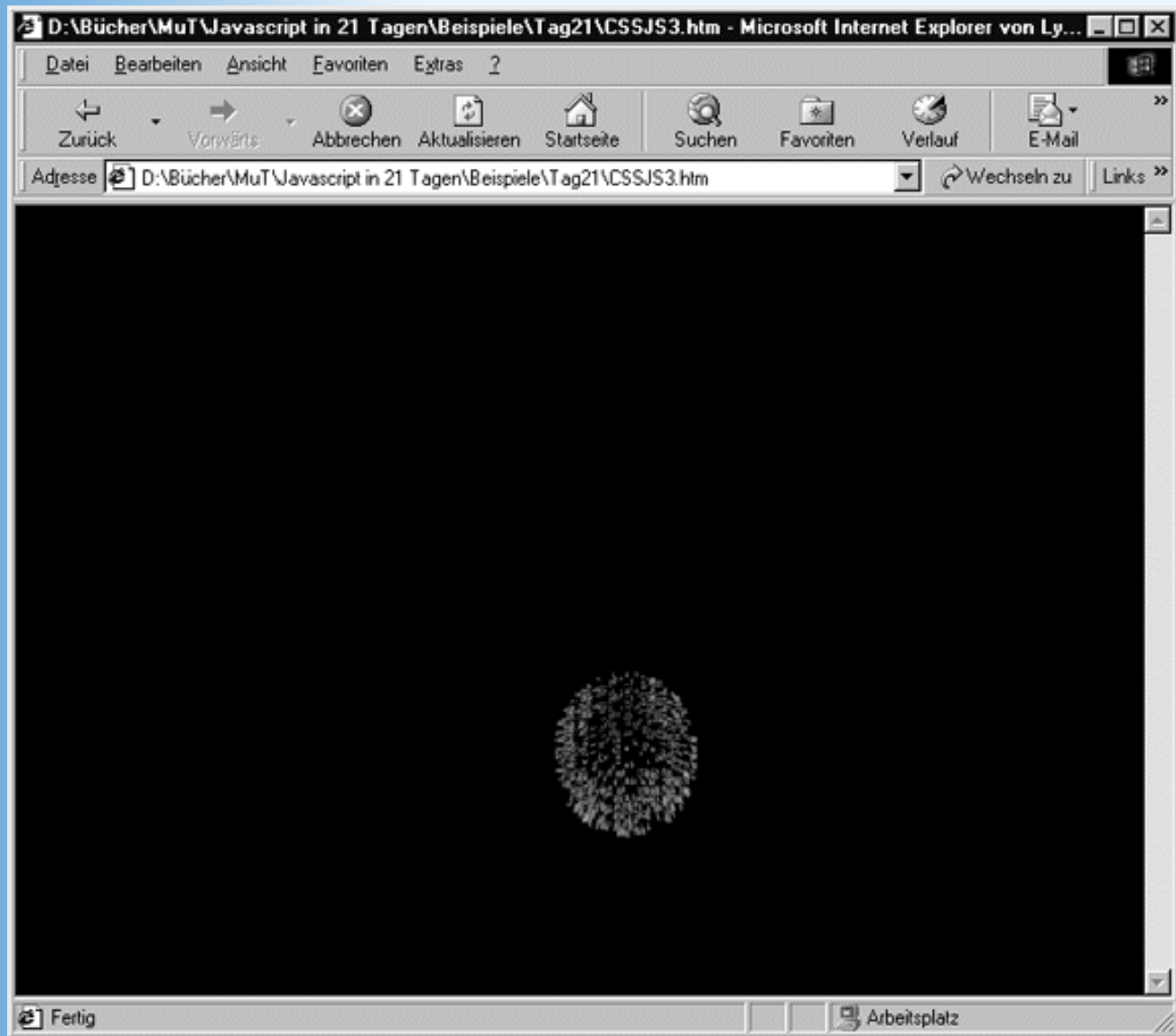


Abbildung 21.27: Beim Zusammenstoß explodieren beide

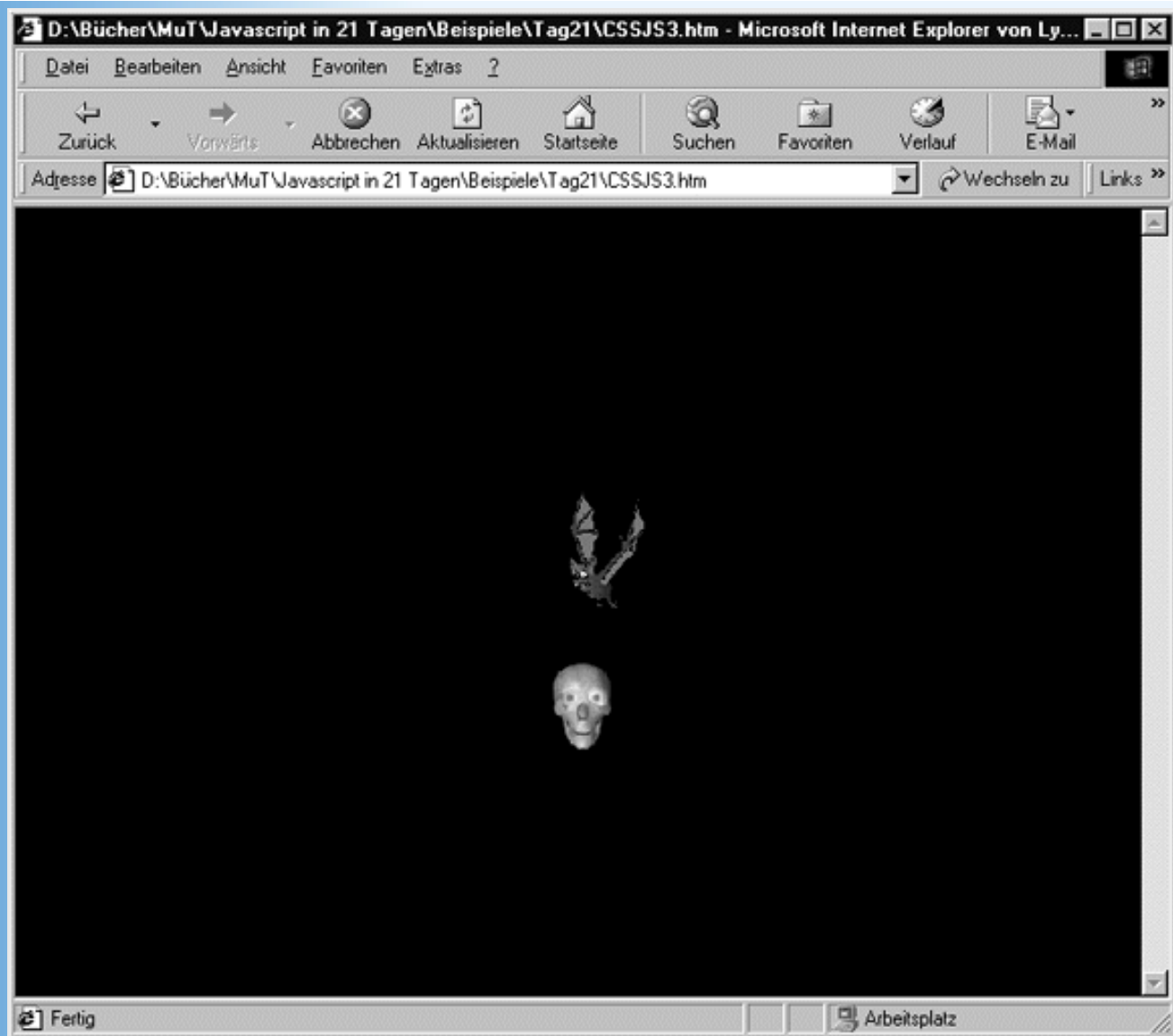


Abbildung 21.28: Die Fledermaus kommt dem Totenkopf näher



Abbildung 21.29: Vor dem Klick sorgen nur die animierten GIFs für Bewegung



Die meisten Webanimationen beruhen auf einer ähnlichen Idee, wie wir sie gerade realisiert haben. Auch Flash-Animationen liegt diese Idee zugrunde, die beliebig verfeinert werden kann.

Zusammenfassung

Am letzten Tag des JavaScript-Lehrgangs haben wir uns das Style-Objekt angeschaut, das bei einigen Eigenschaften oder Methoden des document- Objekts erzeugt wird. Das Style-Objekt eröffnet die Möglichkeit, in Webseiten Dynamic HTML zu realisieren und Style Sheets mit JavaScript zu verbinden.

Style Sheets gibt es in verschiedensten Varianten. Wir haben uns auf die Behandlung von Cascading Style Sheets beschränkt, die die neuen Browser zumindest in den Kernbereichen recht einheitlich implementieren. Die Verbindung von Style Sheets mit einer HTML-Seite kann darüber geschehen, dass man Style Sheets in eine Seite einbettet oder aus einer externen Datei importiert. Die Syntax einer CSS-Deklaration sieht immer gleich aus. Es gibt jedes Mal den folgenden Aufbau:

name : wert

Dabei gibt

name

das zu formatierende Element (den so genannten Selector) an und

wert

die konkrete Stilinformation. Mehrere Deklarationen werden mit Semikolon getrennt. In vielen Situationen werden die Werte auch in geschweifte Klammern gesetzt. Der Selector ist bis auf wenige Ausnahmen einer der existierenden HTML-Tags.

Ein HTML-Tag kann, mit einem Punkt abgetrennt, um die optionale Angabe einer Klasse erweitert werden. Die Syntax sieht so aus:

[HTML - Tag] . [class]

Dies ist die Festlegung der Eigenschaften für eine Instanz eines Elements. Dabei legt man eine Klasse fest und verwendet diese bei der Stilzuweisung. Für class kann jeder beliebige Name genommen werden. Damit ist es möglich, verschiedene Stile für ein und dasselbe HTML-Element zu erstellen. Der ursprüngliche Selector, welcher vor dem Punkt steht, ist das Elternelement zu dieser Instanz.

Eine Klasse kann auch alleine in der Stildeklaration angegeben werden. In diesem Fall muss einem Klassenbezeichner in der Deklaration ein Punkt vorangestellt und bei der konkreten Verwendung ein HTML-Element um den so benannten neuen Parameter erweitert werden.

Grundsätzlich werden CSS-Stilvereinbarungen in verschiedene Kategorien unterteilt, die wir uns in diesem Kapitel angesehen haben:

- Font- Stilvereinbarungen
- Stilvereinbarungen für Farbe und Hintergrund
- Stilvereinbarungen zur Gestaltung von Textformatierungen
- Stilvereinbarungen zur Gestaltung von Textblock-Eigenschaften
- Positionierung mit CSS

Der JavaScript-Zugriff auf CSS kann im Rahmen der Festlegungen erfolgen, die vom Document Object Model festgelegt werden. Wegen der Inkompatibilitäten der unterschiedlichen Ereignisbehandlungsmodelle sowie den verschiedenen Style-Sheet-Konzepte sollten getrennte Projekte für die verschiedenen Browser erstellt werden.

Fragen und Antworten

Frage:

Wenn Effekte aus dem Style-Sheet-Bereich genutzt werden, sollten die Browser-Welten getrennt werden, oder gibt es nicht einen kleinen gemeinsamen Nenner, den man nutzen kann?

Antwort:

Den gibt es schon, aber da die Unterschiede der Browser-Welten so komplex sind, macht es meist mehr Arbeit, jeden einzelnen Effekt auf verschiedenen Browserkonstellationen wasserdicht zu machen als gleich zu trennen. Prinzipiell sollten Sie aber in jedem Fall testen, was das Zeug hält. Animationen würde ich aber auf jeden Fall trennen.

Frage:

Muss man bei einer Animation jede Position einzeln notieren oder kann man da nicht mit Schleifen arbeiten?

Antwort:

Doch. Kann man. Das ist aber nicht ganz so einfach. Am sinnvollsten ist es, wenn man eine Seite neu schreibt. Das heißt, ein Script generiert Positions- und Größenwerte und daraus wird die Seite samt Style-Sheet-Informationen geschrieben. Aber prinzipiell werden Sie oft den Fall vorfinden, dass wirklich jede Position einer Animation in der Seite notiert ist und diese Positionen der Reihe nach aufgerufen werden.

Workshop

Nehmen Sie die Webanimationen aus den letzten Beispielen und verfeinern Sie diese. Eine Erweiterung wäre, wenn man unter Hinzuziehung der random()-Methode einen Zufallsmechanismus generierte, der beispielsweise dazu genutzt wird, die Auswahl der nächsten Position von einem Zufall abhängig zu machen. Etwa wenn die Methode einen Wert größer als 0,25 liefert, wird nach vorne gegangen, ansonsten wieder zurück auf die letzte Position. Oder nehmen Sie feinere Unterteilungen mit mehr Positionen und geringeren Größenveränderungen, die aber schneller aufgerufen werden. Man kann auch den Zufallsmechanismus dazu nutzen, verschiedene mögliche Wege auszuwählen. Eine weitere Verfeinerung wäre, wenn eine Animation endlos lief. Dazu kann man eine Animation einfach rückwärts die Positionen oder Größenveränderungen wieder durchlaufen lassen. Solche Dinge machen vor allem Sinn, wenn man auf Schleifen

zurückgreift.

Kontrollfragen



Im Anhang »Antworten zu den Kontrollfragen« finden Sie die Antworten zu den jeweils am Ende einer Lektion notierten Fragen.

1. Von welchen Eigenschaften oder Methoden des document-Objekts wird das Style-Objekt erzeugt?
2. Wie können Style Sheets in eine Webseite integriert werden?
3. Wie sieht die Syntax von einer CSS-Deklaration immer aus?
4. Was sind die wichtigsten Elemente einer Animation?
5. Ist das korrekt?

```
H6.a
{
color : blue
}
```

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

1
Aber leider nicht nur da.

2
Eine von mehreren denkbaren Lösungen für den Navigator wäre, auch den Text als Grafiken - transparent und möglichst mit kleinem Text - abzuspeichern und über den Button zu positionieren.

3
Damit es keine Überlagerung von verschiedenen Bildern gibt.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Woche 3 Rückblick

Sie haben nun das Ende der JavaScript-Lektionen erreicht. Die dritte Woche stand ganz unter dem Aspekt der Arbeit mit Objekten in JavaScript. In dieser Woche wurde besprochen,

- was Objekte genau sind und wie sie in JavaScript allgemein eingesetzt werden,
- wie eigene Objekte erstellt werden können und was der Sinn davon ist,
- welche Objekte Ihnen direkt zur Verfügung stehen und wie diese für den Zugriff auf Browser, die Anwenderplattform und die Elemente einer Webseite genutzt werden können,
- welche Top-Level-Objekte für die Arbeit unter JavaScript bereitstehen,
- wie eine globale Ereignisbehandlung unter JavaScript funktioniert,
- wie Formulare in einer Webseite eingesetzt werden können und
- wie Stilvereinbarungen in einer Webseite genutzt werden können - mit und ohne JavaScript.

Am Ende der drei Wochen haben Sie so viel von JavaScript und HTML, aber auch Style Sheets erfahren, dass Sie sowohl reine Webseiten erstellen als auch mit JavaScript und Style Sheets ganz nach Ihren Bedürfnissen anpassen, erweitern oder perfektionieren können. Für Ihre weitere Arbeit damit wünsche ich Ihnen viel Erfolg und vor allem Spaß.

Tag A

Wichtige Tabellen zu JavaScript, HTML, Maskierungen, Quellen usw.

Die JavaScript-Token in Bezug auf die ECMA-Norm

JavaScript ist in der Version 1.3 mit der ECMA-262-Norm kompatibel. Dies umfasst folgende Token:

Kategorie	Token
Datenfeldverarbeitung	Array, join, length, reverse, sort
Zuweisungen	Zuweisungsoperatoren und Verbundzuweisungsoperatoren
Boolesche Werte	Boolean
Kommentare	/*...*/ oder //
Konstanten/Literale	NaN, null, true, false, +Infinity, undefined
Ablaufsteuerung	break, continue, for, for...in, if...else, return, while

Datum und Zeit	<code>Date, getDate, getDay, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getYear,,</code>
	<code>getUTCDate, getUTCDay, getUTCFullYear, getUTCHours, getUTCMilliseconds, getUTCMinutes, getUTCMonth, getUTCSeconds,, setDate, setFullYear, setHours, setMilliseconds, setMinutes, setMonth, setSeconds, setTime, setYear,, setUTCDate, setUTCFullYear, setUTCHours, setUTCMilliseconds, setUTCMinutes, setUTCMonth, setUTCSeconds,, toGMTString, toLocaleString, toUTCString, parse, UTC</code>
Deklarationen	<code>function, new, this, var, with</code>

Funktionserstellung	Function, arguments, length
Globale Funktionen	Global, escape, unescape, eval, isFinite, isNaN, parseInt, parseFloat
Math	Math, abs, acos, asin, atan, atan2, ceil, cos, exp, floor, log, max, min, pow, random, round, sin, sqrt, tan, E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2
Zahlen	Number, MAX_VALUE, MIN_VALUE, NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY

Objekterstellung	Object, new, constructor, prototype, toString, valueOf
Operatoren	Addition +, Subtraktion -, Modulo %, Multiplikation *, Division /, Unäre Negation -, Gleichheit ==, Ungleichheit !=, Kleiner als <, Kleiner gleich <=, Größer als >, Größer gleich >=, Logisches AND &&, Logisches OR , Logisches NOT !, Bitweises AND &, Bitweises OR , Bitweises NOT ~, Bitweises Xor ^, Bitweises Left Shift <<, Bitweises Right Shift >>, Vorzeichenloses Right Shift >>>, Trinäre Bedingung ?: , Komma ,, delete, typeof, void, Dekrement --, Inkrement ++
Objekte	Array, Boolean, Date, Function, Global, Math, Number, Object, String

Zeichenfolgen	String, charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, split, toLowerCase, toUpperCase, length

Tabelle A.1: JavaScript-Token

JavaScript erweitert die ECMA-262-Norm noch um einige Elemente. JavaScript 1.3 etwa kennt den Operator für die strikte Gleichheit (=== prüft zwei Ausdrücke, um zu bestimmen, ob sie den gleichen Wert und Datentyp aufweisen) und einen vertikalen Tabulator als Escape-Sequenz (\v oder \u000B) und auch einige nicht in der Norm enthaltenen Anweisungen (label, switch, do...while, export, import).

