

HTL Hollabrunn

In Kooperation mit dem HERDT-Verlag stellen wir Ihnen eine PDF inkl. Zusatzmedien für Ihre persönliche Weiterbildung zur Verfügung. In Verbindung mit dem Programm HERDT|Campus ALL YOU CAN READ stehen diese PDFs nur Lehrkräften und Schüler*innen der oben genannten Lehranstalt zur Verfügung. Eine Nutzung oder Weitergabe für andere Zwecke ist ausdrücklich verboten und unterliegt dem Urheberrecht. Jeglicher Verstoß kann zivil- und strafrechtliche Konsequenzen nach sich ziehen.

Ralph Steyer

1. Ausgabe, Juni 2023

ISBN 978-3-98569-151-7

JavaScript Grundlagen

JAVS_2023



Bevor Sie beginnen ...	4	6 Vordefinierte Objekte	95
		6.1 Grundlagen zu vordefinierten Objekten	95
		6.2 Das Objekt String für Zeichenketten	95
		6.3 Math für mathematische Berechnungen	96
		6.4 Number für Zahlen	99
		6.5 Objekt vom Typ Date für Zeitangaben	100
		6.6 RegExp für reguläre Ausdrücke	103
		6.7 Das Objekt Image	109
		6.8 Arrays	109
		6.9 Zugriff auf Array-Elemente und der Index	113
		6.10 Übungen	116
1 Einführung in JavaScript	6	7 Das DOM-Konzept	117
1.1 Entstehung von JavaScript	6	7.1 Objekte und Hierarchie des DOM	117
1.2 Grundlagen zu JavaScript	6	7.2 Das Objekt window	118
1.3 JavaScript-Versionen	10	7.3 Grundsätzliches zur Struktur des DOM einer Webseite	122
1.4 JavaScript-Aktivierung im Browser testen	11	7.4 Das Objekt document	124
1.5 Nützliche Webseiten	12	7.5 Zugriff auf Inhalte von Elementen in der Webseite	130
1.6 Übung	13	7.6 HTML-Elemente als Unterobjekte von document	132
2 Grundlegende Sprachelemente	14	7.7 Das Objekt history	135
2.1 JavaScript in HTML verwenden	14	7.8 Das Objekt location	136
2.2 Allgemeine Notationsregeln	17	7.9 Das Objektfeld frames	138
2.3 Reservierte Wörter	18	7.10 Das Objekt screen	139
2.4 Bezeichner	19	7.11 Das Objekt navigator	140
2.5 Variablen	21	7.12 Übungen	142
2.6 Konstanten	24	8 Ereignisse	143
2.7 Datentypen	24	8.1 Grundlagen zu Ereignissen	143
2.8 Operatoren	29	8.2 Ereignisbehandlung	144
2.9 Rangfolge der Operatoren	38	8.3 Auf Ereignisse reagieren	145
2.10 Übungen	39	8.4 Das Ereignisobjekt event	151
3 Kontrollstrukturen	40	8.5 Das Ereignisobjekt verwenden	152
3.1 Steuerung des Programmlaufs	40	8.6 Übungen	156
3.2 Anweisungsblock	40	9 Formulare	157
3.3 Auswahl	41	9.1 Grundlagen zu Formularen	157
3.4 Wiederholung	46	9.2 Gemeinsame Methoden und Eigenschaften von Formularelementen	159
3.5 Das KISS-Prinzip	53	9.3 Eingabefelder und Schaltflächen	159
3.6 Übungen	54	9.4 Kontroll- und Optionsfelder	160
4 Funktionen	55	9.5 Auswahllisten	160
4.1 Grundlagen zu Funktionen	55	9.6 Eingaben prüfen	163
4.2 Funktionen mit Parametern	57	9.7 Formulareingaben direkt in JavaScript verwerten	172
4.3 Variable Parameterliste	57	9.8 Übungen	175
4.4 Weitere Möglichkeiten für die Deklaration von Funktionen	60		
4.5 Lokale und globale Variablen	65		
4.6 Vordefinierte Funktionen in JavaScript	69		
4.7 Debuggen von Funktionen	71		
4.8 Übungen	75		
5 Objekte	76		
5.1 Grundlagen von Objekten	76		
5.2 Eigenschaften	80		
5.3 Methoden	88		
5.4 Vererbung	90		
5.5 Anweisungen und Operatoren für Objekte	91		
5.6 Übungen	94		

10 Ajax	176
10.1 Grundlagen zu Ajax	176
10.2 Das XMLHttpRequest-Objekt	176
10.3 Eine HTTP-Anfrage erstellen	177
10.4 Das Datenformat	179
10.5 Daten per Ajax zum Server schicken	180
10.6 Praktische Beispiele	181
10.7 Übung	187
11 Erweiterte JavaScript-Techniken und Ausblick	188
11.1 Hinweise zu JavaScript-Techniken	188
11.2 DHTML	188
11.3 Umgang mit Multimedia	193
11.4 Datenspeicherung im Client	197
11.5 Übung	201
12 Frameworks	202
12.1 Was sind Frameworks?	202
12.2 Einsatz von reinen JavaScript-Frameworks anhand von jQuery	203
12.3 Einsatz von JavaScript-Frameworks mit Entwurfsmuster anhand von Vue.js	207
Stichwortverzeichnis	208

Bevor Sie beginnen ...

Voraussetzungen und Ziele

Zielgruppe

Dieses Buch richtet sich an Webseitenersteller und Webdesigner sowie Programmierer, die JavaScript einsetzen möchten. Dabei spielt es keine Rolle, ob diese z. B. bereits professionelle Webseiten für ein Internetportal oder nur für die eigene Homepage erstellt haben. Auch für Umsteiger aus anderen Programmiersprachen ist das Buch geeignet.

Empfohlene Vorkenntnisse

Für das Verstehen dieses Buches werden grundlegende Kenntnisse in HTML vorausgesetzt. Auf die Bedeutung von HTML-Tags wird nicht eingegangen. Erfahrungen in anderen Programmier- oder Skriptsprachen sind nicht notwendig. Der Umgang mit dem eigenen Betriebssystem und typischen Programmen wie einem Editor oder Browser wird vorausgesetzt.

Hinweise zur Software

- ✓ JavaScript ist nicht von der Verwendung eines bestimmten Betriebssystems abhängig. Sie können Linux, macOS, Windows oder auch ein anderes Betriebssystem verwenden, wenn dafür JavaScript-Unterstützung angeboten wird. Es wird aber davon ausgegangen, dass das Betriebssystem nicht zu alt ist.
- ✓ Sofern nicht anders vermerkt, wird im Folgenden weiter davon ausgegangen, dass Sie einen oder mehrere der unten genannten Browser in aktuellen Versionen verwenden. Beachten Sie, dass die meisten Browser mittlerweile einen äußerst kurzen Versionszyklus einhalten und in einem kurzen Zeitraum mehrere Versionen erscheinen, die sich oft nur geringfügig unterscheiden. Ältere Browser werden in dem Buch nicht mehr berücksichtigt.

Name	Erhältlich unter der Internetadresse:
Edge oder dessen Vorgänger Internet Explorer 11	https://www.microsoft.com/en-us/edge
Mozilla Firefox	https://mozilla-firefox.de.uptodown.com
Opera	https://www.opera.com/de
Apple Safari	https://www.apple.com/de/safari/
Google Chrome	https://www.google.com/chrome/

Sofern Sie serverseitiges JavaScript bzw. JavaScript unabhängig von einem Browser nutzen wollen, ist die Installation von **Node.js** zu empfehlen. Sie finden die neuste Version von Node.js unter <https://nodejs.org/>. Sie erhalten dort einen Installationsassistenten für Ihr Betriebssystem, der Ihnen die einfache Installation des Systems erlaubt.

Node.js ist eine plattformübergreifende Open-Source-JavaScript-Laufzeitumgebung (JavaScript-Engine). Damit kann JavaScript-Code unabhängig von einem Webbrowser ausgeführt und etwa JavaScript bei einem Webserver verwendet werden. Node.js wird in der JavaScript-Laufzeitumgebung V8 ausgeführt, die ursprünglich für Google Chrome entwickelt wurde.

Die Verwendung von JavaScript in Node.js entspricht damit im Wesentlichen der Verwendung in der Konsole eines Browsers. Man ist losgelöst von der HTML-Umgebung und dem DOM einer Webseite und reduziert auf pures JavaScript. Allerdings kann diese Umgebung durch diverse Module und Objekte erweitert werden, was die Einsatzmöglichkeiten von JavaScript auf einen Umfang „normaler“ Programmiersprachen erweitert.

Konventionen

Hervorhebungen im Text

Im Text erkennen Sie bestimmte Programmelemente an der Formatierung. So werden Datei- und Ordernamen, Hyperlinks und Bezeichnungen für Menüs bzw. Menüpunkte und Programmelemente wie Register oder Schaltflächen immer *kursiv* geschrieben und wichtige Begriffe **fett** hervorgehoben.

Courier New	kennzeichnet Programmtext, Programmnamen, Funktionsnamen, Variablennamen, Datentypen, Operatoren etc.
<i>Courier New kursiv</i>	kennzeichnet Zeichenfolgen, die vom Anwendungsprogramm ausgegeben oder in das Programm eingegeben werden.
[]	Bei Darstellungen der Syntax einer Programmiersprache kennzeichnen eckige Klammern optionale Angaben.
/	Bei Darstellungen der Syntax einer Programmiersprache werden alternative Elemente durch einen Schrägstrich voneinander getrennt.

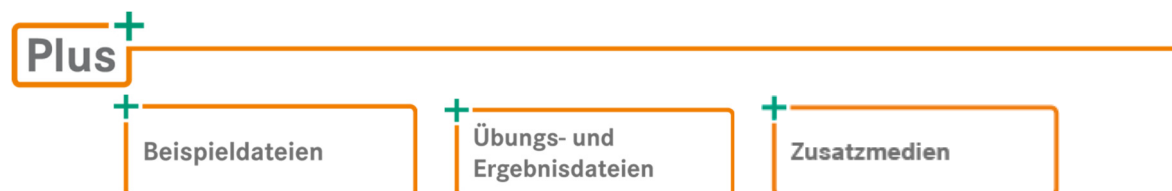
In den abgebildeten Beispielen wird aufgrund der verwendeten Dokumenttyp-Deklaration auf die exakte Schreibweise der öffnenden und schließenden Tags verzichtet. Darum werden z. B. `<p>`-Tags ohne das schließende `</p>`-Tag verwendet.

HERDT BuchPlus – unser Konzept:

Problemlos einsteigen – Effizient lernen – Zielgerichtet nachschlagen

(weitere Infos unter www.herdtd.com/BuchPlus)

Nutzen Sie dabei unsere maßgeschneiderten, im Internet frei verfügbaren Medien:



Wie Sie schnell auf diese BuchPlus-Medien zugreifen können, erfahren Sie unter www.herdtd.com/BuchPlus.

In den Übungsdateien wird teils über die Meta-Angabe `charset` die Zeichencodierung UTF-8 verwendet. Damit ist es nicht mehr notwendig, dass Sie z. B. für deutsche Umlaute die Zeichenreferenz angeben. Dazu wurden die Übungsdateien im Unicode-Format abgespeichert und müssen daher auch in einem Unicode-fähigen Texteditor geöffnet werden, um sie verändern zu können.

1

Einführung in JavaScript

1.1 Entstehung von JavaScript

Die Sprache JavaScript wurde 1995 von der Firma Netscape Communications in Kooperation mit der Firma Sun Microsystems ins Leben gerufen. Der ursprüngliche Name war LiveScript. Aus Marketinggründen wurde ein ähnlicher Name wie für das Sun-Produkt Java gewählt, um eine gemeinsame Vermarktungsstrategie zu gewährleisten. Aufgrund der Ähnlichkeit der Namen JavaScript und Java kommt es häufig zu Verwechslungen. Die Gemeinsamkeiten zwischen den Sprachen sind zwar syntaktisch groß, aber vom Konzept gibt es starke Abweichungen. Eingeführt wurde JavaScript im damaligen Browser Netscape Navigator 2.

Im Zuge der Standardisierung von JavaScript arbeitete die Firma Netscape später mit der ECMA (European Computer Manufacturers Association) zusammen. Die daraus resultierende Spezifikation wurde ECMA-262 mit weiteren Versionen (Editionen) genannt. Die offizielle Version von JavaScript heißt deshalb ECMAScript.

Nach dem Ende der Firma Netscape Communications und der Einstellung des Browsers Netscape Navigator übernahmen die Mozilla Foundation (Browser Firefox) und ECMA die Weiterentwicklung der Skriptsprache.

1.2 Grundlagen zu JavaScript

Wie Sie JavaScript einsetzen können

JavaScript wurde entwickelt, um die statischen Inhalte von Webseiten mit dynamischen Inhalten und Interaktionen zu erweitern. Dies sollte letztendlich dazu dienen, Webseiten attraktiver zu gestalten und den Nutzern eine verbesserte Funktionalität zu bieten. Heutzutage erlangt JavaScript in Zusammenhang mit HTML5 und CSS3 und den modernen **Rich Internet Applications** (RIAs) immer mehr an Bedeutung beim Erstellen von applikationsähnlichen Programmen. Ebenso wird JavaScript immer mehr auf Serverseite bzw. allgemein unabhängig von der Integration in einem Browser eingesetzt. JavaScript wird z. B. für die folgenden Anwendungsfälle genutzt:

- ✓ Bevor Sie die Inhalte von Formularen abschicken, können Sie über JavaScript die korrekte Eingabe der Daten sicherstellen. So können Sie beispielsweise prüfen, ob eine Postleitzahl nur aus Zahlen besteht. Diese Prüfung erfolgt ohne Kommunikation mit dem Webserver. Dieser wird dadurch entlastet, und der Benutzer wird schneller auf Fehler hingewiesen.
- ✓ Sie können vollständige Anwendungen entwickeln, die in einer Webseite direkt beim Nutzer ausgeführt werden. Das können kleine Anwendungen wie z. B. ein Taschenrechner oder Währungsumrechner sein. Aber auch größere Anwendungen wie interaktive Kalender, Portale etc. sind mit JavaScript möglich – insbesondere, wenn diese mit dem Webserver im Hintergrund kommunizieren.
- ✓ Es ist die Darstellung von dynamischen Inhalten möglich, ohne dass dazu Skripts auf dem Webserver oder spezielle Erweiterungen in dem Browser notwendig sind. So können Sie beispielsweise nach der Eingabe von Messdaten ein Diagramm anzeigen oder je nach eingegebenen Werten dynamisch weitere Formularelemente anzeigen bzw. verbergen.
- ✓ Sie können Daten zwischen Browser und Webserver austauschen, ohne dass Sie die Webseite im Browser neu laden müssen. Ein bekannter Einsatz ist hierbei die Anzeige von Suchergebnissen, während Sie noch die Suchanfrage eingeben.
- ✓ Sie können Daten auf dem Client speichern.
- ✓ Sie können den Ort des Besuchers einer Webseite bestimmen und ihm damit lokalisierte Informationen anzeigen.
- ✓ Sie können zahlreiche Daten des Clientrechners auswerten und damit Seiten und Inhalte an die Gegebenheiten beim Besucher anpassen.
- ✓ Auf einem Server, der JavaScript unterstützt, können Sie alle Aktionen durchführen, die man etwa mit speziell ausgerichteten Sprachen und Techniken wie PHP, Perl, Java Server Pages, ASP.NET etc. macht, beispielsweise Daten vom Client entgegennehmen und verwerten oder auf serverseitige Datenbanken zugreifen.
- ✓ JavaScript kann bei Desktop-Applikationen, deren Oberfläche mit einer XML-Struktur beschrieben wird (etwa FXML bei JavaFX), direkt zur Programmierung von Ereignisroutinen eingesetzt werden.

Was ist JavaScript?

JavaScript ist eine Skriptsprache, mit der Sie vollwertig programmieren können, im Gegensatz zur Auszeichnungssprache HTML. Da JavaScript mit Objekten arbeitet, wird auch von einer objektbasierten oder (früher mit Abstrichen) objektorientierten Sprache gesprochen. Da ebenso funktionales sowie prozedurales Programmieren möglich ist, kann JavaScript sehr flexibel eingesetzt werden.

Die Verwendung von JavaScript ist im Gegensatz zu vielen anderen Programmiersprachen recht einfach zu erlernen, da sie weniger Sprachelemente besitzt und viele Dinge automatisch erledigt. Dennoch nimmt die Einarbeitung einige Zeit in Anspruch, da Sie nicht nur die Sprache, sondern auch das Einsatzgebiet kennenlernen müssen. Gerade die Einschätzung der sehr unterschiedlichen Verhaltensweisen von Browsern erforderte in der Vergangenheit viel Erfahrung, und bei richtig komplexen Applikationen können auch JavaScripts sehr umfangreich werden. In der letzten Zeit wird JavaScript auch immer professioneller in großen Projekten eingesetzt und löst zum Teil mächtige Konkurrenztechnologien wie Java oder C# in geeigneten Umgebungen ab. Viele Programmierer mit Erfahrung in C#, Java oder C/C++ eignen sich deshalb JavaScript-Kenntnisse an.

JavaScript einsetzen

JavaScript kann im Browser (clientseitig – beim Benutzer) und in einigen Webservern (serverseitig) sowie in einer allgemeinen JavaScript-Engine eingesetzt werden. Benutzer einer Webseite bekommen die Art des Einsatzes von JavaScript auf einem Webserver nicht mit, da in der Webseite nur Kontakt mit clientseitigem JavaScript besteht und der Webserver eine „Blackbox“ darstellt. JavaScript wird immer noch überwiegend in Browsern genutzt, gewinnt aber jenseits dieses Einsatzzwecks aktuell rasant an Bedeutung.

Die Implementierung von JavaScript erfolgt clientseitig direkt im Browser, indem ein Skript in eine Webseite eingebunden wird. JavaScript-Anwendungen können clientseitig nicht außerhalb, sondern nur innerhalb eines Browsers bzw. einer entsprechenden Ausführungsumgebung ausgeführt werden. Diese Abschirmung vom Rest des Computersystems im Browser wird auch als Sandbox bezeichnet.

Da Browser unter verschiedenen Betriebssystemen zum Einsatz kommen, wird JavaScript auch als plattformübergreifende Sprache bezeichnet. Der Vorteil für Sie ist dabei, dass ein JavaScript-Programm theoretisch unter verschiedenen Browsern und Betriebssystemen gleichermaßen läuft. Dies war lange Zeit aufgrund der unterschiedlichen JavaScript-Versionen und Implementierungen im Browser oft nur für einfache Skripts der Fall, aber mittlerweile geht das auch recht zuverlässig für komplexere Skripts.

Wie wird JavaScript in eine Webseite eingebunden?

Mit JavaScript können Sie unter anderem auf HTML-Elemente zugreifen bzw. HTML-Code erzeugen. Der JavaScript-Code wird dazu in das HTML-Dokument eingefügt oder über eine separate Datei eingebunden. Da sich der JavaScript-Code in einem für Menschen lesbaren Zustand befindet, muss er vom Browser erst interpretiert werden (Prüfung der syntaktischen Korrektheit, Übersetzung, Ausführung).

JavaScript wird in den Quelltext eines HTML-Dokuments eingebunden und gesondert gekennzeichnet. Die Einbindung erfolgt über das Tag `<script>`, das mit einem Ende-Tag abgeschlossen werden muss. Bei der Darstellung des HTML-Dokuments kann der Browser die relevanten Stellen zwischen den beiden `<script>`-Tags als JavaScript identifizieren und sie entsprechend verarbeiten. Browser, die kein JavaScript verstehen oder bei denen JavaScript deaktiviert ist, ignorieren diese Anweisungen.

Browser, die kein JavaScript verstehen, wurden meist vor 1995 entwickelt und werden nicht mehr verwendet.

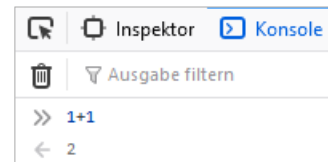
```
<script type="text/javascript">
  <!-- JavaScript-Anweisungen -->
</script>
```

Das Attribut `type` kann entfallen, denn Browser werden als Vorgabe immer JavaScript verwenden. Sie geben mit dem Attribut bei einer vollständigen Notation allgemein den sogenannten MIME-Type (**M**ultipurpose **I**nternet **M**ail **E**xtensions) an. Mit HTML5 wird das Attribut offiziell nicht mehr gefordert. Die Beibehaltung des Attributs ist aber empfehlenswert, um deutlich zu kennzeichnen, dass man mit JavaScript arbeitet.

Wie nutzt man JavaScript ohne Integration in eine Webseite?

Traditionell ist die Verwendung von JavaScript mit einer Integration in eine Webseite verbunden. Aber diese Beschränkung ist schon lange gefallen. JavaScript kann entweder in der Konsole eines Webbrowsers direkt ausgeführt werden oder man verwendet eine JavaScript-Laufzeitumgebung wie Node.js. In beiden Fällen wird keine Webseite als Gerüst bzw. Verankerung des Skripts mehr benötigt. Allerdings können Sie dann auch nicht auf die Bestandteile der Webseite wie Formulare oder Grafiken zugreifen. Auch nutzen einige auf XML basierende Oberflächentechnologien JavaScript zur Programmierung von dynamischen Verhaltensweisen.

Wenn Sie in einem Browser die Konsole öffnen (beispielsweise mit einem Rechtsklick in eine Webseite und Öffnen der Entwicklertools unter Firefox – bei anderen Browsern geht das ähnlich), können Sie dort direkt JavaScript-Anweisungen notieren, z. B. die Berechnung des Ergebnisses von $1 + 1$. Das Ergebnis wird Ihnen direkt in der Konsole angezeigt.



Wenn Sie **Node.js** mit Standardeinstellungen installiert haben, öffnen Sie eine Eingabeaufforderung/Shell/Konsole Ihres Betriebssystems und geben dort `node` ein. Sie gelangen danach in den Eingabemodus von Node.js und können dort, vollkommen analog wie in der Browserkonsole, JavaScript-Befehle eingeben.

```
C:\Users\ralph>node
Welcome to Node.js v18.14.0.
Type ".help" for more information.
> 1+1
2
>
```

Mit der Anweisung `.quit` (beachten Sie den vorangehenden Punkt) beenden Sie die Node.js-Konsole.

Eine JavaScript-Datei können Sie auch als Parameter an Node.js übergeben und damit direkt ausführen, ohne in den Eingabemodus von Node.js gehen zu müssen.

