Auf einen Blick

Inhalt

Teil I: JavaScript lernen

Teil I
JavaScript lernen

1 Einleitung

So können Angreifer beispielsweise Nutzerkennung mit Passwörtern oder auch lokal gespeicherte Daten von privaten und kommerziellen Internetnutzern ausspionieren. [...] Das BSI empfiehlt deswegen den Betreibern von WWW-Servern dringend, vollständig auf JavaScript zu verzichten oder zumindest für ihre sicherheitsbewussten Kunden Alternativangebote bereitzustellen, die ohne aktive Inhalte dargestellt werden können.
Pressemitteilung des Bundesamtes für Sicherheit in der Informationstechnik, BSI (http://www.bsi.bund.de/fachthem/sinet/java99.htm)

Vorsicht ist geboten, wenn Unternehmen für ihre Web-Seiten das Gestaltungssystem JavaScript verwenden. Dahinter verbergen sich kleine Programme, die sich unbemerkt auf dem Kunden-PC einnisten und ihn angreifbar für fremde Zugriffe machen.
BILD am SONNTAG, 21. Mai 2000

JavaScript ist in aller Munde – sowohl im positiven als auch im negativen Sinne. Immer wieder geistern Horrormeldungen wie obige Empfehlung des BSI oder der Unfug aus der BamS durch die Presse. Der Grund dafür ist, dass den Browser-Herstellern – in diesem Falle vor allem Netscape (Netscape Navigator) und Microsoft (Microsoft Internet Explorer) immer wieder Fehler bei der Programmierung ihrer Browser unterlaufen (bei Interesse: **http://www.guninski.com**). Zwar werden eiligst Bugfixes zur Verfügung gestellt, aber der (Image-)Schaden ist schon eingetreten. Andererseits sind keine Fälle bekannt, in denen Sicherheitslecks zu Schäden geführt hätten. So hat die obige Pressemitteilung des BSI in der Fachpresse zu scharfer Kritik geführt; teilweise war von überzogener Panikmache die Rede. Die inhaltliche Qualität der Aussage aus der BamS bedarf keines Kommentars.

Ohne die Diskussion allzu sehr vertiefen zu wollen: Mit JavaScript kann man die eher beschränkten Möglichkeiten von HTML erweitern. Es handelt sich hierbei um eine clientseitige Programmiersprache. Das heißt, alles läuft im Browser ab, und man muss keine besonderen Server-Voraussetzungen erfüllen. Letzterer Punkt ist (noch) ein großer Nachteil von serverseitigen Sprachen wie ASP, Perl, PHP und Konsorten, denn viele Hoster (und vor allem die günstigen Hoster) gestatten keine serverseitige Programmierung. Mit JavaScript ist dies jedoch alles kein Problem.

In den nun folgenden Kapiteln finden Sie eine komplette Einführung in die Sprache, stets anhand von praxisnahen Beispielen illustriert. Ich habe versucht, mich bei den Beispielen auf das Wesentliche – nämlich auf die Java-

Script-Programmierung – zu konzentrieren. Daher sind die Beispiele grafisch bei weitem nicht ausgefeilt, aber sie erfüllen die gestellten Aufgaben.

Jedes Kapitel wird mit Fragen und Übungen abgeschlossen, sodass Sie das Erlernte gleich in die Tat umsetzen können. Kapitel 45 schließlich ist eine komplette Sprachreferenz, die Ihnen auch nach der Lektüre des Buches immer wieder als Nachschlagewerk dienen wird.

MyGalileo

Des Weiteren möchte ich noch auf den Online-Dienst **MyGalileo** verweisen, der im World Wide Web unter **http://www.galileo-press.de** zu erreichen ist. Wenn Sie dieses Buch registrieren, erhalten Sie Zugriff auf die Webseiten, die dieses Buch begleiten. Nach und nach werden dort Inhalte eingefügt, und Sie können mit mir in Kontakt treten. Nutzen Sie diese Gelegenheit; jede Art von konstruktiver Kritik ist willkommen. Wenn Sie einen Fehler finden, teilen Sie mir diesen bitte mit, damit er in einer weiteren Auflage dieses Buches nicht mehr auftritt. Sie helfen damit auch anderen Lesern. In **MyGalileo** werden Sie auch eine aktualisierte Fehlerliste für dieses Buch finden.

1.1 Die Geschichte von JavaScript

Gegen Ende des Jahres 1995 stellte die Firma Netscape die neueste Version ihres Internet-Browsers vor. Der Versionssprung von 1.1 auf 2.0 war berechtigt, da solch revolutionäre Dinge wie etwa die Unterstützung von Frames als neues Feature hinzugekommen waren. Unter anderem war auch eine Sprache namens **LiveScript** eingebaut, mit der man auf HTML-Seiten Einfluss nehmen konnte. Die Syntax dieser Sprache lehnte sich an Java an, und aus marketingtechnischen Gründen wurde die Programmiersprache schließlich in **JavaScript** umbenannt.

Schon bald begann der Siegeszug dieser Programmiersprache. Zwar war die Implementierung im Browser eher mangelhaft (in der Netscape-Version 2.0 konnte man JavaScript nicht einmal deaktivieren, was zu einem bösen Sicherheitsproblem werden sollte), aber mit der Beta-Version des Netscape Navigator 3 wurde die JavaScript-Version 1.1 vorgestellt, die deutlich mehr Möglichkeiten bot. Microsoft wollte nun auch auf den Zug aufspringen und kündigte an, in Version 3 des Internet Explorers ebenfalls eine Skriptunterstützung anzubieten. Aus lizenzrechtlichen Gründen wurde die Sprache **JScript** getauft. Von der Syntax her war sie aber mit JavaScript praktisch identisch.

ECMAScript

Seitdem läuft das übliche Wettrennen zwischen Netscape und Microsoft. Während der Internet Explorer 3 praktisch nur JavaScript 1.0 unterstützte, beherrschte der Internet Explorer 4 schon JavaScript 1.1 – und enthielt einige Features aus dem Sprachschatz von JavaScript 1.2, das mit dem Netscape Navigator 4 eingeführt wurde, der parallel zum »IE4« erschien. Seit dem Netscape Navigator 4.06 gibt es JavaScript in der Version 1.3, die zwar nur rudimentäre Verbesserungen anbietet, sich aber an dem ECMA-262-

Standard, ECMAScript, orientiert. Netscape hatte erkannt, dass man auf Standards setzen muss, und behauptete gleichzeitig, dass der Netscape Navigator 4.06 der Browser sei, der der Spezifikation von ECMAScript am nächsten komme. Microsoft wiederum verkleinerte mit dem Internet Explorer 5 den Rückstand weiter, setzte aber inzwischen eher auf andere Standards, wie **DOM** (**Document Object Model**). Unter dem Codenamen **Mozilla** wurde währenddessen die nächste Netscape-Version als Open Source entwickelt. Das heißt, der Sourcecode lag offen, und jeder konnte an der neuen Version mitentwickeln. Nichtsdestotrotz wurde der Löwenanteil der Arbeit von Netscape-Angestellten geleistet, das »Linux-Wunder« funktioniert nicht überall!

Im Spätherbst 2000 begannen sich die Ereignisse zu überschlagen. Netscape geriet unter Druck, da der Internet Explorer 5.5 sowie der Internet Explorer 6, der als Teil des neuen Microsoft-Betriebssystems Windows XP (Codename: Whistler) angeboten wurde, schnell den aktuellen Zwischenstand des Open-Source-Projekts erreichten. Netscape bastelte eine Installationsroutine und bot eben diesen Zwischenstand als Netscape 6 zum Download an (bezeichnenderweise war zu dieser Zeit eine fast identische Version auf der Mozilla-Website mit der Versionsnummer 0.6 als Download zu haben). Dieser Schritt stieß in der Fachwelt auf große Kritik, denn die Version war noch weit davon entfernt, für den Produktiveinsatz zu taugen, erst die späteren Unterversionen von Netscape 6 und die nächste Version, Netscape 7, sorgten hier für spürbare Besserung. Und – was aus Sicht der Webdesigner viel schlimmer war – die neue Version erwies sich in Sachen JavaScript als nicht abwärtskompatibel, wenn es um »DHTML« ging. So mussten (und müssen immer noch) eine Reihe von Skripten umprogrammiert werden. Doch keine Sorge, denn um eines vorwegzunehmen: In diesem Buch gehen wir ausführlich auf den neuesten Netscape-Browser ein. Die neuen Versionen (Netscape 6 und 7) unterstützen übrigens JavaScript 1.5, das aber nur unwesentliche Änderungen gegenüber den Vorgängerversionen aufweist (es wird eine aktuellere Version von ECMAScript unterstützt).

Die Crux bei der Programmierung von Webseiten besteht darin, dass die beiden großen Browserhersteller immer wieder versucht haben, sich dadurch zu übertrumpfen, dass sie immer neue Features in ihre Browser einbauten. Das ist innerhalb des eigenen Mikrokosmos eine gute Sache, aber wenn die Technik inkompatibel zur Konkurrenz ist, schaut etwa die Hälfte der Websurfer (die nämlich den jeweils anderen Browser benutzen) bildlich gesprochen in die Röhre. Keine Firma kann es sich leisten, ein Web-Angebot nur für die halbe Zielgruppe zu erstellen.

Cross Browser

In diesem Buch geht es nicht nur um die neuesten Effekte und Kniffe. Vielmehr geht es darum, wie Sie Ihre Zielgruppe erweitern können, indem JavaScript-Programme so gestaltet werden, dass sie von den Benutzern beider Browser und auch von den Benutzern älterer Versionen verwendet

werden können. Bei bekannten Fehlern in der Implementierung einzelner Browser werden – soweit möglich – Workarounds angeboten. Ihr Ziel sollte es nicht nur sein, eine technisch eindrucksvolle Seite zu erstellen, sondern auch Ihre Zielgruppe zu erweitern, indem Sie ältere Browser nicht ausschließen oder zumindest vor Fehlermeldungen bewahren.

Sonstige Browser

Es ist ja nicht so, dass nur der Netscape Navigator und der Internet Explorer JavaScript unterstützen. Die Nummer 3 im Browsermarkt, der Opera-Browser, unterstützt JavaScript-Version 1.3, und sogar recht gut. Auch der in StarOffice integrierte Webbrowser sowie die Version 3 von Suns HotJava-Browser sind JavaScript-fähig. Diese Browser haben jedoch einen so kleinen Marktanteil, dass sie im Referenzteil nicht extra berücksichtigt werden. Mit umsichtiger Programmierung lassen sich aber auch hier Fehler vermeiden.

1.2 Voraussetzungen

HTML-Kenntnisse vorausgesetzt

Wenn Sie dieses Buch durcharbeiten wollen, sollten Sie zumindest Kenntnisse in HTML vorweisen können – denn dieses Buch soll nicht künstlich durch einen zusätzlichen HTML-Teil aufgebläht werden. Ebenfalls sollten Sie sicher mit Ihrem Betriebssystem umgehen können und beispielsweise Editoren starten und Dateien speichern können. Apropos Editoren: Im Gegensatz zu HTML gibt es bei der JavaScript-Programmierung keinen großen Markt für Editoren. Hier geschieht das meiste immer noch textbasiert. Die einfachen Texteditoren, die mit dem Betriebssystem mitgeliefert werden (beispielsweise **Notepad** oder **SimpleEdit**), reichen sogar schon aus; wer mehr will, sollte einen der folgenden Editoren ausprobieren, die hier in der Reihenfolge meiner Präferenzen geordnet sind (sie sind aber Shareware und damit nach Ablauf einer Testperiode kostenpflichtig):

- UltraEdit **(http://www.ultraedit.com)** – sehr leistungsfähiger Editor inklusive Syntax-Highlighting (Windows).
- Programmer's File Editor (**http://www.lancs.ac.uk/people/cpaap/pfe/**) – Die Entwicklung wurde mittlerweile eingestellt, aber unter der genannten Adresse ist dieser Editor immer noch erhältlich (Windows).
- NoteTab (**http://www.notetab.ch**) – mehrfach ausgezeichneter Klassiker unter den Texteditoren, auch in kostenfreier Light-Variante erhältlich (Windows).
- BBEdit (**http://www.bbedit.com**) – der Macintosh-Klassiker, auch in einer Light-Version erhältlich.

Kompatibilität

Außerdem sollten Sie möglichst viele Browser installieren. Sie sollten nämlich Ihre JavaScript-Programme auf möglichst vielen Zielplattformen testen, um sicherzustellen, dass die Programme eben nicht nur bei Ihnen laufen. Wenn Sie den Internet Explorer 5 oder 5.5 über den Internet Explorer 4 installieren, müssen Sie bei der Installation unbedingt angeben, dass die alte Version (im so genannten **Kompatibilitätsmodus**) erhalten bleibt. Der

Internet Explorer 3 lässt sich in der 32-Bit-Variante nicht parallel installieren, aber wenn Sie ein wenig beherzt sind, verwenden Sie die 16-Bit-Variante; das funktioniert (angeblich: Alle Installationen sind Ihr Risiko). Der Internet Explorer 6 überschreibt leider ausnahmslos alle Vorgängerversionen. Beim Netscape Navigator ist alles ein wenig einfacher. Die Versionen 7, 6, 4.x und 3 lassen sich problemlos parallel installieren. Ebenfalls sollten Sie den Opera-Browser installieren (**http://www.opera.com**), da dieser auch eine gute JavaScript-Unterstützung hat. Benutzern von Mac OS X steht der Safari-Browser von Apple zur Verfügung, der auf dem Linux-Browser Konqueror basiert. Weitere Informationen zu den verschiedenen Browsern können Sie Kapitel 2 entnehmen.

Während der Entwicklung Ihrer Programme sollten Sie unbedingt den Netscape Navigator verwenden. Diese Empfehlung hat nichts mit meinem persönlichen Geschmack zu tun, sondern beruht auf der Tatsache, dass die Fehlermeldungen des Netscape Navigators viel aussagekräftiger sind als die des Internet Explorers. Läuft ein Programm einmal im Netscape Navigator einigermaßen zuverlässig, dann können Sie natürlich sofort dazu übergehen, es im Internet Explorer zu testen.

Seit der Version 4.06 führt ein JavaScript-Fehler beim Netscape Navigator nicht mehr zu einem extra Warnfenster; stattdessen werden alle Fehlermeldungen in der JavaScript-Konsole angezeigt. Diese wird sichtbar, wenn Sie in der Adresszeile des Browsers `javascript:` eintippen (inklusive des Doppelpunkts) und auf ↵ drücken. Im Netscape 6/7 geht das mit dem Menübefehl **Extras · Web-Entwicklung · JavaScript-Konsole.**

Es bleibt mir nur noch, Ihnen viel Spaß bei der Lektüre dieses Buches und viel Erfolg und Kreativität beim Experimentieren mit JavaScript zu wünschen. Und noch einmal die Bitte: Geben Sie mir Rückmeldung über **MyGalileo**, damit ich Ihre Anregungen für eine Neuauflage in Erwägung ziehen kann. Da ich sehr viele E-Mails bekomme, kann es mit einer Antwort etwas dauern, also werden Sie bitte nicht ungeduldig. Fragen zu Buchthemen beantworte ich gerne, sofern es möglich ist; bei allgemeinen Fragen zu JavaScript sind Sie in den folgenden Newsgroups wohl besser aufgehoben und bekommen auch schneller eine Antwort:

- de.comp.lang.javascript
- comp.lang.javascript

Wenn Ihr Provider keine Newsgroup-Anbindung unterstützt, sollten Sie ihn wechseln oder sich ein kostenloses Konto bei **http://groups.google.de/** oder **http://netnews.web.de/** einrichten lassen.