Daneben gibt es

- bei allen Kernobjekten die Methode toSource,
- beim Object-Objekt die Methoden watch und unwatch (beim Ereignismodell von Netscape von Bedeutung),
- beim Function-Objekt die Eigenschaft arity und die Methoden apply sowie call,
- beim Array-Objekt die Eigenschaften index und input sowie die Methoden concat, pop, push, shift, unshift, slice und splice und
- beim String-Objekt die Methoden concat, match, search, slice und substr.

Dazu kommt noch das Built-in Objekt RegExp samt der dortigen Funktionalitäten. Dies sind insbesondere reguläre Ausdrücke, die zur Erstellung von Suchmustern verwendet werden können. Es gibt diese regulären Ausdrücke:

Regulärer Ausdruck	Beschreibung
\$	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht dem Ende der Eingabe.
(Muster)	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden.
*	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht dem vorhergehenden Zeichen einmal oder mehrere Male.
.	Ein Sonderzeichen bzw. eine Sonderzeichenfolgen, die beim Schreiben von Mustern regulärer Ausdrücke verwendet wird bzw. werden. Entspricht allen Einzelzeichen außer einem Zeilenumbruchzeichen.

?	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht dem vorhergehenden Zeichen kein- oder einmal.
[^m-z]	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Definiert einen ausgeschlossenen Bereich an Zeichen, d.h. diese Zeichen sind in einem Bereich nicht enthalten.
[^xyz]	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Eine Gruppe ausgeschlossener Zeichen. Entspricht allen nicht enthaltenen Zeichen.
[a-z]	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Ein Zeichenbereich. Entspricht allen Zeichen im angegebenen Bereich.
[xyz]	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Eine Gruppe von Zeichen. Entspricht einem beliebigen der enthaltenen Zeichen.
\	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Markiert das folgende Zeichen als maskiertes Sonderzeichen.
\b	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Der gesuchte Ausdruck darf nicht am Rand eines Wortes stehen.
\B	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Der gesuchte Ausdruck darf am Rand eines Wortes stehen.
\d	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht einer Ziffer.
\D	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht einer Nicht-Ziffer.
\f	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht einem Formularvorschubzeichen.
\n	Eine Escape-Sequenz, die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird. Die numerische Angabe n entspricht jeweils einem oktalen Escape-Wert. Oktale Escape-Werte können 1-, 2- oder 3-stellig sein, dürfen aber 256 vom Wert her nicht überschreiten.
\n	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht einem Zeilenumbruchzeichen.

<code>\num</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Die numerische Angabe ist eine positive Ganzzahl, welche das letzte Zeichen so oft wie angegeben wiederholt.
<code>\r</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht einem Wagenrücklaufzeichen.
<code>\s</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht beliebigen nichtdruckbaren Druckzeichen wie Leerzeichen, Tabulator, Formularvorschub etc.
<code>\S</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht beliebigen Nicht-Leerzeichen.
<code>\t</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht einem Tabulatorzeichen.
<code>\v</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht einem vertikalen Tabulatorzeichen.
<code>\w</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht allen Wortzeichen mit Unterstrich.
<code>\W</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht allen Nicht-Wortzeichen.
<code>\xn</code>	Eine Escape-Sequenz, die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird. Die numerische Angabe n entspricht einem hexadezimalen Escape-Wert (genau zwei Stellen).
<code>^</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht dem Anfang der Eingabe.
<code>{n, }</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. n ist eine nicht negative Ganzzahl, welche die minimale Anzahl der geforderten Wiederholungen eines Zeichens oder einer Folge von Zeichen in einem Suchbereich festlegt.
<code>{n, m}</code>	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. n und m sind nicht negative Ganzzahlen, welche die minimale und maximale Anzahl der geforderten Wiederholungen eines Zeichens oder einer Folge von Zeichen in einem Suchbereich festlegen.

{n}	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. n ist eine nicht negative Ganzzahl, welche die exakte Anzahl der geforderten Wiederholungen eines Zeichens oder einer Folge von Zeichen in einem Suchbereich festlegt.
+	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht dem vorhergehenden Zeichen einmal oder mehrere Male.
x y	Ein Sonderzeichen bzw. eine Folge von Sonderzeichen, das/die bei Mustern im Rahmen regulärer Ausdrücke verwendet wird bzw. werden. Entspricht entweder x oder y.

Tabelle A.2: Reguläre Ausdrücke

Die unterstützten Varianten von JavaScript bzw. JScript in den wichtigsten Browserversionen

Die nachfolgende Tabelle zeigt, welche Version von JavaScript im Netscape Navigator unterstützt wird:

Browser	Version
Navigator 2.0	JavaScript 1.0
Navigator 3.0	JavaScript 1.1
Navigator 4.0 - 4.05	JavaScript 1.2
Navigator 4.06 - 4.5	JavaScript 1.3
Navigator 4.6 - 4.ff	teilweise JavaScript 1.4 (soweit offiziell)
Navigator 6	In dieser Ausführung des Navigator soll JavaScript in der Version 1.5 eingeführt werden.

Tabelle A.3: JavaScript-Unterstützung im Navigator

Die nachfolgende Tabelle zeigt, welche Version von JavaScript (was genau genommen JScript ist) im Internet Explorer unterstützt wird:

Browser	Version
Internet Explorer 2.0	keine JavaScript-Unterstützung
Internet Explorer 3.0	JavaScript 1.0
Internet Explorer 3.02	JavaScript 1.1 (mit erheblichen Einschränkungen)
Internet Explorer 4.0	JavaScript 1.2
Internet Explorer 5.0	JavaScript 1.3

Tabelle A.4: JavaScript-Unterstützung im Internet Explorer

JavaScript 1.4 wird derzeit nicht im Internet Explorer unterstützt.

Die folgende Tabelle zeigt noch mal explizite Versionsbeziehungen zwischen JScript als eigenständige Sprache und dem Internet Explorer:

Browser	Version
Internet Explorer 3.0	JScript 1.0
Internet Explorer 4.0	JScript 3.0
Internet Explorer 5.0	JScript 5.0

Tabelle A.5: JScript-Unterstützung im Internet Explorer

Die nachfolgende Tabelle zeigt für den Opera-Browser zwei Versionen, welche JavaScript unterstützen:

Browser	Version
Opera 3.6	JavaScript 1.1
Opera 4.0	JavaScript 1.3 (angekündigt)

Tabelle A.6: JavaScript-Unterstützung im Opera-Browser



Bei allen Tabellen muss beachtet werden, dass es für nahezu alle Zyklen und Techniken bestimmte Konstellationen gibt, wo es Probleme geben kann. Die meisten Browser beachten die offiziellen Standards nicht vollständig.

Die unterstützten HTML-Versionen in den wichtigsten Browserversionen

Die nachfolgende Tabelle zeigt, welche Variante von HTML im Netscape Navigator unterstützt wird:

Browser	Version
Navigator 2.0	HTML 2.0
Navigator 3.0	HTML 3.0/3.2
Navigator 4.0 ff	HTML 4.0

Tabelle A.7: HTML-Unterstützung im Navigator

Die nachfolgende Tabelle zeigt, welche Variante von HTML im Internet Explorer unterstützt wird:

Browser	Version
Internet Explorer 2.0	HTML 2.0

Internet Explorer 3.0	HTML 3.0/3.2
Internet Explorer 4.0 ff	HTML 4.0

Tabelle A.8: HTML-Unterstützung im Internet Explorer



Besonders bei HTML muss beachtet werden, dass sich die meisten Browser wenig um offizielle Standards kümmern und viele Sonderlösungen integriert haben. Zudem werden einige Anweisungen nicht vollständig umgesetzt.

Die Auflistung aller HTML-Elemente und -Attribute

HTML 4.0 beinhaltet eine umfangreiche Anzahl von offiziellen Befehlen und noch mehr Attribute. Hier finden Sie die Tags und zugehörige Attribute.

HTML-Tags

Nachfolgend sollen alphabetisch sämtliche Elemente, die zum offiziellen HTML-4.0-Sprachstandard gehören, aufgelistet werden. Die Auflistung beruht auf der offiziellen Veröffentlichung des W3C (<http://www.w3.org/TR/REC-html40/index/elements.html>).

Wenn im Start- und Ende-Tag keine Angaben stehen, werden die Tags jeweils gefordert. Dabei gilt, dass es Situationen geben kann, wo der Ende-Tag zwar gefordert wird, er aber in der Praxis ohne negative Auswirkung weggelassen werden kann. Umgekehrt gilt: Sofern ein Ende-Tag verboten ist, gibt es keinen Container, der dem Start-Tag nachfolgenden Inhalt enthalten kann. Ein solches Element, wo der Ende-Tag verboten ist, kann aber ohne Probleme mit einem fiktiven Ende-Tag ausgestattet werden (etwa aus dokumentarischen Gründen). Der Ende-Tag wird keinen Fehler erzeugen - er wird durch das Prinzip der Fehlertoleranz einfach ignoriert.

Als *Deprecated* gekennzeichnete Elemente gelten als veraltet. Dafür gibt es im Rahmen des neuen Sprachstandards andere Elemente, die statt dessen verwendet werden sollen. Die Verwendung veralteter Elemente ist aber keinesfalls verboten. Viele der Elemente können durch Attribute erweitert werden. Die Tabelle mit den Attributen finden Sie direkt anschließend.

Tag	Beschreibung
A	Ein Anker.
ABBR	Abkürzung (z.B. WWW, HTTP, USA, RJS etc.)
ACRONYM	Ein Acronym.
ADDRESS	Adressangabe für Informationen über den Autor.
APPLET	Einbindung eines Java-Applets. Deprecated!
AREA	Client-seitiger Imagemap-Bereich. Kein Ende-Tag vorgesehen.
B	Textstil fett.

BASE	Basis-URL des Dokumentes. Kein Ende-Tag vorgesehen.
BASEFONT	Basisschriftgröße. Kein Ende-Tag vorgesehen. Deprecated.
BDO	Deaktivierung der Standardausrichtung des Textes.
BIG	Textstil groß.
BLOCKQUOTE	Zitat (lang).
BODY	Der Dokumentenkörper.
BR	Zeilenumbruch. Kein Ende-Tag vorgesehen.
BUTTON	Pushbutton.
CAPTION	Tabellenbeschriftung.
CENTER	Zentrieren. Deprecated.
CITE	Zitat.
CODE	Textstil für Computercode.
COL	Tabellenspalte. Kein Ende-Tag vorgesehen.
COLGROUP	Tabellenspaltengruppe. Optionaler Ende-Tag.
DD	Definitionsbeschreibung. Optionaler Ende-Tag.
DEL	Gelöschter Text.
DFN	Instanzdefinition.
DIR	Verzeichnisliste. Deprecated.
DIV	Allgemeiner Stilcontainer zur Gestaltung (Ausrichtung) von Elementen.
DL	Definitionsliste.
DT	Definitionsterm. Optionaler Ende-Tag.
EM	Textstil für eine Betonung.
FIELDSET	Formularkontrollgruppe. Gruppiert gleichartige Steuerelemente.
FONT	Lokale Schriftveränderung. Deprecated.
FORM	Formular.
FRAME	Ein Frame. Kein Ende-Tag vorgesehen.
FRAMESET	Die Oberstruktur einer Framegruppe.
H1	Überschrift der Ordnung 1.
H2	Überschrift der Ordnung 2.
H3	Überschrift der Ordnung 3.
H4	Überschrift der Ordnung 4.
H5	Überschrift der Ordnung 5.
H6	Überschrift der Ordnung 6.
HEAD	Dokumenten-Header (Kopfbereich).
HR	Horizontale Trennlinie. Kein Ende-Tag vorgesehen.

HTML	Der äußere Container einer Webseite.
I	Textstil italic (kursiv).
IFRAME	Innenliegendes Unterfenster.
IMG	Bildreferenz. Kein Ende-Tag vorgesehen.
INPUT	Formulareingabe. Kein Ende-Tag vorgesehen.
INS	Eingefügter Text.
ISINDEX	Einzeiliges Prompt (Eingabeaufforderung). Kein Ende-Tag vorgesehen. Deprecated.
KBD	Stil für Text, der von dem Anwender eingegeben wird.
LABEL	Beschriftungstext für ein Formularfeld.
LEGEND	Fieldset-Legende.
LI	Definition eines Listenelementes. Optionaler Ende-Tag.
LINK	Ein Medien-unabhängiger Link. Kein Ende-Tag vorgesehen.
MAP	Client-seitige Imagemap.
MENU	Menüliste. Deprecated.
META	Allgemeine Metainformationen. Kein Ende-Tag vorgesehen.
NOFRAMES	Alternativ angezeigter Inhalt, wenn keine Frames angezeigt werden.
NOSCRIPT	Alternativ angezeigter Inhalt, wenn keine Scripts ausgeführt und angezeigt werden.
OBJECT	Allgemeines Einbettungsobjekt.
OL	Sortierte Liste.
OPTGROUP	Optionsgruppe.
OPTION	Selektierbare Auswahl in einem Formular. Optionaler Ende-Tag.
P	Absatz. Optionaler Ende-Tag.
PARAM	Initialisierungswert für ein Objekt. Kein Ende-Tag vorgesehen.
PRE	Präformatierter Text.
Q	Kurzes Zitat in einer Zeile.
S	Textstil durchgestrichen. Deprecated.
SAMP	Textformatierung für die Darstellung von Beispielen.
SCRIPT	Einbindung von Scriptanweisungen.
SELECT	Auswahlmöglichkeiten in einem Formular.
SMALL	Textstil klein.
SPAN	Allgemeiner Sprach- und Stilcontainer.
STRIKE	Textstil durchgestrichen. Deprecated.
STRONG	Textstil starke Hervorhebung.
STYLE	Einbindung von Stilinformationen (Style Sheets).
SUB	Textstil tiefstellen.

SUP	Textstil hochstellen.
TABLE	Tabelle.
TBODY	Tabellenkörper. Nach offizieller W3C-Aussage ist sowohl die Angabe optional als auch der Ende-Tag.
TD	Tabellendatenzelle. Optionaler Ende-Tag.
TEXTAREA	Mehrzeiliges Textfeld in einem Formular.
TFOOT	Angabe eines Fußbereichs bei Tabellen. Optionaler Ende-Tag.
TH	Tabellenkopfzellen. Optionaler Ende-Tag.
THEAD	Angabe eines Kopfbereichs bei Tabellen. Optionaler Ende-Tag.
TITLE	Dokumententitel.
TR	Tabellenreihe. Optionaler Ende-Tag.
TT	Textstil nichtproportional.
U	Textstil unterstrichen. Deprecated.
UL	Unsortierte Liste.
VAR	Instanz einer Variablen oder eines Programmargumentes.

Tabelle A.9: HTML-Tags

Die HTML-4.0-Attribute

Viele der Tags des HTML-4.0-Sprachstandards können durch Attribute erweitert werden, die sie genauer spezifizieren oder gar erst sinnvoll machen. Die nachfolgende Auflistung enthält alphabetisch sämtliche Attribute, welche zum dem HTML-4.0-Sprachstandard gehören.



Viele Attribute werden mehrfach angegeben. Dies ist deshalb zwingend, weil sich ein identisch benanntes Attribut verschieden auswirken kann. Zusätzlich werden in den Spalten nach dem Attribut Angaben zum Typ des Attributes, ob das Attribut optional eingesetzt werden kann oder ob es bei den zugeordneten Elementen gefordert wird und ob es deprecated ist, gemacht. Auch dies kann sich bei identisch bezeichneten Attributen unterscheiden. Sofern sich Attribute bei verschiedenen HTML-Befehlen gleich auswirken, werden die Elemente zusammengefasst.

Die Auflistung enthält die vollständige Liste der Attribute des HTML-4.0- Standards und basiert auf der offiziell vom W3C (<http://www.w3.org/TR/REC-html40/index/attributes.html>) ausgegebenen Liste. Beachten Sie bitte, dass nicht alle der Attribute in den verschiedenen Browserversionen (auch den aktuellen) umgesetzt sind. Dies gilt insbesondere für die Event- Handler, die hauptsächlich der Netscape Navigator nur sehr eingeschränkt unterstützt.