Etwa führen Sie so die JavaScript-Datei *javascriptdatei.js* aus, wenn Sie sich in dem Verzeichnis mit der JavaScript-Datei befinden:

```
node javascriptdatei.js
```

Sicherheit

Das Ausführen von JavaScript-Code kann vom Anwender in den Einstellungen des Browsers deaktiviert werden. Früher haben einige Benutzer JavaScript auch wegen Sicherheitslücken durch fehlerhafte Implementierungen in den Browsern ausgeschaltet. Die Gefahr durch JavaScript-Anwendungen ist jedoch gering, denn diese laufen im Browser in einer abgeriegelten Umgebung ab, in der schon erwähnten Sandbox. Sie können nicht auf Dateien des lokalen Rechners zugreifen (Ausnahme – dafür explizit vorgesehene Dateien wie der lokale Speicher) und auch keine Benutzerdaten abfragen, die nicht auch über den Browser übermittelt werden können.

Zudem werden die JavaScript-Implementierungen in den aktuellen Browsern immer besser und sicherer. Mittlerweile haben fast alle Anwender JavaScript aktiviert, denn moderne Webseiten sind ohne JavaScript meist nicht mehr zu verwenden, und Browser haben JavaScript in der Standardeinstellung aktiviert. Die Browserhersteller verstecken zudem die Möglichkeiten zur Deaktivierung von JavaScript so, dass typische Anwender sie gar nicht finden.

Wenn Sie JavaScript außerhalb des Browsers ausführen, steht dort die Sandbox des Browsers **nicht** zur Verfügung. Was dann mit JavaScript möglich ist, unterliegt der JavaScript-Engine. Allerdings eröffnet dies viel weitergehende Möglichkeiten, die Sie dann mit JavaScript haben.

1.3 JavaScript-Versionen

Die Sprache JavaScript wird von der Mozilla Foundation immer wieder erweitert, wobei die **praxisrelevante** Entwicklung im Grunde seit dem Jahr 2000 stagniert hat. Die Neuerungen aller späteren Vorschläge der Mozilla Foundation wurden gar nicht oder nicht einheitlich von den unterschiedlichen Browsern interpretiert, mit wenigen Ausnahmen, die aber nicht explizit zu neuen Versionsnummern von JavaScript geführt haben.

Die Weiterentwicklung von JavaScript wird allerdings durch HTML5 vorangetrieben, und viele der proprietären JavaScript-Neuerungen der Mozilla Foundation, die seit dem Jahr 2000 für Firefox und seine Verwandten eingeführt wurden, halten über diesen „Marketingumweg“ Einzug in andere moderne Webbrowser. Auch haben populäre JavaScript-Erweiterungen wie Ajax jenseits der „offiziellen“ JavaScript-Versionen immer wieder für eine Weiterentwicklung der JavaScript-Features gesorgt.



Beachten Sie, dass Microsoft kein JavaScript, sondern eine eigene Skriptsprache mit dem Namen **JScript** verwendet. Diese ist zwar fast identisch zu JavaScript, denn sie basiert wie JavaScript selbst auf der ECMA-Spezifikation. Allerdings enthält JScript einige Erweiterungen und kleinere Abweichungen von JavaScript, was in der Vergangenheit in gewissen Situationen eine besondere Programmierung für den Internet Explorer notwendig machte. Diese Ausnahmesituationen nehmen stark ab, da der Internet Explorer mehr oder weniger Geschichte ist und sich dessen Nachfolger Edge weitgehend standardkonform verhält.

Weiter gibt es noch **TypeScript**. Dies ist eine Open-Source-Programmiersprache, die von Microsoft entwickelt und gepflegt wird. Es ist ein Superset von JavaScript und fügt der Sprache optional sogenannte statische Typisierung (und früher klassenbasiertes objektorientiertes Verhalten – das kann neues JavaScript aber auch) hinzu. In Browsern wird TypeScript aber – sofern unterstützt – in normales JavaScript umgewandelt, bevor die Ausführung erfolgt.

Die jeweiligen Versionen von JavaScript werden von der Mozilla Foundation mit einer fortlaufenden Nummerierung gekennzeichnet und starteten im März 1996 mit der Version 1.0. Die derzeit letzte offizielle Versionsnummer 1.8.5 stammt vom Juli 2010 und ist mit ECMAScript 1.8.1 kompatibel. Tatsächlich wurden die offiziellen Versionsnummern von JavaScript damit quasi „eingefroren“, aber die zugrundeliegende Basis ECMAScript weiter vorangetrieben.

Die zum Zeitpunkt der Bucherstellung aktuellste Version ECMAScript 2022 stammt vom Juni 2022 und wird von den meisten Browsern nicht vollständig unterstützt. Aber das soll keinesfalls bedeuten, dass sich im Umfeld von JavaScript nichts tut.

Vereinheitlichungen in Browsern, Optimierungen der Ausführungsumgebungen und vor allen Dingen die Unterstützung zusätzlicher Features und Definitionen aus modernen Versionen von ECMAScript erfolgen permanent. Die meisten Neuerungen sind allerdings für den Einstieg in JavaScript sowieso von geringer Bedeutung.

JavaScript-Versionen angeben

In modernen Webseiten gibt man normalerweise die verwendete JavaScript-Version nicht mehr an, da alle modernen Browser in etwa die gleichen Features unterstützen. Im Prinzip kann man bei der Einbindung von JavaScript jedoch eine bestimmte JavaScript-Version angeben. Damit können Sie inkompatible Browser von dem verwendeten Skript-Bereich fernhalten. Dazu können Sie die jeweilige Version mit dem Attribut `type` im `<script>`-Tag angeben. Unterstützt ein Browser nur eine niedrigere Versionsnummer, soll der entsprechende JavaScript-Code nicht ausgeführt werden. Andernfalls würden die neuen und damit unbekannten JavaScript-Befehle einen Fehler produzieren. Aber diese Vorsicht ist aus genannten Gründen in der Praxis mittlerweile unbegründet.

```
<script type="text/javascript; version=1.6">
  <!-- JavaScript-Anweisungen -->
</script>
```



Diese Versionsangabe im `<script>`-Tag über einen speziellen Wert im Attribut `type` ist nicht standardisiert und wird zurzeit nur von Mozilla Firefox und dessen Verwandten ausgewertet. Früher hat man bei der Überprüfung der maximal möglichen JavaScript-Version stattdessen das eigentlich veraltete `language`-Attribut verwendet. Die gleichzeitige Verwendung von `type` und `language` muss unterbleiben, da `language` immer ignoriert wird, wenn Sie gleichzeitig auch `type` notieren. Aber die Angabe der Version sollte wie gesagt unterbleiben, wenn es nicht einen unabdingbaren Grund dafür gibt. Im Zweifelsfall handelt man sich damit eher Probleme ein, wenn Browser unnötig ausgegrenzt werden. Denn obwohl moderne Browser viele Features unterstützen, die nach JavaScript 1.5 eingeführt wurden, würde eine Versionsangabe jenseits von JavaScript 1.5 unnötigerweise verhindern, dass die damit referenzierten Skripte ausgeführt werden.

1.4 JavaScript-Aktivierung im Browser testen

Sie können davon ausgehen, dass JavaScript grundsätzlich bei den Anwendern aktiviert ist (Voreinstellung aller Browser) und die meisten Anwender JavaScript nicht deaktivieren. Dennoch können Sie nicht zu 100 % sicher sein, dass JavaScript aktiviert ist. Verwenden Sie das Skript im folgenden Beispiel, um die Aktivierung von JavaScript zu prüfen. Erstellen Sie ein HTML-Dokument, das eine erste einfache JavaScript-Anweisung ausführt. Die Erläuterung der genauen Funktionsweise ist in diesem Moment nicht von Interesse. Sie erfolgt im weiteren Verlauf des Buches.

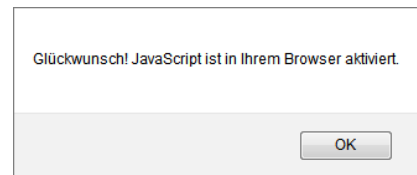
Beispiel: test.html

Geben Sie den folgenden Quellcode in einem Texteditor ein, und speichern Sie ihn in der Datei *test.html*. Nach dem Öffnen des HTML-Dokuments in Ihrem Browser sollte sich ein Popup-Fenster öffnen.

```

<!DOCTYPE html>
① <html>
    <head>
        <meta charset="utf-8">
        <title>Der erste JavaScript-Test</title>
    </head>
    <body>
②     <script type="text/javascript">
③         alert('Glückwunsch! JavaScript ist in Ihrem Browser
            aktiviert.');
```

- ① Das Grundgerüst eines HTML-Dokuments wird eingeleitet.
- ② Dem Browser wird über die Anweisung `<script type="text/javascript">` mitgeteilt, dass die Anweisungen der verwendeten Skriptsprache als JavaScript zu interpretieren sind.
- ③ Die JavaScript-Funktion `alert()` zeigt ein Meldungsfenster an. Als Text wird die angegebene Zeichenkette eingeblendet.
Wird kein Fenster eingeblendet, ist JavaScript nicht aktiviert, sofern Sie keinen Fehler gemacht haben.
- ④ Mit dem schließenden Tag `</script>` wird der JavaScript-Teil beendet.



JavaScript-Meldung im Mozilla Firefox

1.5 Nützliche Webseiten

https://developer.mozilla.org/en/JavaScript	Informationen zu JavaScript
https://wiki.selfhtml.org/wiki/JavaScript	Einführung in JavaScript (SelfHTML, deutsch)
https://www.jswelt.de/	Umfangreiche JavaScript-Sammlung
https://nodejs.org/de	Deutsche Seite von Node.js



Ergänzende Lerninhalte: *Informationen im Internet.pdf*

1.6 Übung

Theoriefragen zu JavaScript

Übungsdatei: --

Ergebnisdatei: *kap01/uebung.pdf*

1. Wo werden JavaScript-Anweisungen ausgeführt?
2. Erklären Sie die Unterschiede der Sprachen TypeScript, JScript, Java und JavaScript.
3. Zählen Sie einige der möglichen Einsatzgebiete von JavaScript auf.

2

Grundlegende Sprachelemente

2.1 JavaScript in HTML verwenden

JavaScript direkt einbinden

Über das `<script>`-Tag können Sie JavaScript-Code in ein HTML-Dokument einfügen. Der JavaScript-Bereich wird durch Angabe des `</script>`-Tags wieder beendet.

```
<script type="text/javascript">
  <!-- Hier werden die JavaScript-Anweisungen eingefügt -->
</script>
```

Der JavaScript-Code kann sowohl im `head`-Bereich (engl.: head = Kopf) als auch im `body`-Bereich (engl.: body = Körper, Rumpf) einer HTML-Datei eingefügt werden. Beim Laden eines HTML-Dokuments werden die JavaScript-Anweisungen sequenziell abgearbeitet. In der Regel werden Skripts im `<head>`-Tag eingefügt, um oft verwendete Funktionen und globale Variablen anzulegen (zu deklarieren). Skripts im `<body>`-Tag werden z. B. dazu genutzt, eine Ausgabe in das HTML-Dokument durchzuführen.

Innerhalb des `<script>`-Tags wird über das Attribut `type` der MIME-Typ der Skriptsprache angegeben, wobei diese Angabe mit HTML5 nicht mehr gefordert wird.

- ✓ Durch die Angabe des optionalen Attributs `defer` (ohne Wert) im `<script>`-Tag teilen Sie dem Browser mit, dass der JavaScript-Code keine Ausgabe im HTML-Dokument erzeugt. Diese Angabe ist rein informell und dient dazu, dass der Browser dadurch das HTML-Dokument schneller darstellen kann, da der JavaScript-Code nicht sofort analysiert (geparst) werden muss, z. B. `<script type="..." defer>`.
- ✓ Bei Bedarf können Sie im `<script>`-Tag das Attribut `lang` setzen, um eine Sprache anzugeben, die in dem Quellcode verwendet wird. Beispielsweise `<script lang="en" ...>`, wenn Sie Englisch in dem Quellcode verwenden.
- ✓ Wenn Sie über JavaScript Ausgaben im HTML-Dokument vornehmen, wird dieser Code erneut interpretiert und die Webseite neu aufgebaut. Auf diese Weise können Sie über JavaScript beispielsweise JavaScript-Code erzeugen, der ebenfalls interpretiert und ausgeführt wird. Für das Erzeugen umfangreicher dynamischer Webseiten ist dies eine gängige Praxis.

- In einer Webseite kann es mehrere Skriptbereiche geben. Diese bilden einen gemeinsamen **Namensraum**. Namensraum (engl. namespace) bedeutet, dass Elemente, die Sie in einem Skriptbereich deklariert haben, auch in den anderen (nachfolgenden) Skriptbereichen verfügbar sind. Ebenso müssen Bezeichner in einem Namensraum eindeutig sein. Das gilt unabhängig von der Art der Einbindung.

Einbinden als externe Dateien

JavaScript-Code können Sie auch als separate Dateien einbinden. Dies ist z. B. nützlich, wenn Sie Funktionen in mehreren HTML-Dokumenten verwenden möchten. Durch die Deklaration der Funktionen in einer Datei steht Ihnen eine zentrale Stelle zur Verfügung, an der Sie Quellcode anpassen können, ohne alle entsprechenden HTML-Dokumente bearbeiten zu müssen.

```
<script type="text/javascript" src="URL"></script>
```

Eigene JavaScript-Dateien werden meist in einem separaten Unterverzeichnis gespeichert. In der Praxis haben sich Verzeichnisstrukturen wie *lib/js* (*lib* steht für Library-Bibliothek), *lib/scripts* oder nur *scripts* oder *js* etabliert.

Sie finden alle JavaScript-Dateien in den BuchPlus-Dateien im Unterverzeichnis *lib/js* der Kapitel.

Die (ausschließliche) Verwendung externer JavaScript-Dateien ist bei modernen Webseiten der Regelfall. Nur damit können Sie die Trennung von Struktur und Funktionalität erreichen, was bei komplexeren Webseiten unabdingbar ist. Beachten Sie, dass in den Beispielen im Buch hin und wieder **rein aus didaktischen Gründen** mit internen Skript-Containern gearbeitet wird. Damit lassen sich Zusammenhänge leichter erklären. In der Praxis können Sie diese internen Skript-Container jederzeit in externe Skripte auslagern.

Über das Attribut `src` beim `<script>`-Tag geben Sie die URL zur Datei an, in der sich der JavaScript-Code befindet. Wie bei der Definition von Hyperlinks können Sie relative und absolute Pfadangaben verwenden. JavaScript-Dateien besitzen standardmäßig die Endung `*.js`, wobei das aber nicht zwingend ist.

Eine JavaScript-Datei kann im `head`-Bereich oder `body`-Bereich eines HTML-Dokuments eingebunden werden. Vorzugsweise sollten Sie diese jedoch im `head`-Bereich einbinden, wenn in der JavaScript-Datei **Deklarationen** (das bedeutet die Einführung von neuen Elementen) durchgeführt werden. Erzeugen Sie jedoch direkt Ausgaben in der JavaScript-Datei, sollte die Datei in den `body`-Bereich eingebunden werden, da Sie im `head`-Bereich einer Webseite keine Ausgaben erzeugen können. Genaugenommen wird der Browser diese Ausgaben automatisch aus dem Header der Webseite in den Körper verlagern.

Beispiel: *javascript.js*

In der Datei befindet sich lediglich eine Ausgabeanweisung, die beim Laden der Webseite bzw. externen JavaScript-Datei einen Text in ein HTML-Dokument schreibt.

```
document.write('Ich befinde mich in einer externen Datei.');
```

- Beim Erstellen von JavaScript-Code in externen Dateien notieren Sie nur die JavaScript-Anweisungen.

Beispiel: *javascript-extern.html*

Binden Sie die JavaScript-Datei *javascript.js* im `body`-Bereich des HTML-Dokuments ein. Dadurch wird die Anweisung in der Datei ausgeführt und die betreffende Zeichenkette an dieser Stelle ausgegeben. Beachten Sie, dass sich die JavaScript-Datei *javascript.js* im Unterverzeichnis *lib/js* befindet.

```
...  
<head>  
  <meta charset="utf-8">  
  <title>JavaScript-Test</title>  
</head>  
<body>  
  <script type="text/javascript" src=  
    "lib/js/javascript.js"></script>  
  ...  
</body>
```

Verwendung in Funktionen und HTML-Tags

Zur Reaktion auf Ereignisse und bei der Verwendung in HTML-Tags können Sie JavaScript-Code auch ohne Angabe des `<script>`-Tags angeben. Dazu gibt es zwei Möglichkeiten:

- ✓ den Eventhandler und
- ✓ die Inline-Referenz.

Reaktion auf Ereignisse mit HTML-Eventhandlern

In HTML gibt es sogenannte **Eventhandler**, um auf gewisse Ereignisse zu reagieren, die bei bestimmten Elementen der Webseite auftreten. Das Ereignis `onclick` wird beispielsweise ausgelöst, wenn der Benutzer auf eine Schaltfläche eines Formulars klickt. In doppelten Anführungszeichen wird der JavaScript-Code angegeben, der in diesem Fall ausgeführt werden soll.

```
<button onclick="alert('Sie haben mich gedrückt');">  
  Klicken Sie auf mich  
</button>
```

Aufruf einer JavaScript-Funktion mit einer Inline-Referenz

Bei einer **Inline-Referenz** wird das Ziel eines Hyperlinks durch eine JavaScript-Funktion definiert. In diesem Fall muss der Text `javascript:` vor den Funktionsnamen gesetzt werden, da die JavaScript-Funktion die statische Angabe der URL ersetzt.

```
<a href="javascript:history.back()">Zurück</a>
```


- ! Beide Varianten (Inline-Referenz und HTML-Eventhandler) zum Aufruf von JavaScript direkt bei HTML-Elementen sind veraltet und sollten in modernen Webseiten nicht mehr verwendet werden. Sie vermischen Struktur und Funktionalität, sind schlecht wartbar und lesbar und vom Umfang der Möglichkeiten sehr eingeschränkt. In alten Skripten werden Sie diese Aufrufe jedoch noch finden. Im Kapitel 8 lernen Sie, wie Sie heutzutage auf Ereignisse reagieren.

Falls JavaScript nicht verfügbar ist

Ist die Verwendung von JavaScript im Browser nicht verfügbar, wurde in der Vergangenheit auf der Webseite oft eine alternative Information angezeigt. Diese wurde zwischen die Tags `<noscript>...</noscript>` eingefügt und mit üblichen HTML-Anweisungen gestaltet. In den meisten Fällen wurde dieser Bereich genutzt, um auf die fehlende JavaScript-Funktionalität des Browsers hinzuweisen. Bei modernen Webseiten wird der Besucher ohne JavaScript-Unterstützung oft automatisch auf eine alternative Webseite weitergeleitet, die ohne JavaScript auskommt. Oder der Besucher wird vollkommen ignoriert und bekommt eine sinnlose Webseite angezeigt.

Ausführungsreihenfolge

JavaScript-Anweisungen werden in der Reihenfolge ausgeführt, in der sie in einem HTML-Dokument oder Skript aufgerufen werden. Dadurch werden JavaScript-Anweisungen, die im `head`-Bereich eines HTML-Dokuments eingebunden werden, **vor** JavaScript-Anweisungen im `body`-Bereich ausgeführt. Führen Sie Ausgaben direkt in das Dokument durch, erscheinen diese beim Laden des HTML-Dokuments an der entsprechenden Stelle.

- ✓ Beim Laden eines JavaScripts werden nur die Anweisungen ausgeführt, die sich nicht innerhalb einer Funktion befinden.

2.2 Allgemeine Notationsregeln

Anweisungen

Über Anweisungen beschreiben Sie die Funktionalität eines Programms. Die Anweisungen werden vom Browser interpretiert und ausgeführt. Eine Anweisung wird in JavaScript durch die Angabe eines Semikolons `;` beendet.

Die letzte Anweisung in einem JavaScript-Block (einer Gruppe von Anweisungen) kann unter Umständen auf das Semikolon verzichten, was zu unklaren Situationen führen kann. Sie sollten deshalb eine Anweisung immer mit einem Semikolon beenden. Im Buch wird davon ausgegangen, dass **jede** Anweisung von einem Semikolon beschlossen wird – mit Ausnahme der letzten Anweisung im Kopf einer `for`-Schleife.

Beispiel

```
document.write("Test");
```

Kommentare

In JavaScript können Sie einzeilige und mehrzeilige Kommentare verwenden. Sie werden vom Browser nicht interpretiert und dienen der Dokumentation des Quellcodes. Quellcode sollten Sie immer so kommentieren, dass Sie nach längerer Zeit den Quellcode nachvollziehen können und dass der Quellcode jederzeit auch von einer anderen Person verstanden und gepflegt werden kann.

//	Diese zwei Zeichen teilen dem Browser mit, dass alle folgenden Zeichen bis zum Ende der Zeile zu ignorieren sind.
/* */	Mehrzeilige Kommentare werden zwischen die Zeichen /* und */ geschrieben. Dabei kennzeichnet die Zeichenfolge /* den Anfang und die Zeichenfolge */ das Ende eines mehrzeiligen Kommentars.

Beispiel für einen einzeiligen Kommentar

```
// Ausgabe der Quadratzahlen von 1 bis 10
for(i = 1; i <= 10; i++) {
  document.write(i * i, "<br />");
}
```

2.3 Reservierte Wörter

Reservierte Wörter, auch **Schlüsselwörter** genannt, sind feste, vorgegebene Wörter der Sprache JavaScript. Da jedes reservierte Wort eine feste, vorgegebene Bedeutung besitzt, dürfen Sie diese nicht als Namen von Variablen oder Funktionen verwenden. Die folgende Übersicht enthält alle reservierten Wörter von JavaScript.

abstract	default	float	let	static	undefined
boolean	delete	for	long	super	var
break	do	function	native	switch	void
byte	double	goto	new	synchronized	volatile
case	else	if	null	this	while
catch	enum	implements	package	throw	with
char	export	import	private	throws	
class	extends	in	protected	transient	
const	false	instanceof	public	true	
continue	final	int	return	try	
debugger	finally	interface	short	typeof	

Nicht alle reservierten Schlüsselwörter haben in JavaScript eine Funktionalität. Zahlreiche Schlüsselwörter sind einfach nur reserviert, etwa, um in zukünftigen Versionen von JavaScript verwendet werden zu können. Die Schlüsselwörter von JavaScript sind an den Schlüsselwörtern von Java orientiert.