Vielleicht finden Sie eine Antwort auf Ihre Frage aber auch in den FAQ der Newsgroup **de.comp.lang.javascript**, die Sie auf der Buch-CD finden. An dieser Stelle möchte ich mich sehr für die Erlaubnis bedanken, die FAQ bei-

legen zu dürfen. Lesen Sie die FAQ unbedingt durch, bevor Sie eine Frage in der Newsgroup stellen!

1.3 Danksagungen zu »JavaScript« (1. Auflage)

Manche Menschen hassen den Danksagungsteil in einem Buch, ich dagegen lese ihn immer als Erstes. Hier folgen die Namen einiger Leute, die mich vor oder während der Bucherstellung in irgendeiner Art und Weise unterstützt oder motiviert haben oder mir anderweitig behilflich waren.

Zunächst einmal gilt der Dank meiner Lektorin Judith Stevens und ihrem Team bei Galileo, die mich fabelhaft betreut haben und stets ein offenes Ohr für meine Wünsche und Anregungen hatten. Herzlichen Dank dafür. Rudolf Krahm ist übrigens »schuld« daran, dass ich überhaupt an dieses Projekt gekommen bin.

Die sprachlichen Korrekturen wurden von Friederike Daenecke ausgeführt. Sie hat dadurch die sprachliche Qualität dieses Buches deutlich gesteigert.

Das schöne Cover wurde von Barbara Thoben entworfen, die Herstellung lag in den Händen von Petra Strauch, in Vertretung Claudia Lucht.

Für die USA-Erlebnisse habe ich folgenden Leuten zu danken: den Verkehrspolizisten von Colby, Tigger und Jerry S. Für China waren das Thommi, Reinhold, Klaus, Axel, Wei Dan und Qu Jian Ben. Zu Hause danke ich neben meiner Familie meinen Freunden Christian, Marianne, Markus und Thorsten; den PP-Kollegen, allen Caipirinha-Anhängern sowie den Sneak-Preview-Fans. Vielen Dank für die »Sabotage« (d. h. Ablenkung von) meiner Arbeit!

Besonders danken möchte ich außerdem noch meinen Freunden Tobias und Matthias, die immer hinter mir stehen und ohne deren Unterstützung ich das Buch wohl nicht hätte schreiben können. Tobias hat übrigens alle Vektorgrafiken für dieses Buch erstellt.

In fachlicher Hinsicht habe ich vieles von dem, was ich weiß, durch das Diskutieren in Mailinglisten und Newsgroups gelernt. Exemplarisch für viele andere möchte ich hier IT und QA nennen, die durch ihr Wissen und ihre Kreativität schon vielen JavaScript-Programmierern geholfen haben.

Last but not least danke ich Thomas Maier, der mir vor ein paar Jahren die Freude am Schreiben wiedergegeben hat, sowie Cecily, die immer an dieses Projekt geglaubt hat.

Christian Wenz, im November 1999

1.4 Danksagungen zu »JavaScript-Rezepte« (1. Auflage)

Die grobe Inhaltsübersicht und der Name des Buchs entstanden am letzten Tag des Jahres 2000 in Ägypten, in der Nähe des Dreiländerecks. Dafür, dass weder das Konzept noch ich danach im Pool oder im Meer versenkt wurden, möchte ich Bettina danken (und natürlich noch für Vieles mehr).

Das Konzept nahm dann immer konkretere Formen an, bis die eigentliche Schreibarbeit dann im Sommer begann, zum Teil in München, zum Teil im toskanischen Exil. Während dieser Zeit hat mich alleine schon die Vorfreude auf den festen Ablauf des Freitagabends zum Arbeiten motiviert: Zunächst das Essen im »Il Piccolo Principe« (mit Andreas, Christian, Hilmar & Special Guests), danach die »Sneak Preview« im Cinema (mit Andreas, Christian, Jörg, Markus, Thorsten, Yvonne & Special Guests). Herzlichen Dank dafür.

Besonderer Dank gebührt des Weiteren Tobias und Matthias, die nicht nur immer hinter mir stehen, sondern mit denen ich auch 18 Stunden am Tag lachen könnte. Danke!

Ebenfalls möchte ich Yvonne danken, die mich während der Bucherstellung toll unterstützt hat.

Die Betreuung von Verlagsseite aus konnte glücklicherweise von meiner Traumkonstellation übernommen werden, die auch schon für die JavaScript-Referenz verantwortlich zeichnete. Das Lektorat hat Judith Stevens-Lemoine übernommen, die sprachliche Korrektur hat Friederike Daenecke durchgeführt (was man, wie ich hoffe, auch beim Durchlesen merkt). Der Umschlag wurde von Barbara Thoben gestaltet.

Die Listings in diesem Buch wurden von Stefan Krumbiegel und Anja Horch überprüft; so konnten noch einige Fehler ausgemerzt werden.

Bleibt mir nur noch, Ihnen viel Spaß beim Lesen und Schmökern in diesem Buch und beim Arbeiten mit den Rezepten zu wünschen!

Christian Wenz, im September 2001

1.5 Vorwort zur 5. Auflage

Eine neue Auflage, und dieses Mal fallen einige Neuerungen bereits beim ersten Blick auf: Das Cover ist neu gestaltet worden, der Buchumfang hat die 1000-Seiten-Grenze überschritten. Was aber hat sich im Inneren getan?

Die größte Neuerung ist der umfangreiche Praxisteil. Damit komme ich dem Wunsch zahlreicher Leser nach, das JavaScript-Buch und den Titel »JavaScript-Rezepte« zu vereinen. Gesagt, getan: Nach der bewährten Einführung in JavaScript werden im hinteren Buchteil zahlreiche fertige JavaScript-

Rezepte vorgestellt, die Sie direkt in eigene Webseiten einbauen können – sozusagen ein Werkzeugkasten für JavaScript-Entwickler. Gewisse Redundanzen sind unvermeidlich, aber sehr gering gehalten. Die Ausrichtungen der verschiedenen Buchteile sind auch verschieden: Am Anfang geht es um eine Vorstellung der Thematik und um die Einführung in die JavaScript-Programmierung; der hintere Buchteil versucht dagegen die Grundlagen in nützliche Mosaiksteine zu kapseln, aus denen dann eine Website mit viel JavaScript erstellt werden kann. Die erste Buchhälfte ist also zum Lernen, die zweite zum Einsetzen und Anwenden (und nebenbei lernen Sie doch noch den einen oder anderen kleinen Kniff lernen). Der dritte und letzte Buchteil enthält eine JavaScript-Objektreferenz, die Lösungen zu den Übungsaufgaben aus dem ersten Teil sowie nützliche Internetquellen.

Auch sonst gibt es viele Neuerungen. Jedes Kapitel wurde komplett durchgesehen, ergänzt und korrigiert. Ganz neue Kapitel gibt es auch. Beispielsweise werfen wir einen Blick über den Tellerrand: Ein Teil von Microsofts ominöser .NET-Strategie ist auch eine Sprache namens JScript .NET, bei der JavaScript deutlich Pate gestanden hat. Zudem wurden einige neue Rezepte hinzugefügt. Wie immer sind Sie dazu aufgerufen, dem Autor oder Verlag gegenüber kundzutun, welche Inhalte Sie gern noch ausführlicher dargestellt finden würden; so können Sie zukünftige Auflagen entscheidend mitgestalten.

Aber auch Konkurrenzbrowser sollen nicht unberücksichtigt bleiben. Der Macintosh-Plattform wird in dieser Auflage eine besondere Bedeutung zuerkannt, insbesondere den neuen Apple-Webbrowser Safari. Unter der Linux-Plattform ist der Konqueror, übrigens die Basis von Safari, sehr verbreitet. Er wird ebenfalls betrachtet. Trotz des eher geringen Marktanteils dieser Webbrowser äußerten viele Leser den Wunsch, diese Browser ausführlicher zu berücksichtigen. Außerdem ist dieses Buch schon in seinen Vorgängerversionen bekannt dafür gewesen, auch ältere oder seltenere Browser zu behandeln. An dieser Stelle ein Hinweis zur aktuellen Marktlage: Der Netscape Navigator 4.x stirbt langsam, aber sicher aus. Die bekannte W3B-Umfrage ermittelte im Jahre 2002, dass der Netscape 4 interessanterweise mit allen neueren Netscape-Versionen inklusive Mozilla gleichauf liegt – eine peinliche Schlappe für das ehrgeizige Mozilla-Projekt. Zwar ist abzusehen, dass der »alte« Netscape mittelfristig deutlich überflügelt werden wird, aber dennoch muss er heutzutage noch berücksichtigt werden. Professionelles Webdesign zeichnet sich gerade dadurch aus, dass eine möglichst große Zielgruppe bedient wird. Die zurzeit um sich greifende Manie – insbesondere in Zeitschriften –, dass JavaScript-Programme ausschließlich im Internet Explorer laufen (nicht einmal in neuen Netscape-Versionen), sorgt dafür, dass Websites nur noch in einem Webbrowser laufen, obwohl es doch so einfach wäre, auch andere Browser zu unterstützen. Ältere Browserversionen, vornehmlich der Internet Explorer und Netscape Navigator, jeweils Version 3 oder früher, haben mittlerweile keinen messbaren Marktanteil mehr. Aus diesem Grund werden Hinweise auf Fehler

und Besonderheiten dieser Versionen nicht mehr in dieser Auflage behandelt, von einigen Randbemerkungen einmal abgesehen.

Bei einem solch umfangreichen Buch können kleinere Fehler leider nie ganz ausgeschlossen werden. Zwar hat dieses Buch eine Reihe von Prüfprozessen durchlaufen, dennoch ist es so gut wie sicher, dass sich der sprichwörtliche Fehlerteufel irgendwo eingeschlichen hat. Wenn Sie also einen Fehler gefunden zu haben, wenden Sie sich über den Verlag an mich. Dann kann der Fehler in der nächsten Auflage ausgemerzt werden. Außerdem werden bekannt gewordene Fehler sowohl im Forum von Galileo Press als auch auf der Support-Seite zum Buch unter **http://www.hauser-wenz.de/support/** veröffentlicht. An genau dieselben Stellen können Sie sich auch wenden, wenn Sie Fragen oder Anregungen zum Buch haben. Ich freue mich auf Ihr Feedback! An dieser Stelle Danke an Felix Siegrist, der viele Verbesserungsvorschläge zum Praxisteil beigesteuert hat.

Allen Neuerungen zum Trotz, eine Sache ist in allen Auflagen gleich geblieben: Judith Stevens-Lemoine hat zum fünften Mal als Lektorin das Projekt geleitet, und Friederike Daeneke hat, ebenfalls zum fünften Mal, als sprachliche Korrektorin meine Ergüsse in lesbare Form gebracht. Der Erfolg des Buchs ist also das Ergebnis eines eingespielten Teams; herzlichen Dank dafür! Und: Danke auch an Sie, dass Sie dieses Buch gekauft haben.

Christian Wenz, im August 2003

P.S.: Besondere Glückwünsche an Yvonne! Sie hat ihr Studium beendet und darf sich jetzt »Dipl.-Ing.« nennen. Ich werde nie wieder etwas gegen FH-Studenten sagen, versprochen!

1.6 Die Icons in diesem Buch

Wichtige **Hinweise** finden Sie in Abschnitten, die mit diesem Symbol gekennzeichnet sind.

Achtung: Abschnitte mit diesem Symbol enthalten eine **Warnung**!

Dieses Icon markiert Abschnitte, in denen Sie auf Besonderheiten der unterschiedlichen **Browserversionen** und **-typen** aufmerksam gemacht werden. Auch die Eigenschaften der verschiedenen **JavaScript-Versionen** werden an diesen Stellen beschrieben.

Hier werden Sie auf Softwarefehler hingewiesen – jeder kennt sie unter demBegriff **Bug.**

Arbeitserleichterungen, Tastenkürzel und Schreibvereinfachungen finden Sie neben diesem Symbol. Hier stoßen Sie auf nicht ganz alltägliche **Profitipps**.

2 Vorbereitung

Take three months to prepare your machines and three months to complete your siege engineering.
– Sun Tzu, The Art Of War

In diesem Kapitel erfahren Sie zunächst, welche Browser überhaupt JavaScript unterstützen und wie Sie Ihr Testsystem einrichten können. Außerdem lernen Sie, wie JavaScript-Code in HTML-Dokumente eingebunden wird. Besondere Beachtung verdienen hierbei die Unterschiede zwischen den einzelnen Browsern.

2.1 Webbrowser

JavaScript ist eine so genannte clientseitige Programmiersprache. Das heißt, JavaScript-Programme werden im Webbrowser ausgeführt. Dies wird möglich, indem JavaScript in HTML eingebettet wird. HTML-Dateien können also JavaScript-Code enthalten. Wenn Sie im World Wide Web unterwegs sind, ist auf der Mehrheit der Seiten, die Sie aufsuchen JavaScript enthalten.

Nicht jeder Browser unterstützt JavaScript, aber die meisten gebräuchlichen Browser tun es. Zunächst ist es interessant zu wissen, welche Webbrowser überhaupt zurzeit gebräuchlich sind.

2.1.1 Netscape Navigator (und Konsorten)

Der wohl erste weithin bekannte Webbrowser ist der Netscape Navigator, der seit Mitte der 90er Jahre für verschiedenste Plattformen verfügbar ist. Als Nachfolger des Mosaic, des Urvaters aller grafischen Webbrowser, war er der erste Webbrowser mit JavaScript-Unterstützung, denn Netscape selbst hat JavaScript erfunden und geschaffen, um die (beschränkten) Möglichkeiten von HTML zu erweitern. Anfangs hieß die Programmiersprache noch LiveScript, was aber die Marketingabteilung von Netscape nicht sonderlich sexy fand. Die »Rettung« kam in Form der Firma Sun, die mit ihrer als plattformunabhängig konzipierten Programmiersprache Java Furore machte. Der clevere Schachzug von Netscape bestand darin, den Namen »Java« zu lizenzieren und schon hieß die Programmiersprache JavaScript. Zugegeben, die Sprachsyntax (Vorschriften, wie ein Programm auszusehen hat, Namen von Kommandos etc.) ist ähnlich, aber ansonsten haben die Sprachen Java und JavaScript nichts, aber auch gar nichts miteinander gemeinsam. Das ist leider auch heute, fast zehn Jahre später, noch nicht überall bekannt, und »Fachleute« reden von Java, meinen aber JavaScript. Die Namensgleichheit ist also eine Marketingmaßnahme und hat keine technische Begründung.

Dennoch gibt es Möglichkeiten, Java und JavaScript zusammenarbeiten zu lassen. In Kapitel 20 wird verraten, wie.

Aber zurück zur Historie des Netscape-Browsers: Version 2 führte JavaScript ein, Version 3 erweiterte die Möglichkeiten, beispielsweise wurden Rollover-Effekte möglich (siehe Kapitel 9). Version 3 gab es zudem als »Gold«-Version, in der auch ein Mailprogramm enthalten war. Netscape hatte damals erkannt, dass eine Kundenbindung an den Browser auch dadurch erzielt werden kann, dass Browser und Mailsoftware untrennbar miteinander verbunden sind. So öffnen sich Links in Mails direkt im zugehörigen Browser.

Abbildung 2.1 Lang´ ist´s her: Netscape 0.91 (nicht JavaScript-fähig)

Version 4 führte zunächst eine Namensänderung ein: Der Browser als Standalone-Produkt hieß weiterhin »Netscape Navigator«; das gesamte Paket inklusive Mailprogramm, Adressbuch und (eine Zeit lang) Terminverwaltung wurde »Netscape Communicator« getauft. Die Entwicklung des Navigator-Produkts, also ohne kundenbindende Zusatzkomponenten, wurde allerdings bald eingestellt, Version 4.08 war die letzte ihrer Art. Das Communicator-Paket dagegen erfuhr auch im Jahre 2003 noch ein Update, wenngleich auch vermutlich das allerletzte: Mit Version 4.8 dürfte die Entwicklung ihr Ende gefunden haben, auch wenn einige Bugs seit Jahren bekannt und unbehoben sind.