Attribut	Beschreibung
abbr	Anwendung bei: TD, TH

	Typ: %Text;
	Verwendung: #OPTIONAL
	Abkürzung für die Kopfzelle einer Tabelle.
accept-charset	Anwendung bei: FORM
	Typ: %Charsets;
	Verwendung: #OPTIONAL
	Liste der unterstützten Zeichensätze bei einem Formular.
accept	Anwendung bei: INPUT
	Typ: %ContentTypes;
	Verwendung: #OPTIONAL
	Liste der unterstützten MIME-Typen für den Datei-Upload.
accesskey	Anwendung bei: A, AREA, BUTTON, INPUT, LABEL, LEGEND, TEXTAREA
	Typ: %Character;
	Verwendung: #OPTIONAL
	Zeichen für den Zugriff auf ein Element (Hotkey).
action	Anwendung bei: FORM
	Typ: %URL;
	Verwendung: #GEFORDERT
	Server-seitige Formularbehandlung
align	Anwendung bei: CAPTION
	Typ: %CAlign;
	Verwendung: #OPTIONAL
	Deprecated
	Ausrichtung einer Beschriftung relativ zu einer Tabelle.
align	Anwendung bei: APPLET, IFRAME, IMG, INPUT, OBJECT
	Typ: %IAlign;
	Verwendung: #OPTIONAL
	Deprecated
	Vertikale oder horizontale Ausrichtung der Position eines Objektes in einer Webseite.
align	Anwendung bei: LEGEND
	Typ: %LAlign;
	Verwendung: #OPTIONAL
	Deprecated
	Relative Angabe zu einem Fieldset.

align	Anwendung bei: TABLE
	Typ: %TAlign;
	Verwendung: #OPTIONAL
	Deprecated
	Position einer Tabelle relativ zum Anzeigefenster im Browser.
align	Anwendung bei: HR
	Typ: (left center right)
	Verwendung: #OPTIONAL
	Deprecated
	Ausrichtung der Überschrift.
align	Anwendung bei: DIV, H1 - H6, P
	Typ: (left center right justify)
	Verwendung: #OPTIONAL
	Deprecated
	Ausrichtung von Elementen.
align	Anwendung bei: COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR
	Typ: (left center right justify char)
	Verwendung: #OPTIONAL
	Ausrichtungsangaben bei einer Tabelle.
alink	Anwendung bei: BODY
	Typ: %Color;
	Verwendung: #OPTIONAL
	Deprecated
	Farbe eines besuchten Links.
alt	Anwendung bei: APPLET
	Typ: %Text;
	Verwendung: #OPTIONAL
	Deprecated
	Alternative Kurzbeschreibung.
alt	Anwendung bei: AREA, IMG
	Typ: %Text;
	Verwendung: #GEFORDERT
	Alternative Kurzbeschreibung.

alt	Anwendung bei: INPUT
	Typ: CDATA
	Verwendung: #OPTIONAL
	Alternative Kurzbeschreibung.
archive	Anwendung bei: OBJECT
	Typ: %URL;
	Verwendung: #OPTIONAL
	Durch Leerzeichen getrennte Archivliste.
archive	Anwendung bei: APPLET
	Typ: CDATA
	Verwendung: #OPTIONAL
	Deprecated
	Durch Komma getrennte Archivliste.
axis	Anwendung bei: TD, TH
	Typ: CDATA
	Verwendung: #OPTIONAL
	Namengruppen von verbundenen Kopfzeilen.
background	Anwendung bei: BODY
	Typ: %URL;
	Verwendung: #OPTIONAL
	Deprecated
	Hintergrund einer Webseite.
bgcolor	Anwendung bei: TABLE
	Typ: %Color;
	Verwendung: #OPTIONAL
	Deprecated
	Hintergrundfarbe für alle Zellen einer Tabelle.
bgcolor	Anwendung bei: TR
	Typ: %Color;
	Verwendung: #OPTIONAL
	Deprecated
	Hintergrundfarbe für eine Zeile einer Tabelle.
bgcolor	Anwendung bei: TD, TH
	Typ: %Color;
	Verwendung: #OPTIONAL

	Deprecated
	Hintergrundfarbe für genau diese Zelle einer Tabelle.
bgcolor	Anwendung bei: BODY
	Typ: %Color;
	Verwendung: #OPTIONAL
	Deprecated
	Hintergrundfarbe des Dokumentes.
border	Anwendung bei: IMG, OBJECT
	Typ: %Length;
	Verwendung: #OPTIONAL
	Deprecated
	Wenn ein Bild oder ein anderes Objekt als Link fungiert, die Angabe, ob es umrandet dargestellt wird.
border	Anwendung bei: TABLE
	Typ: %Pixels;
	Verwendung: #OPTIONAL
	Die Angabe, wie dick der Rahmen um eine Tabelle ist.
cellpadding	Anwendung bei: TABLE
	Typ: %Length;
	Verwendung: #OPTIONAL
	Platzangaben innerhalb Zellen bei einer Tabelle.
cellspacing	Anwendung bei: TABLE
	Typ: %Length;
	Verwendung: #OPTIONAL
	Raum zwischen Zellen bei einer Tabelle.
char	Anwendung bei: COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR
	Typ: %Character;
	Verwendung: #OPTIONAL
	Anordnungszeichen, z.B. char=':'.
charoff	Anwendung bei: COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR
	Typ: %Length;
	Verwendung: #OPTIONAL
	Offset für das Anordnungszeichen.
charset	Anwendung bei: A, LINK, SCRIPT

	Typ: %Charset;
	Verwendung: #OPTIONAL
	Zeichenverschlüsselung von der verbundenen Ressource.
checked	Anwendung bei: INPUT
	Typ: (checked)
	Verwendung: #OPTIONAL
	Angabe, ob ausgewählt oder nicht, und zwar bei Radiobuttons und Checkboxes innerhalb eines Formulars.
cite	Anwendung bei: BLOCKQUOTE, Q
	Typ: %URL;
	Verwendung: #OPTIONAL
	URL für ein Dokument mit Quelltext oder eine Mitteilung.
cite	Anwendung bei: DEL, INS
	Typ: %URL;
	Verwendung: #OPTIONAL
	Eine allgemeine Information über den Grund für den Wechsel der letzten Dokumentenversion.
class	Anwendung bei allen Tags bis auf: BASE, BASEFONT, HEAD, HTML, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: CDATA
	Verwendung: #OPTIONAL
	Durch Leerraum getrennte Liste von Klassen.
classid	Anwendung bei: OBJECT
	Typ: %URL;
	Verwendung: #OPTIONAL
	Identifiziert eine Implementation.
clear	Anwendung bei: BR
	Typ: (left all right none)
	Default: none
	Deprecated
	Kontrolle des Textflusses.
code	Anwendung bei: APPLET
	Typ: CDATA
	Verwendung: #OPTIONAL
	Deprecated
	Die Klassendatei eines Applets.

codebase	Anwendung bei: OBJECT
	Typ: %URL;
	Verwendung: #OPTIONAL
	Die Angabe, wo sich ein Objekt befindet.
codebase	Anwendung bei: APPLET
	Typ: %URL;
	Verwendung: #OPTIONAL
	Deprecated
	Die Angabe, wo sich ein Applet befindet
codetype	Anwendung bei: OBJECT
	Typ: %ContentType;
	Verwendung: #OPTIONAL
	Typ des Codes.
color	Anwendung bei: BASEFONT, FONT
	Typ: %Color;
	Verwendung: #OPTIONAL
	Deprecated
	Textfarbe.
cols	Anwendung bei: FRAMESET
	Typ: %MultiLengths;
	Verwendung: #OPTIONAL
	Eine Liste von Längenangaben bei einem Frameset.
cols	Anwendung bei: TEXTAREA
	Typ: NUMBER
	Verwendung: #GEFORDERT
	Anzahl von Spalten.
colspan	Anwendung bei: TD, TH
	Typ: NUMBER
	Verwendung: Angabe größer 0
	Anzahl von Spalten, welche zu einem Colspan zusammengefasst werden.
compact	Anwendung bei: DIR, MENU
	Typ: (compact)
	Verwendung: #OPTIONAL
	Deprecated
	Reduziert den Platz zwischen Zeichen.

compact	Anwendung bei: DL, OL, UL
	Typ: (compact)
	Verwendung: #OPTIONAL
	Deprecated
	Reduziert den Platz zwischen Zeichen.
content	Anwendung bei: META
	Typ: CDATA
	Verwendung: #GEFORDERT
	Mit dem <META>-Tag assoziierte Information.
coords	Anwendung bei: AREA
	Typ: %Coords;
	Verwendung: #OPTIONAL
	Durch Komma getrennte Liste von Koordinatenangaben.
coords	Anwendung bei: A
	Typ: %Coords;
	Verwendung: #OPTIONAL
	Durch Komma getrennte Liste von Koordinatenangaben, welche bei client-seitigen Imagemaps verwendet werden.
data	Anwendung bei: OBJECT
	Typ: %URL;
	Verwendung: #OPTIONAL
	Referenz auf die Objektdaten.
datetime	Anwendung bei: DEL, INS
	Typ: %Datetime;
	Verwendung: #OPTIONAL
	Datum und Uhrzeit einer Veränderung.
declare	Anwendung bei: OBJECT
	Typ: (declare)
	Verwendung: #OPTIONAL
	Möglichkeit, ein Objekt zu deklarieren, aber noch nicht zu instanzieren.
defer	Anwendung bei: SCRIPT
	Typ: (defer)
	Verwendung: #OPTIONAL
	Hinausschieben der Ausführung eines Scripts.
dir	Anwendung bei allen Tags bis auf: APPLET, BASE, BASEFONT, BDO, BR, FRAME, FRAMESET, HR, IFRAME, PARAM, SCRIPT

	Typ: (ltr rtl)
	Verwendung: #OPTIONAL
	Fließrichtung für gewöhnlichen Text (von rechts nach links oder links nach rechts).
dir	Anwendung bei: BDO
	Typ: (ltr rtl)
	Verwendung: #GEFORDERT
	Fließrichtung für gewöhnlichen Text (von rechts nach links oder links nach rechts).
disabled	Anwendung bei: BUTTON, INPUT, OPTGROUP, OPTION, SELECT, TEXTAREA
	Typ: (disabled)
	Verwendung: #OPTIONAL
	Die Angabe, dass das Element in diesem Kontext nicht aktivierbar ist.
enctype	Anwendung bei: FORM
	Typ: %ContentType;
	Verwendung: »application/x-www- form-urlencoded"
face	Anwendung bei: BASEFONT, FONT
	Typ: CDATA
	Verwendung: #OPTIONAL
	Deprecated
	Durch Komma getrennte Liste von Schriftnamen.
for	Anwendung bei: LABEL
	Typ: IDREF
	Verwendung: #OPTIONAL
	Der ID-Wert eines auszuwählenden Feldes.
frame	Anwendung bei: TABLE
	Typ: %TFrame;
	Verwendung: #OPTIONAL
	Angabe, welcher Teil eines Frame ausgegeben werden soll.
frameborder	Anwendung bei: FRAME, IFRAME
	Typ: (1 0)
	Gibt an, ob ein Frame einen Rahmen hat oder nicht.
headers	Anwendung bei: TD, TH
	Typ: IDREFS
	Verwendung: #OPTIONAL

	Liste von IDs von Kopfzellen.
height	Anwendung bei: IFRAME
	Typ: %Length;
	Verwendung: #OPTIONAL
	Framehöhe.
height	Anwendung bei: IMG, OBJECT
	Typ: %Length;
	Verwendung: #OPTIONAL
	Explizite Höhenangabe für ein Bild oder Objekt. Überschreibt die natürliche Höhe.
height	Anwendung bei: APPLET
	Typ: %Length;
	Verwendung: #GEFORDERT
	Deprecated
	Initialisierungshöhe eines Applets.
height	Anwendung bei: TD, TH
	Typ: %Pixels;
	Verwendung: #OPTIONAL
	Deprecated
	Höhe einer Zelle.
href	Anwendung bei: A, AREA, LINK
	Typ: %URL;
	Verwendung: #OPTIONAL
	URL zu der gelinkten Ressource.
href	Anwendung bei: BASE
	Typ: %URL;
	Verwendung: #OPTIONAL
	URL, der als Basis-URL fungiert.
hreflang	Anwendung bei: A, LINK
	Typ: %LanguageCode;
	Verwendung: #OPTIONAL
	Sprachcode.
hspace	Anwendung bei: APPLET, IMG, OBJECT
	Typ: %Pixels;
	Verwendung: #OPTIONAL
	Deprecated

	Horizontale Abstandsangabe.
http-equiv	Anwendung bei: META
	Typ: NAME
	Verwendung: #OPTIONAL
	HTTP-Antwort für den Header-Namen.
id	Anwendung bei allen Tags bis auf: BASE, HEAD, HTML, META, SCRIPT, STYLE, TITLE
ID	Typ: NAME
	Verwendung: #OPTIONAL
	Dokumentenweit eindeutige ID.
ismap	Anwendung bei: IMG
	Typ: (ismap)
	Verwendung: #OPTIONAL
	Verwendung des Bildes als Server-seitige Imagemap.
label	Anwendung bei: OPTION
	Typ: %Text;
	Verwendung: #OPTIONAL
	Eine Beschriftung für die Verwendung in hierarchischen Menüs.
label	Anwendung bei: OPTGROUP
	Typ: %Text;
	Verwendung: #GEFORDERT
	Eine Beschriftung für die Verwendung in hierarchischen Menüs.
lang	Anwendung bei allen Tags bis auf: APPLET, BASE, BASEFONT, BR, FRAME, FRAMESET, HR, IFRAME, PARAM, SCRIPT
	Typ: %LanguageCode;
	Verwendung: #OPTIONAL
	Typ: Sprachcode.
language	Anwendung bei: SCRIPT
	Typ: CDATA
	Verwendung: #OPTIONAL
	Deprecated
	Name der Scriptsprache.
link	Anwendung bei: BODY
	Typ: %Color;
	Verwendung: #OPTIONAL
	Deprecated

	Farbe der Links.
longdesc	Anwendung bei: IMG
	Typ: %URL;
	Verwendung: #OPTIONAL
	Link zu einer langen Beschreibung.
longdesc	Anwendung bei: FRAME, IFRAME
	Typ: %URL;
	Verwendung: #OPTIONAL
	Link zu einer langen Beschreibung.
marginheight	Anwendung bei: FRAME, IFRAME
	Typ: %Pixels;
	Verwendung: #OPTIONAL
	Randhöhe in Pixel.
marginwidth	Anwendung bei: FRAME, IFRAME
	Typ: %Pixels;
	Verwendung: #OPTIONAL
	Randbreite in Pixel.
maxlength	Anwendung bei: INPUT
	Typ: NUMBER
	Verwendung: #OPTIONAL
	Maximale Anzahl von Zeichen in einem Textfeld.
media	Anwendung bei: STYLE
	Typ: %MediaDesc;
	Verwendung: #OPTIONAL
	Angabe des festgelegten Medientyps.
media	Anwendung bei: LINK
	Typ: %MediaDesc;
	Verwendung: #OPTIONAL
	Angabe des festgelegten Medientyps für Rendering auf diesem Mediaelement.
method	Anwendung bei: FORM
	Typ: (GET POST)
	Verwendung: #OPTIONAL
	Default-Einstellung: GET
	HTTP-Methode zur Anwendung bei der Übertragung eines Formulars.
multiple	Anwendung bei: SELECT

	Typ: (multiple)
	Verwendung: #OPTIONAL
	Die Angabe, ob eine Mehrfachauswahl bei einer Selektionsliste in einem Formular erlaubt ist. Default-Einstellung ist Einfachauswahl.
name	Anwendung bei: BUTTON, TEXTAREA
	Typ: CDATA
	Verwendung: #OPTIONAL
	Der Name eines Elementes. Es kann bei einer Zuweisung über diesen Namen von deren Strukturen (etwa Scripten) angesprochen werden.
name	Anwendung bei: APPLET
	Typ: CDATA
	Verwendung: #OPTIONAL
	Deprecated
	Name eines Applets. Applets können sich über den Namen identifizieren.
name	Anwendung bei: SELECT
	Typ: CDATA
	Verwendung: #OPTIONAL
	Feldname.
name	Anwendung bei: FRAME, IFRAME
	Typ: CDATA
	Verwendung: #OPTIONAL
	Name eines Frames.
name	Anwendung bei: A
	Typ: CDATA
	Verwendung: #OPTIONAL
	Benennung eines Links.
name	Anwendung bei: INPUT, OBJECT
	Typ: CDATA
	Verwendung: #OPTIONAL
	Name im Rahmen eines Formulars.
name	Anwendung bei: MAP
	Typ: CDATA
	Verwendung: #GEFORDERT
	Name zur Referenzierung durch eine Usemap.
name	Anwendung bei: PARAM
	Typ: CDATA

	Verwendung: #GEFORDERT
	Eigenschaftensname.
name	Anwendung bei: META
	Typ: NAME
	Verwendung: #OPTIONAL
	Name einer Metainformation.
nohref	Anwendung bei: AREA
	Typ: (nohref)
	Verwendung: #OPTIONAL
	Angabe, dass innerhalb diese Region keine Aktion stattfindet.
noresize	Anwendung bei: FRAME
	Typ: (noresize)
	Verwendung: #OPTIONAL
	Angabe, ob ein Anwender ein Frame in der Größe verändern darf oder nicht. Wenn das Attribut gesetzt ist, ist es nicht erlaubt.
noshade	Anwendung bei: HR
	Typ: (noshade)
	Verwendung: #OPTIONAL
	Deprecated
	Angabe, ob eine Linie mit einem Schaden dargestellt wird oder nicht (Flag verwendet).
nowrap	Anwendung bei: TD, TH
	Typ: (nowrap)
	Verwendung: #OPTIONAL
	Deprecated
	Unterdrückung von Wortummantelung.
object	Zugeordnete HTML-Befehle: APPLET
	Typ: CDATA
	Verwendung: #OPTIONAL
	Deprecated
	Serialisiert ein Applet.
onblur	Anwendung bei: A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA
	Typ: %Script;
	Verwendung: #OPTIONAL

	Event-Handler. Zuordnung einer Aktion, die stattfindet, wenn das Element den Fokus verloren hat.
onchange	Anwendung bei: INPUT, SELECT, TEXTAREA
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Zuordnung einer Aktion, die stattfindet, wenn der Wert des Elementes sich ändert.
onclick	Anwendung bei allen Tags bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Ein Button wurde geklickt.
ondblclick	Anwendung bei allen Tags bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Ein Button wurde doppelt geklickt.
onfocus	Anwendung bei: A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Ein Element bekommt den Fokus.
onkeydown	Anwendung bei allen Tags bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Eine Taste wurde gedrückt.
onkeypress	Anwendung bei allen HTML-Elementen bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Eine Taste wurde gedrückt und wieder losgelassen.
onkeyup	Anwendung bei allen HTML-Elementen bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE

	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Eine Taste wurde losgelassen.
onload	Anwendung bei: FRAMESET
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Alle Frames sind geladen worden.
onload	Anwendung bei: BODY
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Das Dokument ist geladen worden.
onmousedown	Anwendung bei allen HTML-Elementen bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Ein Button wurde nieder gedrückt.
onmousemove	Anwendung bei allen HTML-Elementen bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Ein Mauszeiger wurde über einem Element bewegt.
onmouseout	Anwendung bei allen HTML-Elementen bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Ein Mauszeiger hat den Bereich eines Elementes verlassen.
onmouseover	Anwendung bei allen HTML-Elementen bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Ein Mauszeiger wurde in den Bereich eines Elementes hinein bewegt.

onmouseup	Anwendung bei allen HTML-Elementen bis auf: APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Ein Mauszeiger wurde losgelassen.
onreset	Anwendung bei: FORM
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler: Das Formular wurde über den Reset -Button gelöscht.
onselect	Anwendung bei: INPUT, TEXTAREA
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler: Text wurde selektiert.
onsubmit	Anwendung bei: FORM
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Das Formular wurde gesendet.
onunload	Anwendung bei: FRAMESET
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Alle Frames wurden wieder entfernt.
onunload	Anwendung bei: BODY
	Typ: %Script;
	Verwendung: #OPTIONAL
	Event-Handler. Das Dokument wurde wieder entfernt.
profile	Anwendung bei: HEAD
	Typ: %URL;
	Verwendung: #OPTIONAL
	Benennung des Verzeichnisses für eine Metainformation.
prompt	Anwendung bei: ISINDEX
	Typ: %Text;
	Verwendung: #OPTIONAL
	Deprecated
	Die Prompt-Anzeige.
readonly	Anwendung bei: TEXTAREA

	Typ: (readonly)
	Verwendung: #OPTIONAL
	Nur Leseerlaubnis.
readonly	Anwendung bei: INPUT
	Typ: (readonly)
	Verwendung: #OPTIONAL
	Nur Leseerlaubnis.
rel	Anwendung bei: A, LINK
	Typ: %LinkTypes;
	Verwendung: #OPTIONAL
	Vorwärtslinkangaben.
rev	Anwendung bei: A, LINK
	Typ: %LinkTypes;
	Verwendung: #OPTIONAL
	Rückwärtslinkangaben.
rows	Anwendung bei: FRAMESET
	Typ: %MultiLengths;
	Verwendung: #OPTIONAL
	Liste von Längenangaben (Anzahl Reihen).
rows	Anwendung bei: TEXTAREA
	Typ: NUMBER
	Verwendung: #GEFORDERT
	Anzahl der Reihen.
rowspan	Anwendung bei: TD, TH
	Typ: NUMBER
	Verwendung: Angabe größer 0
	Anzahl der Reihen, die von einer Zelle überspannt werden sollen.
rules	Anwendung bei: TABLE
	Typ: %TRules;
	Verwendung: #OPTIONAL
	Verhaltensregeln zwischen Zeilen und Spalten in einer Tabelle.
scheme	Anwendung bei: META
	Typ: CDATA
	Verwendung: #OPTIONAL
	Selektiert einen Inhalt.

scope	Anwendung bei: TD, TH
	Typ: %Scope;
	Verwendung: #OPTIONAL
	Wiederholungsbereich der Kopfzellen.
scrolling	Anwendung bei: FRAME, IFRAME
	Typ: (yes no auto)
	Default-Einstellung: auto
	Scrollbalken oder nicht.
selected	Anwendung bei: OPTION
	Typ: (selected)
	Verwendung: #OPTIONAL
	Selektiert.
shape	Anwendung bei: AREA
	Typ: %Shape;
	Default-Einstellung: rect
	Koordinatenangaben.
shape	Anwendung bei: A
	Typ: %Shape;
	Default-Einstellung: rect
	Koordinatenangaben in einer Client-seitigen Imagemap.
size	Anwendung bei: HR
	Typ: %Pixels;
	Verwendung: #OPTIONAL
	Deprecated
	Größenangabe einer Trennlinie.
size	Anwendung bei: FONT
	Typ: CDATA
	Verwendung: #OPTIONAL
	Deprecated
	Größenangabe für eine Schrift. Immer relative Angabe zur Basisschriftgröße. Entweder relativ mit Plus oder Minus von der Basisschriftgröße [+ -]nn - z.B. size="+3" - oder genaue Angabe (genau genommen auch eine Relativangabe) zwischen 1 und 7, z.B. size="6"
size	Anwendung bei: INPUT
	Typ: CDATA
	Verwendung: #OPTIONAL

	Spezifiziert jeden Typ eines Feldes bezüglich seiner Größe.
size	Anwendung bei: BASEFONT
	Typ: CDATA
	Verwendung: #GEFORDERT
	Deprecated
	Basisschriftgröße für jedes Schriftelement.
size	Anwendung bei: SELECT
	Typ: NUMBER
	Verwendung: #OPTIONAL
	Sichtbare Einträge in einer Auswahlliste.
span	Anwendung bei: COL
	Typ: NUMBER
	Verwendung: Angabe größer 0
	Die COL-Attribute sollen sich auf die angegebene Anzahl von Spalten auswirken.
span	Anwendung bei: COLGROUP
	Typ: NUMBER
	Verwendung: Angabe größer 0
	Default-Anzahl von Spalten in einer Colgroup.
src	Anwendung bei: SCRIPT
	Typ: %URL;
	Verwendung: #OPTIONAL
	URL für ein externes Script.
src	Anwendung bei: INPUT
	Typ: %URL;
	Verwendung: #OPTIONAL
	URL für Felder mit Bildern.
src	Anwendung bei: FRAME, IFRAME
	Typ: %URL;
	Verwendung: #OPTIONAL
	Source des Inhaltes eines Frames.
src	Anwendung bei: IMG
	Typ: %URL;
	Verwendung: #GEFORDERT
	URL des Bildes, das eingebettet werden soll.
standby	Anwendung bei: OBJECT

	Typ: %Text;
	Verwendung: #OPTIONAL
	Die Nachricht, die während des Ladevorgangs eines Objektes angezeigt werden soll.
start	Anwendung bei: OL
	Typ: NUMBER
	Verwendung: #OPTIONAL
	Deprecated
	Startnummer einer Aufzählung.
style	Anwendung bei allen HTML-Elementen bis auf: BASE, BASEFONT, HEAD, HTML, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %StyleSheet;
	Verwendung: #OPTIONAL
	Assoziierte Stilinformation.
summary	Anwendung bei: TABLE
	Typ: %Text;
	Verwendung: #OPTIONAL
	Strukturangaben für die Sprachausgabe.
tabindex	Anwendung bei: A, AREA, BUTTON, INPUT, OBJECT, SELECT, TEXTAREA
	Typ: NUMBER
	Verwendung: #OPTIONAL
	Der Tabulatorindex.
target	Anwendung bei: A, AREA, BASE, FORM, LINK
	Typ: %FrameTarget;
	Verwendung: #OPTIONAL
	Die Angabe, wo in dem Frame eine Anzeige erfolgen soll.
text	Anwendung bei: BODY
	Typ: %Color;
	Verwendung: #OPTIONAL
	Deprecated
	Globale Textfarbe im Dokument.
title	Anwendung bei: STYLE
	Typ: %Text;
	Verwendung: #OPTIONAL

	Erklärender Titel. Es sollte nicht mit dem TITLE-Element verwechselt werden, welches Bestandteil des HEADER-Teils einer Webseite ist.
title	Anwendung bei allen HTML-Elementen bis auf: BASE, BASEFONT, HEAD, HTML, META, PARAM, SCRIPT, STYLE, TITLE
	Typ: %Text;
	Verwendung: #OPTIONAL
	Erklärender Titel für das Element, auf das das Attribut angewandt wird (Quickinfo). Es sollte nicht mit dem TITLE-Element verwechselt werden, welches Bestandteil des HEADER-Teils einer Webseite ist.
type	Anwendung bei: A, LINK
	Typ: %ContentType;
	Verwendung: #OPTIONAL
	Erklärender Inhaltstyp.
type	Anwendung bei: OBJECT
	Typ: %ContentType;
	Verwendung: #OPTIONAL
	Inhaltstyp der Daten.
type	Anwendung bei: PARAM
	Typ: %ContentType;
	Verwendung: #OPTIONAL
	Inhaltstyp für Werte, wenn gilt valuetype=ref.
type	Anwendung bei: SCRIPT
	Typ: %ContentType;
	Verwendung: #GEFORDERT
	Typ des Inhalts von der Script-Sprache.
type	Anwendung bei: STYLE
	Typ: %ContentType;
	Verwendung: #GEFORDERT
	Typ des Inhalts von der Style-Sprache.
type	Anwendung bei: INPUT
	Typ: %InputType;
	Default: TEXT
	Welche Art von Eingabe wird gefordert.
type	Anwendung bei: LI
	Typ: %LIStyle;
	Verwendung: #OPTIONAL

	Deprecated
	Stil der Aufzählungsliste.
type	Anwendung bei: OL
	Typ: %OLStyle;
	Verwendung: #OPTIONAL
	Deprecated
	Stil der Nummerierung.
type	Anwendung bei: UL
	Typ: %ULStyle;
	Verwendung: #OPTIONAL
	Deprecated
	Bulletstil.
type	Anwendung bei: BUTTON
	Typ: (button submit reset)
	Default: submit
	Typ der Schaltfläche bei der Verwendung als Formular-Button.
usemap	Anwendung bei: IMG, INPUT, OBJECT
	Typ: %URL;
	Verwendung: #OPTIONAL
	Verwendung als Client-seitige Imagemap.
valign	Anwendung bei: COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR
	Typ: (top middle bottom baseline)
	Verwendung: #OPTIONAL
	Vertikale Ausrichtung in Zellen.
value	Anwendung bei: OPTION
	Typ: CDATA
	Verwendung: #OPTIONAL
	Defaultwert eines Elementinhaltes.
value	Anwendung bei: PARAM
	Typ: CDATA
	Verwendung: #OPTIONAL
	Eigenschaftenwert.
value	Anwendung bei: INPUT
	Typ: CDATA
	Verwendung: #OPTIONAL

	Wert bei Radioboxen und Checkboxes.
value	Anwendung bei: BUTTON
	Typ: CDATA
	Verwendung: #OPTIONAL
	Wird bei der Übermittlung zum Server gesendet.
value	Anwendung bei: LI
	Typ: NUMBER
	Verwendung: #OPTIONAL
	Deprecated
	Setzt eine Aufzählungsnummer.
valuetype	Anwendung bei: PARAM
	Typ: (DATA REF OBJECT)
	Default: DATA
	Beschreibung, wie ein Wert zu interpretieren ist.
version	Anwendung bei: HTML
	Typ: CDATA
	Typ: %HTML.Version;
	Deprecated
	Verwendung: Konstante.
vlink	Anwendung bei: BODY
	Typ: %Color;
	Verwendung: #OPTIONAL
	Deprecated
	Farbe von bereits besuchten Links.
vspace	Anwendung bei: APPLET, IMG, OBJECT
	Typ: %Pixels;
	Verwendung: #OPTIONAL
	Deprecated
	Vertikale Abstandsangabe.
width	Anwendung bei: HR
	Typ: %Length;
	Verwendung: #OPTIONAL
	Deprecated
	Breite der Trennlinie.
width	Anwendung bei: IFRAME

	Typ: %Length;
	Verwendung: #OPTIONAL
	Breite des Frame.
width	Anwendung bei: IMG, OBJECT
	Typ: %Length;
	Verwendung: #OPTIONAL
	Breitenangabe.
width	Anwendung bei: TABLE
	Typ: %Length;
	Verwendung: #OPTIONAL
	Breite der Tabelle.
width	Anwendung bei: APPLET
	Typ: %Length;
	Verwendung: #GEFORDERT
	Deprecated
	Initialisierungsbreite eines Applets.
width	Anwendung bei: COL
	Typ: %MultiLength;
	Verwendung: #OPTIONAL
	Spaltenbreite.
width	Anwendung bei: COLGROUP
	Typ: %MultiLength;
	Verwendung: #OPTIONAL
	Defaultbreite für eingeschlossene Spalten.
width	Anwendung bei: TD, TH
	Typ: %Pixels;
	Verwendung: #OPTIONAL
	Deprecated
	Breite für Zellen.
width	Anwendung bei: PRE
	Typ: NUMBER
	Verwendung: #OPTIONAL
	Deprecated
	Breitenangaben.

Tabelle A.10: HTML-Attribute

Tabelle mit Maskierungen

Die nachfolgende Tabelle gibt die Codes der ISO-5589-1-Norm und ihre zugehörige Maskierung im Rahmen von HTML an. Die Tabelle beginnt erst mit dem Zeichen 32; die fehlenden Zeichen sind reine Steuerzeichen. Die ersten Zeichen bis 127 sind identisch mit dem klassischen ASCII-Code. In der ersten Spalte steht die Codenummer. Aus dieser ergibt sich die numerische Zeichenreferenz (numerical character reference) durch Voranstellen der beiden Zeichen #& und Abschluss mit einem Semikolon (z.B. ! für !). Die zweite Spalte zeigt das dargestellte Zeichen, und - falls eine benannte Zeichenreferenz (character entity reference) definiert ist - steht in der dritten Spalte die entsprechende Codierung (z.B. & für &).

Nummerncode	Zeichen	Character-Entity
32	SP	
33	!	
34	"	"
35	#	
36	\$	
37	%	
38	&	&
39	'	
40	(	
41	)	
42	*	
43	+	
44	,	
45	-	
46	.	
47	/	
48	0	
49	1	
50	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	

58	:	
59	;	
60	<	>
61	=	
62	>	<
63	?	
64	@	
65	A	
66	B	
67	C	
68	D	
69	E	
70	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
80	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
90	Z	
91	[	
92	\	

93	]	
94	^	
95	_	
96	`	
97	a	
98	b	
99	c	
100	d	
101	e	
102	f	
103	g	
104	h	
105	i	
106	j	
107	k	
108	l	
109	m	
110	n	
111	o	
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
120	x	
121	y	
122	z	
123	{	
124		
125	}	
126	~	
127	DEL	

128	—	
129	—	
130		
131		
132	"	
133	...	
134		
135		
136	^	
137		
138		
139		
140		
141	—	
142	—	
143	—	
144	—	
145	'	
146	'	
147	"	
148	"	
149	o	
150	-	
151	-	
152	~	
153	TM	
154		
155		
156		
157	—	
158	—	
159	\xd9	
160	(erzwungenes Leerzeichen)	
161	i	¡
162	¢	¢

163	£	£
164	¤	¤
165	¥	¥
166	¸	¦
167	§	§
168	¨	¨
169	©	©
170	ª	ª
171	"	«
172	¬	¬
173	-	­
174	®	®
175	—	¯
176	°	°
177	±	±
178	²	²
179	³	³
180	´	´
181	µ	µ
182	¶	¶
183	·	·
184	¸	¸
185	¹	¹
186	º	º
187	"	»
188	¼	¼
189	½	½
190	¾	¾
191	¿	¿
192	À	À
193	Á	Á
194	Â	Â
195	Ã	Ã
196	Ä	Ä
197	Å	Å

198	Æ	Æ
199	Ç	Ç
200	È	È
201	É	É
202	Ê	Ê
203	Ë	Ë
204	Ì	Ì
205	Í	Í
206	Î	Î
207	Ï	Ï
208	Ð	Ð
209	Ñ	Ñ
210	Ò	Ò
211	Ó	Ó
212	Ô	Ô
213	Õ	Õ
214	Ö	Ö
215	×	×
216	Ø	Ø
217	Ù	Ù
218	Ú	Ú
219	Û	Û
220	Ü	Ü
221	Ý	Ý
222	Þ	Þ
223	ß	ß
224	à	à
225	á	á
226	â	â
227	ã	ã
228	ä	ä
229	å	å
230	æ	æ
231	ç	ç
232	è	è

233	é	é
234	ê	ê
235	ë	ë
236	ì	ì
237	í	í
238	î	î
239	ï	ï
240	ð	ð
241	ñ	ñ
242	ò	ò
243	ó	ó
244	ô	ô
245	õ	õ
246	ö	ö
247	÷	÷
248	ø	ø
249	ù	ù
250	ú	ú
251	û	û
252	ü	ü
253	ý	ý
254	þ	þ
255	ÿ	ÿ

Tabelle A.11: Maskierungen

Umrechnungstabelle zwischen hexadezimalen und dezimalen Zahlen

Dezimal	Hexadezimal
1	1
2	2
3	3
4	4
5	5
6	6

7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
20	14
30	1E
40	28
50	32
60	3C
70	46
80	50
90	5A
100	64
200	C8
300	12C
400	190
500	1F4
600	258
700	2BC
800	320
900	384
1000	3E8
10000	2710
100000	186A0
1000000	F4240

Tabelle A.12: Umrechnungstabelle hexadezimal - dezimal

MIME-Typen

Die nachfolgende Tabellen gibt die wichtigsten Mime-Typen mit Dateierweiterung und einer kurzen Beschreibung an.

MIME-Typ	Dateinamenerweiterung/en	Beschreibung
application/acad (NCSA)	dwg	AutoCAD-Dateien
application/dxf (CERN)	dxf	AutoCAD-Dateien
application/mif	Transfer interrupted!	

Tag B

Antworten auf die Kontrollfragen

Tag 1

1. Welches ist der meist genutzte Dienst im Internet?

Das World Wide Web.

2. Aus welcher Sprache sind Webseiten aufgebaut?

HTML.

3. Was ist ein Server?

In der Regel ein Programm, das einem Client-Programm bestimmte Funktionalitäten bereitstellt.

4. Gehören Style Sheets zu HTML?

Nein, sie sind eine ergänzende Technologie.

5. Welchen Style-Sheet-Standard unterstützen die meisten modernen Browser zumindest im Ansatz?

CSS.

6. Was ist Unicode?

Ein Klartextformat, das den ASCII-Code erweitert. Es verwendet zwei Byte zur Darstellung von jedem Zeichen.

7. Sind JScript und JavaScript identisch?

Nein, aber sehr ähnlich. JScript ist in Hinsicht auf die WINTEL- Plattform optimiert und dort etwas umfangreicher als pures JavaScript.