2.4 Bezeichner

Bezeichner benennen unter anderem Konstanten, Variablen und Funktionen in JavaScript-Programmen. Man spricht auch von einem **Namen**, aber Bezeichner ist üblicher und mehr technisch. Bei der Programmierung können Sie über einen Bezeichner auf diese Elemente zugreifen. Ein Bezeichner wird vom Programmierer festgelegt, um ein neues Element im Quellcode (beispielsweise eine Variable oder eine Funktion) einzuführen. Das nennt man dann die **Deklaration**.

Dabei gelten für Bezeichner in JavaScript folgende Regeln:

- ✓ Die Bezeichner müssen mit einem Buchstaben, dem Zeichen `$` oder einem Unterstrich `_` beginnen.
- ✓ Bezeichner dürfen nur Buchstaben, Ziffern und die beiden Sonderzeichen `$` und `_` enthalten. Deutsche Umlaute wie ä, ö, ü sind zwar möglich, sollten jedoch nicht eingesetzt werden.
- ✓ Die Groß- und Kleinschreibung ist für den späteren Aufruf von Bedeutung, da JavaScript **case-sensitive** ist, d. h., zwischen Groß- und Kleinschreibung unterscheidet.
- ✓ Bezeichner dürfen keine Leerzeichen enthalten.
- ✓ Reservierte Wörter dürfen nicht als Bezeichner verwendet werden.

Beispiel

```
var kundenNr = 123;    // Richtig, enthält nur Buchstaben
var $Kunden_Nr = 123; // Richtig, enthält $, Buchstaben und den
                        // Unterstrich
var Kunden Nr = 123;   // Falsch, enthält Leerzeichen
var -KundenNr = 123;   // Falsch, beginnt mit einem Minuszeichen
var 0KundenNr = 123;   // Falsch, beginnt mit einer Zahl
var case = 123;        // Falsch, ist ein reserviertes Wort
```

Namenskonventionen

Neben den verbindlichen Regeln für Bezeichner gibt es in JavaScript (wie eigentlich in jeder Programmiersprache) **Konventionen**, die man bei Bezeichnern einhalten sollte. Die Bedeutung dieser Namenskonventionen kann gar nicht genug betont werden.

Die Einhaltung solcher Konventionen ist technisch nicht zwingend (was gerade Einsteigern den Nutzen nicht immer erkennen lässt), aber von sehr großem Vorteil. Denn es ist durchaus von Bedeutung, ob Sie in großen Projekten oder mit mehreren Personen zusammenarbeiten oder nur für sich alleine programmieren. Während eine Verletzung der Namenskonventionen im letzteren Fall nur geringe Probleme macht und im Zweifelsfall nur Sie selbst durch Ihre eigenen Codes nicht mehr durchblicken, kann dies große Projekte ganz zum Scheitern bringen oder zumindest Zeitabläufe verzögern und damit unnötige Kosten verursachen.

Deshalb die dringende Empfehlung, dass Sie Namenskonventionen zu 100 % einhalten sollten. Gerade in JavaScript, das durch seine Freiheiten zwar sehr schlank und flexibel ist, aber ohne strenge Einhaltung dieser Konventionen – gerade für Einsteiger – keinerlei Halt und Orientierung gibt. Mit der Beachtung der Namenskonventionen hingegen sind Syntaxstrukturen in JavaScript eindeutig festgelegt und Quellcode oft alleine aufgrund der gewählten Schreibweise vom Bezeichner eindeutig klar. Es lassen sich so beispielsweise durch eine vereinheitlichte und konsequente Wahl von Groß- und Kleinschreibung sehr wichtige Metainformationen festlegen.

Oder noch einmal anders ausgedrückt – beim Einstieg erscheint die strenge Beachtung der Namenskonventionen vielleicht lästig, aber es zahlt sich aus und erleichtert auf längere Sicht die Programmierung, das Verständnis und später die Pflege von Quellcode ungemein.

Die folgenden Konventionen haben sich in JavaScript etabliert (die genannten Syntaxstrukturen werden in der Folge noch erläutert und an den jeweiligen Stellen werden die Konventionen noch einmal rekapituliert). Beachten Sie, dass es in einigen Zusammenhängen andere Regeln geben kann und insbesondere Microsoft die üblichen JavaScript-Konventionen für sein Habitat an einigen Stellen verändert hat:

- ✓ Bezeichner von Eigenschaften, Variablen und Funktionen/Methoden in JavaScript beginnen immer mit einem Kleinbuchstaben. Allerdings kann es begründete Ausnahmesituationen geben, wo man explizit durch ein abweichendes Großschreiben bestimmte Metainformationen vermitteln will. Diese Abweichungen sollten aber sehr selten erfolgen und wirklich sehr gut gerechtfertigt sein.
- ✓ Bei längeren, zusammengesetzten Bezeichnern verwendet man die **Camelnotation** (auch Höckernotation genannt), bei der jedes neue Teilwort mit einem Großbuchstaben an das vorherige Teilwort angefügt wird. Alternativ kann man auch den Unterstrich zum Zusammenfügen verwenden, aber die Schreibweise ist in JavaScript eher unüblich. Nur bei Konstanten ist sie üblich, denn diese werden vollständig großgeschrieben und damit kann man nicht mit Groß- und Kleinschreibung logische Trennungen vornehmen.
- ✓ Deutsche Umlaute, im Prinzip erlaubte Sonderzeichen außer dem Unterstrich etc. unterbleiben beim Bezeichner.
- ✓ Klassen/Prototypen werden immer mit einem Großbuchstaben begonnen und dann kleingeschrieben. Es gilt weiter die Camelnotation.
- ✓ Klassen/Prototypen werden in Singularform notiert.
- ✓ Konstanten (und nur diese Elemente) werden vollständig großgeschrieben.
- ✓ Da ein Konstruktor in JavaScript immer den gleichen Bezeichner wie die zugehörige Klasse bzw. der Prototyp hat, muss ein Konstruktor immer großgeschrieben werden. Da ein Konstruktor aber in JavaScript eine Funktion ist, widerspricht das der Konvention, dass Funktionen mit einem Kleinbuchstaben beginnen. Die Folge ist fast zwangsläufig und verdeutlicht die Bedeutung der strengen Einhaltung von Namenskonventionen. Wenn Sie diese zu 100 % einhalten, ist die einzige Funktion, die mit einem Großbuchstaben beginnt, ein Konstruktor und umgekehrt muss jede Funktion, die mit einem Großbuchstaben beginnt, eine Konstruktorfunktion sein.

Sie werden die Einhaltung dieser Konventionen in den vorgefertigten Funktionen, Methoden, Klassen/Prototypen oder Konstanten von JavaScript konsequent wiederfinden. Aber auch in zusätzlichen Programmierschnittstellen wie dem DOM.

2.5 Variablen

Variablen können während der Programmausführung unterschiedliche, veränderbare Werte (**Literale** – ein Wert wie 5 oder `true`) annehmen, wie z. B. Zwischen- oder Endergebnisse aus Berechnungen oder Benutzereingaben. Für jede Variable wird ein Speicherplatz im Arbeitsspeicher Ihres Computers reserviert. Im Programm greifen Sie auf diesen Bereich über den Variablennamen zu. Variablen haben die folgenden Eigenschaften:

- ✓ Variablen können an einer beliebigen Stelle in einem JavaScript-Programm durch die Angabe des Variablennamens und der Zuweisung eines Wertes deklariert werden. Obwohl es im Allgemeinen nicht zwingend notwendig ist, (globale) Variablen mit dem einleitenden Schlüsselwort `var` (variable) oder `let` zu deklarieren, dient es unter anderem der besseren Lesbarkeit.
- ✓ Der Unterschied zwischen `var` und dem relativ neu eingeführten `let` in JavaScript ist, dass eine mit `var` eingeführte Variable überall im Code des aktuellen Gültigkeitsbereichs (engl. **Scope**) (global oder lokal innerhalb einer Funktion oder eines Blocks) gültig ist, während eine mit `let` deklarierte Variable innerhalb des Scopes, in dem sie definiert wurde, gültig ist, aber von untergeordneten Scopes verdeckt werden kann. Die Details werden später noch erläutert. Beachten Sie allerdings, dass `let` in alten Browsern nicht unterstützt wird.
- ✓ Wenn man mit einer Anweisung `"use strict"` am Anfang des Skripts oder einer Funktion arbeitet, werden moderne Browser bzw. JavaScript-Engines die Verwendung von `var` bzw. `let` bei der Deklaration einer Variablen erzwingen. Dies ist eine sehr sinnvolle Sicherheitsstrategie. Auch kann diese Anweisung in manchen Konstellationen verhindern, dass eine Variable mehrfach deklariert wird. Beachten Sie, dass die Anweisung `"use strict"`; auch in alten Browsern keine Probleme macht, auch wenn sie dort eventuell nicht beachtet wird. Da es sich nur um einen String handelt, ist die Option auch für alte Browser unproblematisch. Im Buch wird diese Einstellung aus didaktischen Gründen teils verwendet, meist aber weggelassen. In der Praxis sollten Sie diese Einstellung aber **immer** verwenden.
- ✓ Variablen, die Sie über die Angabe `var` bzw. `let` deklarieren und denen Sie keinen Wert zuweisen, haben nach der Deklaration den Zustand bzw. Wert `undefined`. Das ist ein wohldefinierter Zustand, auch wenn der Bezeichner möglicherweise etwas Anderes suggeriert.
- ✓ Variablen, die weder über `var` oder `let` noch durch eine Wertzuweisung deklariert wurden, erzeugen bei der Verwendung einen Laufzeitfehler. So können Sie beispielsweise nicht den Wert einer Variablen auslesen, wenn diese nicht deklariert wurde. Die Interpretation des JavaScript-Codes wird abgebrochen.
- ✓ Mehrere Variablen können in einem Schritt durch Kommata getrennt deklariert werden. Diese Anweisung kann über mehrere Zeilen verteilt werden.
- ✓ Einer Variablen können in JavaScript im Programmverlauf Werte verschiedener Datentypen zugewiesen werden (**lose Typisierung**).

Beispiele für die Deklaration von Variablen

Im Folgenden werden die unterschiedlichen Möglichkeiten der Deklaration einer Variablen gezeigt. Ein Programm wird verständlicher, wenn Variablen ausdrücklich deklariert und initialisiert werden. **Initialisieren** bedeutet, dass eine Variable mit einem Anfangswert belegt wird.

```

k = 10; // die Variable k bekommt den Wert 10 zugewiesen.
        // Gab es die Variable k schon vorher, ist das die
        // Zuweisung eines neuen Werts. Gab es k noch nicht
        // ist das die Deklaration einer Variablen k mit
        // gleichzeitiger Initialisierung
var j; // die Variable j wird deklariert, ist vom Wert aber
        // noch undefined
let l = 0; // die Variable l wird deklariert und mit dem Wert
        // 0 initialisiert
var i, k = 10; // die Variable i ist nach der Deklaration
        // undefined und k bekommt den Wert 10 zugewiesen
var k = 10, i = "Text"; // mehrere Variablen mit Wertzuweisung
        // in einer Deklaration
var k = 10, // mehrere Variablen in einer Deklaration,
    i = "Text"; // aber auf mehrere Zeilen aufgeteilt

```

! Da die Namen von Variablen, Funktionen, Objekten usw. case-sensitive sind (d. h., es wird zwischen groß- und kleingeschriebenen Buchstaben unterschieden), verweisen z. B. die Variablen `Auto` und `auto` auf unterschiedliche Speicherbereiche. Nach Konventionen werden Variablen mit einem Kleinbuchstaben begonnen.

Übliche Schreibweisen für Bezeichner bei Variablen:

<code>vorname</code>	<code>alter</code>	<code>meineEigenschaft1</code>	<code>autoAnhaenger</code>
----------------------	--------------------	--------------------------------	----------------------------

Unübliche Schreibweisen für Bezeichner, die man insbesondere bei der Zusammenarbeit mit anderen Programmierern unterlassen sollte:

<code>Vorname</code>	<code>Alter</code>	<code>Meineeeigenschaft1</code>	<code>Autoanhänger</code>
----------------------	--------------------	---------------------------------	---------------------------

Beispiel: *variablen.html*

In dem Beispiel werden drei Variablen deklariert und deren Wert dann beim Laden der HTML-Datei mit `document.write()` in der Webseite angezeigt.

```

<script type="text/javascript">
  j = 10;
  var k;
  var l = 11;
  document.write("j hat den Wert: ", j, "<br />");
  document.write("k hat den Wert: ", k, "<br />");
  document.write("l hat den Wert: ", l, "<br />");
</script>

```

Da der Variablen `k` kein Wert zugewiesen wurde, wird im Browser als Wert `undefined` ausgegeben.

Die Wertzuweisung zur Variablen `j` würde nicht funktionieren, wenn Sie `"use strict";` als Option notiert hätten, da weder `let` noch `var` zur Einführung verwendet werden.

Beispiel: *variablen.js*

Wenn Sie die Ausgabe der Werte in einer Konsole (vom Browser oder mit Node.js) anzeigen wollen, können Sie eine reine JavaScript-Datei (*variablen.js*) erstellen und statt `document.write()` die Anweisung `console.log()` verwenden. Dann können Sie auch auf die Ausgabe eines HTML-Tags zum Zeilenumbruch verzichten.

```
j = 10;
var k;
var l = 11;
console.log("j hat den Wert: ", j);
console.log("k hat den Wert: ", k);
console.log("l hat den Wert: ", l);
```

Bei Node.js können Sie die JavaScript-Datei einfach als Parameter angeben. Etwa so, wenn Sie sich in dem Verzeichnis mit der JavaScript-Datei (*lib/js*) befinden:

node variablen.js

```
j hat den Wert: 10
k hat den Wert: undefined
l hat den Wert: 11
```

Wertzuweisungen

Variablen sollten vor ihrer ersten Verwendung mit einem Wert belegt werden (Initialisierung). Diese Wertzuweisung erfolgt über das Gleichheitszeichen \equiv (den sogenannten **Zuweisungsoperator**). Diese Werte gelten lediglich zu Anfang eines Programms und können im Programmverlauf jederzeit geändert werden. Sie erreichen bei konsequenter Initialisierung, dass alle Variablen vor der ersten Verwendung gültige Werte besitzen.

Beispiel

Für eine Operation werden drei Variablen deklariert und mit Werten belegt. Dadurch vermeiden Sie Fehler, die durch eine Verwendung der Variablen auftreten, bevor ihnen über einen Ausdruck ein Wert zugewiesen wird.

Beachten Sie auch die Verwendung von `"use strict";` in der ersten Zeile.

```
"use strict";
let summand1 = 1;
let summand2 = 8;
let summe = 0;
summe = summand1 + summand2;
document.write(summe);
```

2.6 Konstanten

JavaScript besitzt vorgegebene Konstanten. Konstanten enthalten während der Programmausführung immer einen unveränderlichen Wert. Sie können also keine zweite Wertzuweisung an eine Konstante vornehmen, nachdem sie initialisiert wurde. Genau wie Variablen werden Konstanten mit einem Namen und einem Datentyp vereinbart. Der Datentyp ergibt sich automatisch aus der Angabe des Wertes. Jede Konstante, die Sie in Ihrem Programm verwenden, muss vorher innerhalb einer Anweisung deklariert und initialisiert werden.

Um in JavaScript eigene Konstanten zu verwenden, müssten Sie Folgendes tun:

- ✓ Konstantenbezeichnern (nach Konvention vollständig großgeschrieben) wird das Schlüsselwort `const` vorangestellt.
- ✓ Bei der Konstantendeklaration muss sofort ein Wert zugewiesen werden. Dieser Wert kann später nicht mehr geändert werden.
- ✓ Konstanten werden per Konvention vollständig großgeschrieben.

Selbstdefinierte Konstanten in JavaScript wurden lange von einigen Browsern nicht oder nur eingeschränkt unterstützt. Mittlerweile kann man selbstdefinierte Konstanten in JavaScript recht zuverlässig verwenden.

2.7 Datentypen

Es ist bei der Programmierung wichtig zu wissen, von was für einem Datentyp eine Variable oder ein Literal ist. Der Datentyp legt fest ...

- ✓ wie viel Speicherplatz für eine Variable oder ein Literal reserviert wird,
- ✓ welche Werte erlaubt sind,
- ✓ wie klein bzw. groß die Zahlen minimal bzw. maximal sein können,
- ✓ welche Operationen mit den Variablen durchgeführt werden können.

Lose Typisierung und Casting

JavaScript verfolgt bei der Zuweisung von Werten an Variablen das Konzept der sogenannten losen Typisierung (engl.: loose typing). Dies bedeutet, dass Sie zwar die Namen von Variablen festlegen, dabei jedoch die Datentypen nicht deklarieren können. Einer Variablen kann nur durch eine Wertzuweisung ein Datentyp zugewiesen werden. Die Typzuweisung einer Variablen erfolgt automatisch.

Grundsätzlich kann eine Variable innerhalb eines Skripts von unterschiedlichem Datentyp sein. Dies bedeutet, dass eine Variable in JavaScript z. B. zuerst eine Zahl sein kann, im nächsten Schritt aber zu einer Zeichenkette werden kann. Dieses Verhalten ist im Rahmen der losen Typisierung möglich. Ein explizites Typumwandeln, das sogenannte **Casting**, ist im Gegensatz zu vielen streng typisierten Sprachen wie Java nicht notwendig.

- ! Lose Typisierung macht den Einstieg in JavaScript für Anfänger einfach, ist aber auch fehleranfällig. Lose Typisierung erhöht die Fehlerwahrscheinlichkeit im Quellcode und macht die Wartung unter Umständen aufwändig. Verwenden Sie deshalb für eine Variable **nur einen Datentyp**.

Die verfügbaren Datentypen in JavaScript

JavaScript besitzt vier Haupt- und zwei Sonderdatentypen (`undefined` und `null`):

Datentyp	Beschreibung
<code>boolean</code>	Werte vom Typ <code>boolean</code> (auch boolesche Werte genannt – nach dem Mathematiker George Boole) sind Wahrheitswerte, die nur einen der beiden Werte <code>true</code> oder <code>false</code> (wahr oder falsch) annehmen. Dieser Datentyp wird im Rahmen von Vergleichen genutzt.
<code>number</code>	Dieser universelle numerische Datentyp kann in JavaScript sowohl Ganzzahl- als auch Gleitkommawerte enthalten. Dies ist recht ungewöhnlich, denn in den meisten Programmiersprachen werden diese Typen getrennt.
<code>object</code>	Der allgemeine Datentyp kann einen Wert eines beliebigen Typs enthalten. Er wird für das Speichern von Objekten verwendet.
<code>string</code>	Dieser Datentyp steht für eine Zeichenkette und kann eine Reihe von alphanumerischen Zeichen enthalten. Maximal sind in JavaScript 256 Zeichen in einer Zeichenkette erlaubt. Bei Bedarf können aber mehrere Zeichenketten mit dem Operator <code>+</code> verknüpft werden. Wichtig ist, dass Sie mit dem Operator auch Werte anderer Datentypen mit dem String verbinden können. Das Resultat ist vom Typ <code>string</code> . Diese Möglichkeit ist bei Ausgaben sehr nützlich.
<code>undefined</code>	Der Sondertyp beschreibt eine nicht definierte Situation. Eine Variable besitzt so lange diesen Wert, bis ihr nach dem Anlegen explizit ein Wert zugewiesen worden ist.
<code>null</code>	Dieser Sondertyp entspricht der Situation, wenn ein Objekt noch keinen Wert hat, und entspricht „keiner Bedeutung“ oder <code>null</code> . Damit unterscheidet sich der Wert von einer leeren Zeichenkette oder dem Literal <code>0</code> . Der Sondertyp <code>null</code> kann beispielsweise zurückgegeben werden, wenn in einem Dialogfenster eine <i>Abbrechen</i> -Schaltfläche betätigt wird.

Ganze Zahlen

Ganze Zahlen besitzen keine Nachkommastellen und können negative sowie positive Werte und den Wert Null annehmen. In JavaScript können ganze Zahlen in einer von drei möglichen Darstellungen eingesetzt werden:

- ✓ **Dezimaldarstellung:** Dies ist die gebräuchlichste Darstellung von Zahlen. Die Zahl wird im Dezimalsystem (Basis 10) dargestellt, d. h., dass die Ziffern 0...9 verwendet werden.
- ✓ **Hexadezimaldarstellung:** Die Zahlen werden im Hexadezimalsystem (Basis 16) dargestellt. Die verwendbaren Ziffern sind 0...9, A...F (A=10, B=11...F=15). Hexadezimale Zahlen werden durch ein einleitendes 0x (bzw. 0X) gekennzeichnet.
- ✓ Die **Oktaldarstellung** (zur Basis 8) ist seit JavaScript 1.5 nicht mehr Bestandteil von JavaScript und wird nur noch aus Gründen der Rückwärtskompatibilität unterstützt. Oktalzahlen werden durch eine vorangestellte 0 (Null) gekennzeichnet.

Dezimal	Hexadezimal
0	0x0
1	0x1
8	0x8
15	0xF
16	0x10
42	0x2A

Gleitkommazahlen

Gleitkommazahlen können Nachkommastellen besitzen. Das Kennzeichen ist in der Darstellung entweder ein Dezimalpunkt, ein Exponent oder beides. Bei einer Fixkommazahl ist die Position des Kommas vorgegeben, bei einer Gleitkommazahl kann das Komma „gleiten“.

Fixkommazahl	Gleitkommazahl
1000.0	1.0E3
2000.4	20004.0E-1
-4000.0	-4e3
-0.004	-4e-3

In JavaScript wird die amerikanische Notation, also ein Punkt (.), verwendet, um die Dezimalstellen einer Zahl kenntlich zu machen. Die deutschen Bezeichner sind also irreführend und „Floatingpoint“ drückt viel deutlicher aus, dass der Punkt zur Trennung verwendet wird.

Die größte und kleinste speicherbare Zahl können Sie z. B. durch folgende Anweisungen ausgeben:

Beispiel

```
document.write("Max: ", Number.MAX_VALUE);
document.write("Min: ", Number.MIN_VALUE);
```



Beachten Sie, dass beim Überschreiten des Wertebereichs der ganzen Zahlen automatisch eine Konvertierung in Gleitkommazahlen erfolgt. Gleitkommazahlen unterliegen immer potenziellen Rundungsfehlern. Beachten Sie dies bei Berechnungen mit großen Zahlen.