Für den JavaScript-Programmierer ist der Netscape 4 insbesondere in Hinblick auf die JavaScript-Unterstützung häufig ärgerlich. Netscape hat in die-

ser Version einige neue, proprietäre (eigene) Erweiterungen eingeführt. Diese fanden teilweise so wenig Zustimmung, dass sie in keinem anderen Browser und auch in keiner neuen Netscape-Version weitergeführt wurden. Insbesondere in den DHTML-Kapiteln (Kapitel 16 bis 19) werden wir mit einigen Kraftakten versuchen, JavaScript-Programme auch auf dem Netscape 4 zum Laufen zu bringen. Anfangs war das noch kein Problem – Netscape 4 war der Marktführer. Das sollte sich jedoch bald ändern, und ab diesem Zeitpunkt wurde es ein Problem.

Abbildung 2.2 Ursache vielen (JavaScript-)Übels: Netscape 4

In der Folgezeit ist einiges passiert. Netscape wurde von AOL gekauft, und die Entwicklung des Netscape wurde zunächst eingestellt. Allerdings wurde ein neuer, zu dieser Zeit noch als revolutionär zu bezeichnender Ansatz gewählt: Das Open-Source-Projekt Mozilla (**http://www.mozilla.org/**) wurde aus dem Boden gestampft. Sein Ziel sollte es sein, einen neuen Netscape-Browser zu entwickeln. Das Wörtchen »Netscape« taucht auf den Mozilla-Seiten selten auf, allerdings sind es größtenteils Netscape- bzw. AOL-Mitarbeiter, die sich an der Entwicklung beteiligen. Als schließlich die ersten Mozilla-Versionen erschienen, war der Marktanteil des Netscape bereits im einstelligen Bereich, denn zu lange hatte sich bei diesem Browser nichts getan. In ihrer Verzweiflung veröffentlichten die AOL-Verantwortlichen eine »Netscape«-Version des Mozilla. Um mit der damals aktuellen Version des Internet Explorer gleichzuziehen, erhielt dieser Netscape die Versionsnummer 6. In Hinblick auf die Produktqualität war diese Version leider ein totaler Reinfall: Die JavaScript-Unterstützung wies zahlreiche Fehler auf, und der Browser war langsam, schwerfällig und stürzte zudem häufig ab.

Der Image-Verlust war enorm, und auch Netscape-Enthusiasten geben heute unverwunden zu, dass Version 6 wirklich schlecht war (was sie nicht daran gehindert hatte, Jahre zuvor jede und jeden zu verteufeln, sollte ein böses Wort gegen den Netscape 6 gesagt werden). Version 7, die einige Zeit später erschien, war deutlich besser, und die aktuelle Version 7.1 ist wirklich gut. Doch das kommt vermutlich zu spät. Der Marktanteil ist weiter stetig gesunken, AOL hat sich außerdem von einer Reihe von Netscape-Mitarbeitern getrennt. Der vorerst letzte Akt in diesem Trauerspiel: Es wurde eine »Mozilla Foundation« gegründet, die sich um die weitere Entwicklung des Open-Source-Projekts kümmern soll. Für den Netscape jedoch hat wohl die letzte Stunde geschlagen: Die Version 7.1 ist aller Voraussicht nach die letzte des einstigen Klassenprimus. Schade, denn die neuen Versionen sind in Hinblick auf die JavaScript-Unterstützung vorbildlich, und der Browser ist auch nicht mehr so langsam wie früher. Allmählich werden auch die Horden von Bugs unter Kontrolle gebracht. Die Zukunft wird zeigen, ob und wie die Weiterentwicklung ohne größere AOL-Unterstützung laufen wird.

Abbildung 2.3 Der (vermutlich letzte) Netscape 7

2.1.2 Microsoft Internet Explorer

Der Microsoft Internet Explorer ist seit einiger Zeit unangefochtener Spitzenreiter im Browsermarkt. Zunächst hatte Microsoft bekanntermaßen das Internet »verschlafen«; die ersten Internet Explorer-Versionen 1.x und 2.x waren kaum zu gebrauchen. Mit dem Internet Explorer 3 änderte sich dies radikal. Der Browser wurde langsam, aber sicher konkurrenzfähig. Viele Eigenschaften des damals technologisch und auch in Sachen Verbreitung

haushoch überlegenen (und in einer Beta-Version vorliegenden) Netscape Navigator 3 wurden integriert. Die Version 3 ist auch die erste Version des »IE«, die JavaScript unterstützt. Die Nachfolgeversion 4 schließlich schloss die technologische Lücke zum Netscape (die Versionsnummern waren lange Zeit gleich, Netscape 4 erschien annähernd zeitgleich zum IE 4). Aufgrund einiger Netscape-Macken (siehe den vorangegangenen Abschnitt) begann der IE mit dieser Version langsam, aber stetig an Marktanteilen zu gewinnen. Zusätzlich zu den technologischen Vorteilen (die mit den Nachfolgeversionen 5.0, 5.5 und 6.0 den Vorsprung noch vergrößern sollten) nutzte Microsoft auch geschickt, aber bedenklich, die Marktmacht auf dem Desktop: Der Internet Explorer ist bei Windows automatisch dabei, er kann nicht oder kaum entfernt werden, und mit etwas »Glück« verlangt eine neue Version von Microsoft Office oder einer anderen Anwendung eine neuere Browserversion.

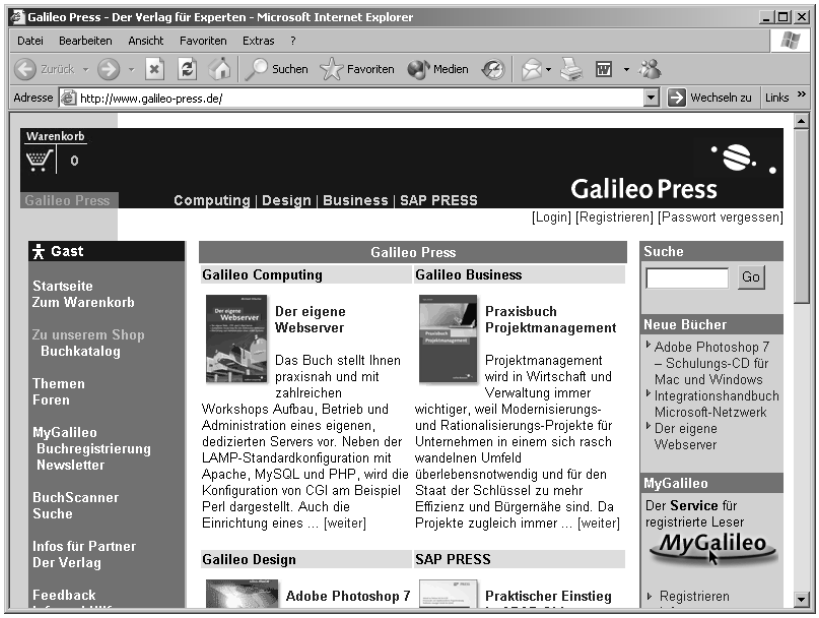

Abbildung 2.4 Der Marktführer: Microsoft Internet Explorer

Mittlerweile kann sich Microsoft über eine eindeutige Marktführerschaft freuen. Allerdings scheint sich auch hier die Browserentwicklung einem Stillstand zu nähern. Microsoft hat Mitte 2003 bekannt gegeben, dass Version 6.0 die wohl letzte sein wird. Neuere Versionen des Internet Explorer – so die Logik aus der Microsoft-Zentrale in Redmond – verlangen eine neue Windows-Version. Wer sich also über die (bekannten) Bugs des IE ärgert, wird sich unter Umständen etwas gedulden müssen, bis sie behoben werden.

Der IE ist häufig im Zentrum der Kritik, weil er (auch aufgrund seiner großen Verbreitung im Markt) oft zum Ziel von Virenautoren wird. Wenn Sie Windows frisch installieren, ist der Internet Explorer anfällig! Abhilfe schafft ein regelmäßiger Besuch bei Windows Update (**http://windowsupdate.microsoft.com/**), um aktuelle Sicherheitspatches zu installieren.

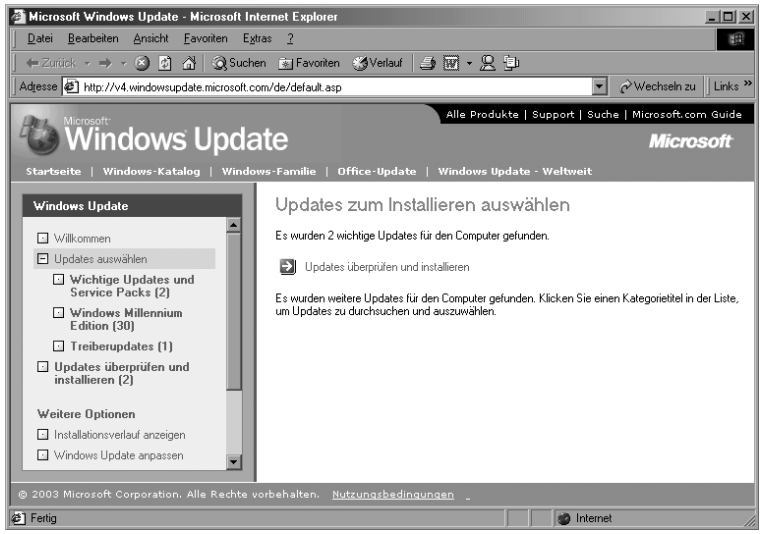

Abbildung 2.5 Windows Update ist fündig geworden.

Abbildung 2.6 Der Internet Explorer unter Mac OS X

Den Internet Explorer gibt es natürlich hauptsächlich für Windows, aber auch Mac-Versionen stehen zur Verfügung, allerdings alle mit Versionsnummer 5. Auch hier ist es fraglich, ob die Entwicklung weitergeht. Immerhin, die OS-X-Version ist neuer als die für OS 9 – aber was heißt das schon? Die Macintosh-Versionen hinken schon immer den Windows-IEs hinterher, und insbesondere die JavaScript-Unterstützung hat Lücken. (siehe Abbildung 2.6)

2.1.3 Opera

Nummer 3 im großen Browserreigen ist der kleine Opera-Webbrowser aus Norwegen. »Klein« deshalb, weil er in seinen früheren Versionen marketingtechnisch geschickt auf eine 1,44-MByte-Diskette kopiert werden konnte, so klein (und auch schnell) war er. Mittlerweile ist der Browser schon größer geworden, aber immer noch äußerst schlank. Die JavaScript-Unterstützung ist spätestens seit Version 7 auch wirklich gut; Version 6 bot häufiger Anlass zum Ärgern aufgrund zahlreicher Fehler. Bei einem maximal einstelligen Marktanteil lohnt es sich hier nicht, alte, fehlerbehaftete Versionen weiterhin zu unterstützen, deswegen wird auch in diesem Buch der Fokus auf Version 7 gelegt (zurzeit, also im August 2003, ist Version 7.11 aktuell, Version 7.20 als Beta).

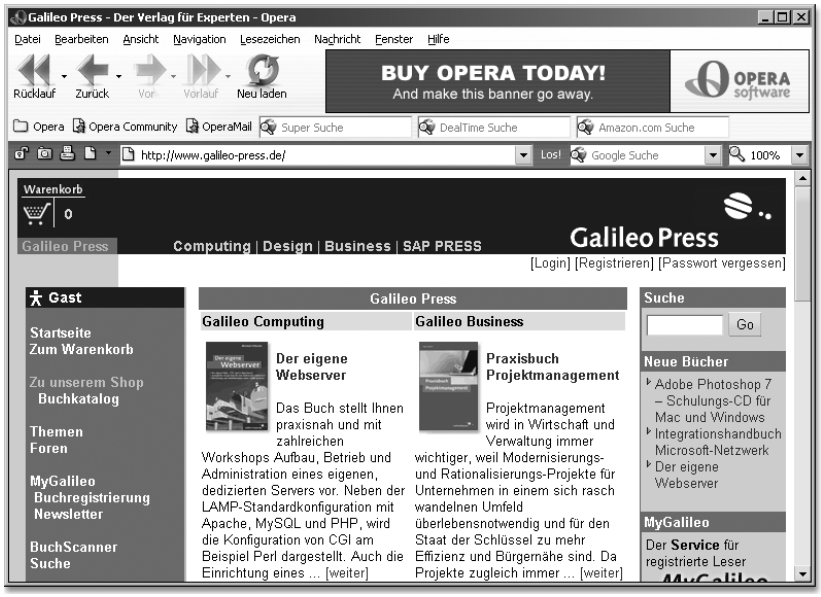

Abbildung 2.7 Der Opera-Browser

Eine »Besonderheit« des Opera ist, dass der Browser zwar kostenlos ist, aber immer ein Werbebanner rechts oben sichtbar ist. Wer eine werbefreie Version will, muss zurzeit 39 $ berappen.

2.1.4 Konqueror

Das freie Betriebssystem Linux gewinnt immer mehr an Fahrt. Zwar ist es von der oft beschworenen Bezwingung von Microsoft Windows auf dem Desktop noch sehr weit entfernt, aber trotzdem ist Linux »in«, und auf Servern hat es Microsoft bereits abgehängt.

Für die grafische Darstellung der Desktop-Oberfläche ist der so genannte Fenstermanager zuständig. Hier gibt es immer wieder Kleinkriege innerhalb der Linux-Gemeinde, welches System denn nun das beste sei, aber auch hier gibt es einen Marktführer: KDE (**http://www.kde.org/**). Ein Teil des Systems ist ein eigener Webbrowser, der Konqueror. Positiv für JavaScript-Entwickler ist, dass die Skriptsprache von Konqueror überaus gut unterstützt wird. Aufgrund des in Zukunft wohl noch weiter steigenden Marktanteils dieses Browsers ist also ein Linux-System Pflicht. Für ganz Faule gibt es sogar Linux-Distributionen, die von CD laufen und keine Installation benötigen. Bekanntestes Beispiel hierfür ist Knoppix (**http://www.knoppix.de/**). Für ernst zu nehmende Tests ist freilich eine »stationäre« Linux-Installation Pflicht. Unter **http://www.linuxiso.org/** finden Sie CD-Images einiger Linux-Distributionen, oder Sie kaufen gleich eine Distribution inklusive Handbuch und Installationssupport.

Abbildung 2.8 Linux-Browser Konqueror, Teil des KDE

2.1.5 Safari

Die Beziehung zwischen Microsoft und Apple ist äußerst undurchsichtig. Apple-Chef Steve Jobs präsentierte vor ein paar Jahren Bill Gates als Retter seines Unternehmens (Microsoft erstand damals Apple-Anteile und spülte so Geld in Jobs´ arg leere Kassen), allerdings ist Microsoft einer der ärgsten Hauptkonkurrenten auf dem Markt. So passt es auch, dass der Internet Explorer bei Mac OS sowie OS X automatisch mit dabei ist, Apple aber seit 2003 ebenfalls einen eigenen Webbrowser anbietet. Sein Name ist Safari, und er steht nur für OS X zur Verfügung. Der Grund: OS X basiert auf Unix, die Nähe zu Linux ist insofern offensichtlich. Apple hat nun keinen neuen Browser von Grund auf programmiert – dies wäre in Anbetracht des technologischen Vorsprungs der Konkurrenz auch ein unkalkulierbares Risiko –, sondern hat einfach den Konqueror auf OS X portiert! So profitieren gleich zwei Gruppen von dieser Entwicklung: Das Konquerer-Projekt (bzw. die KDE-Entwickler) profitiert von Fixes und unter Umständen auch von veröffentlichten Erweiterungen der Apple-Techniker am Konqueror; und Apple selbst freut sich natürlich über einen verfügbaren, frei einzusetzenden Webbrowser sowie über eine große Anzahl von Programmierern, die den Konqueror gratis weiterentwickeln. Die Macintosh-Plattform ist, allen Unkenrufen zum Trotz, nicht totzukriegen und insbesondere in Agenturen sehr weit verbreitet. Der Safari ist also ein Webbrowser, den es – wie den Konqueror auch – zu beobachten gilt.