8. Was ist ein URL?

Eine Adresse in einem vereinheitlichten Schema.

9. Zu welcher HTML-Version gehören Frames?

HTML 4.

10. Wie alt ist das Internet?

1969 startete der Urvater, das ARPANET.

Tag 2

1. Ist XML eine Ablösung für HTML?

Nein.

2. Welche Version von JavaScript ist auf Serverseite aktuell?

1.5.

3. Ist VBScript oder JavaScript weiter verbreitet?

JavaScript.

4. Läuft VBScript im Opera-Browser?

Nein.

5. Welchen JavaScript-Standard unterstützt der Opera-Browser 3.6?

1.1.

6. Sind Java und JavaScript eng verwandt?

Nein, entgegen der Ähnlichkeit im Namen nur sehr weitläufig.

7. Wofür ist CGI die Abkürzung?

»Common Gateway Interface«, was übersetzt »Allgemeine Zugangsschnittstelle« bedeutet.

8. Welche Sprache wird hauptsächlich zur Erzeugung von Server-seitigen Scripten eingesetzt?

Perl.

Tag 3

1. Ist HTML eine Programmiersprache?

Nein. Es fehlen wichtige Elemente wie Variablen und Kontrollstrukturen.

2. Kann man auf das Grundgerüst einer Webseite verzichten?

In Ausnahmesituationen ja, aber in der Regel kann es Probleme geben, wenn das Grundgerüst fehlt.

3. Werden beim Ende-Tag Parameter angegeben?

Auf keinen Fall.

4. Muss man sich bei HTML an Groß- und Kleinschreibungsregeln halten?

Nein. Es ist nur persönlicher Geschmack oder Projektkonvention, wie man da verfährt. Allerdings steht zu erwarten, dass zukünftige Entwicklungen da nicht so tolerant bleiben.

5. Kann man mit HTML absolute Gestaltungsangaben machen?

Nein. Man kann nicht alle Eventualitäten berücksichtigen, die bei den unzähligen Konfigurationsmöglichkeiten von Besuchern einer Webseite auftreten können. Das war und ist auch nicht Sinn und Zweck von HTML. Wer so etwas versucht, hat HTML und das Web nicht verstanden.

Tag 4

1. Finden Sie den Fehler:

```
<SCRIPT language ="JavaScript 1.2">
```

Die Versionsnummer darf nicht durch Leerzeichen abgetrennt werden.

2. Finden Sie den Fehler:

```
<SCRIPT langauge="JavaScript" src="extern.js">  
</SCRIPT>
```

Das Attribut `language` beinhaltet einen Buchstabendreher.

3. Finden Sie den Fehler:

```
<A href="zufall()">Klick</A>
```

Die Inline-Referenz muss vor dem Aufruf einer Funktion die Sprache angeben, also so:

```
<A href="JavaScript:zufall()">Klick</A>
```

4. Finden Sie den Fehler:

```
<STYLE TYPE="text/ccs">
<!--
.meinStil1 {
  font-family: Arial
}
-->
</STYLE>
```

Fieser Fehler - es heißt css.

5. Finden Sie den Fehler:

```
<LINK type="text/css« href="externStil.css">
```

6. Das Attribut rel=stylesheet ist nicht optional. Es muss heißen:

```
<LINK type="text/css« rel=stylesheet href="externStil.css">
```

6. Finden Sie den Fehler:

```
<P> <style="color : gray">
```

Es muss eine einzige Anweisung sein. Die Angabe style ist ein Attribut des Tags: <P style="color : gray">

7. Finden Sie den Fehler:

```
<SCRIPT language="JavaScript" src="extern.js">
alert("Hallo");
</SCRIPT>
```

Der Container muss bei Referenzen auf externe JavaScript-Dateien leer sein.

Tag 5

1. Was bedeutet der Tag <H3>?

Überschrift der Ordnung 3.

2. Wie erzeugt man eine Trennlinie?

Mit dem <HR>-Tag.

3. Mit welchem Tag wird eine Grafik referenziert?

Mit dem -Tag.

4. Wie verhindert man einen Rahmen bei einer als Hyperlink verwendeten Grafik?

Mit dem border-Attribut.

5. Was bedeutet der Tag <I>?

Kursiv.

6. Was bedeutet der Tag ?

Fett.

7. Wie erzeugt man eine Absatzschaltung?

Mit dem <P>-Tag.

8. Was ist ein Anker?

Eine besonders markierte Stelle in einer Webseite, die man mit einem Hyperlink anspringen kann.

9. Wie kann man einen Absatz rechts ausrichten?

Mit der Anweisung <P align=right>.

Tag 6

1. Was macht ++?
Die Zeichenkette ++ ist ein Operator, der den Wert einer Variablen um eins erhöht.
2. Was macht prompt()?
Über prompt() wird beim Laden der Seite ein Benutzerdialog angezeigt, in dem der Besucher eine Eingabe vornehmen kann.
3. Was heißt if-else übersetzt?
Wenn-dann. Das beschreibt die Funktionalität der Schleife auch absolut genau.
4. Warum schreibt man überhaupt über JavaScript Webseiten dynamisch?
In vielen Situationen können bei der Erstellung von Webseiten noch nicht alle Tatsachen bekannt sein, die zum vollständigen Anzeigen aller Fakten notwendig wären. Um diese in die Webseite einarbeiten zu können, muss die Seite vollständig aus JavaScript geschrieben werden.
5. Was macht alert()?
Ausgabe einer Information in Form eines Mitteilungsfensters.
6. Kann man über JavaScript Dinge wie die Bildschirmauflösung auf der Plattform des Anwenders abfragen?
Ja.
7. Kann man über JavaScript Dinge wie die Farbauflösung auf der Plattform des Anwenders verändern?
Nein. Die Beschränkung auf unkritische Aktionen beim Verändern auf der Plattform des Anwenders macht JavaScript so ungefährlich und dessen Akzeptanz wird nicht beeinträchtigt.

Tag 7

1. Was ist ein typographischer Fehler?
Ein Vertipper beim Schreiben des Quelltexts bei einem Schlüsselwort, einer Variablen oder sonst etwas Entscheidendem.
2. Was stimmt nicht an der Variablendeklaration?
`var 5i;`
Ein Variablenbezeichner darf nicht mit einer Zahl beginnen.
3. Was stimmt nicht an der Syntax?
`function test();`
`{`
`document.write('<HTML>');`
`}`
Das Semikolon nach test() ist zu viel.
4. Was stimmt nicht an der Syntax?
`var i;j;k;l;`
Die Trennung der Variablen erfolgt mit Kommata.
5. Was stimmt nicht an der Syntax?
`var i,j,k,l`
Das Semikolon am Ende fehlt.
6. Was stimmt nicht an der Syntax?
`var i j k l;`
Die Trennung der Variablen muss mit Kommata erfolgen.
7. Was lässt sich an der nachfolgenden Syntax ohne großen Aufwand optimieren?
`function schreibWebseite()`
`{`
`document.write('<html>');`
`document.write('<head>');`
`document.write('</head>');`
`document.write('<body link="#9F9F9F« bgcolor="#FFFFFF">');`
`document.write('</body>');`

```
document.write('</html>');  
}
```

Verkürzen des Bezeichners der Funktion, Eliminieren der überflüssigen Zeilenumbrüche und Zusammenfassen der Schreibweise. Etwa so (Zeilenumbrüche und scheinbare Leerzeichen sind nur satztechnisch in dem Buch notwendig):

function

```
sWS(){document.write('<html><head></head><body link="#9F9F9F«  
bgcolor="#FFFFFF"></body></html>');}
```

Tag 8

1. Ist das eine korrekte Variablendeklaration?

```
var i, j, k = 8, z;
```

Nein. Die Wertzuweisung $k = 8$ in der Mitte kann so nicht sein.

2. }

Nein Ein Bezeichner einer Funktion kann nicht mit einer Zahl beginnen "> Stimmt die Deklaration der Funktion?

```
function 2mal()
```

```
{  
alert("Guckmal");  
}
```

Nein. Ein Bezeichner einer Funktion kann nicht mit einer Zahl beginnen.

3. }

Nein Das Semikolon hinter den Klammern ist falsch "> Stimmt die Deklaration der Funktion?

```
function test();  
{  
alert("Guckmal");  
}
```

Nein. Das Semikolon hinter den Klammern ist falsch.

4. Was passt bei dem Funktions-Aufruf nicht?

```
test
```

Jede Funktion oder Methode muss ein Klammernpaar haben.

5. Kann das stimmen?

```
function test()  
{  
return;  
}
```

Nein. Was wird zurückgegeben? Nach return muss ein Wert angegeben werden, der zurückgegeben wird.

Tag 9

1. Betrachten Sie die folgende Anweisung:

```
a + b / c - d
```

In welcher Reihenfolge werden die einzelnen Schritte durchgeführt?

Es gilt Punkt-vor-Strich-Rechnung. Deshalb wird als Erstes b geteilt durch c durchgeführt und dieser Wert dann für die Addition bzw. Subtraktion verwendet. Die Reihenfolge der Abarbeitung hier ist egal, da die Operationen gleiche Priorität haben. Da aber bei gleicher Wertigkeit von Operatoren in einer Anweisungszeile die Bewertung von links nach rechts erfolgt, wird zuerst addiert und dann subtrahiert.

2. Was bedeutet der Operator `||` ?

Logisches Oder.

3. Was bedeutet der Operator `!=` ?

Vergleichsoperator auf Ungleich.

4. }

Ja Auch wenn Sie jetzt stutzen (das sollten Sie tun!) Das Script ist syntaktisch richtig Es wird nur - sehr wahrscheinlich - nicht das tun, was geplant ist Das einfache Gleichheitszeichen ist der Zuweisungsoperator und nicht der normalerweise an dieser Stelle verwendete Vergleichsoperator Es wird dennoch keine Fehlermeldung durch den Interpreter geben Was passiert, ist, dass der Variablen a der Wert 5 erfolgreich zugewiesen wird Die Aktion wird also als wahr bewertet Das wiederum bedeutet, die Bedingung ist immer erfüllt Egal ob a vorher den Wert hatte oder sogar, ob es vorher eine Variable a überhaupt gab Testen Sie es aus So etwas ist einer der tückischsten Fehler, die bei der Verwechslung von = und == auftreten können "> Ist das Script richtig?

```
if(a=5)
{
  alert("Test");
}
```

Ja. Auch wenn Sie jetzt stutzen (das sollten Sie tun!). Das Script ist syntaktisch richtig. Es wird nur - sehr wahrscheinlich - nicht das tun, was geplant ist. Das einfache Gleichheitszeichen ist der Zuweisungsoperator und nicht der normalerweise an dieser Stelle verwendete Vergleichsoperator. Es wird dennoch keine Fehlermeldung durch den Interpreter geben. Was passiert, ist, dass der Variablen a der Wert 5 erfolgreich zugewiesen wird. Die Aktion wird also als wahr bewertet. Das wiederum bedeutet, die Bedingung ist immer erfüllt. Egal ob a vorher den Wert hatte oder sogar, ob es vorher eine Variable a überhaupt gab. Testen Sie es aus. So etwas ist einer der tückischsten Fehler, die bei der Verwechslung von = und == auftreten können.

5. Was bedeutet der Operator && ?

Logisches Und.

6. Ist der nachfolgende Bezeichner ein Schlüsselwort?

Else

Nein. Ein solches Schlüsselwort gibt es nicht! Widerspruch? Beachten Sie Groß- und Kleinschreibung.

7. Was bedeutet der Operator *= ?

Multiplikations-Zuweisungsoperator.

8. Ist der nachfolgende Bezeichner ein Schlüsselwort?

funktion

Nein, aber es gibt das Schlüsselwort function.

9. Was bedeutet der Operator % ?

Modulo-Operator.

10. Ist der nachfolgende Bezeichner ein Schlüsselwort?

do

Ja.

11. Ist der nachfolgende Bezeichner ein Schlüsselwort?

false

Ja.

12. Was bedeutet der Operator += ?

Additions-Zuweisungsoperator.

13. Ist der nachfolgende Bezeichner ein Schlüsselwort?

For

Nein. Beachten Sie Groß- und Kleinschreibung.

14. Ist der nachfolgende Bezeichner ein Schlüsselwort?

retrun

Nein, aber es gibt das Schlüsselwort return.

Tag 10

1. Stimmt die Syntax?

```
for(zaehler==0;zaehler<=10;zaehler++)
```

Nein, es muss der Zuweisungsoperator verwendet werden (zaehler=0).

2. Gibt es die Sprunganweisung Return?

Nein. Immer wieder: Groß- und Kleinschreibung beachten.

3. Stimmt die Syntax?

```
for(zaehler=0;zaehler<=10;zaehler)
```

Jein. Stimmen schon (von der Syntax her), aber es wird wohl nicht ganz das gemacht, was erwartet wird. Das Intervall zum Verändern der Zählvariablen ist 0 und damit wird eine Endlosschleife erzeugt. Und zwar eine ganz brutale, denn der Browser wird unter Umständen heftigst abgeschossen.

4. Stimmt die Syntax?

```
if(i<10)
```

Aber klar doch. Warum denn nicht? ;)

5. Was ist der Unterschied zwischen break und continue?

Die Anweisung break bricht eine Schleife ab und gibt die Kontrolle aus der Schleife heraus, continue startet unmittelbar einen neuen Schleifendurchlauf.

6. ¹ der Gleichheitsoperator stehen "> Stimmt die Syntax?

```
while(a=8)
```

Auch hier gilt jein. Dies ist wieder eine Endlosschleife. Hier muss wahrscheinlich¹ der Gleichheitsoperator stehen.

Tag 11

1. Ja, klar doch ;-) "> Unterstützt der Netscape Navigator diese Syntax?

```
<BODY onLoad="a()">
```

Ja, klar doch ;-).

2. Nein "> Unterstützt der Netscape Navigator diese Syntax?

```
<HR onClick="a()">
```

Nein.

3. Ja "> Unterstützt der Netscape Navigator diese Syntax?

```
<A href="test.htm" onMouseOver="a()">
```

Ja.

4. Nein "> Unterstützt der Netscape Navigator diese Syntax?

```
<BUTTON onFocus="c()" onBlur="d()">Click </BUTTON>
```

Nein.

5. Ja "> Unterstützt der Netscape Navigator diese Syntax?

```
<INPUT onChange="change()">
```

Ja.

6. Ja "> Unterstützt der Netscape Navigator diese Syntax?

```
<FORM Name="meinFormular" onReset="return wirklich()">
```

Ja.

7. Ja "> Unterstützt der Netscape Navigator diese Syntax?

```
<BODY onUnload="a()">
```

Ja.

8. Nein "> Unterstützt der Netscape Navigator diese Syntax?

```
<H1 onSelect ="a()">Willkommen</H1>
```

Nein.

Tag 12

1. Wie viele verschiedene Listenformen gibt es in HTML?

Eine Bullet-Liste und fünf nummerische (normal, zweimal römisch und zweimal mit Buchstaben).

2. Was stimmt nicht an dem Listing?

```
<UL>
```

```
<LI type="i">Erstens</LI>
```

```
<LI>Zweitens</LI>
```

```
<LI>Drittens</LI>
```


Die Typvereinbarung muss im äußeren Container erfolgen.

3. Was stimmt nicht an dem Listing?

Erstens

Zweitens

Drittens

Ende-Tag des äußeren Containers passt nicht zum Anfangs-Tag.

4. Was stimmt nicht an dem Listing?

<TABLE>

<TR>

<TH>Mon</TH>

<TH>DI </TH>

</TR>

<TD>123</TD>

<TD>234</TD>

</TR>

</TABLE>

Der Beginn des zweiten Zeilen-Containers fehlt.

5. Ab welcher HTML-Version werden Tabellen unterstützt?

HTML 3.2.

Tag 13

1. Was macht isInfinite()?

Test auf Unendlichkeit.

2. Funktioniert das?

if(a==NaN)

Nein. Es kann nie direkt auf NaN geprüft werden.

3. Ja, schon Aber es wird wahrscheinlich nicht das geprüft, was gewünscht ist Es wird einfach darauf geprüft, ob der Wert einer Variablen mit einem bestimmten String identisch ist Das ist keine Prüfung auf undefiniert ">

Funktioniert das?

if(a=="undefined")

Ja, schon. Aber es wird wahrscheinlich nicht das geprüft, was gewünscht ist. Es wird einfach darauf geprüft, ob der Wert einer Variablen mit einem bestimmten String identisch ist. Das ist keine Prüfung auf undefiniert.

4. Was liefert parseInt(45.678)?

Den Wert 45.

5. Was liefert parseInt(DM 45.67)?

Den Wert NaN.

6. Was liefert parseFloat(45.67 DM)?

Den Wert 45.67.

7. Funktioniert das?

if(a==undefined)

Nein. Es sei denn, es gibt eine Variable mit Bezeichner undefined.

8. Funktioniert das?

if((typeof a) == undefined)

Nein. Es muss auf einen String mit Inhalt undefined verglichen werden.

Tag 14

1. `<NOEMBED>>Hier läuft eigentlich ein Video</NOEMBED> ">` Was stimmt hier nicht?

`<EMBED src="video001.avi" width=300 height=300`

`<NOEMBED>>Hier läuft eigentlich ein Video.</NOEMBED>`

Der `<NOEMED>`-Tag ist fälschlicherweise in den `<EMBED>`-Tag mit hineingeschrieben worden. Er muss außerhalb stehen.

2. Ist das korrekt?

`<EMBED type="application/x-java-applet« code="MeinApplet.class« width="300« height="200">`

Ja.

3. Im Internet Explorer wird die angegebene Sound-Datei beim Laden der Webseite als Hintergrundmusik abgespielt (zweimal) Im Navigator und in kompatiblen Browsern wird die Angabe ignoriert "> Was tut folgender HTML-Tag?