Beispiel: *rundungsproblem.js*

```
k = 12345678901234567890;
// liefert 12345678901234567000, da die Zahl zuerst in eine
// Gleitkommazahl konvertiert und danach gerundet wird !
console.log("k hat den Wert: ", k);
```

Zeichenketten (Strings)

Eine Zeichenkette (String) ist eine Folge von Zeichen, die in JavaScript in einfache oder doppelte Anführungszeichen eingeschlossen ist. Somit gibt es zwei gleichwertige Formen von Anführungszeichen. Beachten Sie, dass gleiche Arten von Anführungszeichen verwendet werden. Beginnen Sie eine Zeichenkette mit einem doppelten Anführungszeichen `"`, müssen Sie diese auch mit einem doppelten Anführungszeichen beenden.

Möchten Sie den Text: Und ich fragte ihn: "Hallo, wie geht es dir?" auf dem Bildschirm ausgeben, müssen Sie die Zeichenkette mit einem einfachen Anführungszeichen `'` beginnen und beenden, da sich die doppelten Anführungszeichen bereits in der auszugebenden Zeichenkette befinden. Alternativ können Sie die auszugebenden doppelten Anführungszeichen auch maskieren, was etwas später noch erklärt wird.

Angabe in JavaScript	Ausgabe
"Dies ist eine Zeichenkette."	Dies ist eine Zeichenkette.
'Das ist auch eine Zeichenkette.'	Das ist auch eine Zeichenkette.
'Die Zahl "123" ist auch eine Zeichenkette.'	Die Zahl "123" ist auch eine Zeichenkette.
""So funktioniert es auch", sagte er."	'So funktioniert es auch', sagte er.

Ein wichtiger Unterschied zwischen Zeichenketten und numerischen Werten liegt in der **Funktion der Operatoren**. Die Elemente einer Zeichenkette können nicht arithmetisch miteinander verknüpft werden. Bei ganzen Zahlen und Gleitkommazahlen entspricht das Zeichen `+` der mathematischen Addition. Werden zwei Zeichenketten mit dem Zeichen `+` verknüpft, werden sie zu einer Zeichenkette zusammengefasst (Stringverkettung). In allen anderen mathematischen Operationen werden die Zeichenketten als Zahlenwerte interpretiert. Können Zeichenketten nicht in einen Zahlenwert umgewandelt werden, liefert die Berechnung den Wert NaN (not a number = keine Zahl) zurück.

```
var zZahl = 1 ;
var zeichen = '1';
var ergebnis = zahl + zahl;           // ergibt 2;
ergebnis = zeichen + zeichen;         // ergibt '11';
ergebnis = zeichen - zeichen;         // ergibt 0;
ergebnis = '1a' * zahl;               // ergibt NaN
```

Steuerzeichen in Zeichenketten

Innerhalb einer Zeichenkette können Sie Sonderzeichen angeben, um beispielsweise einen Zeilenumbruch durchzuführen oder einen Tabulator einzufügen. Diese Sonderzeichen werden mit einem Backslash `\` eingeleitet und als Escape-Sequenzen oder **Steuerzeichen** bezeichnet.



Die Wirkung der Steuerzeichen ist teils nur innerhalb eines Textes zu erkennen, der nicht in der Webseite angezeigt wird, z. B. bei einer Meldung über die Funktion `alert()` oder mit `console.log()`. Bei der Ausgabe von Text im Browser müssen Sie stattdessen für einige Steuerzeichen HTML-Tags verwenden, da der Browser Tabulatoren und Zeilenumbrüche im HTML-Code als Leerzeichen interpretiert.

Steuerzeichen	Bedeutung
<code>\n</code>	new line: Zeilenumbruch (neue Zeile)
<code>\r</code>	return: „Wagenrücklauf“: Der Cursor steht in der nächsten Zeile wieder an Position 1. In JavaScript hat dies keine weitere Bedeutung.
<code>\t</code>	Tabulator
<code>\f</code>	form feed: Seitenvorschub. In JavaScript hat dies keine weitere Bedeutung.
<code>\b</code>	backspace: ein Zeichen zurück
<code>\"</code>	doppeltes Anführungszeichen, auch innerhalb von doppelten Anführungszeichen, z. B. <code>alert ("\"")</code> ;
<code>\'</code>	einfaches Anführungszeichen, auch innerhalb von einfachen Anführungszeichen, z. B. <code>alert ('\'')</code> ;
<code>\\</code>	Backslash

Boolesche Werte (Wahrheitswerte)

Die booleschen Werte sind `true` (wahr) und `false` (falsch). Wahrheitswerte werden eingesetzt, wenn ein Wert nur zwei Zustände annehmen kann, z. B. Licht an oder Licht aus. Ausdrücke geben häufig einen booleschen Wert zurück, z. B. beim Vergleichen von Zahlen. Der Vergleich `2 > 3` liefert das Ergebnis `false`, weil die Zahl 2 nicht größer ist als 3.

Boolescher Wert	Bedeutung
<code>true</code>	„Wahr“, die Bedingung ist erfüllt, z. B. <code>2 < 3</code> .
<code>false</code>	„Falsch“, die Bedingung ist nicht erfüllt, z. B. <code>2 > 3</code> .



Beachten Sie, dass auch Zahlenwerte in JavaScript für boolesche Ausdrücke stehen. Das Literal 0 oder auch der Leerstring stehen für `false`, und alle numerischen Werte ungleich 0 und andere Werte, die nicht dem Nullwert entsprechen, entsprechen dem Fall, dass Sie dort `true` notieren.

Objekte

Objekte sind Datenelemente, die Eigenschaften und objektgebundene Funktionen (Methoden) besitzen können (siehe Kapitel 5). In JavaScript werden Ihnen diverse vordefinierte Objekte angeboten. Sie können in JavaScript aber auch eigene Objekte erzeugen.

2.8 Operatoren

Operatoren sind Zeichen, die verwendet werden, um Berechnungen durchzuführen oder Verknüpfungen und Vergleiche zwischen Variablen oder Literalen herzustellen. Man nennt das dann auch Operationen zwischen Operanden.

Eine Operation arbeitet in der Regel mit einem oder zwei Operanden. Entsprechend wird von unären oder binären Operatoren gesprochen. Es gibt auch einen Operator mit drei Operanden, der triadischer oder ternärer Operator genannt wird.

Ein Operator erzeugt immer einen **Ergebniswert**. Manche Operatoren können nur in Verbindung mit Variablen eines bestimmten Datentyps (sinnvoll) eingesetzt werden.

Operator	Datentyp
Arithmetischer Operator	Zahlen
Vergleichsoperator	Zahlen, Strings (Zeichenketten), boolesche Werte
Verknüpfungsoperator, Konkatenationsoperator	Alle Datentypen – das Ergebnis ist aber immer ein String
Logischer Operator	Boolesche Werte
Bit-Operator	Zahlen, boolesche Werte
Zuweisungsoperator	Alle

Arithmetische Operatoren

Arithmetische Operatoren dienen zur Berechnung eines Wertes. Dazu wird eine Operation auf einen oder mehrere Operanden angewendet.

Name	Operator	Syntax	Beispiel	Wert
Addition	+	Summand1 + Summand2	7 + 4	11
Subtraktion	-	Minuend - Subtrahend	7 - 4	3
Multiplikation	*	Faktor1 * Faktor2	7 * 4	28
Division	/	Dividend / Divisor	7 / 4	1.75
Modulo	%	Dividend % Divisor	7 % 4	3
Negation	-	-Operand	-(2 + 5)	-7
Inkrement	++	++Variable; Variable++	x = 10; ++x; y = 135; y++	11 136
Dekrement	--	--Variable; Variable--	x = 10; --x; y = 135; y--	9 134

Die Grundrechenarten

Für die vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division verwendet JavaScript die Zeichen $+$, $-$, $*$ und $/$. Bei den zugehörigen Operatoren handelt es sich um binäre Operatoren, die eine Zahl als Ergebniswert liefern.

Modulo

Der Modulo-Operator berechnet den Rest einer ganzzahligen Division. Da JavaScript keinen Operator für die ganzzahlige Division besitzt, müssen Sie diesen mithilfe des Modulo-Operators selbst nachbilden.

Beispiel

Bei der ganzzahligen Division der Zahl 23 mit der Zahl 5 bleibt ein Rest von 3 ($23 = 4 * 5 + 3$), d. h., die Zahl 5 ist viermal in der Zahl 23 enthalten. Der Rest von 3 ist nicht mehr ganzzahlig durch 5 teilbar.

```
23 % 5 // liefert den Wert 3
(23 - (23 % 5)) / 5 // liefert den Wert 4 = ganzzahlige Division
```

Modulo ist in JavaScript auch auf Gleitkommazahlen anwendbar. Die folgenden Anweisungen sind also gültig:

```
7.1 % 5
7.1 % 3.3
```

Sie sollten allerdings niemals Modulo mit Gleitkommazahlen ausführen, da es zu Rundungsproblemen kommen kann, die an der internen Darstellung der Gleitkommazahlen hängen und nicht zu verhindern sind.

Inkrement und Dekrement

Der Inkrementoperator sowie der Dekrementoperator sind unäre Operatoren, d. h., sie werden nur auf einen Operanden angewendet. Das Inkrement $++$ bewirkt, dass der Wert des Operanden um 1 erhöht wird. Das Dekrement $--$ bewirkt, dass der Wert um 1 reduziert wird. Die Operatoren können dabei vor oder nach dem Operanden gesetzt werden. Bei letzterer Variante besitzt der Operand während der Operation den bisherigen Wert und wird erst danach erhöht, in komplexen Ausdrücken kann sich dies jedoch auf das zurückgegebene Ergebnis auswirken.

Die Angabe vor dem Operanden wird als Präfix-, die nach dem Operanden als Postfix-Notation bezeichnet. Im ersten Fall wird die Operation vor jeder weiteren Berechnung ausgeführt, in der Postfix-Notation erst nach der Berechnung des Ausdrucks.

Beispiel: inkdek.html

In dem Beispiel werden Variablen mit denselben Werten unterschiedlich inkrementiert und dekrementiert. Beachten Sie die daraus resultierenden unterschiedlichen Ergebnisse.

```

<script type="text/javascript">
①  var x = 5; var y = 1.5;
    document.write("<br />x = " + x + "; y = " + y);
②  document.write("<br />x++ + y ergibt: " + (x++ + y));
    x = 5; y = 1.5;
    document.write("<p>x = " + x + "; y = " + y);
③  document.write("<br />x + ++y ergibt: " + (x + ++y));
    x = 5; y = 1.5;
    document.write("<p>x = " + x + "; y = " + y);
④  document.write("<br />x-- + y ergibt: " + (x-- + y));
    x = 5; y = 1.5;
    document.write("<p>x = " + x + "; y = " + y);
⑤  document.write("<br />x + --y ergibt: " + (x + --y));
</script>

```

- ① Die Variablen x und y werden mit den Werten 5 und 1.5 initialisiert.
- ② Zuerst werden die Werte von x und y addiert und ausgegeben, erst danach wird x automatisch um den Wert 1 erhöht.
- ③ Die Variable y wird schon vor der Addition um 1 erhöht und dann erst mit dem Wert von x addiert.
- ④ Die Werte von x und y werden addiert und ausgegeben. Dann erst wird x um 1 verringert.
- ⑤ Erst nachdem y um 1 verringert wurde, werden beide

```

x = 5; y = 1.5
x++ + y ergibt: 6.5

x = 5; y = 1.5
x + ++y ergibt: 7.5

x = 5; y = 1.5
x-- + y ergibt: 6.5

x = 5; y = 1.5
x + --y ergibt: 5.5

```

Verschiedene Inkrement- und Dekrementberechnungen

Beispiel: inkdek.js

Hier folgt die reine JavaScript-Version, die Sie in einer Konsole ausführen können. Durch den Verzicht auf HTML-Ausgaben wird der Code reduziert und die reine JavaScript-Logik ist deutlicher zu erkennen.

In der ersten Zeile finden Sie allerdings "use strict";, um deren Einsatz noch einmal in der Praxis zu zeigen. Da aber alle Variablen sowieso mit var „sauber“ deklariert werden, ist diese Sicherheitseinstellung hier nur prophylaktisch zu sehen.

```

"use strict";
var x = 5;
var y = 1.5;
console.log("x = " + x + "; y = " + y);
console.log("x++ + y ergibt: " + (x++ + y));
x = 5;
y = 1.5;
console.log("x = " + x + "; y = " + y);

```

```
console.log("x + ++y ergibt: " + (x + ++y));
x = 5;
y = 1.5;
console.log("x = " + x + "; y = " + y);
console.log("x-- + y ergibt: " + (x-- + y));
x = 5;
y = 1.5;
console.log("x = " + x + "; y = " + y);
console.log("x + --y ergibt: " + (x + --y));
```

Vergleichsoperatoren

Vergleichsoperatoren werden benutzt, um die Werte von zwei Operanden miteinander zu vergleichen. Werden Vergleichsoperatoren auf Zeichenketten angewendet, ist die Reihenfolge der Zeichen im Zeichensatz von Bedeutung. Ziffern werden z. B. niedriger eingestuft als Buchstaben und Großbuchstaben niedriger als Kleinbuchstaben. Nachfolgend sind die einzelnen Zeichen nach der Wertigkeit sortiert, wobei das Zeichen `!` das niedrigste und das Zeichen `~` das hochwertigste Zeichen ist.

```
!"#$%&'()*+,-./
0123456789
:;<=>?@
ABCDEFGHIJKLMNPOQRSTUVWXYZ
[\\]^_`
abcdefghijklmnopqrstuvwxyz
{|}~
```

Folgende Vergleichsoperatoren werden unterschieden:

Name	Operator	Syntax	Beispiel	Wert
gleich	<code>==</code>	<code>Operand1 == Operand2</code>	<code>true == false</code> <code>2 + 4 == 6</code>	false true
ungleich	<code>!=</code>	<code>Operand1 != Operand2</code>	<code>"Hund" != "HUND"</code> <code>2 + 4 != 6</code>	true false
strikte (Un-) Gleichheit	<code>===</code> <code>!==</code>	<code>Operand1 === Operand2</code> <code>Operand1 !== Operand2</code>	<code>2 + 4 === 6</code> <code>"Hund" !== "HUND"</code>	true true
kleiner als	<code><</code>	<code>Operand1 < Operand2</code>	<code>4 < "5"</code> <code>"ADE" < "ABC"</code>	true false
größer als	<code>></code>	<code>Operand1 > Operand2</code>	<code>"abd" > "abc"</code> <code>"X" > 10</code>	true false
kleiner/gleich	<code><=</code>	<code>Operand1 <= Operand2</code>	<code>6 <= 7</code> <code>"abc" <= "abc"</code>	true true
größer/gleich	<code>>=</code>	<code>Operand1 >= Operand2</code>	<code>"6" >= "6"</code> <code>"Buch" >= "buch"</code>	true false

Wenn Sie zwei Operanden auf strikte Gleichheit prüfen, müssen diese den gleichen Wert und den gleichen Datentyp besitzen. Wird auf einfache Gleichheit geprüft, können auch Typumwandlungen zu einer Auswertung von `true` führen. Im Falle von strikter Gleichheit ist dies nicht möglich. Das wird auch **Identität** genannt.

```
console.log(2 == "2"); // liefert true, da nur Werte geprüft
                      // werden
console.log(2 === "2"); // liefert false, da unterschiedliche
                      // Typen vorliegen
```

Verknüpfungsoperator

Dieser Operator, der auch Konkatenationsoperator (engl.: concatenate = verknüpfen) oder Stringverkettungsoperator genannt wird, verknüpft zwei Zeichenketten und liefert die zusammengesetzte Zeichenkette als Ergebniswert.

Name	Operator	Syntax	Beispiel	Wert
Verbinden	+	Operand1 + Operand2	"Zimmer" + "pflanze"	Zimmerpflanze
			x = 10; "x-Wert:" + x	x-Wert:10



Das Zeichen für den Verknüpfungsoperator kann ohne den Kontext **nicht** vom Berechnungsoperator für eine Addition unterschieden werden. Deshalb sollten Sie beim Programmieren darauf achten, ob Ihre Variablen Zahlen oder Zeichenketten enthalten. Wird der Operator \oplus verwendet, wird er als Stringverkettungsoperator interpretiert, wenn die Variablen nicht explizit als Zahlen angegeben werden. Diese Tatsache führt zu dem Ergebnis, dass die Anweisungen `"1" + "1"` und `1 + "1"` den Wert `"11"` ergeben und nicht die Zahl 2. Dieses Verhalten wird „polymorph“ (vielgestaltig) genannt.

Logische Operatoren

Im Gegensatz zu den Vergleichsoperatoren werden mit den logischen Operatoren die booleschen Wahrheitswerte `true` und `false` miteinander verknüpft. Der Ergebniswert eines logischen Ausdrucks besteht aus einem booleschen Wert.

Name	Operator	Syntax	Beispiel	Wert
UND	&&	Operand1 && Operand2	true && true false && true true && false false && false	true false false false
ODER		Operand1 Operand2	true true false true true false false false	true true true false

Name	Operator	Syntax	Beispiel	Wert
NICHT	!	!Operand	!true !false	false true
ENTWEDER ODER XOR	^	Operand1 ^ Operand2	true ^ true false ^ true true ^ false false ^ false	false true true false

- ✓ Das logische UND ist ein binärer Operator. UND verknüpft zwei Operanden und liefert nur dann `true` als Ergebnis, wenn beide Operanden den Wert `true` besitzen. Ansonsten liefert UND den Rückgabewert `false`.
- ✓ Auch beim logischen ODER handelt es sich um einen binären Operator. ODER liefert den Ergebniswert `true`, wenn mindestens ein Operand den Wert `true` besitzt. Wenn beide Operanden den Wert `false` besitzen, wird als Ergebnis der Wert `false` geliefert.
- ✓ NICHT ist ein unärer Operator. Dieser Operator kehrt einen booleschen Wert um. Die Anwendung auf den Wert `false` liefert den Wert `true` und umgekehrt. NICHT beeinflusst immer den Operanden, der syntaktisch nach ihm steht. Aus diesem Grund wird auch von einem Präfixoperator gesprochen.
- ✓ Der selten genutzte Operator ENTWEDER ODER (XOR) liefert `true`, wenn mindestens ein Operand den Wert `true` besitzt. Wenn beide Operanden den Wert `false` oder `true` besitzen, wird als Ergebnis der Wert `false` geliefert.

Short-Circuit-Auswertung

Ausdrücke werden von links nach rechts ausgewertet. Wenn sich an der Auswertung des Gesamtausdrucks durch die Auswertung weiter rechts stehender Teilausdrücke nichts mehr ändern kann, wird die Auswertung abgebrochen. Diese Funktionalität wird auch als Short-Circuit-Auswertung oder Lazy Evaluation bezeichnet.

Beispiel

Bei der Verwendung des ODER-Operators wird die Auswertung abgebrochen, wenn der erste Teilausdruck den Wert `true` liefert. Die folgenden Ausdrücke ändern nichts mehr am Rückgabewert `true`. Im Falle der Verwendung des UND-Operators ändert sich das Gesamtergebnis nicht mehr, wenn der erste Teilausdruck den Wert `false` liefert, da für den Rückgabewert `true` beide Teilausdrücke `true` zurückliefern müssen.

```
(4 > 3) || (5 > 6) // (5 > 6) wird nicht mehr ausgewertet
(3 > 4) && (testeIrgendetwas()) // testeIrgendetwas wird nicht
                                // mehr ausgeführt
```



Beachten Sie bei dieser Form der Auswertung, dass Sie keine Funktionen in den Ausdrücken verwenden, die unbedingt ausgeführt werden müssen. Die Funktion `testeIrgendetwas()` wird im Beispiel nicht ausgeführt, da bereits der Ausdruck `(3 > 4)` den Wert `false` liefert und damit das Ergebnis des gesamten Ausdrucks in jedem Fall `false` ist.

Bit-Operatoren

Bit-Operatoren werden auf Zahlen und boolesche Werte angewendet, die entsprechend ihrer binären Darstellung verknüpft werden (die Bits können z. B. für bestimmte Eigenschaften stehen). Die booleschen Werte `true` und `false` entsprechen dabei den Werten 1 und 0. Da die Operatoren in der JavaScript-Praxis eher geringe Bedeutung haben, werden sie im Rahmen dieses Buches nicht behandelt.



Ergänzende Lerninhalte: *Bit-Operatoren.pdf*

Zuweisungsoperatoren

Der einfachste Zuweisungsoperator ist das Gleichheitszeichen (=). Sie können damit den Wert von Variablen festlegen.

```
var a = 42.1;
```

Der Zuweisungsoperator kann in Verbindung mit anderen Operatoren eingesetzt werden und gilt für alle Datentypen. Eine häufige Operation ist es, bestehende Werte in Variablen zu verändern und erneut zuzuweisen.

```
a = a + 5;      // erhöhe den Wert in der Variablen a um 5
```

Da solche Anweisungen sehr häufig benötigt werden, gibt es für alle genannten binären Operatoren eine Kurzschreibweise, um die Operatorfunktion mit der Zuweisung zu verknüpfen. Das letzte Beispiel kann daher auch kürzer geschrieben werden:

```
a += 5;        // erhöhe den Wert in der Variablen a um 5
```

Operator	Wirkung	Beispiel	Ergebnis
=	Einfache Wertzuweisung	<code>a = 5;</code>	<code>a = 5</code>
+=	Addieren und Zuweisen	<code>a = 5;</code> <code>a += 5;</code>	<code>5 + 5</code> <code>a = 10</code>
+=	Anfügen und Zuweisen	<code>a = 'Ei';</code> <code>a += 'dotter';</code>	<code>'Ei' + 'dotter'</code> <code>a = 'Eidotter'</code>
-=	Subtrahieren und Zuweisen	<code>a = 5;</code> <code>a -= 2;</code>	<code>5 - 2</code> <code>a = 3</code>
*=	Multiplizieren und Zuweisen	<code>a = 5;</code> <code>a *= 2</code>	<code>5 * 2</code> <code>a = 10</code>
/=	Dividieren und Zuweisen	<code>a = 20;</code> <code>a /= 4</code>	<code>20 / 4</code> <code>a = 5</code>
%=	Modulo-Operation und Zuweisen	<code>a = 5;</code> <code>a %= 3</code>	<code>5 % 3</code> <code>a = 2</code>
&=	Bitweises UND und Zuweisen	<code>a = 5;</code> <code>a &= 3</code>	<code>0101 & 0011</code> <code>a = 1</code>

Operator	Wirkung	Beispiel	Ergebnis
<code>^=</code>	Bitweises XOR und Zuweisen	<code>a = 5;</code> <code>a ^= 3;</code>	0101 ^ 0011 <code>a = 6</code>
<code> =</code>	Bitweises ODER und Zuweisen	<code>a = 5;</code> <code>a = 3;</code>	0101 0011 <code>a = 7</code>
<code><<=</code>	Bitweises Linksverschieben und Zuweisen	<code>a = 5;</code> <code>a <<= 2</code>	0101 <code>a = 20</code>
<code>>>=</code>	Bitweises Rechtsverschieben und Zuweisen	<code>a = 40;</code> <code>a >>= 2;</code>	101000 <code>a = 10</code>
<code>>>>=</code>	Zero-Fill-Rechtsverschiebung und Zuweisen	<code>a = 100;</code> <code>a >>>= 4;</code>	1100100 <code>a = 6</code>

Bedingungsoperator

Mithilfe des Bedingungsoperators können einige bedingungsabhängige Anweisungen verkürzt dargestellt werden.