Abbildung 2.9 Apples eigener Browser: Safari (auf Konqueror-Basis)

Von diesen fünf Browsern einmal abgesehen, gibt es nur noch wenige Browser mit einer nennenswerten JavaScript-Unterstützung. Sun hat den HotJava-Browser im Angebot, einen komplett Java-basierten Webbrowser, der allerdings überhaupt keine nennenswerte Verbreitung (und auch keine besonders nennenswerte JavaScript-Unterstützung) aufweist. Fazit: Die zuvor genannten Webbrowser werden für ein professionelles Testsystem benötigt, weitere noch nicht, zumindest zu diesem Zeitpunkt noch nicht.

2.1.6 Marktanteile

»Glaube nie einer Statistik, die du nicht selbst gefälscht hast« – dieser Spruch wurde lange Zeit irrtümlich Winston Churchill zugeschrieben. Trotzdem ist an dieser Aussage etwas dran, denn je nach ideologischer Ausrichtung einer Website und deren Zielgruppe sind die Marktanteile von Browsern andere. Abhilfe schaffen hier unabhängige Instanzen, die global über das Internet untersuchen, welcher Webbrowser welche Verbreitung hat.

In Deutschland ist hier insbesondere die Unternehmensberatung Fittkau & Maaß zu nennen, die regelmäßig die W3B-Umfrage (**http://www.w3b.de/**) durchführt und dabei auch die Browserverteilung untersucht. Im März 2003 ergab diese Untersuchung die folgenden Marktanteile:

Browser	Marktanteil
Internet Explorer	90,8%
Netscape/Mozilla	7,7%
Andere	1,5%

Tabelle 2.1 Browsermarktanteile gemäß W3B[1]

Interessant sind hier zusätzliche Details: Der Mozilla hat demnach nur eine Verbreitung von 0,82%, und mehr als die Hälfte aller Netscape-Nutzer verwenden die »uralte« Version 4 (also kann von einem Netscape-4-Anteil von mindestens 4% ausgegangen werden; bei Netscape 6/7 sind es weniger als 4%). Der alte Netscape ist also weiter verbreitet als die neuen Versionen! Deswegen wird auch in diesem Buch großes Augenmerk darauf gelegt, dass möglichst alle Beispiele gerade im Netscape 4 problemlos laufen, allen Makeln und Mankos dieses Browsers zum Trotz. Wer sich damit schmückt, neuere Netscapes zu unterstützen, aber dabei die alten vernachlässigt, ignoriert die tatsächlich vorhandenen Marktgegebenheiten.

Während W3B sich auf deutsche Websurfer beschränkt, führt OneStat.com (**http://www.onestat.com/**) weltweite Untersuchungen durch. Ende Juli 2003 veröffentlichte OneStat.com folgende Zahlen:

1 Quelle: http://www.heise.de/newsticker/data/anw-06.03.03-001/

Browser	Marktanteil
Internet Explorer	95,4%
Netscape	2,5%
Mozilla	1,6%
Andere	2,1%

Tabelle 2.2 Browsermarktanteile gemäß OneStat.com[2]

Auch hier ist eine weitere Aufschlüsselung der Ergebnisse interessant. Der Internet Explorer 6 hat einen Marktanteil von 66,3%, vor den Versionen 5.5 (14,5%) und 5.0 (12,7%). Dies zeigt, dass auch ältere Versionen des IE noch betrachtet werden müssen (später hierzu mehr). Der alte IE4 ist kaum mehr messbar (0,8%), liegt aber immer noch vor seinem einstigen Konkurrenten, dem Netscape 4 (0,6%). Interessanterweise ist davon die Rede, dass Netscape insgesamt auf 2,5% kommt, dass der Netscape 4 aber der beliebteste aller Netscape-Browser sei. So ist es fraglich, wie die 2,5% erreicht werden. Es lässt sich aber auch aus dieser Statistik schließen, dass der Netscape 4 mit den neueren Netscape-Versionen zumindest gleichauf liegt. Extrem klein ist noch der Marktanteil des Safari-Browsers (0,25%), aber dieser steht ja erst am Anfang. Ebenfalls nur geringe Marktanteile weist der Opera auf (0,6% für Version 6, andere Versionen werden nicht genannt).

OnStat.com und W3B versuchen jeweils, einen repräsentativen Querschnitt zu ermitteln. Bei Special-Interest-Seiten sehen die Marktanteile natürlich anders aus. Wenn Sie beispielsweise ein Info-Portal für Linux-Nutzer betrachten, wird dort der Konqueror-Anteil höher und der Anteil des Internet Explorer niedriger sein; auf der Microsoft-Homepage sieht das sicher ganz anders aus. Die Zahlen von OneStat.com und W3B dagegen entsprechen wohl dem tatsächlichen Querschnitt.

Ein Makel haben diese Erhebungen trotzdem. Viele (schlechte) JavaScript-Programmierer setzen für ihre Websites den Internet Explorer voraus und ignorieren »kleinere« Browser. Aus diesem Grund bieten beispielsweise Konqueror und Opera an, dass der Browser der Website gegenüber als Internet Explorer ausgegeben wird. Teilweise ist es sogar möglich, die Browser-Identifikation (also wie der Browser heißt und welche Versionsnummer er hat) selbst, als Benutzer, anzugeben. Wenn dies nicht erkannt und abgefangen wird, verfälscht eine solche Angabe natürlich die Statistik zugunsten des Internet Explorer. An der Führungsposition des Microsoft-Browsers besteht natürlich trotzdem kein Zweifel.

2 Quelle: http://www.onestat.com/html/aboutus_pressbox23.html

2.1.7 Testsystem

Um es noch einmal zu betonen: Je mehr Browser unterstützt werden, desto weniger Nutzer suchen enttäuscht oder verärgert das Weite. Im Verlauf dieses Buchs werden Sie sehen, dass es gar nicht so schwer ist, eine solche Unterstützung umzusetzen. Voraussetzung ist jedoch dafür, dass Sie Zugriff auf entsprechende Testsysteme haben.

Windows ist natürlich Pflicht. Dort können zumindest die Netscape-Versionen 4, 6, 7 parallel installiert werden, zusätzlich noch der jeweils aktuelle Mozilla (er ist meist dem Netscape mindestens um eine Nasenlänge voraus). Die Netscape-Versionen 1 bis 3 sind ebenfalls noch erhältlich (z.B. im Browser-Archive von evolt.org unter **http://browsers.evolt.org/?navigator/**), allerdings ist ihr Marktanteil so stark gesunken, dass es mittlerweile nicht mehr notwendig ist, sie zu unterstützen. Netscape 4 allerdings ist – wie zuvor gesehen – mit der wichtigste Testbrowser neben dem Internet Explorer; aus Gründen der Stabilität und zum Schutz Ihrer Nerven sollten Sie Version 4.8 verwenden.

Der Internet Explorer ermöglicht keine parallele Installation diverser Versionen. Außerdem werden beispielsweise für den Internet Explorer 5 keine Updates mehr angeboten (außer, Sie setzen Windows 2000 ein), und der Support (einschließlich Updates) für Internet Explorer 5.5 wird voraussichtlich Ende 2003 eingestellt. Dennoch zeigen die Statistiken deutlich, dass die Versionen 5 und 5.5 noch eine relevante Marktposition haben. Deswegen sollten Sie hierfür ein extra Testsystem (oder eine zweite Windows-Partition) einrichten. Bedenken Sie aber, dass ein »Downgrade« nicht möglich ist. Windows XP beispielsweise wird mit dem Internet Explorer 6 ausgeliefert, hier können Sie keine Vorgängerversion installieren.

Wenn Sie den Internet Explorer 4 installiert haben, können Sie die Version 5 oder 5.5 parallel installieren. Dazu müssen Sie bei den Installationsoptionen angeben, dass Sie den IE4 weiterhin im Kompatibilitätsmodus ausführen möchten. Ab dem Internet Explorer 6 ist dieser Modus nicht mehr möglich. Wenn Sie die Möglichkeit haben und ohnehin ein Testsystem für IE5 oder IE5.5 einrichten, sollten Sie sich aber diesen minimalen Aufwand leisten und Ihre JavaScript-Programme auch auf dem alten Internet Explorer 4 testen. Der uralte Internet Explorer 3 kann übrigens auch parallel installiert werden – in der 16-Bit-Version (Windows 3.1x). Dieser Aufwand ist allerdings heutzutage wahrlich nicht mehr notwendig.

Auf einem Macintosh-System ist natürlich der IE Pflicht, ebenfalls die Mac-Versionen von Netscape und Mozilla und neuerdings auch der Safari. Zusätzlich ist noch ein Linux-System empfehlenswert, auf dem neben den Netscape-Derivaten vor allem der Konqueror Beachtung verdient. Sie benötigen hierzu aber unbedingt den KDE als Fenstermanager.

Sie sehen also: Es ist eine Menge Aufwand erforderlich, aber es lohnt sich, um einen möglichst großen Besucherstamm bedienen zu können. Auch bei

der Erstellung dieses Buches wurde auf ein umfangreiches Testsystem zurückgegriffen. Sie erhalten also eine Reihe von Tipps, wie Sie Kompatibilitätsprobleme umschiffen können.

2.2 Verwendung von <script>

Nun aber endlich zur Erstellung von JavaScript. Wie bereits erwähnt, wird JavaScript in HTML integriert; Sie arbeiten also hauptsächlich mit HTML-Dateien, die Sie in einem einfachen Texteditor erstellen können.

JavaScript-Kommandos können an mehreren Stellen einer HTML-Datei untergebracht werden:

- zwischen den Tags `<script>` und `</script>`
- in einer externen Datei
- in Form eines HTML-Links
- als Parameter von HTML-Tags

In den folgenden Abschnitten werden die einzelnen Möglichkeiten der Reihe nach vorgestellt und erläutert.

Als Beispiel hierzu dient die Anweisung `document.write("The weather means the seasons")`, die den Text `"The weather means the seasons"` ausgibt. Warum dieser Befehl so funktioniert, erfahren Sie in den nächsten Kapiteln; fürs Erste müssen Sie mir einfach vertrauen.

Die nahe liegendste Methode, JavaScript-Befehle auszuführen, besteht darin, das[3] `<script>`-Tag zu verwenden. Folgender Code sorgt dafür, dass »The weather means the seasons« ausgegeben wird:

```
<script>
document.write("The weather means the seasons");
</script>
```

Befehle werden in JavaScript untereinander – einer pro Zeile – dargestellt. Wenn Sie mehrere Kommandos in einer Zeile unterbringen wollen, müssen Sie die Anweisungen durch ein Semikolon voneinander trennen. Im Gegensatz zu anderen Programmiersprachen (beispielsweise Java) muss aber keineswegs jedes Kommando mit einem Strichpunkt enden. In der ersten Version der JavaScript-Sprachspezifikation war das Semikolon am Ende jeder Anweisung strikt vorgeschrieben. Inzwischen wurde aber davon Abstand genommen, und jeder JavaScript-Programmierer hat seinen eigenen Stil. Prinzipiell haben die Strichpunkte den Sinn, dem JavaScript-Interpreter (also dem Bestandteil des Browsers, der den JavaScript-Code ausführt) mitzuteilen, an welcher Stelle eine Anweisung endet. Es gibt auch Programmiersprachen, bei denen das Zeilenende das Ende einer Anwei-

3 Der Tag oder das Tag? Manche Autoren bevorzugen »das Tag«, um eine eindeutige Abgrenzung vom (Wochen-)Tag zu gewährleisten; andere bevorzugen »der Tag«. Ich habe mich auf die Seite der Mehrheit geschlagen: »das Tag«.

sung markiert. In JavaScript ist beides möglich. Um den Code sauber zu halten und um bei Programmierfehlern schneller die Fehlerquelle zu finden, verzichte ich **nicht** auf optionale Strichpunkte. Es ist Ihnen aber natürlich freigestellt, sich einen anderen Stil anzueignen – insbesondere, wenn Sie bereits Erfahrungen in einer Programmiersprache gesammelt haben, in der keine Strichpunkte vorkommen (z. B. Visual Basic/VBScript/VB.NET).

Die beiden folgenden Anweisungen sind also äquivalent. Einmal stehen die beiden Befehle in verschiedenen Zeilen, einmal in einer Zeile.

```
<script>
document.write("The weather ");
document.write("means the seasons");
</script>
```

und

```
<script>
document.write("The weather "); document.write("means the
   seasons");
</script>
```

JavaScript-Code wird hierbei vom JavaScript-Interpreter des verwendeten Browsers ausgeführt. Betrachten Sie zum Beispiel folgendes HTML-Dokument:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script>
document.write("The weather means the seasons");
</script>
</body>
</html>
```

Wenn es vom Browser interpretiert worden ist und dieser Browser JavaScript unterstützt, verhält es sich so wie folgendes HTML-Dokument:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
The weather means the seasons
</body>
</html>
```

Nehmen Sie es mir bitte nicht übel, wenn die ersten Beispiele in diesem Kapitel nicht unbedingt die breite Funktionspalette von JavaScript demonstrieren.

2.2.1 Das language-Attribut

Das obige Beispiel ist streng genommen etwas unsauber. Das `<script>`-Tag eignet sich auch für andere Programmiersprachen, die in HTML-Dokumente eingebettet werden können, beispielsweise für Visual Basic Script (VBScript) oder JScript. Dazu dient das Attribut `language` des `<script>`-Tags. Ist es nicht gesetzt – wie im obigen Beispiel –, so wird angenommen, dass die zwischen den Tags stehenden Kommandos in JavaScript verfasst wurden (deswegen funktioniert obiges Beispiel auch). Aber um auf Nummer Sicher zu gehen – es könnte ja sein, dass eine neue Version des Microsoft Internet Explorer als Standardsprache VBScript annimmt –, schreiben wir:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript">
document.write("The weather means the seasons");
</script>
</body>
</html>
```

Wie Sie in Kapitel 1 erfahren haben, gibt es mehrere Versionen von JavaScript. Sie können im `language`-Attribut auch explizit eine der Versionen angeben. Der folgende Code wird nur von Browsern ausgeführt, die JavaScript Version 1.1 unterstützen (das sind insbesondere der Netscape Navigator ab Version 3 und der Microsoft Internet Explorer ab Version 4):

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript1.1">
document.write("The weather means the seasons");
</script>
</body>
</html>
```

Ältere Browser, beispielsweise der Internet Explorer 3, ignorieren den JavaScript-Befehl und geben nichts aus.

Zur Zeit der Drucklegung (August 2003) sind die in Tabelle 2.3 aufgeführten Parameter gültig.