`<BGSOUND src="test.wav" loop=2>`

Im Internet Explorer wird die angegebene Sound-Datei beim Laden der Webseite als Hintergrundmusik abgespielt (zweimal). Im Navigator und in kompatiblen Browsern wird die Angabe ignoriert.

4. Was muss bei der Angabe von Parametern für ein Java-Applet unbedingt beachtet werden?
Groß- und Kleinschreibung des Namens von dem Parameter und - bei Strings - u.U. auch des Wertes.

5. Versteht der Internet Explorer LiveConnect?

Nein.

6. Ist das korrekt?

`<APPLET code=ArcTest.class height=400>`

Hier sollte ein Applet stehen

`</APPLET>`

Nein. Die Angabe der Breite eines Applets ist nicht optional.

7. Was ist der Unterschied zwischen den Java-Methoden `println()` und `print()`?
Während `println()` einen Zeilenvorschub bewirkt, wird `print()` eine Ausgabe des spezifizierten Inhalts ohne Zeilenvorschub auslösen.

8. Was bewirkt folgende Syntax?

`var meinFrame = new Packages.java.awt.Frame();`

Aus JavaScript heraus wird der Konstruktor der Java-Klasse `Frame` aufgerufen, um ein Fenster-Objekt zu erzeugen, auf das man aus JavaScript zugreifen kann.

Tag 15

1. Was macht `this`?

Schlüsselwort für den Zugriff auf das Objekt, welches eine Funktionalität aufruft.

2. Ist das korrekt?

`var test = New Date();`

Nein. Das Schlüsselwort `new` wird klein geschrieben.

3. Stimmt das?

`window.status =
document.forms[0].elements.value *
document.forms[0].elements.value;`

Nein. Es muss sich bei `elements` um Objektfelder handeln. Also so etwas wie `elements[0]`.

4. Stimmt der Methodenaufruf?

`window.close;`

Nein. Ein Methodenaufruf muss (!!) immer (!!) Klammern haben.

5. Stimmt das?

```
with(document)
{
write(»Mircoweich«);
write(»<BR>«);
}
```

Ja.

6. Was ist ein Objekt?

Ein Objekt ist eine in sich abgeschlossene und eigenständige Einheit aus zusammengehörenden Anweisungen und Daten, die als Softwaremodell ein Ding aus der realen Welt mit all seinen für die Behandlung eines Problems notwendigen Eigenschaften und Verhaltensweise beschreiben soll.

7. Was ist ein Konstruktor?

Eine besondere Methode bzw. Funktion, die ein neues Objekt erzeugt, wenn man sie entsprechend anwendet.

Tag 16

1. Was ist der Unterschied zwischen frame und frames[]?

frame ist die Objektdекlaration, welche ein Frameobjekt repräsentiert, frames[] das Objektfeld mit den konkreten Objekten.

2. Ja Obwohl es kaum sinnvoll ist Es wird ein leeres Fenster geöffnet Sie können damit aber recht leicht einen Browser abschießen, wenn Sie solche Anweisungen in einer Endlosschleife abschicken "> Geht das?

```
open("", "");
```

Ja. Obwohl es kaum sinnvoll ist. Es wird ein leeres Fenster geöffnet. Sie können damit aber recht leicht einen Browser abschießen, wenn Sie solche Anweisungen in einer Endlosschleife abschicken.

3. Welcher Browser reagiert bei self.close(); mit einer Warnmeldung - der Navigator oder der Internet Explorer?
Der Internet Explorer

4. Ja "> Stimmt das?

```
<FRAMESET rows="55%, 1%, 1%, 5%">
```

Ja.

5. "width=300; height=250; screenX=100; screenY=100");

Die Optionen werden mit Kommata und nicht mit Semikola getrennt "> Was stimmt hier nicht?

```
open("bild1.htm", "DiesistkeinleeresFenster",
"width=300; height=250; screenX=100; screenY=100");
```

Die Optionen werden mit Kommata und nicht mit Semikola getrennt.

6. Für was steht self?

Für das aktuelle Fenster - eine »read-only«-Eigenschaft.

7. Was macht die Methode home()?

Sie ruft die Startseite des Browsers auf.

8. Stimmt das?

```
<FRAMESET cols="25%, 75%">
<FRAME src= "Li.htm" name="links" >
<FRAME src= "Re.htm" name="rechts" >
</FRAMESET>
```

Ja.

9. Versuchen Sie den Fehler zu finden:

```
<FRAMESET cols="30%, 70%">
  <FRAME src="li.htm">
  <FRAMESET rows="30%, 70%">
    <FRAME src="131.gif">
  <FRAMESET cols="50%, 50%, 50%" frameborder=0>
    <FRAME src= "li.htm" noresize>
  <FRAMESET cols="50%, 50%">
```



```
<FRAME src= "132.gif">
<FRAME src= "132.gif">
</FRAMESET>
<FRAME src= "131.gif" scrolling=no>
</FRAMESET>
<FRAME src="132.gif">
</FRAMESET>
</FRAMESET>
```

Die Zeile `<FRAMESET cols="50%, 50%">` muss in rows unterteilen.

Tag 17

1. Was stimmt hier nicht?
`window.captureEvents(Event.Click);`
Es heißt `Event.CLICK`.
2. Stimmt das?
`window.captureEvents(`
`Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP)`
Ja.
3. Es ist korrekt Das ist HTML und da ist Groß- und Kleinschreibung egal "> Ist das korrekt oder müssen for und event groß geschrieben werden?
`<SCRIPT for=document event=onclick`
`language="JavaScript">`
Es ist korrekt. Das ist HTML und da ist Groß- und Kleinschreibung egal.
4. Was macht `releaseEvents()`?
Die Methode erlaubt das Ignorieren eines aufgefangenen Ereignisobjekts vom angegebenen Typ.
5. Was macht `routeEvent()`?
Die Methode erlaubt das Weiter- bzw. Umleiten eines aufgefangenen Ereignisobjekts zu einem angegebenen Zielobjekt.
6. Stimmt das?
`window.captureEvents(Event.ONCLICK);`
Nein. Es heißt `Event.CLICK`.

Tag 18

1. Für was steht NaN?
»Not a Number«.
2. Diese explizite Abfrage auf exakte Gleichheit ist nur für kleinere, private Home-pages angemessen Im professionellen Bereich dürfte man solche Abfragen nicht finden, denn die Eigenschaft `appName` kann verändert werden Dies machen viele »Sponsoren«, indem sie beispielsweise die Eigenschaft auf "Netscape von T-OhneLeine" setzen Eine Abprüfung auf Gleichheit mit "Netscape" durch Scripte wird also oft false liefern Man sollte deshalb nicht einfach nur auf Gleichheit prüfen, sondern darauf, ob eine bestimmte Zeichenkette im Wert der Eigenschaft enthalten ist "> Was ist das Problem an folgendem Vergleich?
`if(navigator.appName == "Netscape")`
Diese explizite Abfrage auf exakte Gleichheit ist nur für kleinere, private Home-pages angemessen. Im professionellen Bereich dürfte man solche Abfragen nicht finden, denn die Eigenschaft `appName` kann verändert werden. Dies machen viele »Sponsoren«, indem sie beispielsweise die Eigenschaft auf "Netscape von T-OhneLeine" setzen. Eine Abprüfung auf Gleichheit mit "Netscape" durch Scripte wird also oft false liefern. Man sollte deshalb nicht einfach nur auf Gleichheit prüfen, sondern darauf, ob eine bestimmte Zeichenkette im Wert der Eigenschaft enthalten ist.
3. Was macht `getDate()`?
Gibt den Monatstag eines Datumsobjekts als Zahl zurück.

4. Ja Das ist die Definition eines Function-Objekts "> Geht das?
berechne = new Function("a","b","return a * b + 5");
Ja. Das ist die Definition eines Function-Objekts.
5. Was macht random() und zu welchem Objekt gehört die Methode?
Math. Die Methode generiert eine Zufallszahl zwischen 0 und 1.
6. Für was steht NEGATIVE_INFINITY?
»Negativ-Unendlich«.
7. Für was steht availLeft und zu welchem Objekt gehört es?

Das Objekt ist screen, und availLeft ist eine Eigenschaft mit der Angabe der X-Koordinate von dem ersten verfügbaren Pixel.

8. Was macht slice() und zu welchem Objekt gehört die Methode?
String. Die Methode extrahiert einen Teilstring von dem angegebenen Index bis zum Ende oder einem optionalen Ende-Index. Dieser kann auch ein negativer Wert sein, wenn ab dem Ende des Strings rückwärts gezählt werden soll.

Tag 19

1. Stimmt das?
var i = new image();
Nein, der Konstruktor muss groß geschrieben werden.
2. Für was ist DOM die Abkürzung?
»Document Object Model«.
3. Für was ist JSSS die Abkürzung?
»JavaScript-Assisted Style Sheets«.
4. Was passiert im Netscape Navigator, wenn er das Objekt all vorfindet?
Er reagiert mit einer Fehlermeldung.
5. Was ist referrer?
Eine Eigenschaft mit dem URL der HTML-Datei, von dem aus die aktuelle HTML-Datei aufgerufen wurde.

Tag 20

1. Was ist <FORM>?
Der Beginn eines Formular-Containers.
2. Was ist <OPTION>?
Ein Container für einen Listeneintrag.
3. Was ist multiple?
Ein Attribut, das Multiselektion erlaubt.
4. Ist das richtig?
<SELECT name="Tiere" size=1>
<OPTION>Hund</OPTION>
<OPTION>Katze</OPTION>
<OPTION>Maus</OPTION>
</SELECT>

Ja. Das ist ein einzeliges Selektionsfeld.
5. Mit welcher HTML-Anweisung wird ein mehrzeiliges Eingabefeld realisiert?
Mit dem <TEXTAREA>-Container.
6. Wie unterscheiden sich Radiobuttons von Checkbuttons ?
Durch type="radio" und type="checkbox".
7. Was ist options?
Ein Objektfeld, das in der Objekthierarchie unter elements angeordnet ist. Darüber hat man vollständigen Zugriff

auf Auswahllisten innerhalb eines Formulars.

Tag 21

1. Von welchen Eigenschaften oder Methoden des document-Objekts wird das Style-Objekt erzeugt?
`document.classes`
`document.contextual`
`document.ids`
`document.tags`
2. Wie können Style-Sheets in eine Webseite integriert werden?
Direkte Notation im <STYLE>-Container.
Import externer Stilinformationen über den <STYLE>-Container.
Die <LINK>-Referenz.
Inline-Referenz
3. Wie sieht die Syntax einer CSS-Deklaration immer aus?
Es gibt jedes Mal den folgenden Aufbau:
`name : wert`
Dabei gibt name das zu formatierende Element an und wert die konkrete Stilinformation. Mehrere Deklarationen werden mit Semikolon getrennt.
4. Was sind die wichtigsten Elemente einer Animation?
Die Methode `setTimeout()`, Positionsangaben, Größenangaben und die Möglichkeit, Bilder dynamisch auszutauschen. Eventuell noch animierte GIFs.
5. Ist das korrekt?
`H6.a { color : blue }`
Ja.

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

1

Falls es nicht bewusst verwendet wird.

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Tag C

Die CD zum Buch

Auf der beiliegenden CD finden Sie u.a. Folgendes:

Das Verzeichnis EBOOKS enthält dieses Buch im HTML-Format - falls Sie gerne an der Maschine bleiben oder mit dem Laptop unterwegs Zeit zum Lernen finden.

Das Eintippen des Source-Codes der Beispiellistings ist aus Gründen höheren Lerneffekts nur zu empfehlen. Lassen Sie sich also von den Programmen im Unterverzeichnis Source weniger zur Bequemlichkeit als vielmehr zum Spielen und Ausprobieren verführen.

Damit Sie Ihre Werke wie im Buch vorgeführt gleich mit verschiedenen Browsern testen können, liefert sie die CD-ROM unter Browser gleich mit. Hier finden Sie den Netscape Navigator 4.7, den Internet Explorer 5.5 und verschiedene Trial-Versionen von Opera für Windows, Linux und BeOS.

Im Verzeichnis Programme schließlich finden Sie den beschriebenen Editor Phase 5 sowie Trial-Versionen von Flash und Dreamweaver.

Stichwortverzeichnis

Symbols

<A>

- [<A>](#)

<APP>-

- [<APP>-](#)

<APPLET>

- [<APPLET>](#)
- [<APPLET>](#)

- [](#)

<BGSOUND>

- [<BGSOUND>](#)

<big>

- [<big>](#)

<blockquote>

- [<blockquote>](#)

<BODY>

- [<BODY>](#)

**
**

- [
](#)

<cite>

- [<cite>](#)

<code>

- [<code>](#)

<dfn>

- [<dfn>](#)

<DIV>

- [<DIV>](#)

<DOCTYPE>

- [<DOCTYPE>](#)

- [](#)

<EMBED>

- [<EMBED>](#)

- [](#)

<FORM>

- [<FORM>](#)

<FRAME>

- [<FRAME>](#)

<FRAMESET>

- [<FRAMESET>](#)

<H1>

- [<H1>](#)

<H2>

- [<H2>](#)

<H3>

- [<H3>](#)

<H4>

- [<H4>](#)

<H5>

- [<H5>](#)

<H6>

- [<H6>](#)

<HEAD>

- [<HEAD>](#)

<HR>

- [<HR>](#)

<i>

- [<i>](#)

- [](#)

<INPUT>

- [<INPUT>](#)

<italic>

- [<italic>](#)

<kbd>

- [<kbd>](#)

<LAYER>

- [<LAYER>](#)
- [<LAYER>](#)

- [](#)

<META>

- [<META>](#)

praktische Beispiele

- [praktische Beispiele](#)

<NOBR>

- [<NOBR>](#)

<NOFRAMES>

- [<NOFRAMES>](#)

<NOSCRIPT>

- [<NOSCRIPT>](#)

<OBJECT>

- [<OBJECT>](#)
- [<OBJECT>](#)

- [](#)

<OPTION>

- [<OPTION>](#)

<P>

- [<P>](#)

<PARAM>

- [<PARAM>](#)

<samp>

- [<samp>](#)

<SCRIPT>

- [<SCRIPT>](#)

<SELECT>

- [<SELECT>](#)

<small>

- [<small>](#)

- [](#)

- [](#)

<strike>

- [<strike>](#)

- [](#)

<STYLE>

- [<STYLE>](#)

<sub>

- [<sub>](#)

<sup>

- [<sup>](#)

<TABLE>

- [<TABLE>](#)

<TD>

- [<TD>](#)

<TEXTAREA>

- [<TEXTAREA>](#)

<TH>

- [<TH>](#)

<TITLE>

- [<TITLE>](#)

<TR>

- [<TR>](#)

<tt>

- [<tt>](#)

<u>

- [<u>](#)

- [](#)

<var>

- [<var>](#)

A

Abfrage der Eigenschaften von HTML-Elementen

- [Abfrage der Eigenschaften von HTML-Elementen](#)

abs()

- [abs\(\)](#)

ActiveX-Controls

- [ActiveX-Controls](#)

Einbindung

- [Einbindung](#)

Additionsoperator

- [Additionsoperator](#)

Additions-Zuweisungsoperator

- [Additions-Zuweisungsoperator](#)

Advanced Research Projects Agency, s. ARPA

- [Advanced Research Projects Agency, s. ARPAAEinführung & Grundlagen](#)

alert()

- [alert\(\)](#)

praktische Beispiele

- [praktische Beispiele](#)

all

Objekt

- [Objekt](#)

praktische Beispiele

- [praktische Beispiele](#)

- [praktische Beispiele](#)

anchor()

- [anchor\(\)](#)

anchor, Objekt

- [anchor, Objekt](#)

anchors, Objektfeld

- [anchors, Objektfeld](#)

Animation

- [Animation](#)

JavaScript

- [JavaScript](#)

JavaScript und Style Sheets

- [JavaScript und Style Sheets](#)

Animation mit CSS

- [Animation mit CSS](#)

- [Animation mit CSS](#)

- [Animation mit CSS](#)

Animierte GIFs

- [Animierte GIFs Einführung & Grundlagen](#)

Anweisungen

- [Anweisungen](#)

applet, Objekt

- [applet, Objekt](#)

applets, Objektfeld

- [applets, Objektfeld](#)

ARPA

- [ARPA Einführung & Grundlagen](#)

ARPANET

- [ARPANET Einführung & Grundlagen](#)

Array, Objekt

- [Array, Objekt](#)
- [Array, Objekt](#)

Arrays

- [Arrays](#)

ASCII-Zeichensatz

- [ASCII-Zeichensatz](#)

ASP

- [ASP Das Wesen von JavaScript](#)

Ausdruck

- [Ausdruck](#)

Ausdrucksanweisungen

- [Ausdrucksanweisungen](#)

B

back()

- [back\(\)](#)
- [back\(\)](#)

big()

- [big\(\)](#)

Bildmanipulationen

- [Bildmanipulationen](#)
- [Bildmanipulationen](#)

Binäre Operatoren

- [Binäre Operatoren](#)

bitweise

AND-Operator

- [AND-Operator](#)

Komplement-Operator

- [Komplement-Operator](#)

OR-Operator

- [OR-Operator](#)

XOR-Operator

- [XOR-Operator](#)

bitweise Operatoren

- [bitweise Operatoren](#)

bitweise Verschiebung

- [bitweise Verschiebung](#)

blink()

- [blink\(\)](#)

Blockanweisung

- [Blockanweisung](#)

blur()

- [blur\(\)](#)
- [blur\(\)](#)

Body, Webseite

- [Body, Webseite](#)

bold()

- [bold\(\)](#)

Boolean

- [Boolean](#)

Objekt

- [Objekt](#)
- [Objekt](#)

borderWidths()

- [borderWidths\(\)](#)

Botschaften

Objekte

- [Objekte](#)

break

- [break](#)

Browser

Aufgaben

- [Aufgaben Einführung & Grundlagen](#)

Browserversionen

- [Browserversionen](#)

Button

Objekte

- [Objekte](#)

Bytecode

Java

- [Java Das Wesen von JavaScript](#)