Ausdruck ? Truewert : Falsewert

Der Operator benötigt drei Operanden und heißt deshalb auch ternärer Operator. Er benötigt:

- ✓ einen logischen Ausdruck, der den Wert `true` oder `false` liefert,
- ✓ einen Rückgabewert eines beliebigen Datentyps, der zurückgegeben wird, wenn der Ausdruck den Wert `true` liefert (`if`-Zweig),
- ✓ einen Rückgabewert eines beliebigen Datentyps, der zurückgegeben wird, wenn der Ausdruck den Wert `false` liefert (`else`-Zweig).

Operator	Wirkung	Beispiel	Ergebniswert
<code>? :</code>	Bedingung	<code>stunde > 12 ? 'P.M.' : 'A.M.'</code>	Falls <code>stunde</code> größer 12 ist, wird die Zeichenkette <code>'P.M.'</code> , ansonsten <code>'A.M.'</code> geliefert.

Typenkonvertierung und `typeof`-Operator

JavaScript ist eine lose typisierte Sprache. Der Interpreter wandelt Operanden automatisch in die entsprechenden Datentypen um, damit eine Operation ohne Fehler ausgeführt werden kann.

Beispiel	Ergebniswert	Erklärung
<code>"Text plus Zahl " + 7</code>	<code>"Text plus Zahl 7"</code>	Konvertierung der Zahl 7 in einen String
<code>"Text plus Zahl " + 7 * 7</code>	<code>"Text plus Zahl 49"</code>	"Punkt vor Strich", dann Konvertierung

Beispiel	Ergebniswert	Erklärung
<code>7 + 7 + " Text plus Zahl "</code>	<code>"14 Text plus Zahl "</code>	Abarbeitung der Reihe nach
<code>"Text plus Zahl " + 7 + 7</code>	<code>"Text plus Zahl 77"</code>	Abarbeitung der Reihe nach
<code>"7" * 7</code>	49	Konvertierung der Ziffer 7 in die Zahl 7

Aufgrund der automatischen Typenkonvertierung kann es passieren, dass der aktuelle Typ einer Variablen nicht eindeutig klar ist. Der Operator `typeof` ermöglicht es, den Typ einer Variablen auszulesen. Er liefert einen der folgenden Datentypen, die eingangs besprochen wurden:

- ✓ `number` für Zahlen
- ✓ `string` für Zeichenketten
- ✓ `boolean` für Wahrheitswerte
- ✓ `undefined` für nicht initialisierte Variable

Beispiel	Rückgabewert als Typ
<code>var variable = 5; typ = typeof variable;</code>	<code>number</code>
<code>var variable = "Hallo"; typ = typeof variable;</code>	<code>string</code>
<code>var variable = true; typ = typeof variable;</code>	<code>boolean</code>
<code>typ = typeof variable1;</code>	<code>undefined</code>

Außerdem gibt es den Typ `function` für Funktionen und den Typ `object`, der beispielsweise bei Arrays zurückgeliefert wird.

Beispiel	Rückgabewert als Typ
<code>var variable = new function("5+2"); typ = typeof variable;</code>	<code>function</code>
<code>var variable = [1, 2, 3, 4]; typ = typeof variable;</code>	<code>object</code>

2.9 Rangfolge der Operatoren

Die Operatoren sind in ihrer Rangordnung festgelegt und werden dementsprechend nacheinander abgearbeitet:

Rangstufe (1 = höchste Priorität)	Operatorzeichen	Operatorname	Operatortyp
1	() []	Klammer	Gruppierung
2	! ++ -- - ~	NICHT Inkrement und Dekrement Negation NICHT	Logischer Operator Arithmetischer Operator Arithmetischer Operator Bit-Operator
3	* / %	Multiplikation Division Modulo-Operator	Arithmetische Operatoren
4	+	Aneinanderreihung	Konkatenationsoperator
5	+ -	Addition Subtraktion	Arithmetische Operatoren
6	<< >> >>>	Linksverschiebung Rechtsverschiebung Zero-Fill-Rechtsverschiebung	Bit-Operatoren
7	< > <= >=	Kleiner als Größer als Kleiner gleich Größer gleich	Vergleichsoperatoren
8	== === != !==	Werte sind gleich Werte sind ungleich	Vergleichsoperatoren
9	&	UND	Bit-Operator
10	^	ENTWEDER ODER	Bit-Operator
11		ODER	Bit-Operator
12	&&	UND	Logischer Operator
13		ODER	Logischer Operator
14	? :	Bedingung	Konditionaler Operator
15	= += -= *= /= %= &= ^= = <<= >>= >>>=	Zuweisungsoperatoren	Zuweisungsoperatoren
16	, (Komma)	Aneinanderreihung	

Geklammerte Ausdrücke besitzen die höchste Priorität und werden zuerst berechnet. Zur besseren Lesbarkeit und zur Vermeidung von Fehlern sollten Sie alle Ausdrücke, die mehr als zwei Operanden besitzen, durch das Setzen von Klammern erweitern.

<code>a = 8 + 2 * 7 + 3</code>	<code>// a = 8 + 14 + 3 = 25</code>
<code>a = (8 + 2) * (7 + 3)</code>	<code>// a = 10 * 10 = 100</code>

2.10 Übungen

Übung 1: Verschiedene Datentypen

Übungsdatei: --

Ergebnisdatei: *kap02/uebung1.html*

1. Geben Sie jeweils ein Beispiel für eine Zahl, eine Zeichenkette und einen booleschen Wert an.

Übung 2: Boolesche Werte

Übungsdatei: --

Ergebnisdatei: *kap02/uebung2.html*

1. Welchen Wert haben die folgenden Ausdrücke?
 - ✓ `3 < 8`
 - ✓ `true && false`
 - ✓ `6 + 4 * 2 - 1`
 - ✓ `"Hund" != "Katze"`

Übung 3: Steuerzeichen in Zeichenketten

Übungsdatei: --

Ergebnisdatei: *kap02/uebung3.html*

1. Erstellen Sie eine Zeichenkette, die durch die Verwendung von Steuerzeichen die folgende Ausgabe in einer Webseite durch die Funktion `document.write()` erzeugt:
 - ✓ einen Zeilenumbruch,
 - ✓ ein Zitat in Anführungszeichen.

Also z. B.:

Cato

"Ceterum censeo Carthaginem esse delendam"

3

Kontrollstrukturen

3.1 Steuerung des Programmablaufs

Eine Anweisung ist ein Teil des Programms oder Skripts, das ausgeführt wird, um eine bestimmte Aktion durchzuführen. Mehrere Anweisungen werden in sogenannten Anweisungsblöcken oder kurz Blöcken zusammengefasst. In JavaScript werden die Anweisungen normalerweise von oben nach unten sequenziell abgearbeitet. Oft ist es jedoch erforderlich, dass Programmteile mehrmals oder gar nicht abgearbeitet werden sollen.

Mit bedingten Ausführungen und Schleifen können Sie von der sequenziellen Abarbeitung abweichen. Ob Anweisungen innerhalb der Schleife ausgeführt werden, ist dann von einer bestimmten Bedingung abhängig. Die Anweisungen können dabei auch mehrfach ausgeführt werden.

3.2 Anweisungsblock

Eine Anweisung ist die kleinste ausführbare Einheit eines Programms. In JavaScript werden Anweisungen mit einem Semikolon (Strichpunkt) abgeschlossen (Ausnahmen gibt es, sollten aber meist nicht verwendet werden). Ein Skript kann es erforderlich machen, dass eine Folge von mehreren Anweisungen unter bestimmten Bedingungen ausgeführt werden muss. In diesem Fall werden die betreffenden Anweisungen in einem Anweisungsblock eingeschlossen.

```
{  
  Anweisungsblock mit beliebig vielen einzelnen Anweisungen  
}
```

Der Anweisungsblock wird durch geschweifte Klammern begonnen und beendet: `{ }`. Die Blöcke können auch verschachtelt sein, um Anweisungen innerhalb einer Funktion zu beschreiben. Dies bedeutet, ein geklammerter Anweisungsblock kann wiederum einen geklammerten Anweisungsblock beinhalten.

3.3 Auswahl

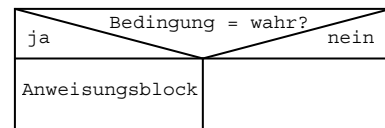
Nachfolgend finden Sie eine Aufstellung verschiedener Auswahlmöglichkeiten von JavaScript.

Einfache Bedingung

Im einfachsten Fall soll ein Anweisungsblock nur dann ausgeführt werden, wenn eine Bedingung erfüllt ist. Eine Bedingung wird durch einen Ausdruck definiert, der als Ergebnis `true` oder `false` (wahr oder falsch) liefert. Die einfache Bedingung ist dadurch gekennzeichnet, dass nach einer Bedingungsabfrage eine Anweisung oder ein Anweisungsblock ausgeführt wird oder nicht.

Wenn die Bedingung erfüllt ist, dann werden die Anweisungen ausgeführt. Ist die Bedingung nicht erfüllt, werden sie nicht ausgeführt. In JavaScript kann dieses Konstrukt mit der `if`-Anweisung gebildet werden:

```
if (Bedingung) {  
    Anweisungsblock;  
}
```



- ✓ Das Schlüsselwort `if` leitet die einfache Bedingung ein.
- ✓ Die Bedingung selbst steht in runden Klammern.
- ✓ Als Bedingung ist ein logischer Ausdruck anzugeben.
- ✓ Es wird geprüft, ob der Ausdruck den Wert `true` oder `false` zurückliefert, d. h., ob die nachfolgenden Anweisungen innerhalb der geschweiften Klammern ausgeführt werden sollen.
- ✓ Wenn der Ausdruck den Wert `false` hat, werden alle Anweisungen innerhalb des Anweisungsblocks ignoriert.
- ✓ Anstatt des Anweisungsblocks kann auch eine einzelne Anweisung verwendet werden, sofern nur eine Anweisung notwendig ist. Es ist aber guter Programmierstil, immer einen Anweisungsblock mit geschweiften Klammern zu notieren.
- ✓ Nach dem Ende des Anweisungsblocks ist die `if`-Anweisung beendet und alle weiteren Anweisungen werden unabhängig von der Bedingung abgearbeitet.

Beispiel

```
var angabe, einheit;  
angabe = "Gewicht";  
if (angabe == "Gewicht") {  
    einheit = "kg";  
}
```

Wenn die Variable `angabe` die Zeichenkette `Gewicht` enthält, wird der Variablen `einheit` die Maßeinheit `kg` zugewiesen.

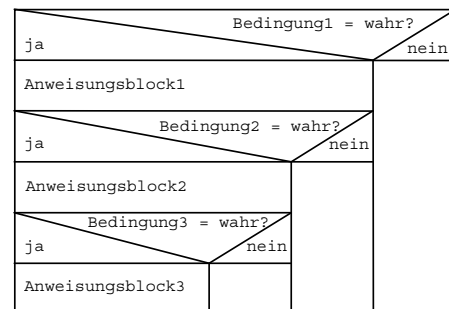
Soll in Abhängigkeit der Bedingung nur eine Anweisung ausgeführt werden, können die geschweiften Klammern entfallen. Von dieser Möglichkeit sollten Sie allerdings Abstand nehmen. Wenn Sie später weitere Anweisungen bedingt ausführen wollen, fehlt der Block, um diese darin zu notieren. Es ist eine häufige Fehlerquelle, dass man die neue Anweisung dann **nicht bedingt**, sondern **immer** ausführt, da man die nachträgliche Ergänzung der Klammern vergisst und die neue Anweisung dann nicht in Abhängigkeit von der Bedingung ausgeführt wird.

```
if (angabe == "Gewicht") einheit = "kg";
```

Verschachtelte if-Anweisungen

Das Ausführen eines Anweisungsblocks oder einer Anweisung kann von mehreren Bedingungen in mehreren Stufen abhängig sein. Hierzu werden die Bedingungsabfragen ineinander geschachtelt. Ist die erste Bedingung erfüllt, wird kontrolliert, ob auch die nächste Bedingung zutrifft. Trifft eine Bedingung nicht zu (*false*), wird die gesamte verschachtelte Auswahl verlassen.

```
if (Bedingung1) { // erste Stufe
    Anweisungsblock 1;
    if (Bedingung2) { // zweite Stufe
        Anweisungsblock 2;
        if (Bedingung3) { // dritte Stufe
            Anweisungsblock 3;
        }
    }
}
```

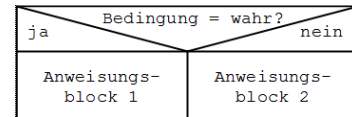


- ✓ Für die einzelnen if-Anweisungen gelten die gleichen Regeln wie bei der einfachen Bedingung.
- ✓ Beginnt eine verschachtelte if-Anweisung mit einer öffnenden geschweiften Klammer, muss sie auch wieder durch eine schließende geschweifte Klammer beendet werden. Zur besseren Lesbarkeit werden die einzelnen Verschachtelungen durch Einrücken des Quelltextes kenntlich gemacht.
- ✓ Der erste Anweisungsblock wird abgearbeitet, wenn die erste Bedingung zutrifft. Innerhalb des Blocks erfolgt dann die Prüfung der zweiten Bedingung. Trifft sie zu, erfolgt die Abarbeitung des zweiten Anweisungsblocks. Das Gleiche gilt für den dritten Anweisungsblock. Dieser wird nur abgearbeitet, wenn die erste, zweite und dritte Bedingung zutreffen.
- ✓ Sobald eine Bedingung nicht zutrifft, werden die im darauffolgenden Anweisungsblock enthaltenen Anweisungen nicht mehr ausgeführt. Stattdessen wird der Anweisungsblock übergangen und mit der nächsten Anweisung fortgefahren.

if-else-Anweisung

Die if-else-Anweisung beschreibt nicht nur, was passieren soll, wenn eine Bedingung erfüllt ist, sondern enthält auch die Anweisungen für den Fall, dass sie nicht erfüllt ist. Dazu wird nach dem Anweisungsblock der if-Anweisung ein weiterer Anweisungsblock hinzugefügt, der durch das Schlüsselwort else eingeleitet wird.

```
if (Bedingung) {  
    Anweisungsblock 1;  
}  
else {  
    Anweisungsblock 2;  
}
```



- ✓ Nach der Einleitung der if-Anweisung steht der Bedingungsausdruck.
- ✓ Trifft die Bedingung zu, werden alle Anweisungen des if-Blocks abgearbeitet. Der Anweisungsblock nach dem Schlüsselwort else wird nicht abgearbeitet.
- ✓ Liefert die Bedingung den Wert false, wird der Anweisungsblock nach else abgearbeitet. Der erste Anweisungsblock bleibt unberücksichtigt.

Beispiel

```
if (angabe == "Gewicht") {  
    einheit = "kg";  
}  
else {  
    einheit = "km";  
}
```

Wenn die Variable `angabe` die Zeichenkette `Gewicht` enthält, wird der Variablen `einheit` die Maßeinheit `kg` zugewiesen. Ansonsten erhält die Variable `einheit` die Maßeinheit `km` für die Länge.

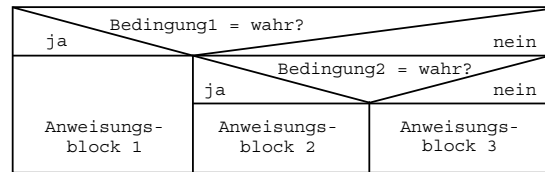
Mehrstufige if-else-Anweisungen

Bei einer mehrstufigen Bedingung wird einer von mehreren Anweisungsblöcken in Abhängigkeit von verschiedenen Bedingungen ausgeführt.

Wenn die erste Bedingung erfüllt ist, wird der erste Anweisungsblock ausgeführt. Ansonsten wird von mehreren Anweisungsblöcken derjenige durchgeführt, für den die Bedingung zutrifft. Sie können die mehrstufige Bedingung verwenden, wenn es mehr als zwei Auswahlmöglichkeiten gibt.

Die Einleitung der jeweils nächsten Bedingung erfolgt durch `else if`.

```
if (Bedingung1) {
    Anweisungsblock 1;
}
else if (Bedingung2) {
    Anweisungsblock 2;
}
else {
    Anweisungsblock 3;
}
```



- ✓ Es gelten die gleichen Regeln wie bei einer einfachen `if-else`-Anweisung. Die mehrstufige Struktur lässt sich an der Anweisung `else if` erkennen.
- ✓ Die letzte Stufe stellt die `else`-Anweisung dar. Die Anweisungen dieser Alternative werden ausgeführt, wenn keiner der zuvor geprüften Bedingungsausdrücke den Wert `true` geliefert hat. Diese Stufe ist optional.
- ✓ Der erste Anweisungsblock, dessen Bedingung zutrifft, wird abgearbeitet. Danach findet keine Bedingungsprüfung mehr statt. Nach Abarbeitung eines Anweisungsblocks wird jede `if`-Anweisung verlassen.

Beispiel

```
if (angabe == "Gewicht") {
    einheit = "kg";
}
else if (angabe == "Länge") {
    einheit = "km";
}
else {
    einheit = "s";
}
```

Wenn die Variable `angabe` die Zeichenkette `Gewicht` enthält, wird der Variablen `einheit` die Maßeinheit `kg` zugewiesen. Beinhaltet die Variable `angabe` den Wert `Länge`, erhält die Variable `einheit` den Wert `km`. Treffen die vorherigen Bedingungen nicht zu, erhält die Variable `einheit` die Maßeinheit `s`.

Mehrseitige Bedingung

Möchten Sie zwischen mehreren Fällen unterscheiden und die Ergebnisse entsprechend auswerten, können Sie statt mehrerer `if-else` die Schlüsselwörter `switch` und `case` verwenden. Die Auswahlanweisung ist aus folgenden Gründen empfehlenswert:

- ✓ Die Anwendung von `switch-case` ist kompakter und besser lesbar als mehrere `if-else`-Anweisungen.
- ✓ Man kann bei Bedarf eine „Fall-Through“-Anweisung erstellen. Bei dieser werden alle Anweisungen ab dem ersten Treffer abgearbeitet.

Bei einer Fallauswahl, auch Selektion genannt, wird bei der Verwendung von `switch-case` der Wert einer Variablen ausgewertet und in Abhängigkeit von diesem Wert eine Anweisung bzw. ein Anweisungsblock ausgeführt. Die Variable, deren Inhalt geprüft werden soll, wird auch als Selektor bezeichnet.

Der Selektor in einer `switch-case`-Anweisung kann immer nur auf **exakte Übereinstimmung** geprüft werden. Ein Vergleich auf größer oder kleiner ist nicht möglich. Diese Einschränkung erfordert bei Bedarf, dass Sie auf die `if-else`-Anweisung zurückgreifen müssen.

```
switch (Selektor) {
  case Wert1: Anweisungen; break;
  case Wert2: Anweisungen; break;
  ...
  default:    Anweisungen; break;
}
```

Variablen = ?			
Wert1	Wert2	Wert3	sonst
Anweisungen	Anweisungen	Anweisungen	Anweisungen

- ✓ `switch` und der in Klammern stehende und zu überprüfende Selektor leiten einen Anweisungsblock ein, der mehrere `case`-Anweisungen enthalten kann. Mindestens eine `case`-Anweisung ist zwingend.
- ✓ Innerhalb der `switch`-Abfrage werden die verschiedenen `case`-Abfragen definiert. Diese beinhalten den möglichen Wert der Bedingung. Stimmt ein Wert des Selektors mit dem Wert einer `case`-Abfrage überein, werden die entsprechenden Anweisungen nacheinander ausgeführt.
- ✓ Über das optionale Schlüsselwort `break` beenden Sie die Ausführung des aktuellen Anweisungsblocks und somit auch die `switch`-Abfrage. Das Schlüsselwort kann entfallen, wenn Sie eine „Fall-Through“-Anweisung erstellen wollen, um alle Anweisungen ab dem ersten Treffer auszuführen.
- ✓ Mit dem optionalen `default`-Block definieren Sie Anweisungen, die ausgeführt werden, wenn keine der vorherigen `case`-Abfragen mit dem Selektor übereinstimmt.
- ✓ Die Treffer sind nicht sortiert, sondern können vom Programmierer frei angeordnet werden. Theoretisch können Sie sogar mehrere `case`-Abfragen mit den gleichen Trefferwerten notieren, aber das ist nicht sinnvoll.

Beispiel

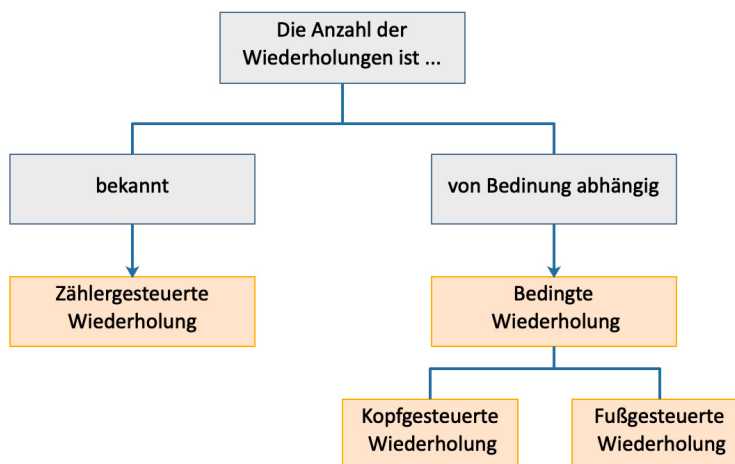
```
switch (angabe) {
  case "Gewicht": einheit = "kg"; break;
  case "Länge"   : einheit = "km"; break;
  case "Zeit"    : einheit = "s";  break;
  default       : einheit = "";    break;
}
```

In Abhängigkeit vom Wert der Variable `angabe` werden die einzelnen Wertzuweisungen für die Variable `einheit` durchgeführt. Hat die Variable `angabe` den Wert `Gewicht`, erhält die Variable `einheit` den Wert `kg` usw. Trifft keine der drei `case`-Abfragen zu, wird der Variablen `einheit` über den `default`-Zweig eine leere Zeichenkette zugewiesen.