Parameter	Bedeutung
`JavaScript`	Jeder Browser, der JavaScript unterstützt
`JavaScript1.1`	Alle Browser, die mindestens die JavaScript-Version 1.1 unterstützen (ab NN3, IE4)
`JavaScript1.2`	Alle Browser, die mindestens die JavaScript-Version 1.2 unterstützen (ab NN4, IE5)
`JavaScript1.3`	Alle Browser, die mindestens die JavaScript-Version 1.3 unterstützen (ab Netscape Navigator 4.06, IE5)
`JavaScript1.4`	Ab Netscape 6
`JavaScript1.5`	Ab Netscape 6

Tabelle 2.3 Die Parameter für <script language="…">

Mit der folgenden HTML-Seite können Sie überprüfen, welche JavaScript-Versionen der jeweilige Browser unterstützt:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript">
document.write("Der Browser unterstützt JavaScript
   <hr>");
</script>
<script language="JavaScript1.1">
document.write("Der Browser unterstützt JavaScript
   v1.1<hr>");
</script>
<script language="JavaScript1.2">
document.write("Der Browser unterstützt JavaScript
   v1.2<hr>");
</script>
<script language="JavaScript1.3">
document.write("Der Browser unterstützt JavaScript
   v1.3<hr>");
</script>
<script language="JavaScript1.4">
document.write("Der Browser unterstützt JavaScript
   v1.4<hr>");
```

```
</script>
<script language="JavaScript1.5">
document.write("Der Browser unterstützt JavaScript
   v1.5<hr>");
</script>
</body>
</html>
```

In Abbildung 2.10 sehen Sie beispielsweise, was der Netscape Navigator 4.78 ausgibt.

Beachten Sie bei obigem Beispiel, dass man mit `document.write()` insbesondere auch HTML-Code, in diesem Fall das `<hr>`-Tag für eine horizontale Linie, ausgeben kann.

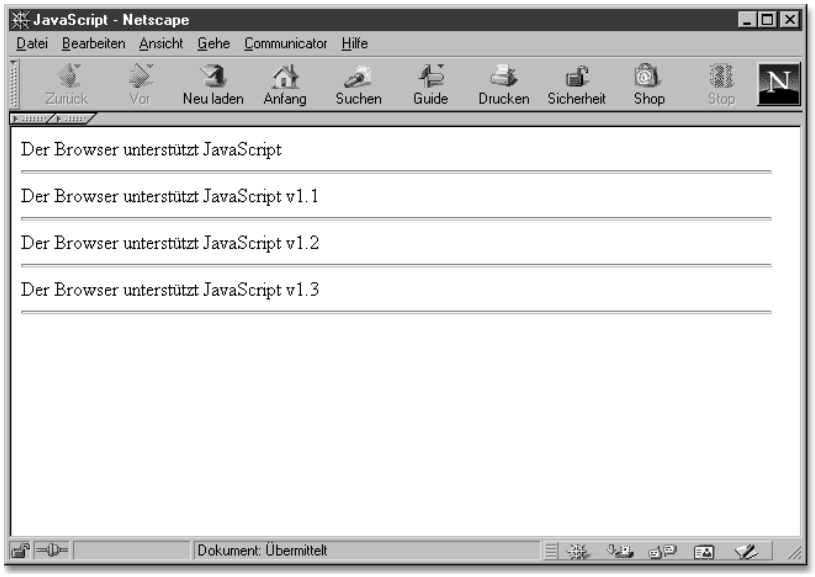

Abbildung 2.10 Die vom Netscape Navigator 4.78 unterstützten JavaScript-Versionen

In Tabelle 2.4 sehen Sie, welche Parameter ausgewählte Browserversionen erkennen bzw. unterstützen (Browserverfügbarkeit Stand August 2003):

Browser	Unterstützte Parameter
Netscape 4.00-4.05	JavaScript1.0 bis JavaScript1.2
Netscape 4.06-4.8	JavaScript1.0 bis JavaScript1.3
Netscape 6/7	JavaScript1.0 bis JavaScript1.5

Tabelle 2.4 Die von ausgewählten Browsern unterstützten Parameter <script language="...">

Browser	Unterstützte Parameter
`Internet Explorer 4`	JavaScript1.0 bis JavaScript1.2
`Internet Explorer 5`	JavaScript1.0 bis JavaScript1.3
`Internet Explorer 5.5`	JavaScript1.0 bis JavaScript1.3
`Internet Explorer 6`	JavaScript1.0 bis JavaScript1.3
`Opera 5.x/6/7`	JavaScript1.0 bis JavaScript1.4
`Konqueror/Safari`	JavaScript1.0 bis JavaScript1.4

Tabelle 2.4 Die von ausgewählten Browsern unterstützten Parameter <script language="…"> (Forts.)

Durch die Verwendung spezieller **language**-Attribute können Sie Fehlermeldungen vermeiden, die bei der Verwendung von zu modernen JavaScript-Kommandos erscheinen würden. Wenn Sie also Sprachelemente von JavaScript verwenden, die erst ab Version 1.1 unterstützt werden, sollten Sie das `language`-Attribut auf `"JavaScript1.1"` setzen; ältere Browser ignorieren dann die Befehle völlig.

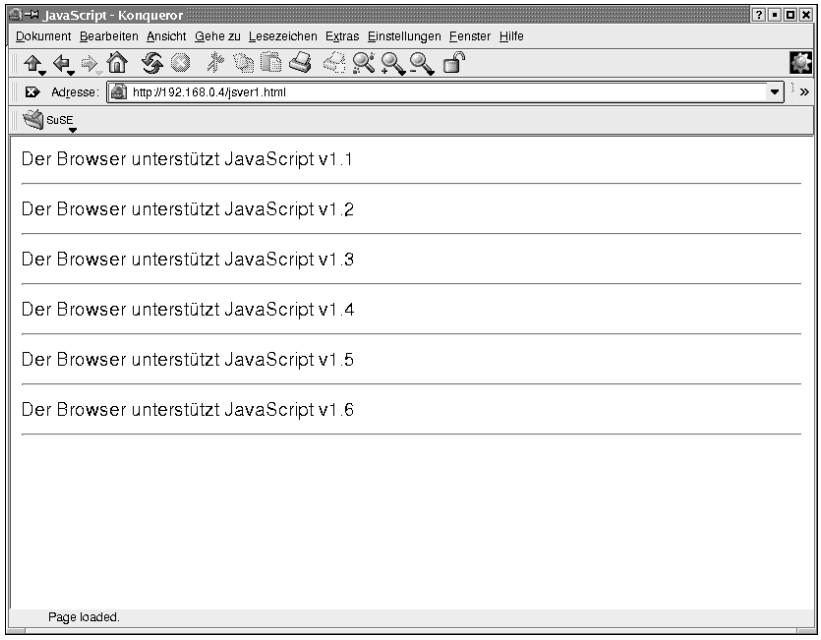

Abbildung 2.11 Konqueror kennt (angeblich) schon JavaScript 1.6!

Das stimmt leider nicht immer. Einige Versionen des Netscape Navigator 3 beispielsweise haben einen Bug und versuchen, auch JavaScript-Blöcke mit `language="JavaScript1.2"` und `language="JavaScript1.3"` zu inter-

pretieren. Auch ältere Versionen des Internet Explorer führen manchmal Code aus, der nicht für sie bestimmt ist. Besonders »schlimm« ist in diesem Zusammenhang jedoch Konqueror und damit auch Safari. Es scheint zu genügen, dass der Wert des `language`-Attributs mit `"JavaScript"` beginnt. Fügt man beispielsweise in obiges Listing noch einen Block mit `<script language="JavaScript1.6">` ein, würde auch dieser ausgeführt werden, obwohl es diese Sprachversion nicht gibt (siehe Abbildung 2.11).

2.2.2 Browser ohne JavaScript

So schön ein Programm auch auf dem eigenen Rechner laufen mag – es kommt darauf an, dass es beim Kunden und bei allen Besuchern (nun gut, sagen wir, bei den meisten Besuchern) der Website läuft. In vielen Firmennetzwerken ist es beispielsweise so, dass JavaScript aus Sicherheitsgründen nicht aktiviert werden darf. Die Beispiele von oben sehen dann so aus wie in Abbildung 1.12.

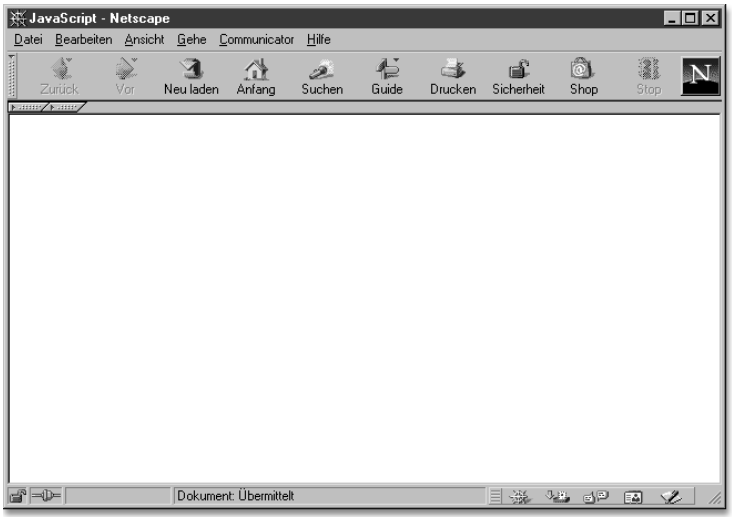

Abbildung 2.12 Die Ausgabe bei deaktiviertem JavaScript: Gähnende Leere

Bei ganz alten Browsern ist es noch schlimmer, zuweilen sieht man sogar den JavaScript-Code. Sie können sich aber hier mit einem kleinen Trick behelfen: Mit `<!--` wird ein HTML-Kommentar eingeleitet; alles dahinter wird vom HTML-Interpreter ignoriert, jedoch nicht vom JavaScript-Interpreter! Es empfiehlt sich also, JavaScript-Code durch folgenden Befehl einzuleiten:

```
<script language="JavaScript"><!--
```

Nun stellt sich die Frage, wie der HTML-Kommentar beendet wird. Probieren Sie einmal die einfachste Variante aus, nämlich einfach ein Kommentar-Ende-Tag (`-->`) vor dem `</script>`:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript"><!--
document.write("The weather means the seasons");
--></script>
</body>
</html>
```

Wenn Sie diese Seite im Browser laden, sollten Sie eine Fehlermeldung erhalten. Der Grund: Der JavaScript-Interpreter interpretiert `-->` als JavaScript-Befehl und liefert eine Fehlermeldung.

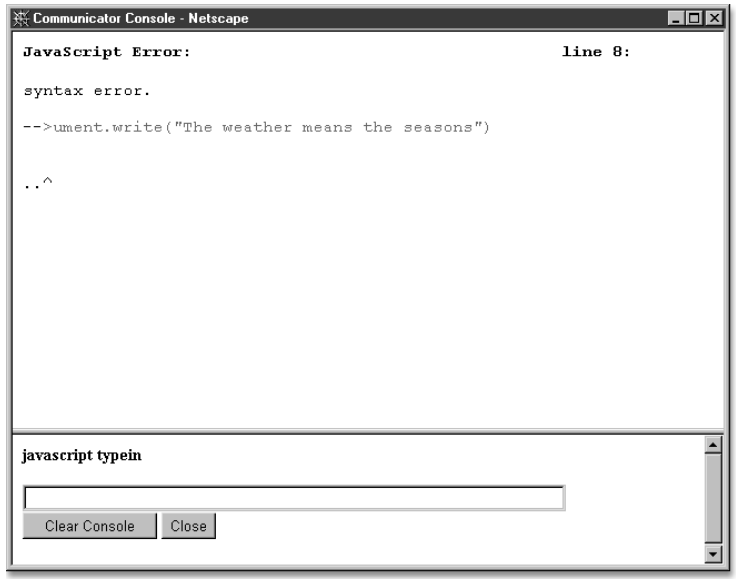

Abbildung 2.13 Fehlermeldung des Netscape Navigator

Aber auch hier gibt es einen kleinen Trick, mit dem Sie dieses Hindernis aus dem Weg räumen können. Mit `//` leitet man einen Kommentar im JavaScript-Code ein. Die Verwendung von Kommentaren ist bei der Programmierung sehr wichtig, damit man auch Wochen später noch weiß, was man damals eigentlich beabsichtigt hat, und damit auch andere Menschen mit dem Code arbeiten können. Es gibt zwei Arten von Kommentaren:

- `//`: Hiermit wird ein einzeiliger Kommentar eingeleitet; alles hinter den beiden Querstrichen in der jeweiligen Zeile wird vom JavaScript-Interpreter ignoriert.
- `/*` und `*/`: Hiermit wird ein mehrzeiliger Kommentar eingeleitet; alles nach `/*` wird vom JavaScript-Interpreter ignoriert, bis die Zeichenfolge `*/` kommt und den Kommentar abschließt.

Im folgenden Listing sehen Sie Beispiele für Kommentare:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript">
// Hier wird auf generelle JavaScript-Unterstützung geprüft
document.write("Der Browser unterstützt
JavaScript<hr>");
</script>
<script language="JavaScript1.1">
/* Hier geht es um JavaScript 1.1 */
document.write("Der Browser unterstützt JavaScript
   v1.1<hr>");
</script>
<script language="JavaScript1.2">
// JavaScript 1.2 wird überprüft,
// und zwar gründlich
document.write("Der Browser unterstützt JavaScript
   v1.2<hr>");
</script>
<script language="JavaScript1.3">
/* JavaScript 1.3 wurde
   mit dem Netscape Navigator 4.06 eingeführt */
document.write("Der Browser unterstützt JavaScript
   v1.3<hr>");
</script>
<script language="JavaScript1.4">
document.write("Der Browser unterstützt JavaScript
   v1.4<hr>");
</script>
<script language="JavaScript1.5">
document.write("Der Browser unterstützt JavaScript
   v1.5<hr>");
</script>
</body>
</html>
```

Kommen wir zum ursprünglichen Problem zurück. Der Browser gibt eine Fehlermeldung aus, weil `-->` als JavaScript-Code interpretiert wird und nicht als HTML-Element. Wenn dem `-->` jedoch ein `//` vorangestellt wird, ignoriert der JavaScript-Interpreter diesen Code, der HTML-Interpreter jedoch stellt fest, dass der Kommentar zu Ende ist. Folgender Code wird von Browsern, die JavaScript unterstützen, ausgeführt; Browser, die kein JavaScript unterstützen oder bei denen JavaScript deaktiviert ist, sehen einen HTML-Kommentar, ignorieren das Innere und geben folglich nichts aus, auch keinen puren JavaScript-Code.

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript"><!--
document.write("The weather means the seasons");
//--></script>
</body>
</html>
```

Browser und Textausgabe

Sie sehen an den obigen Beispielen, dass es relativ einfach ist, bei Browsern, die JavaScript unterstützen, einen Text auszugeben. Der andere Weg ist aber auch möglich. Es gibt hierfür (ab Netscape Navigator 3 bzw. Internet Explorer 3) ein besonderes HTML-Element, `<noscript>`, das so ähnlich wie `<noframes>` funktioniert. Damit sind folgende Szenarien denkbar:

- Der Browser unterstützt kein JavaScript, egal, ob das `<noscript>`-Tag bekannt ist oder nicht. Es wird notfalls ignoriert, und die darauf folgenden HTML-Elemente werden interpretiert (bzw. der Inhalt wird angezeigt).
- Der Browser unterstützt JavaScript, und es ist auch eingeschaltet. Dann wird alles, was zwischen `<noscript>` und `</noscript>` steht, nicht dargestellt.
- Der Browser unterstützt JavaScript, es ist jedoch ausgeschaltet. Dann wird auch all das, was zwischen `<noscript>` und `</noscript>` steht, dargestellt.