C

captureEvents()

- [captureEvents\(\)](#)
- [captureEvents\(\)](#)
- [captureEvents\(\)](#)

Cascading Style Sheets, s. CSS

- [Cascading Style Sheets, s. CSS](#)

case

- [case](#)

Casting

- [Casting](#)

CGI

- [CGI Das Wesen von JavaScript](#)

Formulare

- [Formulare](#)

charAt()

- [charAt\(\)](#)

charCodeAt(Index)

- [charCodeAt\(Index\)](#)

Checkbox

Objekte

- [Objekte](#)

clearTimeout()

- [clearTimeout\(\)](#)

click()

- [click\(\)](#)

Client

- [Einführung & Grundlagen](#)

Client-seitige Scriptsprachen

- [Client-seitige Scriptsprachen Das Wesen von JavaScript](#)

close()

- [close\(\)](#)
- [close\(\)](#)
- [close\(\)](#)

CMS

- [CMS Einführung & Grundlagen](#)

Common Gateway Interface, s. CGI

- [Common Gateway Interface, s. CGI Das Wesen von JavaScript](#)

Composer

- [Composer Einführung & Grundlagen](#)

concat()

- [concat\(\)](#)
- [concat\(\)](#)

confirm()

- [confirm\(\)](#)

praktische Beispiele

- [praktische Beispiele](#)

Container

HTML

- [HTML](#)

Content Management

- [Content Management Einführung & Grundlagen](#)

contextual()

- [contextual\(\)](#)

praktische Beispiele

- [praktische Beispiele](#)

continue

- [continue](#)

Cookies

- [Cookies](#)

cos()

- [cos\(\)](#)

CSS

- [CSS](#)

- [CSS](#)

Anwendungsregeln

- [Anwendungsregeln](#)

Deklaration

- [Deklaration](#)

Farbe und Hintergrund

- [Farbe und Hintergrund](#)

JavaScript-Zugriff

- [JavaScript-Zugriff](#)

Positionierung

- [Positionierung](#)

Schriftformatierungseigenschaften

- [Schriftformatierungseigenschaften](#)

Textblockgestaltung

- [Textblockgestaltung](#)

Textgestaltung

- [Textgestaltung](#)

CuteFTP

- [CuteFTP Einführung & Grundlagen](#)

D

Date

Objekt

- [Objekt](#)

- [Objekt](#)

praktische Beispiele

- [praktische Beispiele](#)

- [praktische Beispiele](#)

Datentypen

- [Datentypen](#)

Debugging

- [Debugging](#)

default

- [default](#)

Deklarationsanweisungen

- [Deklarationsanweisungen](#)

Dekrement-Operator

- [Dekrement-Operator](#)

deprecated

- [deprecated](#)
- [deprecated](#)

DHTML

- [DHTML](#)

Dienstprotokolle

- [Dienstprotokolle Einführung & Grundlagen](#)

Divisionsoperator

- [Divisionsoperator](#)

Divisions-Zuweisungsoperator

- [Divisions-Zuweisungsoperator](#)

DNS-Konzept

- [DNS-Konzept Einführung & Grundlagen](#)

document

Objekt

- [Objekt](#)
- [Objekt](#)

Document Objekt Model, s. DOM

- [Document Objekt Model, s. DOM](#)

Document Type Definition, s. DTD

- [Document Type Definition, s. DTD](#)

DOM

- [DOM](#)

Domain

- [Domain Einführung & Grundlagen](#)

Domain Name System, s. DNS

- [Domain Name System, s. DNSEinführung & Grundlagen](#)

DOT-Notation

- [DOT-Notation](#)

do-while-Schleife

- [do-while-Schleife](#)

Dreamweaver

- [Dreamweaver Einführung & Grundlagen](#)

Dreamweaver 3.0

- [Dreamweaver 3.0 Einführung & Grundlagen](#)

DTD

- [DTD](#)

Dynamic-HTML, s. DHTML

- [Dynamic-HTML, s. DHTML](#)

dynamische Farbveränderungen über CSS

- [dynamische Farbveränderungen über CSS](#)

dynamische Kennzeichnung von Links über CSS

- [dynamische Kennzeichnung von Links über CSS](#)

dynamische Positionsveränderungen über CSS

- [dynamische Positionsveränderungen über CSS](#)

E

ECMA-262-Norm

- [ECMA-262-Norm](#)

JavaScript

- [JavaScript](#)

Einbindung in eine Webseite

- [Einbindung in eine Webseite](#)

Eingebettete Formatvorlagen

- [Eingebettete Formatvorlagen](#)

elements

Objektfeld

- [Objektfeld](#)
- [Objektfeld](#)

else

- [else](#)

Endlosschleifen

- [Endlosschleifen](#)

Ereignis

was ist ein

- [was ist ein](#)

Ereignisbehandlung

- [Ereignisbehandlung](#)

Ereignismodell

- [Ereignismodell](#)

JavaScript

- [JavaScript](#)

Erfinder von JavaScript

- [Erfinder von JavaScript Einführung & Grundlagen](#)

escape()

JavaScript-Funktion

- [JavaScript-Funktion](#)

praktische Beispiele

- [praktische Beispiele](#)

eval()

JavaScript-Funktion

- [JavaScript-Funktion](#)

event

Auswerten

- [Auswerten](#)

Objekt

- [Objekt](#)

- [Objekt](#)

praktische Beispiele

- [praktische Beispiele](#)

Event-Handler

- [Event-Handler](#)

expliziter Aufruf

- [expliziter Aufruf](#)

Eventhandling

- [Eventhandling](#)

Events

- [Events](#)

exp()

- [exp\(\)](#)

Extensible Markup Language, s. XML

- [Extensible Markup Language, s. XMLDas Wesen von JavaScript](#)

Extensible Stylesheet Language, s. XSL

- [Extensible Stylesheet Language, s. XSLDas Wesen von JavaScript](#)

externer Aufruf

- [externer Aufruf](#)

F

Fehler

Hilfe durch den JavaScript-Interpreter

- [Hilfe durch den JavaScript-Interpreter](#)

Hilfe durch Syntaxhighlighting

- [Hilfe durch Syntaxhighlighting](#)

Kontrollausgaben

- [Kontrollausgaben](#)

Laufzeitfehler

- [Laufzeitfehler](#)

logische

- [logische](#)

syntaktische

- [syntaktische](#)

typographische

- [typographische](#)

Fehlervermeidung

- [Fehlervermeidung](#)

FileUpload

Objekte

- [Objekte](#)

find()

- [find\(\)](#)

fixed()

- [fixed\(\)](#)

Flash

- [Flash Einführung & Grundlagen](#)

focus()

- [focus\(\)](#)
- [focus\(\)](#)

fontcolor()

- [fontcolor\(\)](#)

fontsize()

- [fontsize\(\)](#)

for...in-Schleife

- [for...in-Schleife](#)
- [for...in-Schleife](#)

form

Objekt

- [Objekt](#)

- [Objekt](#)

Formatvorlagen

eingebettete

- [eingebettete](#)

forms

Objektfeld

- [Objektfeld](#)

- [Objektfeld](#)

Formulare

- [Formulare](#)

Auswahllisten

- [Auswahllisten](#)

Button

- [Button](#)

Checkbutton

- [Checkbutton](#)

Datumseingaben

- [Datumseingaben](#)

einzeilige Eingabefelder

- [einzeilige Eingabefelder](#)

Hintergründe

- [Hintergründe](#)

JavaScript-Zugriff

- [JavaScript-Zugriff](#)

mehrzeilige Eingabefelder

- [mehrzeilige Eingabefelder](#)

Passworteingaben

- [Passworteingaben](#)

praktische Beispiele

- [praktische Beispiele](#)

- [praktische Beispiele](#)

- [praktische Beispiele](#)

Radiobutton

- [Radiobutton](#)

Standard-Dateiauswahlfenster

- [Standard-Dateiauswahlfenster](#)

for-Schleife

- [for-Schleife](#)

forward()

- [forward\(\)](#)
- [forward\(\)](#)

frame

Objekt

- [Objekt](#)
- [Objekt](#)

Frames

- [Frames](#)

Eigenschaften

- [Eigenschaften](#)

JavaScript-Zugriff

- [JavaScript-Zugriff](#)

Jokerzeichen

- [Jokerzeichen](#)

Objektfeld

- [Objektfeld](#)

praktische Beispiele

- [praktische Beispiele](#)
- [praktische Beispiele](#)

reservierte Fensternamen

- [reservierte Fensternamen](#)

Verweise

- [Verweise](#)

fromCharCode()

- [fromCharCode\(\)](#)

Frontpage 2000

- [Frontpage 2000 Einführung & Grundlagen](#)

Frontpage Express

- [Frontpage Express Einführung & Grundlagen](#)

FTP

- [FTP Einführung & Grundlagen](#)

Function

Objekt

- [Objekt](#)
- [Objekt](#)

function

- [function](#)

Funktionen

- [Funktionen](#)

Bezeichner

- [Bezeichner](#)

Rückgabewert

- [Rückgabewert](#)

vorgefertigte JavaScript-

- [vorgefertigte JavaScript-](#)

Fusion 5.0

- [Fusion 5.0 Einführung & Grundlagen](#)

G

GET

- [GET](#)

getClass()

JavaScript-Funktion

- [JavaScript-Funktion](#)

getDate()

- [getDate\(\)](#)

getDay()

- [getDay\(\)](#)

getHours()

- [getHours\(\)](#)

getMinutes()

- [getMinutes\(\)](#)

getMonth()

- [getMonth\(\)](#)

getSeconds()

- [getSeconds\(\)](#)

getSelection()

- [getSelection\(\)](#)

getTime()

- [getTime\(\)](#)

getTimeZoneoffset()

- [getTimeZoneoffset\(\)](#)

getYear()

- [getYear\(\)](#)

GIF

Animator

- [Animator Einführung & Grundlagen](#)

Construction Set

- [Construction Set Einführung & Grundlagen](#)

Format

- [Format](#)

Gleichheitsoperator

- [Gleichheitsoperator](#)

go()

- [go\(\)](#)

Grafik Workshop

- [Grafik Workshop Einführung & Grundlagen](#)

Größer-als-oder-gleich-Operator

- [Größer-als-oder-gleich-Operator](#)

Größer-als-Operator

- [Größer-als-Operator](#)

Grundgerüst

Webseite

- [Webseite](#)

H

handleEvent()

- [handleEvent\(\)](#)
- [handleEvent\(\)](#)
- [handleEvent\(\)](#)

Header

Webseite

- [Webseite](#)

Hexadezimalsystem

- [Hexadezimalsystem](#)

Hidden

Objekte

- [Objekte](#)

Hintergrundbild

Webseite

- [Webseite](#)

history

Objekt

- [Objekt](#)
- [Objekt](#)

home()

- [home\(\)](#)

Hot Metal Pro

- [Hot Metal Pro Einführung & Grundlagen](#)

HotJava

- [HotJava Einführung & Grundlagen](#)

HTML

- [HTML Einführung & Grundlagen](#)

1.0

- [1.0 Einführung & Grundlagen](#)

2.0

- [2.0 Einführung & Grundlagen](#)

3.0/3.2

- [3.0/3.2 Einführung & Grundlagen](#)

4.0

- [4.0 Einführung & Grundlagen](#)

Auflistung aller HTML-4.0-Attribute

- [Auflistung aller HTML-4.0-Attribute](#)

Auflistung aller HTML-4.0-Befehle

- [Auflistung aller HTML-4.0-Befehle](#)

Browserversionen

- [Browserversionen](#)

Dateiendung

- [Dateiendung](#)

Farbangaben

- [Farbangaben](#)

Hintergrundbild

- [Hintergrundbild](#)

Listen

- [Listen](#)

Maskieren

- [Maskieren](#)

Prinzip der Fehlertoleranz

- [Prinzip der Fehlertoleranz Einführung & Grundlagen](#)

Steueranweisungen

- [Steueranweisungen](#)

Tabelle mit Maskierungen

- [Tabelle mit Maskierungen](#)

Tabellen

- [Tabellen](#)

Verschachtelung von Tags

- [Verschachtelung von Tags](#)

Weiterleitung einer Seite

- [Weiterleitung einer Seite](#)

Wizards

- [Wizards Einführung & Grundlagen](#)

Zeilen- und Absatzschaltungen

- [Zeilen- und Absatzschaltungen](#)

HTTP

- [HTTP Einführung & Grundlagen](#)

Hyper Text Transport Protocol, s. HTTP

- [Hyper Text Transport Protocol, s. HTTP Einführung & Grundlagen](#)

if-Anweisung

- [if-Anweisung](#)

if-else-Operator

- [if-else-Operator](#)

if-Schleife

praktische Beispiele

- [praktische Beispiele](#)

image

Objekt

- [Objekt](#)

images

Objektfeld

- [Objektfeld](#)

praktische Beispiele

- [praktische Beispiele](#)

important

CSS

- [CSS](#)

Style Sheets

- [Style Sheets](#)

indexOf()

- [indexOf\(\)](#)

praktische Beispiele

- [praktische Beispiele](#)

Infinity

- [Infinity](#)

praktische Beispiele

- [praktische Beispiele](#)

Infixschreibweise

- [Infixschreibweise](#)

Information Hiding

- [Information Hiding](#)

Inkrement-Operator

- [Inkrement-Operator](#)

praktische Beispiele

- [praktische Beispiele](#)

- [praktische Beispiele](#)

Inline-Referenz

- [Inline-Referenz](#)

JavaScript

- [JavaScript](#)

Style Sheets

- [Style Sheets](#)

Instanzen

- [Instanzen](#)

Instanzvariablen

- [Instanzvariablen](#)

Interpretation

was ist

- [was ist Einführung & Grundlagen](#)

Interpreter

- [Interpreter Einführung & Grundlagen](#)

IP-Nummer

- [IP-Nummer Einführung & Grundlagen](#)

IP-Protokoll

- [IP-Protokoll Einführung & Grundlagen](#)

isFinite()

JavaScript-Funktion

- [JavaScript-Funktion](#)

isNaN()

JavaScript-Funktion

- [JavaScript-Funktion](#)

ISO-5589-1-Norm

- [ISO-5589-1-Norm](#)

ISO-8859-1-Norm

- [ISO-8859-1-Norm](#)

italics()

- [italics\(\)](#)

Iterationsanweisungen

- [Iterationsanweisungen](#)

J

Java

- [Java](#)

JavaScript-Zugriff über LiveConnect

- [JavaScript-Zugriff über LiveConnect](#)

offizielle Definition

- [offizielle Definition Das Wesen von JavaScript](#)

Java PlugIn

- [Java PlugIn](#)

Java Virtual Machine, s. JVM

- [Java Virtual Machine, s. JVMDas Wesen von JavaScript](#)

Java-Applets

Einbindung

- [Einbindung](#)

JavaScript-Zugriff

- [JavaScript-Zugriff](#)

JavaArray

Objekt

- [Objekt](#)

Java-Console

- [Java-Console](#)

javaEnabled()

- [javaEnabled\(\)](#)

JavaObject

Objekt

- [Objekt](#)

JavaPackage

Objekt

- [Objekt](#)

JavaScript

- [JavaScript Das Wesen von JavaScript](#)

JavaScript-Assisted Style Sheets, s. JSSS

- [JavaScript-Assisted Style Sheets, s. JSSS](#)

Java-Servlets

- [Java-Servlets Das Wesen von JavaScript](#)

join()

- [join\(\)](#)

JPEG-Format

- [JPEG-Format](#)

JSript

Vergleich zu JavaScript

- [Vergleich zu JavaScript Einführung & Grundlagen](#)

JSSS

- [JSSS](#)
- [JSSS](#)

JVM

- [JVM Das Wesen von JavaScript](#)

K

Klassen

- [Klassen](#)

Klassenvariablen

- [Klassenvariablen](#)

Kleiner-als-oder-gleich-Operator

- [Kleiner-als-oder-gleich-Operator](#)

Kleiner-als-Operator

- [Kleiner-als-Operator](#)

konditionaler Operator

- [konditionaler Operator](#)
- [konditionaler Operator](#)

Konstruktor

- [Konstruktor](#)

L

lastIndexOf()

- [lastIndexOf\(\)](#)

layer

Objekt

- [Objekt](#)

layers

praktische Beispiele

- [praktische Beispiele](#)

link

Objekt

- [Objekt](#)

link()

- [link\(\)](#)

links

Objektfeld

- [Objektfeld](#)

Listen

- [Listen](#)
- [Listen](#)

Literale

- [Literale](#)

LiveConnect

- [LiveConnect](#)

LiveScript

- [LiveScript Das Wesen von JavaScript](#)

LiveWire

- [LiveWire Das Wesen von JavaScript](#)

location

Objekt

- [Objekt](#)

- [Objekt](#)

log()

- [log\(\)](#)

logischer AND-Operator

- [logischer AND-Operator](#)

logischer NOT-Operator

- [logischer NOT-Operator](#)

logischer OR-Operator

- [logischer OR-Operator](#)

M

margins()

- [margins\(\)](#)

Maskierung

HTML

- [HTML](#)

match()

- [match\(\)](#)

Math

Objekt

- [Objekt](#)

- [Objekt](#)

max()

- [max\(\)](#)

Metainformationen

praktisches Beispiel

- [praktisches Beispiel](#)

Webseite

- [Webseite](#)

Methoden

- [Methoden](#)

MILNET

- [MILNET Einführung & Grundlagen](#)

Mime-Typ

- [Mime-Typ](#)

contentType

Objekt

- [Objekt](#)

Objektfeld

- [Objektfeld](#)

Tabelle

- [Tabelle](#)

min()

- [min\(\)](#)

Mocha

- [Mocha Das Wesen von JavaScript](#)

Modulo-Operator

- [Modulo-Operator](#)

Modulo-Zuweisungsoperator

- [Modulo-Zuweisungsoperator](#)

Mosaic

- [Mosaic Einführung & Grundlagen](#)