3.4 Wiederholung

Oftmals müssen die gleichen Anweisungen mehrmals wiederholt werden. Meistens ist dabei auch nicht vorhersehbar, wie oft dies ausgeführt werden soll. Solche Wiederholungen werden in der Regel in Schleifen (auch Iterationen genannt) ausgeführt. Eine Schleife besteht aus einer Schleifensteuerung und einem Schleifenkörper. Die Schleifensteuerung gibt an, wie oft oder unter welcher Bedingung die Anweisungen abgearbeitet werden. Im Schleifenkörper sind die zu wiederholenden Anweisungen enthalten.

Es gibt verschiedene Arten von Wiederholungen.



Die Wiederholungen werden entweder gezählt, wenn die Anzahl bekannt ist, oder in Abhängigkeit einer Bedingung ausgeführt. Welche Schleifenart verwendet werden sollte, hängt von der Art der Aufgabe ab.

Zählergesteuerte Wiederholung

Eine zählergesteuerte Wiederholung realisieren Sie über eine Zählvariable, die von der Schleife bei jedem Durchlauf in der Regel um den Wert 1 erhöht wird.



Eine Zählvariable kann auch erniedrigt oder um einen anderen Wert als 1 erhöht werden. Diese Fälle werden in der Praxis kaum benötigt. Eine eventuell damit geplante Logik gehört nicht in die Schleifensteuerung, sondern in den Schleifenkörper.

Eine zählergesteuerte Wiederholung können Sie bequem mit der `for`-Schleife realisieren. Allerdings können Sie auch mit allen anderen Schleifenformen, die im Folgenden besprochen werden, zählergesteuerte Schleifen erstellen. Die `for`-Schleife wird mit dem Schlüsselwort `for` eingeleitet. Diese Einleitung wird von drei optionalen Anweisungen gefolgt. Mit dem ersten Ausdruck initialisieren Sie die Zählvariable. Über den zweiten Ausdruck können Sie die Abbruchbedingung der Schleife und im dritten Ausdruck die Schrittweite des Zählers festlegen.

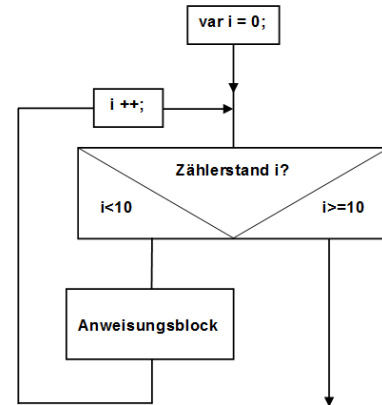
Eine `for`-Schleife hat den folgenden Aufbau:

```

for (Initialisierung; Bedingung; Aktualisierung) {
    Anweisung1;
    Anweisung2;
}
  
```

- ✓ Das Schlüsselwort `for` leitet die Zählschleife ein.
- ✓ In Klammern `()` und `[]` eingeschlossen und durch Semikolon `;` voneinander getrennt werden der Startwert der Schleife (`i=0`), die Abbruchbedingung (`i<10`) und die Zählweise (`i++`) festgelegt.
- ✓ Die Anweisungen im Schleifenkopf sind jeweils optional. Die Semikolons sind jedoch immer anzugeben.
- ✓ Sie können in jedem Bereich des Kopfs der `for`-Schleife mehrere Ausdrücke durch Kommata getrennt angeben (das ist jedoch schlechter Programmierstil), z. B.

```
for(var i = 0, j = 0; i < 10, j < 10; i++, j++)
```



Beispiel: `for.html`

Es wird der Wert der Zählvariable und somit die Anzahl der Schleifendurchläufe ausgegeben.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>for-Schleifen</title>
</head>
<body>
  <h3>Die for-Schleife</h3>
  <script type="text/javascript">
    ① for(var i = 0; i < 10; i++) {
    ②   document.write("i: ", i, "; Durchlauf: ", i+1,
      "<br />");
    }
  </script>
</body>
</html>

```

- ① Die Schleife beginnt mit dem Zählerwert 0. In jedem Durchlauf wird die Zählvariable `i` um den Wert 1 erhöht (`i++`). Besitzt die Variable `i` den Wert 10, wird der Schleifendurchlauf beendet und die Anweisungen nach der Schleife werden ausgeführt.
- ② Im Browser werden der aktuelle Wert der Zählvariablen und der Durchlauf ausgegeben. Dadurch, dass die Zählung bei 0 beginnt, muss bei der Ausgabe des Durchlaufs der Wert 1 hinzuaddiert werden.

Die for-Schleife

```

i: 0; Durchlauf: 1
i: 1; Durchlauf: 2
i: 2; Durchlauf: 3
i: 3; Durchlauf: 4
i: 4; Durchlauf: 5
i: 5; Durchlauf: 6
i: 6; Durchlauf: 7
i: 7; Durchlauf: 8
i: 8; Durchlauf: 9
i: 9; Durchlauf: 10

```

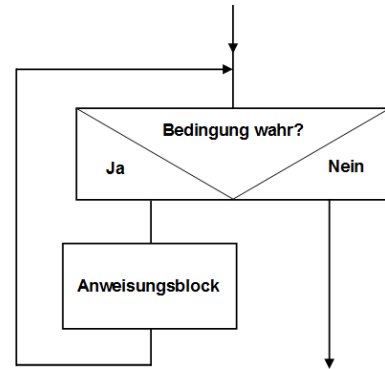
Zehn Schleifendurchläufe
mit einer `for`-Schleife

Kopfgesteuerte Wiederholung

In der `while`-Schleife wird das Ausführen von Anweisungen von der Gültigkeit eines beliebigen Ausdrucks abhängig gemacht. Der Ausdruck liefert einen booleschen Wert zurück. Ist der Wert `true`, werden die folgenden Anweisungen ausgeführt. Bei dem Rückgabewert `false` wird die Ausführung nach der `while`-Anweisung fortgesetzt. Der Ausdruck wird in jedem Durchlauf zu Beginn der `while`-Anweisung ausgewertet.

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

Das Schlüsselwort `while` leitet die kopfgesteuerte Wiederholung ein. Es folgt in runden Klammern `()` ein Ausdruck, der einen booleschen Wert liefert. Ist dieser `true`, werden die Anweisungen im Schleifenkörper ausgeführt. Nach dem Ausdruck folgt eine Anweisung oder ein ganzer Anweisungsblock. Meist arbeitet man zur Beschreibung einer Bedingung – wie bei der `for`-Schleife – mit einer Zählvariablen, die im Inneren des Schleifenkörpers erhöht wird.



Nachdem das Programm die Anweisungen im Schleifenkörper ausgeführt hat, wird der Ausdruck am Anfang der Schleife erneut geprüft. Falls dieser den Wert `false` zurückliefert, wird der Schleifenkörper übergangen und die erste Anweisung nach der Schleife durchgeführt. Bei Rückgabe von `true` wird der Schleifenkörper erneut ausgeführt.

Eine `while`-Schleife wird nicht in jedem Fall durchlaufen. Liefert der Ausdruck gleich zu Beginn den Wert `false`, wird keine Anweisung der Schleife ausgeführt.

Beispiel: *while.html*

In diesem Beispiel wird das Beispiel der `for`-Schleife über die `while`-Schleife realisiert.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>while-Schleifen</title>
</head>
<body>
  <h3>Die while-Schleife</h3>
  <script type="text/javascript">
    ①   var i = 0;
    ②   var count = 0;
    ③   while (i < 10) {
```



```

④      document.write("i: ", i, "; Durchlauf: ", ++count,
        "<br />"); i++;
    }
    </script>
</body>
</html>

```

- ① Die Variable `i` wird initialisiert und mit dem Startwert 0 belegt.
- ② Die Variable `count` zum Zählen der Schleifendurchläufe erhält den Wert 0.
- ③ Die `while`-Schleife darf nur so oft durchlaufen werden, wie der Wert der Variablen `i` kleiner 10 ist. Beim ersten Durchlauf ist diese Bedingung im Beispiel erfüllt.
- ④ Im Browser werden der aktuelle Wert der Zählvariablen und der Durchlauf ausgegeben. Vor der Ausgabe des Schleifendurchlaufs wird der Wert der Variablen `count` um eins erhöht.

Die while-Schleife

```

i: 0; Durchlauf: 1
i: 1; Durchlauf: 2
i: 2; Durchlauf: 3
i: 3; Durchlauf: 4
i: 4; Durchlauf: 5
i: 5; Durchlauf: 6
i: 6; Durchlauf: 7
i: 7; Durchlauf: 8
i: 8; Durchlauf: 9
i: 9; Durchlauf: 10

```

*Zehn Schleifendurchläufe
mit einer while-Schleife*

Fußgesteuerte Wiederholung

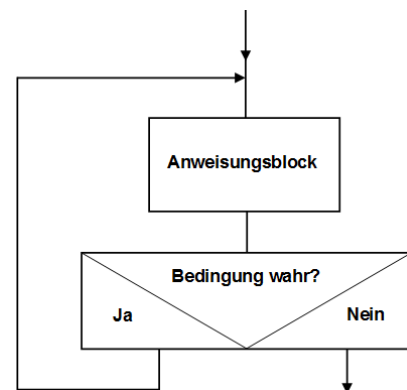
Bei einer fußgesteuerten Schleife, auch `do-while`-Schleife genannt, wird der Anweisungsblock in jedem Fall mindestens einmal ausgeführt. Erst am Ende der Schleife wird getestet, ob der Durchlauf wiederholt wird oder nicht. Ansonsten ist die Schleife identisch zur `while`-Schleife.

```

do {
    Anweisungsblock;
} while (Bedingung);

```

Der Bedingungsausdruck, der erfüllt werden soll, befindet sich am Ende der Schleife. Der Schleifenkörper wird ausgeführt, solange der Bedingungsausdruck am Ende der Schleife den Wert `true` liefert. Ist die Bedingung schon nach dem ersten Durchlauf nicht erfüllt, wird die Schleife dennoch mindestens einmal ausgeführt.



Beispiel: *dowhile.html*

Im folgenden Beispiel wird die Ausgabeanweisung genau einmal ausgeführt, da der Ausdruck der `do-while`-Schleife bereits bei der ersten Auswertung den Wert `false` liefert.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>do-while-Schleife</title>
</head>
<body>
  <h3>Die do-while-Schleife</h3>
  <script type="text/javascript">
    ①   var i = 5;
    ②   var count = 0;
    ③   do {
    ④     document.write("i: ", i, "; Durchlauf: ", ++count,
        "<br />");
    ⑤     i++;
    ⑥   } while (i < 5);
  </script>
</body>
</html>

```

- ① Die Variable `i` wird initialisiert und mit dem Startwert 5 belegt.
- ② Die Variable `count` zum Zählen der Schleifendurchläufe erhält den Wert 0.
- ③ Mit dem Schlüsselwort `do` wird die fußgesteuerte Wiederholung eingeleitet. Zu Beginn der Schleife findet noch keine Bedingungsprüfung statt.
- ④ Wie in den vorherigen Beispielen werden der Wert der Zählvariablen und die aktuelle Schleifenanzahl ausgegeben.
- ⑤ Die Zählvariable `i` wird um 1 erhöht.
- ⑥ Am Ende der Schleife wird der Ausdruck ausgewertet. Da die Bedingung `i < 5` nicht erfüllt ist, wird die Schleife verlassen.

Die do-while-Schleife

i: 5; Durchlauf: 1

*Einmaliger Durchlauf
einer do-while-Schleife*

Schleifensteuerung mit Sprunganweisungen

Die `break`- und `continue`-Anweisungen bieten die Möglichkeit, eine Schleife oder einen Schleifendurchlauf vorzeitig abubrechen. Man nennt sie deshalb Sprunganweisungen.

Das Schlüsselwort `break` verlässt die gesamte Schleife und beginnt mit der Anweisung, die der abgebrochenen Schleife unmittelbar folgt. Die Anweisung `continue` dagegen beendet einen aktuellen Schleifendurchlauf, beendet aber die Schleife nicht, sondern fährt mit dem nächsten Schleifendurchlauf fort.

- Im Gegensatz zur `break`-Anweisung kann ein falsch platziertes `continue` zu Endlosschleifen führen. Deshalb sollten Sie es mit Vorsicht benutzen.

Beispiel: `continue_break.html`

Das Beispiel zeigt die Auswirkungen beider Schlüsselwörter. Zuerst wird mit `continue` vorzeitig ein neuer Schleifendurchlauf erzwungen. Im zweiten Teil des Beispiels wird mit `break` eine `while`-Schleife vorzeitig verlassen.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>continue und break</title>
</head>
<body>
  <h3>continue</h3>
  <script type="text/javascript">
    ① var count = 0;
    ② for (var i = 0; i < 5; i++) {
    ③   if (i == 2) continue;
    ④   document.write("i: ", i, "; Ausgabe: ", ++count,
      "<br />");
    }
  </script>
  <h3>break</h3>
  <script type="text/javascript">
    ⑤ var i = 0;
    var count = 0;
    ⑥ while (i < 5) {
    ⑦   if (i == 2) break;
    ⑧   document.write("i: ", i, "; Ausgabe: ", ++count,
      "<br />");
    ⑨   i++;
    }
  </script>
```

- ① Die Variable `count` zum Zählen der Schleifendurchläufe wird initialisiert.
- ② Die `for`-Schleife soll fünfmal durchlaufen werden (`i=0 . . . 4`).
- ③ Über die `if`-Anweisung wird der aktuelle Wert der Zählvariablen überprüft. Wenn diese den Wert 2 besitzt, wird über die Anweisung `continue` zum Schleifenbeginn verzweigt.
- ④ Im Browser werden der aktuelle Wert der Zählvariablen und der Durchlauf ausgegeben. Vor der Ausgabe des Schleifendurchlaufs wird der Wert der Variablen `count` um 1 erhöht.

- ⑤ Im zweiten Teil des Beispiels werden die Variable `i` sowie die Variable `count` initialisiert.
- ⑥ Die `while`-Schleife soll wiederholt werden, solange der Wert von `i` kleiner als 5 ist.
- ⑦ Besitzt die Variable `i` beim Test innerhalb der `if`-Anweisung den Wert 2, wird die `while`-Schleife vorzeitig über die `break`-Anweisung verlassen.
- ⑧ Es werden der aktuelle Zählerstand und der aktuelle Schleifendurchlauf ausgegeben.
- ⑨ Die Zählvariable `i` wird um den Wert 1 erhöht.

continue

`i: 0; Ausgabe: 1`
`i: 1; Ausgabe: 2`
`i: 3; Ausgabe: 3`
`i: 4; Ausgabe: 4`

break

`i: 0; Ausgabe: 1`
`i: 1; Ausgabe: 2`

*Vorzeitiges Unterbrechen
von Schleifendurchläufen*

Beispiel

Um die Anzahl der möglichen Passworteingaben in einem Formularfeld zu begrenzen, können Sie eine Schleife verwenden.

```
var versuch = 0;
do {
  if (versuch === 3) break;
  nutzereingabe = passwortabfrage();
  versuch++;
} while (nutzereingabe !== passwort);
```

Die `while`-Schleife wird so lange durchlaufen, bis das Passwort richtig eingegeben wurde. Die Abfrage `if (versuch == 3) break;` bricht die Schleife jedoch ab, wenn drei Versuche erfolglos waren.

JavaScript stellt auch **benannte** `break`- und `continue`-Anweisungen bereit, um damit über die Angabe einer Sprungmarke gezielt aus Schleifen (auch aus verschachtelten) herauszuspringen. Diese Technik sollten Sie allerdings nie anwenden, denn sie ist fehlerträchtig, kaum wartbar oder lesbar und erinnert an den Spaghetticode früherer Jahre. Im Buch wird auf diese Sprungmarken nicht eingegangen.



Ergänzende Lerninhalte: *JavaScript-Techniken für ältere Skripte.pdf*



Achten Sie bei der Arbeit mit Schleifen immer darauf, dass das Abbruchkriterium auch tatsächlich eintreten kann. Ansonsten wird der Schleifenkörper unendlich oft ausgeführt oder so lange, bis ein Fehler bei der Zuweisung entsteht. Dies kann z. B. der Fall sein, wenn Sie zu einer Zahl einen konstanten Wert addieren. Bei unendlicher Ausführung wird der Wertebereich dieser Zahl überschritten.

3.5 Das KISS-Prinzip

Gerade als Anfänger neigt man bei der Programmierung dazu, eher kompliziert zu programmieren. Dabei sollte Programmierung kein Selbstzweck sein und nicht demonstrieren, was man alles schon an Kniffen und Programmierdingen kann, sondern vorgegebene Aufgaben lösen. Auf der anderen Seite schaut man auch oft zu stark auf die reine Lösung eines Problems und vergisst eine spätere Wartung und die Verständlichkeit des Quellcodes. Eine gute Programmierung löst eine Aufgabe zuverlässig, aber auch performant, ressourcenschonend und wartbar.

Etabliert hat sich in der professionellen Programmierung das sogenannte **KISS-Prinzip**: Das KISS-Prinzip (englisch *Keep it simple, stupid* oder auch *Keep it simple [and] stupid*, *Keep it short and simple*, *Keep it simple and smart*, *Keep it simple and straightforward* oder *Keep it simply stupid*) fordert, zu jedem Problem eine möglichst einfache Lösung anzustreben. Wenn es mehrere Möglichkeiten für einen bestimmten Sachverhalt gibt, dann ist diejenige Variante zu bevorzugen, die am einfachsten ist und mit den wenigsten Annahmen, Ausnahmen und Variablen auskommt. Insbesondere in JavaScript hilft die Reduzierung seines Programmierstils auf diese Idee, sowohl professioneller und gleichzeitig viel einfacher zu programmieren.

Dabei gibt es keine „verbindlichen“ Regeln, um diese Idee umzusetzen – aber einige „unverbindliche“ Ansätze, die in diese Richtung gehen und teils schon erwähnt wurden:

- ✓ Halten Sie sich zu 100 % an Namenskonventionen.
- ✓ Allgemein sollte jede Form von Ausnahme im Programmierstil vermieden werden. Programmieren Sie „erwartbar“. Dazu zählt beispielsweise, dass Sie sich an gängige Notationen halten, die die meisten anderen Programmierer verwenden. So wird etwa die Zählvariable einer Schleife `i` lauten. Eine darin verschachtelte Schleife mit einer Zählvariablen wird diese `j` nennen, eine wieder darin verschachtelte `k` usw.
- ✓ Alle Zählvariablen von Schleifen werden mit dem Wert 0 initialisiert und um den Wert 1 erhöht.
- ✓ Variablen werden grundsätzlich initialisiert.
Ausnahme: Sie wollen explizit einen undefinierten Zustand für eine gewisse Logik nutzen. Aber dann wird ja in JavaScript explizit mittels `var` oder `let` die „Initialisierung“ auf `undefined` vorgenommen.
- ✓ Globale Variablen werden – soweit es geht – vermieden.
- ✓ Vermeiden Sie es, den Datentyp einer Variable zu ändern.
- ✓ In Entscheidungsstrukturen, Schleifen etc. wird immer eine Blockanweisung notiert – auch wenn da nur eine einzige Anweisung notwendig ist und man auf die Notation eines Blocks verzichten könnte.
- ✓ Alle Anweisungen werden mit einem Semikolon beendet, auch wenn man darauf verzichten kann. Wenige Ausnahmen wie der Inkrement-Teil der `for`-Schleife sind aber zugelassen.
- ✓ Wenn Sie öffnende und schließende geschweifte Klammern benötigen, notieren Sie diese immer an der gleichen Stelle. Wenn Sie etwa einen Block mit einer geschweiften Klammer eröffnen, steht diese Klammer entweder in einer neuen Zeile oder direkt am Ende der vorherigen Zeile. Es ist – sofern es keine Richtlinien in einem Projekt gibt – gleichgültig, welche Variante Sie wählen. Aber nutzen Sie diese konsequent und ohne Ausnahme.
- ✓ Wiederholen Sie sich nicht. Das Konzept hat auch einen eigenen Namen (DRY – Don't repeat yourself). Erstellen und/oder nutzen Sie (getestete) Strukturen (Funktionen, Objekte, Methoden, etc.).

3.6 Übungen

Übung 1: Fallauswahl mit **switch**

Übungsdatei: --

Ergebnisdatei: *kap03/uebung1.html*

1. Prüfen Sie über eine mehrseitige Fallauswahl, ob das HTML-Dokument über einen auf Netscape basierenden Browser wie Firefox oder den Internet Explorer betrachtet wird. Wird keiner der angegebenen Browser verwendet, ist dessen Name anzuzeigen. Den Namen der Browser ermitteln Sie über `navigator.appName`, z. B. `document.write(navigator.appName)` ;
Beachten Sie, dass ganz neue Versionen vom Internet Explorer diese Kennung nicht mehr als „Microsoft Internet Explorer“ bereitstellen.