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript"><!--
document.write("The weather means the seasons");
//--></script>
```

```
<noscript>
Ihr Browser kann mit JavaScript nichts anfangen, oder es ist
   ausgeschaltet!
</noscript>
</body>
</html>
```

Denken Sie immer auch an diejenigen Besucher, die JavaScript deaktiviert haben oder deren Browser (man denke nur an die eingeschränkten Browser von Handhelds) kein JavaScript unterstützt. Erstellen Sie notfalls eine Version Ihrer Website, die auch ohne JavaScript funktioniert.

Abbildung 2.14 zeigt, dass das tatsächlich funktioniert: Sie sehen obiges Dokument im klassischen Text-Webbrowser Lynx.

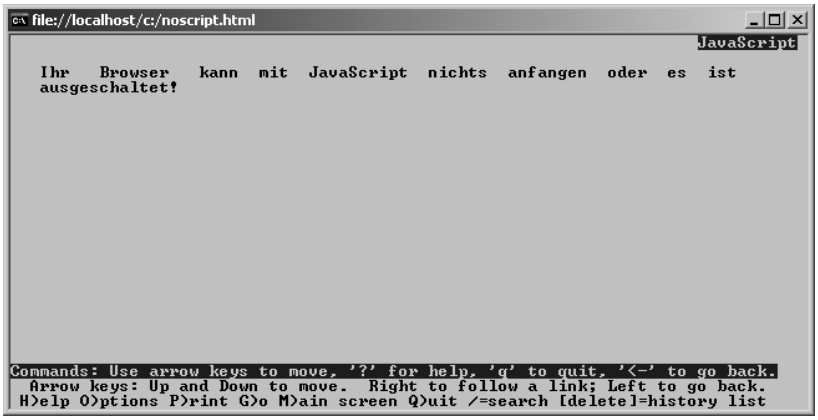

Abbildung 2.14 Kein JavaScript beim Text-Browser Lynx

2.2.3 Externe Dateien

Wenn Sie später einmal JavaScript-Programme schreiben, die auf mehreren unterschiedlichen Seiten benötigt werden, wäre es eigentlich ziemlich töricht, dasselbe Skript in mehrere Seiten zu kopieren – der Aufwand bei Änderungen am Skript wäre beträchtlich, da mehrere Dateien geöffnet und geändert werden müssten.

Es gibt hier auf den ersten Blick einen Ausweg, der aber auf den zweiten Blick auch seine Nachteile hat. Zuerst zur grauen Theorie: Man kann beim `<script>`-Tag im Attribut `src` den Namen einer externen Datei mit JavaScript-Kommandos angeben. Als Dateiendung hat sich hierbei **.js** durchgesetzt. Es empfiehlt sich, für alle externen Dateien ein eigenes Verzeichnis anzulegen, damit diese alle gesammelt an einem zentralen Ort zu finden sind.

Angenommen, folgende Datei ist auf dem Webserver im virtuellen Verzeichnis **js** unter dem Namen **weather.js** gespeichert:

```
//erste externe JavaScript-Datei
document.write("The weather means the seasons");
```

Sie wird folgendermaßen in ein HTML-Dokument eingebunden, um dieselbe Wirkung zu erzielen wie das Dokument aus dem vorigen Beispiel:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript" src="/js/weather.js">
</script>
</body>
</html>
```

Natürlich kann auch hier das `language`-Attribut gesetzt werden. Bei dem folgenden Dokument wird nur etwas ausgegeben, wenn der Browser JavaScript Version 1.1 unterstützt:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript1.1" src="/js/weather.js">
</script>
</body>
</html>
```

Dieses Vorgehen birgt aber auch einen kleinen Fallstrick: Probieren Sie doch einmal Folgendes in Ihrem Browser aus:

```
<html>
<body>
<script language="JavaScript" src="/js/weather.js"><!--
document.write("<br>Invincibility is in oneself,
vulnerability is in the opponent");
//--></script>
</body>
</html>
```

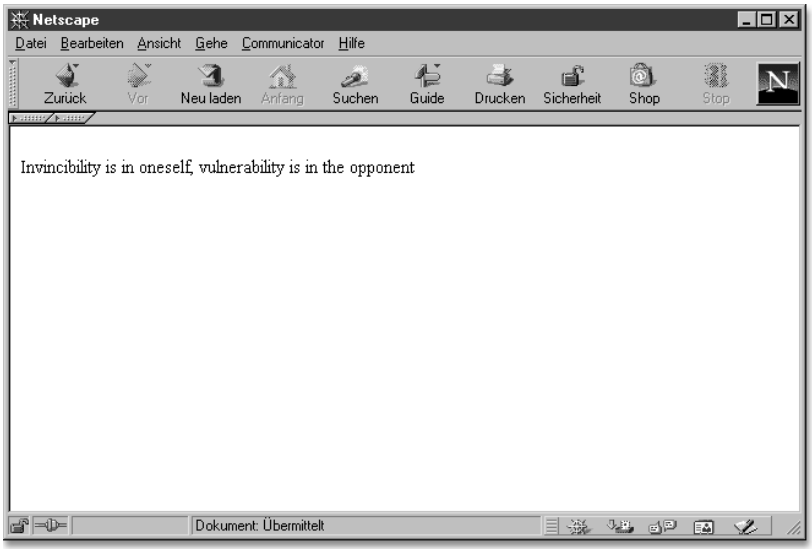

Abbildung 2.15 Der Text wird nur einmal angezeigt.

Das Ergebnis sehen Sie in Abbildung 2.15: Der Inhalt des `<script>`-Elements wird ignoriert. Der Grund: Ist das `src`-Attribut des `<script>`-Tags gesetzt, wird eingeschlossener JavaScript-Code nicht betrachtet; ist `src` nicht gesetzt, so wird der eingeschlossene Code ausgeführt. Um also beide Sätze auszugeben, muss das HTML-Dokument folgendermaßen abgeändert werden:

```
<html>
<body>
<script language="JavaScript" src="/js/weather.js"></script>
<script language="JavaScript"><!--
document.write("<br>Invincibility is in oneself,
   vulnerability is in the opponent");
//--></script>
</body>
</html>
```

Obwohl externe Dateien sehr praktisch und auch recht weit verbreitet sind, haben sie einen Nachteil, den viele gar nicht kennen: Der Netscape Navigator 2 kann mit dem `src`-Attribut nichts anfangen, und der Internet Explorer 3 hat immer wieder Probleme damit (und im lokalen Betrieb, also ohne Webserver, funktioniert es überhaupt nicht). Die ersten Versionen des Internet Explorer 3 haben dieses Feature gar nicht unterstützt, während es in den Versionen 3.02 und 3.03 hin und wieder funktioniert. Es hilft manchmal, eine vollständige URL anzugeben (also `src="http://www.ihrefirma.de/skript.js"`), aber auch das funktioniert hin und wie-

der nicht. Setzen Sie also Ihr Skript für den Internet Explorer 3 in das HTML-Dokument selbst, oder testen Sie es intensiv. Kontaktieren Sie außerdem den Betreiber Ihres Webservers; er muss ihn so konfigurieren, dass Dateien mit der Endung **.js** mit dem MIME-Typ **application/x-javascript** verknüpft werden.

Sollten in diesem Kapitel externe Dateien verwendet werden, so dient das in der Regel der Übersichtlichkeit des Codes. Natürlich muss verhindert werden, dass ältere Browser eine Fehlermeldung ausgeben. Da der Internet Explorer 3 immer seltener verwendet wird (ohnehin nur noch auf unter Windows 95 laufenden Systemen möglich), verliert auch diese »Gefahr« zunehmend an Brisanz.

2.3 JavaScript-Links

JavaScript-Befehle werden oft aufgrund von Benutzereingaben ausgeführt. Eine Möglichkeit besteht darin, eine Aktion durch einen Mausklick zu starten. Bevor ein paar Beispiele vorgestellt werden, muss noch ein neuer JavaScript-Befehl eingeführt werden. Mit `window.alert("Invincibility is in oneself, vulnerability is in the opponent")` wird ein modales Fenster ausgegeben, das den Text »Invincibility is in oneself, vulnerability is in the opponent« anzeigt. Je nach verwendetem Browser sieht das etwas anders aus; in Abbildung 2.16 sehen Sie die Darstellung im Netscape Navigator 4 und in Abbildung 2.17 im Safari-Browser. Sie sehen daran, dass Sie auf das grafische Layout keinen Einfluss nehmen können, denn das wird vom jeweiligen Browser übernommen.

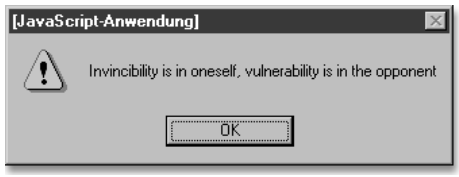

Abbildung 2.16 Ein Warnfenster mit dem Netscape Navigator 4

Abbildung 2.17 Dieselbe Meldung, diesmal im Mac-Browser Safari

Protokolle HTML-Links können auf URLs mit den verschiedensten Protokollen verweisen, so zum Beispiel **http:**, **ftp:**, **news:** oder **mailto:**. Nun kommt ein

weiteres Protokoll hinzu, das jedoch lediglich im Zusammenhang mit JavaScript eine Bedeutung hat: **javascript:**. Folgender Link gibt »Invincibility is in oneself, vulnerability is in the opponent« in einem modalen Fenster aus, wenn man darauf klickt:

```
<A HREF="javascript:window.alert('Invincibility is in
oneself, vulnerability is in the opponent');">
```

Beachten Sie die Anführungszeichen. In JavaScript ist es prinzipiell egal, ob Sie einfache oder doppelte Anführungszeichen verwenden – Hauptsache, Sie hören so auf, wie Sie angefangen haben.

Folgendes ist also völlig korrekt:

```
document.write("The weather means the seasons");
document.write('<br>Invincibility is in oneself,
vulnerability is in the opponent');
```

Falsch ist dagegen:

```
document.write("The weather means the seasons ');
document.write('<br>Invincibility is in oneself,
vulnerability is in the opponent");
```

Obwohl man mit dem Pseudo-Protokoll **javascript:** bequem JavaScript-Befehle aufrufen kann, gibt es doch einen kleinen Nachteil: Browser, die kein JavaScript unterstützen, geben eine Fehlermeldung aus. Im folgenden Abschnitt wird diesem Missstand abgeholfen.

2.4 Event-Handler

Ohne anderen Kapiteln allzu sehr vorgreifen zu wollen – wenn Sie bestimmte Aktionen ausführen, beispielsweise die Maus bewegen oder auf einen Link klicken, tritt JavaScript-intern ein so genanntes **Ereignis** (engl. **event**) ein. Manche dieser Ereignisse können Sie mit JavaScript abfangen und darauf reagieren. Dazu benötigen Sie **Event-Handler**, die als Attribute mancher HTML-Tags in den HTML-Code eingebunden werden. Eine vollständige Auflistung aller Event-Handler finden Sie in der Referenz; davor werden die wichtigsten Handler in den jeweiligen Kapiteln anhand eines Beispiels vorgestellt.

Grundsätzlich gilt Folgendes: Ein Event-Handler beginnt stets mit `on`. Der Event-Handler, der zuständig ist, wenn auf einen Link geklickt wird, heißt `onclick`. Wenn folgender Link angeklickt wird, wird ein modales Fenster geöffnet:

```
<a href="irgendwohin.html" onclick="alert(
'Invincibility is in oneself, vulnerability is in the
   opponent');">Sun Tzu</a>
```

Vom Prinzip her wunderbar – Browser, die JavaScript unterstützen, führen den Code aus, folgen danach aber dem Link (in späteren Kapiteln dazu mehr). Ältere Browser »sehen« nur den Link und versuchen, das Ziel des Links aufzurufen. Das ist aber in der Regel nicht erwünscht. Das `href`-Attribut des Links muss gesetzt werden, sonst wird er nicht angezeigt. Es gibt aber eine Möglichkeit, einen Link anzugeben, der keine neue Seite lädt:

```
<a href="#" onclick="alert('Invincibility is in oneself,
vulnerability is in the opponent');">Sun Tzu</a>
```

Diesen Trick sollten Sie sich gut merken!

2.5 JavaScript-Entities

Die letzte Möglichkeit, um JavaScript-Code in das HTML-Dokument einzubetten, wird sehr selten verwendet, was unter anderem auch daran liegt, dass hier nur Werte von HTML-Attributen gesetzt werden können – und dass es nur mit dem Netscape Navigator funktioniert. Sie kennen bereits HTML-Entities, die mit `&` beginnen und mit `;` enden. Ein Beispiel ist die HTML-Entity für Ä, `&Auml;`. JavaScript-Entities sind ganz ähnlich aufgebaut: Sie beginnen mit `&`, enden mit `;`, und dazwischen steht ein JavaScript-Ausdruck. Für dieses Beispiel wird ein weiteres neues JavaScript-Kommando eingeführt (genauer: eine Objekteigenschaft, aber dazu später mehr): `location.protocol` gibt das verwendete Protokoll der aktuellen Seite vor, also beispielsweise `file:` bei lokalen Dateien und `http:` bei Dateien aus dem World Wide Web. In dem folgenden Dokument wird das `value`-Attribut eines Texteingabefelds auf den Wert von `location.protocol` gesetzt. Die JavaScript-Entity heißt `&{location.protocol};` – beachten Sie insbesondere, dass der JavaScript-Ausdruck in geschweifte Klammern eingeschlossen werden muss!

```
<html>
<head>
<title>JavaScript-Entities</title>
</head>
<body>
<form>
Protokoll: <input type="text"
value="&{location.protocol};">
</form>
</body>
</html>
```

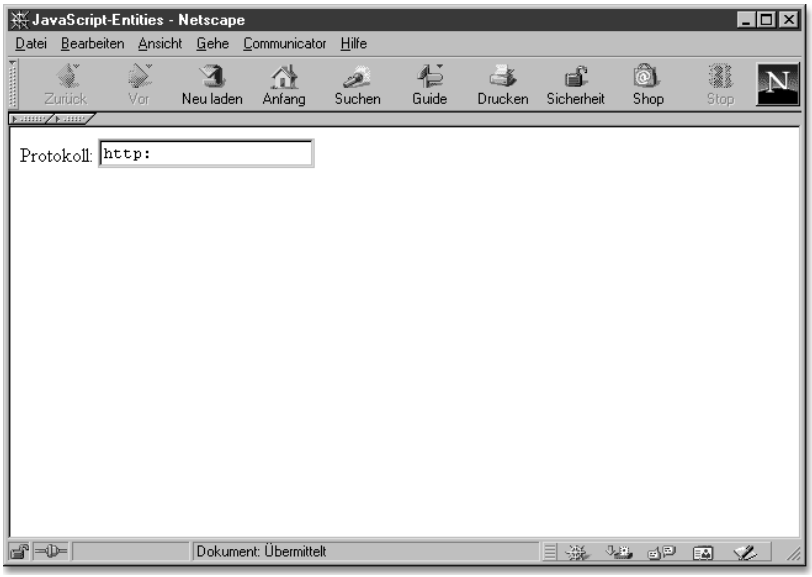

Abbildung 2.18 HTML-Attribute setzen mit Entities

3 Programmierung I

Es war einmal vor langer Zeit,
und diese Zeit ist schon lange, lange her,
etwa letzten Freitag,
als Winnie-der-Pu ganz allein
unter dem Namen Sanders in einem Wald wohnte.
– A. A. Milne, Pu der Bär, deutsche Übersetzung von Harry Rowohlt

Gut gerüstet

In diesem und dem folgenden Kapitel werden die Grundprinzipien der Programmierung an sich und der Programmierung mit JavaScript im Besonderen erläutert. An Praxisbeispielen ist dieses Kapitel recht arm, aber Sie werden die hier vorgestellten Techniken in den weiteren Kapiteln noch des Öfteren benötigen. Sie erhalten hier das Rüstzeug, das die Grundlage für alle kommenden Kapitel ist.

Grundbegriffe der Programmierung werden hier ebenfalls erläutert. Leser, die bereits Erfahrungen mit der einen oder anderen Programmiersprache haben, mögen diesen kleinen Exkurs verzeihen, aber so soll auch Neulingen die Chance geboten werden, in die Materie einzusteigen. Allerdings sind die Erklärungen recht knapp gehalten, und nur das Nötigste wird hierzu erläutert. Schließlich geht es ja darum, möglichst schnell brauchbare Anwendungen zu schreiben.

3.1 Variablen

Bei der Programmierung müssen immer wieder Daten zwischengespeichert werden. Hierzu bedient man sich so genannter **Variablen** (in manchen Büchern – aber sehr wenigen – werden sie auch als »Veränderliche« bezeichnet).

3.1.1 Namensgebung

Jede Variable wird anhand ihres Namens angesprochen. Bei der Namensgebung haben Sie größtenteils freie Hand. Ein Variablenname besteht aus einer Folge von Buchstaben, Ziffern und dem Unterstrich (_). Das erste Zeichen darf jedoch keine Ziffer sein. Außerdem wird zwischen Groß- und Kleinschreibung unterschieden. Die Bezeichner `Pooh`, `pooh` und `POOH` sind also verschiedene Variablen. Beispiele für Variablennamen sind etwa:

- `Winnie_The_Pooh`
- `WinnieThePooh`
- `Winnie2002`
- `_Winnie_`
- `Honigtopf`

Unbrauchbar sind dagegen die folgenden Namen:

- `1Winnie` (beginnt mit einer Ziffer)
- `Winnie Pooh` (Leerzeichen)
- `Winnie-Pooh` (Bindestrich)
- `Honigtöpfe` (Umlaut)

JavaScript-Schlüsselwörter und -begriffe dürfen Sie nicht als Variablennamen verwenden. Ein Beispiel hierfür ist etwa `alert`, das Sie im vorigen Kapitel schon einmal in Aktion gesehen haben.

Um einer Variablen einen Wert zuzuweisen, wird das Gleichheitszeichen verwendet. Links vom Gleichheitszeichen steht der Variablenname, rechts davon der neue Wert (oder eine andere Variable, deren Wert dann zugewiesen wird).

3.1.2 Numerische Variablen

Es gibt verschiedene Typen von Variablen. Zahlenwerte werden wie im amerikanischen Zahlensystem mit einem Dezimalpunkt statt einem Dezimalkomma angegeben:

```
Pi = 3.14159265;
Mauerfall = 1989;
MinusHundert = -100;
```

3.1.3 Zeichenketten

Sehr oft werden Zeichenketten, auch **Strings** genannt, verwendet. Die zweite Bezeichnung kommt daher, dass es ein so benanntes Objekt in JavaScript gibt, aber dazu später mehr. In diesem Buch werden beide Begriffe äquivalent verwendet.

Ein String wird von Anführungszeichen eingeschlossen, entweder von einfachen (Apostrophen) oder doppelten. Hierbei ist zu beachten, dass unbedingt gerade Anführungszeichen (⇧+2) und Apostrophe (⇧+#) verwendet werden. Im Gegensatz zu beispielsweise Perl oder PHP ist es hier egal, ob einfache oder doppelte Anführungszeichen verwendet werden, Hauptsache, die Zeichenkette wird mit derselben Art von Anführungszeichen beendet, wie sie eingeleitet worden ist.

```
WinniesFreund = "Piglet";
WinniesAndererFreund = 'Tigger';
```

Die verschiedenen Anführungszeichen haben unter anderem den folgenden Sinn: Wenn Sie beispielsweise ein Apostroph in einer Zeichenkette verwenden wollen, können Sie diese Zeichenkette ja schlecht mit Apostrophen eingrenzen, da der JavaScript-Interpreter dann nicht weiß, wo die Zeichenkette aufhört. In diesem Fall müssen Sie die andere Sorte von Anführungszeichen verwenden:

```
WinnieSagt = "It's me, Winnie the Pooh";
```

Wenn man aber beispielsweise beide Arten von Anführungszeichen in einer Zeichenkette verwenden muss, kommt man in Schwierigkeiten. Hier hilft der Backslash (\) weiter. Das Zeichen, das dem Backslash folgt, wird »entwertet«, d. h. es nimmt in der Zeichenkette keine besondere Bedeutung ein. Beim Anführungszeichen oder Apostroph bedeutet das: Die Zeichenkette wird hiermit nicht beendet.

```
WinnieSagt = 'It\'s me, Winnie the Pooh';
```

Wenn man nun den Backslash selbst in der Zeichenkette verwenden will, muss man auch ihn entwerten:

```
WindowsVerzeichnis = "C:\\WINDOWS";
```

Mit dem Backslash können auch einige besondere Zeichen dargestellt werden. Tabelle 3.1 zeigt eine Übersicht:

Ausdruck	Bedeutung
\r	Wagenrücklauf
\n	Neue Zeile
\t	Tabulator
\b	Backspace (Löschtaste)
\f	Seitenvorschub

Tabelle 3.1 Sonderzeichen innerhalb von Zeichenketten

3.1.4 Boolesche Variablen

Oft ist man nicht an einem Zahlenwert oder einem String interessiert, sondern an einem Wahrheitswert, also wahr oder falsch. Man spricht hier auch von booleschen Variablen. Solche Variablen können als Wert nur `true` (wahr) oder `false` (falsch) annehmen. Später werden Sie sehen, dass boolesche Werte indirekt auch bei einigen anderen JavaScript-Konstrukten vorkommen.

```
Ist2000einSchaltjahr = true;
Ist2004einSchaltjahr = true;
Ist3000einSchaltjahr = false;
```

3.1.5 Variablendeklaration

Wie Sie bereits gesehen haben, kann mit dem Gleichheitszeichen einer Variablen ein Wert zugewiesen werden. Ein Variablenname kann auch öfter verwendet werden, und ihm kann auch mehrmals ein Wert zugewiesen werden. Es handelt sich dann jedoch immer um dieselbe Variable. Sehr oft

findet man in der Praxis das Schlüsselwort `var` vor der ersten Verwendung einer Variablen. Dies dient zur Initialisierung der Variablen und wird ab hier der Übersichtlichkeit halber in diesem Buch konsistent verwendet. So sieht man auf den ersten Blick, welche Variable schon einmal deklariert worden ist und welche nicht. Später, bei der Einführung von Funktionen, wird das Schlüsselwort noch eine besondere Bedeutung erhalten.

Beachten Sie, dass Sie das Schlüsselwort `var` nur einmal pro Variable verwenden sollten. Richtig ist also Folgendes:

```
var AnzahlHonigtoepfe = 5;
// Befehle zum Einkaufen von mehr Vorräten
AnzahlHonigtoepfe = 6;
```

Nicht so gut ist dagegen:

```
var AnzahlHonigtoepfe = 5;
// Befehle zum Einkaufen von mehr Vorräten
var AnzahlHonigtoepfe = 6;
```

Der Grund: Bei der Verwendung von `var` wird die Variable neu initialisiert. Das heißt, die alte Variable wird gelöscht und eine neue Variable erstellt. Wenn die Variable aber bereits existiert, ist der Einsatz von `var` unnötig.

In diesem und auch in einigen anderen Beispielen in diesem Buch sind Programmzeilen, die gegenüber einem vorherigen Listing verändert wurden, durch Fettdruck hervorgehoben.

3.2 Operatoren

Durch Operatoren wird eine gewisse Anzahl von Variablen miteinander kombiniert. Beispiele für Operatoren sind die Grundrechenarten. Durch den Plus-Operator werden zwei Zahlen miteinander kombiniert, und als Ergebnis erhält man die Summe dieser beiden Zahlen. Man unterscheidet – auch je nach Typ der beteiligten Variablen – verschiedene Arten von Operatoren.

3.2.1 Arithmetische Operatoren

Diese Art von Operatoren arbeitet mit numerischen Variablen. Sie sollten also sicherstellen, dass auch wirklich Zahlenvariablen vorliegen, sonst könnten Sie eine Fehlermeldung erhalten. In Kapitel 12, »Formulare II«, finden Sie Techniken, wie man Zahlenvariablen als solche erkennen kann. Tabelle 3.2 zeigt alle arithmetischen Operatoren anhand eines Beispiels.

Operator	Beschreibung	Beispiel	Ergebnis (Wert von a)
+	Addition	a = 7 + 4	11
-	Substraktion	a = 7 – 4	3
*	Multiplikation	a = 7 * 4	28
/	Division	a = 7 / 4	1.75
%	Modulo (Restrechnung)[1]	a = 7 % 4	3
-	Negation	b = 7 a = –b	–7

Tabelle 3.2 Arithmetische Operatoren

Will man eine Variable um einen bestimmten Wert erhöhen, kann man sich des folgenden Konstrukts bedienen:

```
AnzahlHonigtoepfe = AnzahlHonigtoepfe + 5;
```

Der Variablen `AnzahlHonigtoepfe` wird als Wert der alte Wert dieser Variablen plus fünf zugewiesen. Der Wert der Variablen wird also de facto um fünf erhöht. In der Praxis kommt es sehr häufig vor, dass der Wert einer Variablen um genau eins erhöht oder verringert werden soll; für diesen Fall sieht JavaScript eine Abkürzung vor:

- `AnzahlHonigtoepfe++` erhöht den Wert der Variablen um eins.
- `AnzahlHonigtoepfe--` verringert den Wert der Variablen um eins.

Die Operatoren ++ und `--` können auch direkt vor dem Variablennamen stehen. Der Unterschied liegt in der Reihenfolge, in der diese Operation im Vergleich mit anderen Operationen ausgeführt werden soll. Am Beispiel des Operators ++ soll das einmal durchexerziert werden; `--` verhält sich analog.

Das Endergebnis des Standalone-Ausdrucks (des »allein stehenden« Ausdrucks)

```
AnzahlHonigtoepfe++;
```

hat zunächst denselben Effekt (nämlich: Erhöhung um 1) wie

```
++AnzahlHonigtoepfe;
```

Einen Unterschied stellt man jedoch fest, wenn der Ausdruck bei einer Zuweisung verwendet wird:

```
var AnzahlHonigtoepfe = 5;
var Anzahl = ++AnzahlHonigtoepfe;
var Anzahl2 = AnzahlHonigtoepfe++;
```

1 Siehe Beispiel: 7 Modulo 4 liefert 3, weil 7 bei der Division durch 4 den Rest 3 lässt. Analog ist beispielsweise 11 Modulo 4 auch 3.

Welchen Wert hat `Anzahl`, welchen Wert hat `Anzahl2`?

Die (vielleicht etwas überraschende) Antwort lautet: `Anzahl` hat den Wert `6`, `Anzahl2` hat auch den Wert `6`. Betrachten Sie zunächst die zweite Zeile:

```
var Anzahl = ++AnzahlHonigtoepfe;
```

Der ++-Operator steht vor dem Variablennamen. Das bedeutet hier, dass zunächst diese Operation (`AnzahlHonigtoepfe` um eins erhöhen) ausgeführt und dann der neue Wert (`6`) der Variablen `Anzahl` zugewiesen wird.

Bei der dritten Zeile ist es genau andersherum:

```
var Anzahl2 = AnzahlHonigtoepfe++;
```

Zuerst wird der Variablen `Anzahl2` der (aktuelle) Wert von `AnzahlHonigtoepfe` zugewiesen, dann wird der Wert von `AnzahlHonigtoepfe` um eins vergrößert.

Wenn man den Wert einer Variablen nicht um exakt eins erhöhen oder verringern will, kann man sich einer anderen Abkürzung bedienen. Diese Abkürzung existiert für jede der vier Grundrechenarten sowie für den Modulo-Operator:

Operator	Bedeutung	Langform	Kurzform
+=	Addition	a = a + b	a += b
–=	Subtraktion	a = a – b	a –= b
*=	Multiplikation	a = a * b	a *= b
/=	Division	a = a / b	a /= b
%=	Modulo	a = a % b	a %= b

Tabelle 3.3 Abkürzungen für arithmetische Operationen

Auch in JavaScript gilt: Punktrechnung geht vor Strichrechnung. Multiplikationen und Divisionen werden also vor Additionen und Subtraktionen ausgeführt. Der folgende Ausdruck liefert daher `7` und nicht `9`:

```
var PunktVorStrich = 1 + 2 * 3;
```

3.2.2 Boolesche Operatoren

Mit Logikoperatoren (oder booleschen Operatoren) kann man Wahrheitswerte miteinander verknüpfen. Die Bedeutung der Operatoren ist die mathematische Bedeutung, nicht unbedingt die umgangssprachliche Bedeutung. Aus diesem Grund werden die einzelnen Operatoren hier explizit vorgestellt.

UND (&&)

Nur, wenn beide (bzw. alle) beteiligten Variablen den Wert `true` haben, liefert die Operation `true` zurück, ansonsten `false`.

```
var t = true;
var f = false;
var bool1 = t && f;  //liefert false
var bool2 = t && t;  //liefert true
```

ODER (||)

Ist eine der beteiligten Variablen `true`, so liefert die Operation `true` zurück. Das Ergebnis ist nur dann `false`, wenn alle Variablen den Wert `false` haben. Hier liegt ein Unterschied zum Deutschen vor, denn dort bedeutet »oder« eher »Entweder-oder«: Das Ergebnis ist nur dann `true`, wenn genau eine der beteiligten Variablen den Wert `true` hat.

```
var t = true;
var f1 = false;
var f2 = false;
var bool1 = t || f1 || f2;  //liefert true
var bool2 = f1 || f2;       //liefert false
```

NEGATION (!)

Der Negationsoperator macht `true` zu `false` und `false` zu `true`.