- [Mosaic Einführung & Grundlagen](#)

moveBy()

- [moveBy\(\)](#)

moveTo()

- [moveTo\(\)](#)

MP3

Einbindung in eine Webseite

- [Einbindung in eine Webseite](#)

Multiplikationsoperator

- [Multiplikationsoperator](#)

Multiplikations-Zuweisungsoperator

- [Multiplikations-Zuweisungsoperator](#)

Multipurpose Internet Mail Extensions, s. MIME-Typ

- [Multipurpose Internet Mail Extensions, s. MIME-Typ](#)

N

NaN

- [NaN](#)

praktische Beispiele

- [praktische Beispiele](#)

navigator

Objekt

- [Objekt](#)

- [Objekt](#)

netscape

Objekt

- [Objekt](#)

new

- [new](#)

- [new](#)

null

- [null](#)

Number

- [Number](#)

Objekt

- [Objekt](#)

- [Objekt](#)

Number()

JavaScript-Funktion

- [JavaScript-Funktion](#)

O

Object

- [Object](#)

Objektdaten

- [Objektdaten](#)

Objektdeklaration

- [Objektdeklaration](#)

- [Objektdeklaration](#)

Objekte

- [Objekte](#)

all

- [all](#)

anchor

- [anchor](#)

applet

- [applet](#)

Array

- [Array](#)

- [Array](#)

Boolean

- [Boolean](#)
- [Boolean](#)

Date

- [Date](#)
- [Date](#)

Definition eigener

- [Definition eigener](#)

document

- [document](#)
- [document](#)

Eigenschaften

- [Eigenschaften](#)

event

- [event](#)
- [event](#)

form

- [form](#)
- [form](#)

frame

- [frame](#)
- [frame](#)

Function

- [Function](#)
- [Function](#)

history

- [history](#)
- [history](#)

image

- [image](#)

java

- [java](#)

JSONArray

- [JSONArray](#)

JsonObject

- [JsonObject](#)

JavaPackage

- [JavaPackage](#)

layer

- [layer](#)

link

- [link](#)

location

- [location](#)
- [location](#)

Math

- [Math](#)
- [Math](#)

Methoden

- [Methoden](#)

mimeType

- [mimeType](#)

navigator

- [navigator](#)
- [navigator](#)

netscape

- [netscape](#)

Number

- [Number](#)
- [Number](#)

Packages

- [Packages](#)

plugin

- [plugin](#)

RegExp

- [RegExp](#)
- [RegExp](#)
- [RegExp](#)

screen

- [screen](#)
- [screen](#)

String

- [String](#)

- [String](#)

Style

- [Style](#)
- [Style](#)

sun

- [sun](#)

window

- [window](#)
- [window](#)

Objektfelder

- [Objektfelder](#)

Objektinstanz

Erstellen

- [Erstellen](#)

Objektmethoden

- [Objektmethoden](#)

Objektvariablen

- [Objektvariablen](#)

onAbort

Event-Handler

- [Event-Handler](#)

onBlur

Event-Handler

- [Event-Handler](#)

onChange

Event-Handler

- [Event-Handler](#)

onClick

Event-Handler

- [Event-Handler](#)

onDbClick

Event-Handler

- [Event-Handler](#)

onError

Event-Handler

- [Event-Handler](#)

onFocus

Event-Handler

- [Event-Handler](#)

onKeyDown

Event-Handler

- [Event-Handler](#)

onKeyPress

Event-Handler

- [Event-Handler](#)

onKeyUp

Event-Handler

- [Event-Handler](#)

onLoad

Event-Handler

- [Event-Handler](#)

onMouseDown

Event-Handler

- [Event-Handler](#)

onMouseMove

Event-Handler

- [Event-Handler](#)

onMouseOut

Event-Handler

- [Event-Handler](#)

onMouseOver

Event-Handler

- [Event-Handler](#)

onMouseUp

Event-Handler

- [Event-Handler](#)

onReset

- [onReset](#)

Event-Handler

- [Event-Handler](#)

onSelect

Event-Handler

- [Event-Handler](#)

onSubmit

- [onSubmit](#)

Event-Handler

- [Event-Handler](#)

onUnload

Event-Handler

- [Event-Handler](#)

open()

- [open\(\)](#)
- [open\(\)](#)
- [open\(\)](#)

Operanden

- [Operanden](#)

Operator

konditionaler

- [konditionaler](#)
- [konditionaler](#)

typeof

- [typeof](#)

Operatoren

- [Operatoren](#)

arithmetische

- [arithmetische](#)

bitweise

- [bitweise](#)

einstellige arithmetische

- [einstellige arithmetische](#)

Inkrement/Dekrement

- [Inkrement/Dekrement](#)

logische Vergleichsoperatoren

- [logische Vergleichsoperatoren](#)

Prioritäten

- [Prioritäten](#)

Vergleichsoperatoren

- [Vergleichsoperatoren](#)

Zuweisungsoperatoren

- [Zuweisungsoperatoren](#)

options

Objektfeld

- [Objektfeld](#)

- [Objektfeld](#)

P

Packages

Objekt

- [Objekt](#)

paddings()

- [paddings\(\)](#)

Paint Shop Pro

- [Paint Shop Pro Einführung & Grundlagen](#)

Parameter

HTML

- [HTML](#)

parent

reservierter Fensternamen

- [reservierter Fensternamen](#)

parse()

- [parse\(\)](#)

parseFloat()

JavaScript-Funktion

- [JavaScript-Funktion](#)

Parser

- [Parser Einführung & Grundlagen](#)

parsInt()

JavaScript-Funktion

- [JavaScript-Funktion](#)

Password

Objekte

- [Objekte](#)

Perl

- [Perl Das Wesen von JavaScript](#)

plugin

Objekt

- [Objekt](#)

PlugIns

Einbindung

- [Einbindung](#)

plugins

Objektfeld

- [Objektfeld](#)

plugins.refresh()

- [plugins.refresh\(\)](#)

pop()

- [pop\(\)](#)

Positionierungen

Style Sheets

- [Style Sheets](#)

POST

- [POST](#)

Postfix-Operatoren

- [Postfix-Operatoren](#)

pow()

- [pow\(\)](#)

Practical Extraction and Report Language, s. Perl

- [Practical Extraction and Report Language, s. PerlDas Wesen von JavaScript](#)

Präfix-Operatoren

- [Präfix-Operatoren](#)

print()

- [print\(\)](#)

Prinzip der Fehlertoleranz

- [Prinzip der Fehlertoleranz Einführung & Grundlagen](#)

prompt()

- [prompt\(\)](#)

praktische Beispiele

- [praktische Beispiele](#)

prototype

- [prototype](#)

Prozeduren

- [Prozeduren](#)

Punkt-Notation

- [Punkt-Notation](#)

push()

- [push\(\)](#)

Quellen

- [Quellen](#)

R

Radio

Objekte

- [Objekte](#)

random()

- [random\(\)](#)

RegExp

Objekt

- [Objekt](#)
- [Objekt](#)
- [Objekt](#)

releaseEvents()

- [releaseEvents\(\)](#)
- [releaseEvents\(\)](#)

reload()

- [reload\(\)](#)

replace()

- [replace\(\)](#)
- [replace\(\)](#)

reservierte Fensternamen

- [reservierte Fensternamen](#)

Reset

Objekte

- [Objekte](#)

reset()

- [reset\(\)](#)

resizeBy()

- [resizeBy\(\)](#)

resizeTo()

- [resizeTo\(\)](#)

return

- [return](#)
- [return](#)

reverse()

- [reverse\(\)](#)

RGB-Werte

- [RGB-Werte](#)

round()

- [round\(\)](#)

routeEvent()

- [routeEvent\(\)](#)
- [routeEvent\(\)](#)

Router

- [Router Einführung & Grundlagen](#)

S

Schlüsselworte

- [Schlüsselworte](#)

screen

Objekt

- [Objekt](#)
- [Objekt](#)

praktische Beispiele

- [praktische Beispiele](#)

Scripte

Einbindung in eine Webseite

- [Einbindung in eine Webseite](#)

externer Aufruf

- [externer Aufruf](#)

Inline-Aufruf

- [Inline-Aufruf](#)

scrollBy()

- [scrollBy\(\)](#)

scrollTo()

- [scrollTo\(\)](#)

search()

- [search\(\)](#)

praktische Beispiele

- [praktische Beispiele](#)

Select

Objekte

- [Objekte](#)

select

Objekte

- [Objekte](#)

select()

- [select\(\)](#)

Selector

CSS

- [CSS](#)

self

reservierter Fensternamen

- [reservierter Fensternamen](#)

semantische Auszeichnung

- [semantische Auszeichnung Das Wesen von JavaScript](#)

Server

- [Server Einführung & Grundlagen](#)

Server-seitige Scriptsprachen

- [Server-seitige Scriptsprachen Das Wesen von JavaScript](#)

setDate()

- [setDate\(\)](#)

setHours()

- [setHours\(\)](#)

setMinutes()

- [setMinutes\(\)](#)

setMonth()

- [setMonth\(\)](#)

setSeconds()

- [setSeconds\(\)](#)

setTime()

- [setTime\(\)](#)

setTimeout()

- [setTimeout\(\)](#)

setYear()

- [setYear\(\)](#)

Setzen der Eigenschaften von Bildern

- [Setzen der Eigenschaften von Bildern](#)

Setzen der Eigenschaften von HTML-Elementen

- [Setzen der Eigenschaften von HTML-Elementen](#)

SGML

- [SGML Einführung & Grundlagen](#)

shift()

- [shift\(\)](#)

Sicherungsmaßnahmen

- [Sicherungsmaßnahmen](#)

sin()

- [sin\(\)](#)

slice()

- [slice\(\)](#)
- [slice\(\)](#)

small()

- [small\(\)](#)

sort()

- [sort\(\)](#)

splice()

- [splice\(\)](#)

split()

- [split\(\)](#)

sqrt()

- [sqrt\(\)](#)

Standardobjekte

- [Standardobjekte](#)

Statuszeile

Zugriff aus JavaScript

- [Zugriff aus JavaScript](#)

stop()

- [stop\(\)](#)

strike()

- [strike\(\)](#)

String

- [String](#)

Objekt

- [Objekt](#)
- [Objekt](#)

String()

JavaScript-Funktion

- [JavaScript-Funktion](#)

Structured Generalized Markup Language, s. SGML

- [Structured Generalized Markup Language, s. SGMLEinführung & Grundlagen](#)

Style

Objekt

- [Objekt](#)
- [Objekt](#)

Style Sheets

- [Style Sheets Einführung & Grundlagen](#)
- [Style Sheets](#)

direkte Einbindung

- [direkte Einbindung](#)

Inline-Referenz

- [Inline-Referenz](#)

Link-Notation

- [Link-Notation](#)

Positionierung

- [Positionierung](#)

was sind

- [was sind](#)

sub()

- [sub\(\)](#)

Submit

Objekte

- [Objekte](#)

submit()

- [submit\(\)](#)

substr()

- [substr\(\)](#)

substring()

- [substring\(\)](#)

Subtraktionsoperator

- [Subtraktionsoperator](#)

Subtraktions-Zuweisungsoperator

- [Subtraktions-Zuweisungsoperator](#)

Sun

- [Sun Einführung & Grundlagen](#)

sun

Objekt

- [Objekt](#)

sup()

- [sup\(\)](#)

Super NoteTab

- [Super NoteTab Einführung & Grundlagen](#)
- [Super NoteTab Das Wesen von JavaScript](#)

switch-Fallunterscheidung

- [switch-Fallunterscheidung](#)

Syntaxhighlighting

- [Syntaxhighlighting Einführung & Grundlagen](#)

T

Tabellen

- [Tabellen](#)
- [Tabellen](#)

JavaScript

- [JavaScript](#)

Tags

HTML

- [HTML](#)

taint()

JavaScript-Funktion

- [JavaScript-Funktion](#)

tan()

- [tan\(\)](#)

TCP/IP

- [TCP/IP Einführung & Grundlagen](#)

Telnet

- [Telnet Einführung & Grundlagen](#)

Ternäre Operatoren

- [Ternäre Operatoren](#)

Test, ob aktiviert

- [Test, ob aktiviert](#)

Text

Objekte

- [Objekte](#)

Textarea

Objekte

- [Objekte](#)

this

- [this](#)

toGMTString()

- [toGMTString\(\)](#)

Token

- [Token](#)
- [Token](#)

toLocaleString()

- [toLocaleString\(\)](#)

toLowerCase()

- [toLowerCase\(\)](#)

top

reservierter Fenstername

- [reservierter Fenstername](#)

Top-Level-Eigenschaften

- [Top-Level-Eigenschaften](#)

JavaScript

- [JavaScript](#)

Top-Level-Funktionen

- [Top-Level-Funktionen](#)

toSource()

- [toSource\(\)](#)

toString()

- [toString\(\)](#)

toUpperCase()

- [toUpperCase\(\)](#)

Transmission Control Protocol/Internet Protocol, s. TCP/IP

- [Transmission Control Protocol/Internet Protocol, s. TCP/IPEinführung & Grundlagen](#)

typeof-Operator

- [typeof-Operator](#)

praktische Beispiele

- [praktische Beispiele](#)

Typkonvertierung

- [Typkonvertierung](#)

U

Unäre Operatoren

- [Unäre Operatoren](#)

undefined

- [undefined](#)
- [undefined](#)
- [undefined](#)

unescape()

JavaScript-Funktion

- [JavaScript-Funktion](#)

praktische Beispiele

- [praktische Beispiele](#)

Ungleichheitsoperator

- [Ungleichheitsoperator](#)

UNICODE

- [UNICODE Einführung & Grundlagen](#)

Unicode

- [Unicode](#)

Uniform Resource Locator, s. URL

- [Uniform Resource Locator, s. URLEinführung & Grundlagen](#)

unshift()

- [unshift\(\)](#)

untaint()

JavaScript-Funktion

- [JavaScript-Funktion](#)

Upload

- [Upload Einführung & Grundlagen](#)

URL

- [URL Einführung & Grundlagen](#)

Usenet

- [Usenet Einführung & Grundlagen](#)

UTC()

- [UTC\(\)](#)

V

valueOf()

- [valueOf\(\)](#)

var

- [var](#)

Variablen

- [Variablen](#)

Datentypen

- [Datentypen](#)

Deklaration

- [Deklaration](#)

globale

- [globale](#)

Initialisierung

- [Initialisierung](#)

lokale

- [lokale](#)

Typkonvertierung

- [Typkonvertierung](#)

Wertzuweisung

- [Wertzuweisung](#)

VBScript

- [VBScript Einführung & Grundlagen](#)

was ist

- [was ist Das Wesen von JavaScript](#)

Vergleichsoperatoren

- [Vergleichsoperatoren](#)

Versionsangabe

- [Versionsangabe](#)

Video

Einbindung in eine Webseite

- [Einbindung in eine Webseite](#)

Visual Basic

- [Visual Basic Das Wesen von JavaScript](#)

W

W3C

- [W3C Einführung & Grundlagen](#)

Web-Publishing-Assistent

- [Web-Publishing-Assistent Einführung & Grundlagen](#)

Webseiten

dynamischer Aufbau

- [dynamischer Aufbau](#)

Optimierung

- [Optimierung](#)

Zugriff auf den Browser

- [Zugriff auf den Browser](#)

Zugriff auf die Besucherplattform

- [Zugriff auf die Besucherplattform](#)

Weiterleiten einer Webseite

- [Weiterleiten einer Webseite](#)

JavaScript

- [JavaScript](#)

Weiterleitung einer Webseite

<META>-Tag

- [<META>-Tag](#)

while-Schleife

- [while-Schleife](#)

praktische Beispiele

- [praktische Beispiele](#)

window

Eigenschaften

- [Eigenschaften](#)

Event-Handler

- [Event-Handler](#)

Objekt

- [Objekt](#)

- [Objekt](#)

reservierter Fensternamen

- [reservierter Fensternamen](#)

with()

- [with\(\)](#)

Wizards

HTML

■ [HTML Einführung & Grundlagen](#)

World Wide Web-Consortium

- [World Wide Web-Consortium Einführung & Grundlagen](#)

Wrapper

- [Wrapper](#)

write()

- [write\(\)](#)

writeln()

- [writeln\(\)](#)

WS_FTP95

- [WS_FTP95 Einführung & Grundlagen](#)

WYSIWYG

- [WYSIWYG Einführung & Grundlagen](#)

X

XML

- [XML Das Wesen von JavaScript](#)

XSL

- [XSL Das Wesen von JavaScript](#)

Z

Zugriff auf die Statuszeile

- [Zugriff auf die Statuszeile](#)

Zugriff auf Formulare

- [Zugriff auf Formulare](#)

Zugriff auf Frames

- [Zugriff auf Frames](#)

Zugriff auf Java

- [Zugriff auf Java](#)

Zugriff auf Style Sheets

- [Zugriff auf Style Sheets](#)

Zuweisungsoperatoren

- [Zuweisungsoperatoren](#)

arithmetische

- [arithmetische](#)

© [Markt+Technik Verlag](#), ein Imprint der Pearson Education Deutschland GmbH

Wir sind an Ihrer Meinung interessiert!

Als Teil unseres Bestrebens, Bücher höchster Qualität zu produzieren, ist SAMS an Ihrer Meinung interessiert.

Sie können Ihre Kommentare, Ideen oder Vorschläge zur Verbesserung zukünftiger Ausgaben an die unten angegebene Adresse oder per Fax an 089-46003-330 schicken.

Online-Verbindung bekommen Sie über das Internet <http://www.mut.com>. Schon jetzt vielen Dank – Ihre Kommentare werden uns dabei helfen, weiterhin die besten Computerbücher herauszugeben.

SAMS

Markt&Technik

[Erik Franz](#)

Hans-Pinsel-Str. 9B

85540 Haar bei München

Fragen zur HTML-Version?

Markt&Technik

[Harry Jörg](#)

Hans-Pinsel-Str. 9b

85540 Haar bei München