Übung 2: Schleifendurchlauf mit **for**

Übungsdatei: --

Ergebnisdatei: *kap03/uebung2.html*

1. Geben Sie über eine `for`-Schleife die Quadrate der Zahlen 1...10 aus.

Übung 3: Bedingungsgemäßer Abbruch einer kopfgesteuerten Wiederholung

Übungsdatei: --

Ergebnisdatei: *kap03/uebung3.html*

1. Geben Sie so lange über eine `while`-Schleife die Quadrate der Zahlen 1...n aus, bis das Produkt größer als 1000 ist.

4

Funktionen

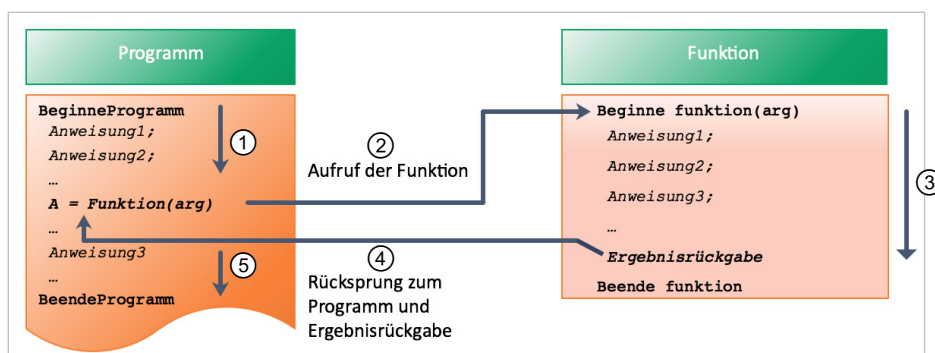
4.1 Grundlagen zu Funktionen

Funktionen sind eigenständige Unterprogramme, die von Anweisungen direkt aus dem Skript oder der Webseite aufgerufen werden. Diese beinhalten z. B. immer wiederkehrende Anweisungen. Anstatt wiederholt denselben Code anzugeben, wird er in eine Funktion ausgelagert und kann über den Funktionsaufruf beliebig oft aufgerufen werden.

Beispiel

Funktionen können dem Aufrufer einen Wert zurückliefern und daher in Ausdrücken verwendet werden. Man nennt diesen Wert einen **Rückgabewert**. Wie in der Mathematik entsprechen die Funktionen dabei einem bestimmten Wert selbst, z. B.

```
var resultat = 0;
resultat = addiere(2, 3) // resultat hat nach dem Aufruf der
                        Funktion den Wert 5
```



Aufruf einer Funktion

- ✓ Das Skript wird bis zum Aufruf einer Funktion abgearbeitet ①.
- ✓ Der Funktionsaufruf ② erzwingt einen Sprung in die angegebene Funktion.
- ✓ Jetzt werden die Anweisungen der Funktion abgearbeitet ③.
- ✓ Mit dem Verlassen der Funktion wird zurück zum Aufrufer gesprungen, dem das Ergebnis der Funktion übergeben wird ④. Im Beispiel wird der Rückgabewert der Variablen A zugewiesen.
- ✓ Das Skript wird weiterverarbeitet ⑤.

Syntax einer Funktion

```
function funktionsname([Parameter 1, Parameter 2,... Parameter n])
{
    Anweisung1;
    Anweisung2;
    Anweisung3;
    return Wert;
}
```

- ✓ Die Anweisung `function` leitet eine Funktion ein.
- ✓ `funktionsname` ist der Bezeichner der Funktion, mit dem diese im weiteren Verlauf aufgerufen werden kann. Nach Konvention werden Bezeichner einer Funktion kleingeschrieben. Damit geben Sie einen sogenannten Funktionszeiger (auch **Funktionsreferenz** genannt) an. Der Funktionsname ist optional und kann nur bei sogenannten **anonymen Funktionen** oder **Lambda-Ausdrücken** weggelassen werden. Diese liefern immer einen Funktionszeiger als Rückgabewert, über den sie dann verwendet werden können.
- ✓ In den runden Klammern `()` werden optional Parameter angegeben, die einen beliebigen Typ besitzen können und deren Werte innerhalb der Funktion verarbeitet werden sollen. Die Klammern müssen bei der Deklaration in jedem Fall angegeben werden, auch wenn die Funktion keine Parameter erwartet, z. B. `function warnung()`.
- ✓ Innerhalb der geschweiften Klammern `{ }` werden die einzelnen Anweisungen angegeben.
- ✓ Mit dem optionalen Schlüsselwort `return` und einem ebenfalls optionalen Rückgabewert können Sie eine Funktion an jeder Stelle wieder verlassen.
- ✓ Eine Funktion muss vor ihrer ersten Verwendung deklariert werden. Befinden sich die Funktionsdefinition und der Funktionsaufruf in verschiedenen `<script>`-Abschnitten, müssen Sie sicherstellen, dass die Funktion vor ihrem Aufruf deklariert worden ist und dem JavaScript-Interpreter somit bekannt ist.

Der Funktionsaufruf

Nach der Funktionsdeklaration kann die Funktion per Funktionsaufruf ausgeführt werden. Funktionen werden über den Namen aufgerufen, mit dem sie deklariert wurden. Wenn die Funktion aufgerufen wird, werden die Anweisungen in dieser Funktion abgearbeitet und eventuelle Ergebnisse über das Schlüsselwort `return` dem Aufrufer übergeben.

Im Folgenden wird zwischen `Anweisung2` und `Anweisung3` die Funktion `funktionsname()` ausgeführt und ihr Rückgabewert der Variable `x` zugewiesen.

```
Anweisung1;
Anweisung2;
x = funktionsname();
Anweisung3;
```

Funktionen werden nur dann ausgeführt, wenn sie explizit aufgerufen werden.

4.2 Funktionen mit Parametern

Beim Aufruf einer Funktion können Werte übergeben werden, mit denen in der Funktion gearbeitet werden soll. Diese Werte werden Parameter genannt und sind in der Funktion lokale Variablen. Es können beliebig viele Parameter beim Funktionsaufruf übergeben werden. Die Parameter werden bei der Definition und beim Aufruf durch Kommata getrennt in runden Klammern angegeben. Die Übergabe der Parameter erfolgt immer als Wert, d. h., Änderungen der lokalen Parameterwerte innerhalb der Funktion haben keine Auswirkung auf den Wert des übergebenen Parameters außerhalb der Funktion. Für den Aufruf einer Funktion ist jedoch ausschließlich der Bezeichner (die Funktionsreferenz) relevant, nicht aber die Parameterliste. Das bedeutet, dass man bei einem Funktionsaufruf nicht die gleiche Anzahl an Parametern angeben muss, wie bei der Funktionsdeklaration notiert wurden.

Der Aufruf einer Funktion mit Parameterübergabe hat den folgenden Aufbau:

```
function funktionsname (Parameter1, Parameter2) {  
    // JavaScript - Anweisungen  
    return rueckgabewert;  
}  
Anweisung1;  
Anweisung2;  
x = funktionsname(Parameter1, Parameter2);  
Anweisung3;
```

Zuerst wird die Funktion deklariert, die in diesem Fall über das Schlüsselwort `return` einen Wert zurückliefert. Zwischen `Anweisung2` und `Anweisung3` wird die Funktion aufgerufen und es werden die Werte `Parameter1` und `Parameter2` übergeben. Der JavaScript-Interpreter verzweigt somit in die vorher deklarierte Funktion, führt die verschiedenen JavaScript-Anweisungen aus und liefert einen Wert zurück. Dieser Rückgabewert wird im oberen Beispiel in der Variablen `x` für weitere Bearbeitungsschritte gespeichert.

4.3 Variable Parameterliste

Wenn Sie bei der Deklaration einer Funktion keine Parameter spezifizieren, kommen Sie dennoch an eventuell beim Aufruf angegebene Übergabewerte heran. Über das in jeder Funktion immer standardmäßig verfügbare Feld `arguments` können Sie ebenfalls auf alle Parameter einer Funktion zugreifen. Der Zugriff erfolgt dabei allerdings über den Index des Parameters und nicht über den Parameternamen. Die Anzahl der übergebenen Parameter erhalten Sie über `arguments.length`. Auf diese Weise können Sie einer Funktion eine variable Parameterliste übergeben.

Beispiel: [args_funktion.html](#)

Die Funktion `test()` besitzt im folgenden Beispielcode laut Deklaration zwei Parameter. Beim Aufruf der Funktion werden jedoch noch weitere Parameter übergeben. Durch das Auswerten der Anzahl der Parameter (`arguments.length`) werden die Inhalte der zusätzlichen Parameter angezeigt.

```
function test(par1, par2) {
    document.write("Parameter 1 mit dem Wert: " + par1 + "<br />");
    document.write("Parameter 2 mit dem Wert: " + par2 + "<br />");
    for(var i = 2; i < arguments.length; i++) {
        document.write("Zusätzlicher Parameter " + (i - 1) + "
            mit dem Wert: " + arguments[i] + "<br />");
    }
}
test(10, 11, 12, 13, 14, 15, 16);
```

Beispiel: *funktion.html*

Es wird eine Funktion deklariert, die den Flächeninhalt eines Kreises anhand der folgenden mathematischen Formel berechnet. Die Funktion `flaeche()` soll anhand eines vorgegebenen Durchmessers den entsprechenden Flächeninhalt des Kreises zurückliefern.

$$A = \pi / 4 * d^2$$

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Funktionsaufrufe</title>
  </head>
  <body>
    <h3>Berechnen einer Kreisfläche</h3>
    <p>nach der Formel: <code>A = &pi; /
      4 * d<sup>2</sup></code></p>
    <script type="text/javascript">
      ① function flaeche(d) {
      ②   var pi = Math.PI;
      ③   var a = (pi / 4) * d * d;
      ④   return a;
      ⑤ }
      ⑥ var ergebnis = 0;
      ⑦ var durchmesser = 10;
      ⑧ ergebnis = flaeche(durchmesser);
      document.write("d=", durchmesser, " m; A=", ergebnis,
        "m<sup>2</sup><br />");
    </script>
  </body>
</html>
```

- ① Die Funktion zum Berechnen der Kreisfläche erhält den Namen `flaeche`, mit dem sie innerhalb des HTML-Dokuments aufgerufen werden kann. Als Übergabeparameter erhält sie den entsprechenden Kreisdurchmesser, der in der Funktion über die Variable `d` genutzt werden kann.
- ② Mit der Angabe von `Math.PI` wird auf die in JavaScript hinterlegte Konstante für π zugegriffen und diese der lokalen Variablen `PI` zugewiesen. `Math` ist dabei die Klasse für die internen mathematischen JavaScript-Funktionalitäten (`PI` ist eine Eigenschaft der Klasse `Math`, siehe Kapitel 5).

Beachten Sie, dass man Variablen auch großschreiben kann (wie hier) und die Abweichung vom Namenskonzept durchaus in Ausnahmefällen sinnvoll sein kann, wenn damit bestimmte Metainformationen vermittelt werden sollen. Die Bezeichner `PI` und `A` sind mit mathematischen Metainformationen behaftet und dann ist dieser Bruch der Namenskonventionen sogar sehr sinnvoll.

- ③ Der Flächeninhalt wird nach der mathematischen Formel berechnet und in der Variablen `A` abgelegt.
- ④ Über `return` wird die errechnete Kreisfläche an den Aufrufer ⑦ zurückgegeben.
- ⑤ In der Variablen `ergebnis` soll der berechnete Wert abgelegt werden.
- ⑥ Die Variable `durchmesser` wird mit dem Durchmesser des Kreises initialisiert.
- ⑦ Die Funktion `flaeche()` wird mit dem Wert des Durchmessers aufgerufen. Der Interpreter verzweigt in die Funktion und führt die dortigen JavaScript-Anweisungen aus. Der Rückgabewert der Funktion, also das Ergebnis der Berechnung, wird der Variablen `ergebnis` übergeben.
- ⑧ `document.write()` gibt das Ergebnis in das HTML-Dokument aus, das im Browser angezeigt wird.

Berechnen einer Kreisfläche

nach der Formel: $A = \pi / 4 * d^2$

`d=10m;`

`A=78.53981633974483 m2`

Mathematische Berechnung

Beispiel: `funktion_schleife.html`

Das vorherige Beispiel soll erweitert werden, indem der Funktionsaufruf durch eine `for`-Schleife wiederholt wird und somit mehrere Kreisflächen berechnet werden. Zudem wird die Funktion `flaeche()` auf eine Anweisung reduziert.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Funktionsaufrufe</title>
  </head>
  <body>
    <h3>Berechnen mehrerer Kreisflächen</h3>
    <script type="text/javascript">
      ① function flaeche(d) {
      ②   return (Math.PI / 4 * d * d);
    }
  </script>
</body>
</html>
```

```

③   for (dm = 0; dm <= 100; dm +=5) {
④       document.write("d=", dm, " m; A=", flaeche(dm),
           " m<sup>2</sup><br />");
       }
    </script>
</body>
</html>

```

- ① Die Funktion `flaeche()` mit dem Übergabeparameter `d` für den zu berechnenden Durchmesser wird eingeleitet.
- ② Das Ergebnis der Berechnung wird als Ausdruck dem Schlüsselwort `return` zugewiesen und an den Aufrufer zurückgegeben.
- ③ Über eine `for`-Schleife sollen die verschiedenen Kreisflächen in 5er-Schritten berechnet werden. Der vorgegebene Durchmesser liegt dabei zwischen 0 und 100 Metern.
- ④ Da der Aufruf der Funktion `flaeche()` den berechneten Wert zurückliefert, können Sie diesen auch direkt über `document.write()` ausgeben.

Berechnen mehrerer Kreisflächen

nach der Formel: $A = \pi / 4 \cdot d^2$

d=0 m; A=0 m²
 d=5 m; A=19.634954084936208 m²
 d=10 m; A=78.53981633974483 m²
 d=15 m; A=176.71458676442586 m²
 d=20 m; A=314.1592653589793 m²
 d=25 m; A=490.8738521234052 m²
 d=30 m; A=706.8583470577008 m²
 d=35 m; A=962.1127642575813 m²
 d=40 m; A=1256.6370614359172 m²
 d=45 m; A=1589.903174380165 m²
 d=50 m; A=1963.4954084936208 m²
 d=55 m; A=2376.8539816339745 m²
 d=60 m; A=2829.981633974483 m²
 d=65 m; A=3321.903174380165 m²
 d=70 m; A=3852.70614359172 m²
 d=75 m; A=4422.42575813 m²
 d=80 m; A=5031.127642575813 m²
 d=85 m; A=5678.8539816339745 m²
 d=90 m; A=6365.70614359172 m²
 d=95 m; A=7091.793174380165 m²
 d=100 m; A=7853.981633974483 m²

Mehrfacher Aufruf einer Funktion

4.4 Weitere Möglichkeiten für die Deklaration von Funktionen

Funktionsdeklaration von Bedingungen abhängig machen

Eine Funktionsdeklaration kann auch von einer Bedingung abhängig gemacht werden. Auf diese Weise können Sie z. B. Funktionen mit gleichen Namen, aber unterschiedlichen Funktionalitäten definieren.

Beispiel: *if_funktion.html*

Die Wirkungsweise der Funktion `operation()` wird vom Inhalt der Variablen `name` abhängig gemacht. Auf diese Weise können Sie die Funktionalität eines Programms auf andere Weise vom Wert einer Variablen abhängig machen.

```

name = "Addition";
if(name == "Addition") {
    function operation(zahl1, zahl2) { return zahl1 + zahl2; }
}
else {
    function operation(zahl1, zahl2) { return zahl1 - zahl2; }
}
document.write(operation(10, 11));

```

Die Möglichkeit, die Funktionsdeklaration von Bedingungen abhängig zu machen, ist ab JavaScript 1.5 verfügbar, wird aber in der Praxis selten eingesetzt.

Vorgabewert von Parametern

Bei einer Funktionsdeklaration kann man Parametern bereits einen Vorgabewert zuweisen (Vorgabeparameter). Wenn der Parameter beim Aufruf nicht gesetzt wird, wird der Vorgabewert verwendet. Beachten Sie, dass Vorgabeparameter nur **am Ende** der Parameterliste stehen können.

Beispiel: *vorgabe.js*

```
function multi(a, b = 1) {  
    return a * b;  
}  
console.log(multi(4, 5))  
console.log(multi(4))
```

Wenn der zweite Parameter beim Aufruf der Funktion nicht angegeben wird, wird dafür der Wert 1 verwendet.

Anonyme Funktionen

Durch die Deklaration von Funktionen in Ausdrücken haben Sie eine weitere Möglichkeit, die Programmausführung dynamisch zu ändern. Statt eines Funktionsnamens wird die gesamte Funktionsdeklaration im Ausdruck angegeben. Der einzige Unterschied zur üblichen Funktionsdeklaration liegt im Weglassen des Funktionsnamens. Deshalb werden diese Funktionen auch als **anonyme Funktionen** bezeichnet.

Bei modernen RIAs, bei echter objektorientierter Programmierung mit JavaScript und JavaScript-Frameworks wie jQuery sowie auch bei jeder modernen Ereignisbehandlung haben anonyme Funktionen eine immense Bedeutung.

Beispiel: *anonyme_funktion.html*

In der ersten Anweisung wird die Funktionsdeklaration einer Variablen zugewiesen. Der Variablenname steht somit für den Funktionsnamen und kann wiederum in Ausdrücken verwendet werden. Im zweiten Beispiel wird der Funktion `tueEtwas()` als Parameter eine Funktion übergeben. Die übergebene Funktion wird in der `return`-Anweisung mit dem Wert 10 als Parameter aufgerufen.

Der Funktionsaufruf der Funktion `tueEtwas()` beinhaltet im ersten Fall die vollständige anonyme Funktionsdeklaration. Im zweiten Fall wird die Variable `quadrat` übergeben, die wiederum auf eine Funktionsdeklaration verweist.

```
var quadrat = function (zahl) { return zahl * zahl; }  
document.write(quadrat(10));  
function tueEtwas(f) {  
    return f(10);  
}  
document.write(tueEtwas(function(value) {  
    return value * value; })); // oder  
document.write(tueEtwas(quadrat));
```

Anonyme Funktionen werden sehr oft beim Eventhandling (der Reaktion auf Ereignisse) eingesetzt.

Innere Funktionen – Closures

JavaScript erlaubt auch den Aufbau **verschachtelter Funktionen**. Dabei wird eine Funktion innerhalb einer anderen Funktion deklariert. Für geschachtelte Funktionen gelten folgende Regeln:

- ✓ Die innere verschachtelte Funktion kann nur über die äußere aufgerufen werden und ist für direkte Zugriffe unzugänglich!
- ✓ Die innere Funktion kann die Variablen oder andere innere Funktionen der äußeren Funktion verwenden.
- ✓ Die äußere Funktion hat keinen Zugang zu den lokalen Variablen der inneren Funktion.
- ✓ Die innere Funktion ist für die äußere Funktion eine lokale Variable.

Zur Laufzeit erzeugt der JavaScript-Interpreter beim Aufruf der äußeren Funktion den Code der inneren Funktion. Dabei entsteht ein sogenanntes **Closure** (Einschluss) der inneren Funktion – das ist der Code der Funktion und eine Referenz auf alle Variablen, die von der inneren Funktion benötigt werden. Ein Closure kombiniert den Programmcode mit der lexikalischen Umgebung – d. h., das Closure „merkt“ sich die Umgebung, in der es erzeugt wurde.

```
function aussen() {  
    var a = 3;  
    document.write("<h3>Verdoppeln des Werts: ");  
    function innen1() {  
        innen2();  
        document.write(a);  
    }  
    function innen2() {  
        a *= 2;  
    }  
    innen1();  
    document.write("!</h3>");  
}  
ausen();
```

- ✓ In dem Skriptbereich wird eine Funktion `ausen()` deklariert, die direkt hinter der Deklaration aufgerufen wird. Damit wird eine dynamisch generierte Überschrift der Ordnung 3 erzeugt und ausgegeben. In der Funktion `ausen()` gibt es eine lokale Variable `a`, die mit dem Wert 3 initialisiert wird. Dazu finden Sie zwei innere Funktionen `innen1()` und `innen2()`.
- ✓ Die innere Funktion `innen2()` verdoppelt den Wert der Variable `a`. Es wird also auf eine lokale Variable `a` aus dem Geltungsbereich der äußeren Funktion zugegriffen.
- ✓ Auch in der inneren Funktion `innen1()` wird auf die lokale Variable `a` aus dem Geltungsbereich der äußeren Funktion zugegriffen. Zusätzlich wird aber davor die andere innere Funktion `innen2()` aufgerufen.
- ✓ Der Aufruf der inneren Funktion `innen1()` erfolgt im Geltungsbereich der äußeren Funktion, womit also auch indirekt `innen2()` aufgerufen wird.

Mit Closures kann man in JavaScript das Konzept der **Datenkapselung** unterstützen, das im Rahmen der objektorientierten Programmierung zu den Grundprinzipien zählt. Innere Funktionen werden auch oft als anonyme Funktionen erstellt.

Lambda-Ausdrücke

In vielen Programmiersprachen haben die letzte Zeit sogenannte Lambda-Ausdrücke Einzug gehalten, die in direktem Bezug zu Funktionen stehen, deren Deklaration aber oftmals vereinfachen. Ein Lambda-Ausdruck ist ebenso wie eine anonyme Funktion eine Art Funktion ohne einen eigenen Namen. Obwohl es im Detail in einigen Programmiersprachen Unterschiede zwischen Lambda-Ausdrücken und anonymen Funktionen gibt (nicht nur von der Syntax), sind beide Techniken oft gleich oder zumindest von der Wirkung weitgehend identisch.

Statt einen Funktionsnamen zu verwenden, wird ein Lambda-Ausdruck (wie eine anonyme Funktion) ohne Bezeichner notiert. Ein Lambda-Ausdruck kann ein oder mehrere Argumente (Parameter) akzeptieren und einen Rückgabewert liefern. Allerdings ist die Funktionalität eines Lambda-Ausdrucks in der Regel sehr einfach. Beispielsweise wird nur ein Ausdruck für eine Berechnung formuliert und das Ergebnis als Rückgabewert geliefert. Für komplexere Aufgaben eignen sich Lambda-Ausdrücke in der Regel nicht. Sie haben aber andere Stärken.

In vielen Programmiersprachen können Lambda-Ausdrücke verwendet werden, um Funktionen in Form von Funktionsreferenzen als Argumente an andere Funktionen oder Methoden zu übergeben oder um Funktionen innerhalb anderer Funktionen zu definieren. Lambda-Ausdrücke sind besonders nützlich in funktionalen Programmiersprachen, in denen Funktionen als „First-Class Citizens“ betrachtet werden. Das sind Objekte, die an andere Funktionen übergeben oder von ihnen zurückgegeben werden können.

- ✓ In JavaScript beginnen Sie einen Lambda-Ausdruck nicht mit einem spezifischen Schlüsselwort, wie in vielen anderen Sprachen. Stattdessen wird der **Pfeiloperator** `=>` verwendet.
- ✓ Es darf **kein** Zeilenumbruch zwischen Parametern und dem Pfeiloperator stehen.
- ✓ Lambda-Ausdrücke (Pfeilfunktionen) können entweder einen „knappen“ oder einen gewöhnlichen Funktionskörper oder Blockkörper haben.