```
var t = true;
var f = false;
var bool1 = !t;  //liefert false
var bool2 = !f;  //liefert true
```

Short Evaluation

Wie Sie bereits gesehen haben, genügt genau eine Variable mit dem Wert `true`, damit das Ergebnis einer Oder-Verknüpfung ganz sicher den Wert `true` hat. Analog liefert eine Und-Verknüpfung auf jeden Fall den Wert `false`, wenn eine Variable den Wert `false` hat.

JavaScript – zumindest die existierenden Implementierungen – benutzt hier das Prinzip der so genannten Short Evaluation (wörtlich: kurze Auswertung). Bei einer Und- bzw. Oder-Verknüpfung werden die beteiligten Variablen von links nach rechts durchgegangen. Sollte bei einer dieser Variablen aufgrund ihres Werts das Ergebnis der gesamten Operation schon feststehen, wird der Rest nicht weiter ausgewertet.

Hier ein Beispiel:

```
var f = false;
var bool = f && andereVariable;
```

Auf `andereVariable` wird hier gar nicht erst zugegriffen: Die Variable `f` ist `false`, damit kann die `&&`-Verknüpfung (logisches Und) nur `false` als Ergebnis haben.

Vergleichsoperatoren

Vergleichsoperatoren werden meistens bei Zahlenwerten verwendet. Auch bei Zeichenketten sind diese Vergleiche möglich. Hier richtet sich die Rangfolge der einzelnen Zeichen (welches Zeichen ist »größer« als ein anderes?) nach dem ASCII-Code des Zeichens.

Operator	Beschreibung	Beispiel	Ergebnis (Wert für a)
==	Gleich	a = (3 == 4) a = ("Pooh" == "Piglet")	false
!=	Ungleich	a = (3 != 4) a = ("Pooh" != "Piglet")	true
>	Größer als	a = (3 > 4)	false
<	Kleiner als	a = (3 < 4)	true
>=	Größer oder gleich	a = (3 >= 4)	false
<=	Kleiner oder gleich	a = (3 <= 4)	true

Tabelle 3.4 Vergleichsoperatoren

Eine häufige Fehlerquelle ist die Verwechslung der Zuweisung = mit dem Vergleichsoperator ==. Ab JavaScript Version 1.3 gibt der Interpreter eine Fehlermeldung aus, wenn offensichtlich ein Vergleich durchgeführt werden soll, aber der Zuweisungsoperator verwendet wird.

3.2.3 String-Operatoren

Auch mit Zeichenketten kann man »rechnen«; man kann zwei Zeichenketten aneinander hängen. Hierzu wird auch der Plus-Operator (+) verwendet:

```
var Vorname = "Winnie";
var Nachname = "Pooh";
var Baer = Vorname + " " + Nachname;  //liefert "Winnie Pooh"
```

Ansonsten kann man mit Zeichenketten nicht rechnen. Dennoch sollen an dieser Stelle noch drei Möglichkeiten vorgestellt werden, um mit Zeichenketten etwas anzufangen:

- `Zeichenkette.length`: Liefert die Anzahl der Zeichen in einer Zeichenkette zurück.
- `Zeichenkette.charAt(x)`: Liefert das Zeichen an Position `x` in der Zeichenkette zurück. Dabei beginnt die Zählung bei 0, das vierte Zeichen erhält man also mit `Zeichenkette.charAt(3)`.

- `Zeichenkette.substring(start, ende)`: Liefert eine Teilzeichenkette zurück, und zwar ab dem Zeichen an der Position `start` (die Zählung beginnt wieder bei 0) und bis zu dem Zeichen vor dem Zeichen an der Position `ende`.

Hierzu ein kleines Beispiel. In den Variablen `a` und `b` stehen das erste Zeichen der Zeichenkette und die folgenden Zeichen:

```
var z = "Winnie The Pooh";
var a = z.charAt(0);  //a == "W"
var b = z.substring(1, z.length);  //b == "innie The Pooh"
```

3.2.4 Umwandlung zwischen den Variablentypen

Die vorgestellten Operatoren können auch dazu verwendet werden, Umwandlungen zwischen den einzelnen Variablentypen durchzuführen. JavaScript ist in Sachen Variablentypus nicht so strikt wie andere Programmiersprachen. Eine Variable kann auch ihren Typ während des Programmablaufs ändern. Beispielsweise werden Sie in einem späteren Kapitel feststellen, dass Formulareingaben stets als Zeichenketten vorliegen. Wenn Sie sich aber sicher sind, dass die Zeichenkette eine korrekt formatierte Zahl enthält, können Sie JavaScript dazu zwingen, die Variable als Zahlenwert zu betrachten. Der Trick besteht darin, die Variable mit eins zu multiplizieren (oder 0 zu addieren). Eine Multiplikation kann nur mit Zahlenwerten durchgeführt werden, sodass JavaScript die Variable in eine Zahl umwandelt – und eine Multiplikation mit eins ändert am Wert der Zahl auch nichts.

Außerdem ist es manchmal notwendig, eine boolesche oder eine numerische Variable in eine Zeichenkette umzuwandeln. Diesmal muss der **Konkatenationsoperator** (Verkettungsoperator), das Plus, verwendet werden. Indem eine Variable mit einer leeren Zeichenkette konkateniert (verkettet) wird, erhält man als Ergebnis eine Zeichenkette, ändert aber ansonsten den Wert der Variablen nicht.

```
var AnzahlHonigtoepfe = "5";
AnzahlHonigtoepfe *= 1;  //Zeichenkette in Zahl
var wahrheitswert = true;
wahrheitswert += "";  //Wahrheitswert in Zeichenkette
var Anzahl = 6;
Anzahl += "";  //Zahl in Zeichenkette
```

JavaScript führt zwar eine automatische Typenkonvertierung durch, aber nicht immer in die gewünschte Richtung:

```
var siebenundvierzig = "47";
var summe = siebenundvierzig + 11;  // "4711", nicht 58
```

JavaScript stellt auch ein paar Funktionen zur Verfügung, um Umwandlungen durchzuführen. Durch `parseInt()` wird eine Zeichenkette in eine

(ganzzahlige) Zahl umgewandelt, durch `parseFloat()` in eine Fließkommazahl. Innerhalb der runden Klammern wird die Zeichenkette angegeben:

```
var zahl1 = parseInt("47");  //liefert 47 als Zahl
var zahl2 = parseInt("47.11");  //liefert 47,11 dezimal
```

Hierauf gehen wir an späterer Stelle noch einmal en detail ein.

3.3 Kontrollstrukturen: Schleifen

Hin und wieder kommt es vor, dass eine Anweisung mehrmals ausgeführt werden muss, beispielsweise bei einer Aufzählung. Hierzu bietet JavaScript mehrere Kontrollstrukturen an. Mit JavaScript Version 1.2 (sowie beim Internet Explorer 4) wurden neue Kontrollstrukturen eingeführt, die aber mit dem Befehlssatz von JavaScript 1.0 vollständig nachgebildet werden können.

3.3.1 For-Schleifen

Diese Art der Schleife führt eine Anweisung eine (in der Regel) bestimmte Anzahl von Malen aus. Die Syntax sieht dabei folgendermaßen aus:

```
for (Initialisierung; Bedingung; Befehlsfolge){
   //Anweisungen
}
```

Die `for`-Schleife hat drei Parameter:

- **Initialisierung**: Oft läuft bei einer Schleife eine Zählvariable mit, die die Anzahl der Wiederholungen zählt. Diese Variable kann hier initialisiert werden. Sollen mehrere Variablen initialisiert werden, so werden die einzelnen Anweisungen durch Kommata voneinander getrennt.
- **Bedingung**: Die `for`-Schleife wird so lange ausgeführt, bis diese Bedingung nicht mehr erfüllt ist.
- **Befehlsfolge**: Nach jedem Durchlauf der Anweisungen wird diese Befehlsfolge (in der Regel ein Befehl; mehrere Befehle werden durch Kommata voneinander getrennt) ausgeführt. Wenn die Schleife irgendwann enden soll, sollten hier in der Regel Befehle ausgeführt werden, die nach einer bestimmten Anzahl von Durchläufen die Bedingung (den zweiten Parameter) nicht mehr erfüllbar machen.

Die geschweiften Klammern um den Anweisungsblock sind dann zwingend, wenn der Block aus mehr als einem Befehl besteht. Handelt es sich um nur einen Befehl, so kann man die geschweiften Klammern weglassen. Man sollte aber zumindest den Code einrücken, damit das Ganze übersichtlich und lesbar bleibt.

Der folgende Code gibt zehnmal `Winnie` aus. Dabei wird eine Zählvariable mit `0` initialisiert und in jedem Schleifendurchlauf um eins erhöht. Die Abbruchbedingung prüft, ob die Zählvariable kleiner als zehn ist. Vor dem

elften Durchlauf wird die Variable auf zehn erhöht, und die Schleife wird verlassen.


```
for (var i=0; i<10; i++)
   document.write("Winnie<br>");
```

Sehr oft wird die Zählvariable auch direkt in der Schleife verwendet. Der folgende Code gibt alle Quadratzahlen von 0 = 0 bis 10 = 100 aus:

```
for (var i=0; i<=10; i++){
   document.write("Das Quadrat von " + i + " ist: ");
   document.write(i*i + "<br>");
}
```

Sie sehen hierbei die Verwendung des Plus-Operators: Hier werden eine Zeichenkette und eine Zahl zusammengefügt. Das Ergebnis ist eine Zeichenkette, die Zahl wird also in eine Zeichenkette umgewandelt.

Beachten Sie ebenso, dass Sie einen Zeilenumbruch nicht mit `\r\n` angeben sollten, sondern mit dem entsprechenden HTML-Tag, `<br>`. Bei Tags müssen Sie noch eine Besonderheit beachten, aber hierzu kommen wir erst im nächsten Kapitel.

Wie bereits erwähnt wurde, kann man auch mehrere Zählvariablen verwenden, die man dann durch Kommata voneinander trennen muss. Anwendungen dafür gibt es ziemlich selten, hier folgt ein sehr praxisfremdes Beispiel. Es werden zwei Zählvariablen verwendet, `i` und `j`. Die erste enthält eine Zahl, die zweite eine Zeichenkette. Der Zahlenwert wird in eine Zeichenkette umgewandelt und an `j` angehängt. Sobald `j` mehr als 15 Zeichen enthält, wird die Schleife verlassen.

```
for (var i=0, j=""; j.length<=15; i++, j += i)
   document.write(i + " - " + j + "<br>");
```

Dieses Programm gibt Folgendes auf dem Bildschirm aus:

```
0 -
1 - 1
2 - 12
3 - 123
4 - 1234
5 - 12345
6 - 123456
7 - 1234567
8 - 12345678
9 - 123456789
10 - 12345678910
11 - 1234567891011
12 - 123456789101112
```

Vor dem nächsten Schleifendurchlauf würde an die Zeichenkette `13` angehängt, die Länge würde dadurch auf 17 Zeichen anwachsen, also wird die Schleife hier verlassen.

3.3.2 Do-While-Schleife

Nicht immer weiß man, wie oft ein Anweisungsblock hintereinander ausgeführt werden soll. Stattdessen will man den Block so lange ausführen, bis eine Bedingung nicht mehr erfüllt ist. Im folgenden Beispiel sollen in einer Zeichenkette alle `A`s durch `B`s ersetzt werden. Hierbei ist die Methode `Zeichenkette.indexOf(Teilstring)` nützlich; diese gibt nämlich zurück, an welcher Position in der Zeichenkette der Teilstring das erste Mal vorkommt. Ist der Teilstring nicht in der Zeichenkette enthalten, wird `-1` zurückgegeben. Das erste Zeichen steht, wie auch schon vorher, an Position 0. Diese Art von Schleife gibt es jedoch erst seit Netscape Navigator 4 und Internet Explorer 4.

Die Syntax ist sieht wie folgt aus:

```
do {
   //Anweisungsblock
} while (Bedingung);
```

Der Anweisungsblock wird ausgeführt, und dann wird die Bedingung überprüft. Ist sie erfüllt, wird der Block erneut ausgeführt (und dann wird wieder die Bedingung geprüft); andernfalls wird die Schleife verlassen.

```
var Zk = "AXAYAZ"  //Zk steht für "Zeichenkette"
do{
   io = Zk.indexOf("A");
   Zk = Zk.substring(0, io) + "B" + Zk.substring(io+1,
   Zk.length);
} while (Zk.indexOf("A")>-1);
```

Nach dem Durchlauf enthält `Zk` den Wert `BXBYBZ`.

Beachten Sie insbesondere, dass die Schleife auf jeden Fall einmal ausgeführt wird! Im Beispiel führt das zu einem Fehler, wenn die Zeichenkette zu Anfang überhaupt kein `A` enthält. Im Folgenden lernen Sie Methoden kennen, um diesen Fehler zu vermeiden.

Noch ein Wort zu der Zuweisung: Wenn Sie bei einer Zeichenkette `z` das Zeichen an der Stelle `x` in das Zeichen `X` ändern wollen (die Zählung beginnt wie immer bei 0), kommen Sie mit folgender Anweisung weiter:

```
z = z.substring(0, x) + "X" + z.substring(x+1, z.length);
```

Alle Zeichen vor und hinter dem zu ändernden Zeichen bleiben durch die beiden `substring()`-Anweisungen erhalten.

3.3.3 While-Schleife

Schon seit JavaScript Version 1.0 gibt es eine weitere Form der Schleifen, und zwar `while`-Schleifen (ohne `do`). Die Syntax ist der von `do-while`-Schleifen sehr ähnlich, der Unterschied steckt im Detail:

```
while (Bedingung){
   //Anweisungsblock
}
```

Die Bedingung wird hier **vor** dem Durchlaufen des Anweisungsblocks überprüft. Im Beispiel von oben, bei der Ersetzung aller `A`s durch `B`s, ist das sehr nützlich, da hier der Anweisungsblock nicht ausgeführt wird, wenn die Zeichenkette von Anfang an keine `A`s enthält:

```
var Zk = "AXAYAZ"  //Zk steht für "Zeichenkette"
while (Zk.indexOf("A")>-1){
   io = Zk.indexOf("A")
   Zk = Zk.substring(0, io) + "B" + Zk.substring(io+1,
   Zk.length)
}
```

3.3.4 For-In-Schleife

Diese Schleife wird recht selten verwendet, und an dieser Stelle fehlt Ihnen noch das Grundwissen über Objekte, um Ihnen eine ausreichende Erklärung geben zu können. Prinzipiell sei gesagt, dass man mit der Schleife durch alle Eigenschaften eines Objekts und alle Elemente einer Variablensammlung (eines Arrays, dazu später mehr) laufen kann. Die folgende Schleife gibt das `name`-Attribut aller Elemente eines Formulars wieder. Spätestens in Kapitel 7, »Formulare I«, werden Sie diesen Code verstehen, vorerst aber müssen Sie mir blind vertrauen. Die `for-in`-Schleifen werden ohnehin sehr selten eingesetzt.

```
for (e in document.forms[0].elements)
   document.write(e.name+"<br>")
```

3.3.5 Schleifensteuerung

Eine Schleife muss nicht unbedingt so oft durchlaufen werden, wie vorgesehen ist. Angenommen, in einer Schleife wird eine Zeichenkette auf das Vorhandensein eines bestimmten Zeichens überprüft (das geht mit `indexOf()` sehr schnell, aber darauf gehen wir an dieser Stelle nicht ein). Sobald das Zeichen gefunden worden ist, muss die Schleife nicht unbedingt weiter ausgeführt werden, denn das Ergebnis (»Das Zeichen ist in der Zeichenkette enthalten«) steht ja jetzt schon fest. Aus diesem Grund gibt es den Befehl `break`, der das sofortige Verlassen der aktuellen Schleife veranlasst. Der Interpreter fährt also hinter dem aktuellen Anweisungsblock fort.

Ebenso kann es bei Schleifen, insbesondere bei `for`-Schleifen, immer wieder vorkommen, dass man zum nächsten Schleifendurchlauf springen möchte (beispielsweise, wenn man genau weiß, dass dieser Durchlauf nicht das gewünschte Ergebnis bringt) und den Rest der Schleife aus Effizienzgründen nicht ausführen lassen will. Der entsprechende Befehl heißt `continue`.

Wieder ist es leider so, dass Ihr momentanes Wissen noch nicht ausreicht, um hier ein sinnvolles Beispiel anzugeben. Am Ende des nächsten Kapitels werden Sie aber in der Lage sein, die Befehle `break` und `continue` einzusetzen.

Allgemein wird die Verwendung von `break` und `continue` als eher schlechter Programmierstil angesehen; zu sehr ähneln diese Befehle dem Spaghetti-Code aus alten BASIC-Zeiten[2]. In der Praxis werden die beiden Befehle jedoch durchaus angewandt, und zumindest im JavaScript-Bereich ist meiner Meinung nach nichts dagegen einzuwenden.

3.4 Fragen & Aufgaben

1. Welche der folgenden Variablennamen sind gültig?
   - `Winnie-The-Pooh`
   - `Piglet1`
   - `1Rabbit`
   - `Christopher Robin`
   - `_1a`
   - `break`
2. Woran scheitert dieser Aufruf?

   ```
   <a href="#" onclick="alert(\"I love Tigger\");">
   ```
3. Was sind die prinzipiellen Unterschiede zwischen einer `while`- und einer `do-while`-Schleife?
4. Erzeugen Sie einen JavaScript-Code, der in einer Zeichenkette alle Leerzeichen durch `%20` ersetzt.

2 Bei den Ur-Versionen von BASIC wurde jede Programmzeile anhand ihrer Nummer identifiziert. Schleifen gab es nur in sehr eingeschränkter Form, sodass einer der häufigsten Befehle GOTO hieß. Mit ihm konnte eine Zeilennummer direkt angesprungen werden. Dadurch wurde der Quellcode sehr unübersichtlich und auch unsauber.