Die nachfolgenden Beispielcodes einiger Lambda-Ausdrücke in JavaScript sollen das Konzept verdeutlichen.

Beispiel: *lambda1.js*

Die Lambda-Ausdrücke in dem Beispiel zeigen Deklarationen ohne Parameter. Da ein Lambda-Ausdruck eine Funktionsreferenz zurückgibt, kann man diese einer Variablen zuweisen. Der Aufruf erfolgt über den Namen der Variablen, gefolgt von einem Klammernpaar. Das ist dann wie ein gewöhnlicher Funktionsaufruf zu sehen.

```
var lambda1 = () => 4;
var lambda2 = () => {
  return 4;
}
var lambda3 = _ => 4;
```

```
console.log(lambda1()); // Aufruf der 1. Lambda-Funktion
console.log(lambda2()); // Aufruf der 2. Lambda-Funktion
console.log(lambda3()); // Aufruf der 3. Lambda-Funktion
```

Die erste Deklaration eines Lambda-Ausdrucks zeigt eine einfache Deklaration ohne Parameter mit Zuweisung zu einer Variablen und Rückgabe von einem Literal. Dies ist die Verwendung der Notation eines „knappen“ Funktionskörpers. In einem knappen Funktionskörper ist lediglich ein Ausdruck nötig, und eine implizite Rückgabe wird automatisch angehängt. Die Parameterliste für eine parameterlose Funktion muss mit einem Klammernpaar deutlich gemacht werden.

Die zweite Deklaration eines Lambda-Ausdrucks ohne Parameter mit Zuweisung zu einer Variablen und Rückgabe von einem Literal zeigt die explizite Verwendung von `return`. Der Körper eines Lambda-Ausdrucks kann in geschweifte Klammern gesetzt werden (gewöhnlicher Funktionskörper) und muss es auch, wenn dort nicht nur einfache Ausdrücke stehen (etwa explizites `return`). In einem Blockkörper muss dann aber auch eine explizite Rückgabe-Anweisung verwendet werden.

Variante 3 ist eine alternative Deklaration eines Lambda-Ausdrucks ohne Parameter mit `_` statt leerer Klammern für den Parameterausdruck.

Beispiel: *lambda2.js*

Die Lambda-Ausdrücke in dem Beispiel zeigen die Deklaration mit einem Parameter.

```
var lambda1 = x => x * x;
var lambda2 = (x) => x * x;

console.log(lambda1(11)); // Aufruf der 1. Lambda-Funktion
console.log(lambda2(11)); // Aufruf der 2. Lambda-Funktion
```

Die erste Deklaration eines Lambda-Ausdrucks zeigt die Verwendung eines nichtgeklammerten Übergabewerts und die Zuweisung zu einer Variablen sowie Rückgabe eines berechneten Werts (Multiplikation des übergebenen Werts mit sich selbst).

Die zweite Deklaration eines Lambda-Ausdrucks ist fast identisch, nur steht der Übergabewert in Klammern.

Beispiel: *lambda3.js*

Das dritte Beispiel zeigt die Verwendung mehrerer Parameter.

```
var lambda1 = (x, y) => x * y;
console.log(lambda1(11, 2)); // Aufruf der Lambda-Funktion
```

Die Deklaration eines Lambda-Ausdrucks mit mehreren Übergabewerten kann diese dann im Ausdruck verwenden. Wichtig: Bei mehreren Parametern **müssen** diese geklammert werden.

4.5 Lokale und globale Variablen

Die Variable `d` für die Angabe des Durchmessers aus den vorherigen Beispielen zur Flächenberechnung besitzt einen Gültigkeitsbereich (engl. **Scope**), der auf die Funktion beschränkt ist, d. h., außerhalb der Funktion kann nicht auf den Wert von `d` zugegriffen werden. Solche Variablen werden als lokale Variablen bezeichnet.

Globale Variablen, auf die jederzeit zugegriffen werden kann, sollten immer außerhalb von Funktionen deklariert werden. Jede Variable, die außerhalb von einer Funktion deklariert wird, ist global. In JavaScript ist aber auch jede Variable global, die in einer Funktion ohne das vorangestellte Schlüsselwort `var` oder `let` eingeführt wird. Diese Möglichkeit sollten Sie nicht nutzen, da damit die Ausführungsgeschwindigkeit eines Skripts enorm verschlechtert wird und andere negative Randwirkungen auftreten können. Deshalb sollten Sie in Funktionen immer nur lokale Variablen mit dem vorangestellten Schlüsselwort `var` oder `let` deklarieren.

Beispiel: *funktion_lokal_global.html*

In diesem Beispiel soll der Zugriff auf lokale und globale Variablen veranschaulicht werden. Dazu werden verschiedene einfache Berechnungen durchgeführt und deren Ergebnisse angezeigt.

```
<body>
  <h3>Lokale und globale Variablen</h3>
  <script type="text/javascript">
    ①   var x = 10;
        var y = 10;
        var z = 1;
    ②   function produkt(x, y) {
    ③     var ergebnis = x * y;
    ④     document.write("<br />Werte innerhalb der Funktion:
        x = " + x + "; y = " + y + "; z = " + z + "<br />");
        document.write("Ergebnis in der Funktion:
        x * y = " + ergebnis + "<br />");
        return ergebnis;
    }
    ⑤   document.write("Werte außerhalb der Funktion:
        x = " + x + "; y = " + y + "; z = " + z + "<br />");
    ⑥   var meinProdukt = x * y * produkt(3, 4);
    ⑦   document.write("<br />Ergebnis außerhalb der Funktion:
        x(global) * y(global) * (Ergebnis lokal)
        = " + meinProdukt);
  </script>
</body>
```

- ① Die Variablen `x`, `y` und `z` werden initialisiert. Da sie nicht innerhalb einer Funktion festgelegt werden, sind sie globale Variablen.

- ② Die Funktion `produkt()` wird deklariert, in der zwei Variablen miteinander multipliziert werden sollen. Die Übergabeparameter werden in der Funktion über die Variablen `x` und `y` angesprochen. Übergabeparameter sind ja immer in der Funktion als lokale Variablen zu sehen und diese beiden überschreiben damit in der Funktion die Werte der globalen Variablen.
- ③ Zusätzlich wird in der Funktion die Variable `ergebnis` mit dem vorangestellten `var` deklariert und mit dem Produkt von `x` und `y` belegt. Diese Variable kann außerhalb der Funktion nicht verwendet werden, da sie als lokale Variable nur in der Funktion gültig ist.
- ④ Zur besseren Darstellung werden die Werte der einzelnen Variablen ausgegeben. Innerhalb der Funktion haben `x` und `y` die Werte 3 und 4. Die globale Variable `z` hat den Wert 1, da sie in der Funktion nicht überschrieben wurde.
- ⑤ Zu Beginn werden auch außerhalb der Funktion die Werte für `x`, `y` und `z` ausgegeben.
- ⑥ Die Variable `meinProdukt` enthält die Berechnungen der globalen Variablen `x` und `y` sowie die lokalen Funktionsvariablen mit den Werten 3 und 4. Obwohl während der Ausführung der Funktion `produkt()` die lokalen Variablen `x` und `y` verwendet und verändert wurden, hatte dies keinen Einfluss auf den Wert 10 der globalen Variablen `x` und `y`. Das Ergebnis ist 1200 ($10 * 10 * (3 * 4)$).
- ⑦ Das Ergebnis dieser Berechnung wird über die Browserausgabe sichtbar gemacht.

Auswirkung von lokalen und globalen Variablen

Generell zeugt es von gutem, strukturiertem Programmierstil, wenn Funktionen völlig ohne globale Variablen auskommen. Solche Funktionen sind auch in hohem Maße wiederverwendbar, da ihre Verarbeitungen nur von den Parametern abhängen, die ihnen beim Aufruf explizit übergeben werden.

Blocklokal mit `let` versus `var`

An verschiedenen Stellen wurde bereits darauf hingewiesen, dass Variablen immer mit `var` oder `let` deklariert werden sollten. Aber die Unterschiede zwischen `let` und `var` wurden bisher nur angedeutet.

Mit `let` deklarierte Variablen sind blocklokal, mit `var` deklarierte Variablen innerhalb des gesamten Scopes gültig, wo sie deklariert wurden. Doch was bedeutet das?

Die nachfolgenden Beispiele sollen dies verdeutlichen und dabei ebenfalls noch einmal den Bezug zur globalen Gültigkeit beleuchten.

Beispiel: varlet1.js

```
function test() {  
    a++;  
    console.log(a);  
}  
let a = 1;  
test();  
test();  
test();
```

Die Deklaration der Variablen `a` im globalen Kontext macht diese auch innerhalb einer Funktion verfügbar. Der dreifache Aufruf der Funktion `test()` erhöht den Wert dieser globalen Variablen jeweils um den Wert 1.

Das ist die Ausgabe des Beispiels:

```
2  
3  
4
```

Bei dieser Verwendung gibt es keinen Unterschied, ob die Variable `a` außerhalb der Funktion mit `let` oder `var` deklariert wurde oder ganz ohne eines der beiden Schlüsselworte. Sie ist in jedem Fall global.

Beispiel: varlet2.js

```
let a = 1;  
function eins() {  
    let a = 1;  
    console.log(a);  
    a++;  
    console.log(a);  
}  
console.log(a);  
eins();  
console.log(a);
```

Die Ausgabe von diesem Code ist:

```
1  
1  
2  
1
```

Die Deklaration der Variablen `a` im globalen Kontext macht diese auch innerhalb einer Funktion verfügbar, aber dort wird sie mit `let` redefiniert. Das verdeckt die global deklarierte Variable durch eine lokale Variable mit gleichem Namen.

Zwar wird in der Funktion `eins()` der Wert von `a` dann um den Wert 1 erhöht. Aber das ist die lokale Variable. Nachdem die Funktion verlassen wurde, sieht man an dem Wert des global deklarierten `a`, dass der Wert immer noch 1 ist.

Bei dieser Verwendung gibt es keinen Unterschied, ob die Variable `a` sowohl global als auch lokal mit `let` oder `var` deklariert wurde. Auch kann man `let` oder `var` für die globale Variable weglassen und dann in der Funktion `a` mit `let` oder `var` redefinieren (siehe `varlet2_1.js` bis `varlet2_5.js`).

Wohl aber gibt es einen Unterschied, wenn Sie in der Funktion `var` und `let` weglassen. Vollkommen gleichgültig, ob Sie dann die globale Variable mit `let` oder `var` deklarieren oder ohne eines der beiden Schlüsselworte (siehe `varlet2_6.js` bis `varlet2_8.js`). In dem Fall ist die Ausgabe:

```
1
1
2
2
```

In der Funktion **verändern** Sie die **globale** Variable und arbeiten nicht mit einer lokalen Variablen.

Doch bisher zeigt sich noch kein Unterschied zwischen `let` und `var`. Das ändert das abschließende Beispiel.

Beispiel: `varlet3.js`

```
var a = 1;

function eins() {
  let a = 2;
  console.log(a); {
    let a;
    a++;
  }
  console.log(a);
}
console.log(a);
eins();
console.log(a);
```

Die Ausgabe von diesem Code ist:

```
1
2
2
1
```

Mit `let` können Sie in allen Konstellationen innerhalb eines Blocks (!) eine Deklaration vornehmen, die sowohl nur da gültig ist als auch eine eventuelle äußere Variable gleichen Namens temporär verdeckt. Der Block ist der Scope.

In dem Beispiel wird in der Funktion `eins()` `a` mit `let` als lokale Variable deklariert und mit dem Wert 2 initialisiert. Die globale Variable wird davon verdeckt, aber nicht geändert. Innerhalb der Funktion wird danach aber ein Block eingeführt und dort mit `let` erneut eine Deklaration der Variablen `a` vorgenommen, die nun in dem Scope gültig ist. Die Erhöhung in dem Block verändert also nicht die lokale Variable der umgebenden Funktion, die zwar ebenso in der Funktion, aber außerhalb von dem Block deklariert wurde.

Mit `var` geht das nicht, was Sie mit *varlet3_1.js* überprüfen können. Die erneute Deklaration innerhalb des Blocks verdeckt die lokale Variable nicht (was dann auch nicht wirklich eine neue Deklaration ist – trotz `var`) und deshalb wird diese auch mit `a++` erhöht.

Beachten Sie, dass Sie mit `var` schon blocklokale Variablen deklarieren können, wenn Sie innerhalb von Blöcken eine Deklaration vornehmen. Aber es darf außerhalb dieser Blöcke keine Deklaration einer Variable gleichen Namens vorhanden sein. Diese Einschränkung hat `let` nicht.

4.6 Vordefinierte Funktionen in JavaScript



Es gibt in JavaScript einige vordefinierte Funktionen, die Sie direkt in Ihren Skripten aufrufen können. Die folgenden Funktionen sind in JavaScript bereits vorhanden. Die Funktionsnamen sind reserviert und dürfen deswegen nicht für eigene Funktionsnamen verwendet werden.

<code>eval</code> (Zeichenkette)	Die übergebene Zeichenkette wird als JavaScript-Code ausgewertet und das Ergebnis der Ausführung zurückgegeben. Wenn Sie beispielsweise die Anweisung <code>document.write(eval("3 + 4 * 6"))</code> ; notieren, wird am Bildschirm das Ergebnis des Rechenausdrucks ausgegeben. Die Funktion kann auch zur Erzeugung eines Objekts genutzt werden, wenn dieses in einem String abgebildet wurde. Die Funktion gilt aber als Sicherheitsrisiko und ist auch von der Ausführungsgeschwindigkeit schlecht. Sie sollten den Einsatz vermeiden.
<code>encodeURIComponent</code> , <code>decodeURI</code> , <code>encodeURIComponent</code> <code>decodeURIComponent</code>	Diese Funktionen dienen der Ersetzung von Zeichen in URLs bzw. in Zeichenketten. Der Aufruf von <code>decodeURI("t\"est")</code> ergibt z. B. das Ergebnis <code>t"est</code> . Eine ausführliche Beschreibung dieser Funktionen finden Sie im Mozilla Developer Network unter https://developer.mozilla.org/en/JavaScript/Guide/Functions#escape_and_unescape_functions .
<code>escape</code> (Zeichenkette) <code>unescape</code> (Zeichenkette)	Über diese Funktionen können Sie Zeichenketten konvertieren, die Steuerzeichen (ASCII-Wert von 0 bis 31) enthalten. Diese Funktionen sollten nicht mehr genutzt werden. Greifen Sie stattdessen auf die Funktionen <code>encodeXXX</code> und <code>decodeXXX</code> zurück.
<code>isNaN(Wert)</code>	Mit der Funktion <code>isNaN()</code> (is Not a Number = ist keine Zahl) können Sie testen, ob ein übergebener Wert eine ungültige Zahl ist. Ist der Wert eine Zeichenkette oder ein boolescher Wert, wird <code>true</code> , ist er eine Zahl, wird <code>false</code> zurückgeliefert. Da Sie erfahrungsgemäß häufig testen müssen, ob eine Funktion numerisch ist oder nicht, ist diese Funktion besonders wichtig. Auf den Wert <code>NaN</code> selbst können Sie nicht mit einem Vergleich testen, obwohl dieser als Ergebnis eines mathematischen Ausdrucks geliefert wird, wenn dieser keine Zahl ergibt. Ein Vergleich <code>if (a == NaN)</code> ist nicht möglich, verwenden Sie deshalb die Funktion <code>isNaN()</code> .

<code>isFinite(Wert)</code>	Es wird getestet, ob der übergebene Wert innerhalb des Zahlenbereichs liegt, den JavaScript verarbeiten kann. Liegt der Wert außerhalb oder ist der Wert eine Zeichenkette, wird <code>false</code> zurückgeliefert, andernfalls <code>true</code> .
<code>Number(Objekt)</code> <code>String(Objekt)</code>	Über diese Funktionen können Sie ein Objekt in eine Zahl oder eine Zeichenkette konvertieren.
<code>parseFloat</code> <i>(Zeichenkette)</i>	Wandelt eine Zeichenkette in eine Dezimalzahl um und gibt diese als numerischen Wert zurück. Diese Funktion gibt NaN (Not a Number = keine Zahl) zurück, wenn die Zeichenkette nicht mit den Zeichen <code>-</code> , <code>+</code> , <code>.</code> oder einer Ziffer beginnt. Die Konvertierung der Zeichenkette in eine Zahl endet mit dem ersten ungültigen Zeichen oder dem Ende der Zeichenkette.
<code>parseInt</code> <i>(Zeichenkette)</i>	Wandelt eine Zeichenkette in eine ganze Zahl um und gibt diese als numerischen Wert zurück. Diese Funktion gibt NaN (Not a Number = keine Zahl) zurück, wenn die Zeichenkette nicht mit den Zeichen <code>-</code> , <code>+</code> oder einer Ziffer beginnt. Die Konvertierung der Zeichenkette in eine Zahl endet mit dem ersten ungültigen Zeichen oder dem Ende der Zeichenkette.

Beispiel: *parseFloat_parseInt.html*

Die Anwendung der beiden Funktionen `parseFloat()` und `parseInt()` soll demonstriert werden. Als Eingabewerte werden Zeichenketten verwendet, die korrekte Zahlenwerte und gemischte Werte besitzen.

	<pre> <body> <h3>parseFloat() und parseInt()</h3> <script type="text/javascript"> var a = "50"; var b = "5.6abc"; document.write('
a = "', a, '"; b = "', b, '"'); ① document.write('
c = a + b'); document.write('
c = ', a + b); ② document.write('
c = parseInt(a) + parseFloat(b)'); document.write('
c = ', parseInt(a) + parseFloat(b)); ③ document.write('
c = parseFloat(a) + parseInt(b)'); document.write('
c = ', parseFloat(a) + parseInt(b)); </script> </body> </pre>
--	--

- ① Nach der Initialisierung und der Ausgabe der Variablenwerte werden die beiden Zeichenketten `a` und `b` zunächst aneinandergefügt.

- ② Erst durch die beiden Funktionen `parseInt()` und `parseFloat()` können die Zeichenketten in Zahlen umgewandelt und arithmetisch addiert werden. Die Variable `a` wird in die Ganzzahl 50 und die Variable `b` durch `parseFloat()` in die Dezimalzahl 5.6 umgewandelt. Der Zeichenkettenteil `abc` wird abgeschnitten. Dadurch ergibt sich der mathematische Wert $50 + 5.6 = 55.6$.
- ③ Die Funktionen werden mit vertauschten Werten aufgerufen. Zuerst wird `a` als Dezimalzahl interpretiert und `b` als Ganzzahl. Die Konvertierung für die Variable `b` wird nach dem Punkt abgeschnitten. Dadurch ergibt sich der mathematische Wert $50 + 5 = 55$.

parseFloat() und parseInt()

```
a = "50"; b = "5.6abc";
c = a + b
c = 505.6abc

c = parseInt(a) + parseFloat(b)
c = 55.6

c = parseFloat(a) + parseInt(b)
c = 55
```

Zeichenkette in Zahlen umwandeln

In HTML gibt es keine Eingabefelder, die speziell für Zahlen genutzt werden können. Die Eingaben werden immer als Zeichenkette betrachtet. Möchten Sie mit den Eingaben rechnen, können Sie mit `isNaN()` z. B. überprüfen, ob die Eingabe als Zahl interpretiert werden kann. Danach haben Sie die Möglichkeit, die Zeichenkette über `parseFloat()` oder `parseInt()` in eine Zahl des entsprechenden Typs umzuwandeln.

4.7 Debuggen von Funktionen

In diesem Abschnitt erfahren Sie, wie Sie Fehler in JavaScript-Code finden und beseitigen. Die Art und Weise dieser Fehlersuche wird Debugging (deutsch: Entwanzen) genannt.

Beispiel: *debug_variable.html*

Um das Debuggen von Funktionen zu testen, erstellen Sie das folgende JavaScript-Beispiel:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Debugging</title>
</head>
<body>
  <script type="text/javascript">
    var +variable = true;
  </script>
</body>
</html>
```

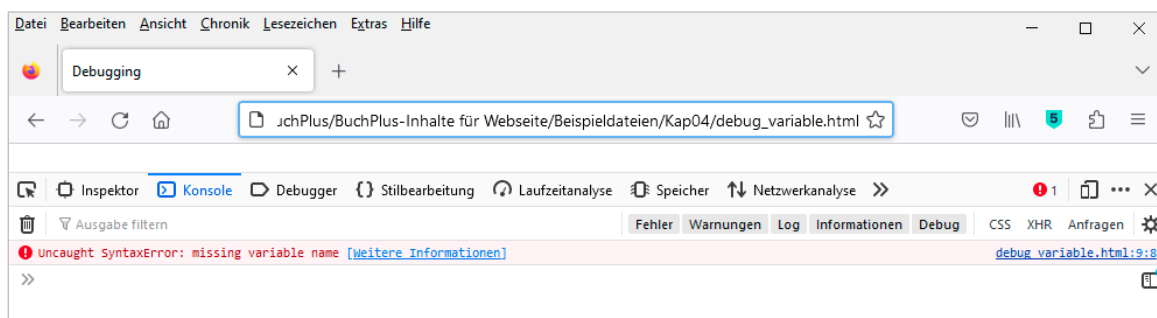
In diesem Beispiel wird der Name der Variablen **absichtlich** falsch geschrieben. Ihr wird das Sonderzeichen `+` vorangesetzt, das nicht in Variablennamen genutzt werden darf.

Entwicklertools, Fehlerkonsole & Co

Für das Debuggen (die Suche nach Fehlern) stellen alle modernen Browser eine Fehlerkonsole beziehungsweise Web-Konsole oder gar vollständige Entwicklertools mit ganz mächtigen Möglichkeiten zur Verfügung, mit denen Sie unter anderem auch Fehler in einem JavaScript sehen können. Wie Sie konkret an diese Tools eines Browsers gelangen, ist jedoch von dem Browser wie auch seiner genauen Version abhängig. Die Zugangswege der Browser über ihre verschiedenen Versionen haben sich immer wieder geändert, und Sie sind ggf. auf die Hilfe im Browser oder aktuelle Quellen im Internet angewiesen, um den Zugangsweg zu finden, zumal auch die Möglichkeiten moderner Browser zur Analyse von Webseiten mittlerweile weit darüber hinausgehen, im Vergleich zu den einfachen Fehlerkonsolen von vor wenigen Jahren.

Bei Firefox werden die neuen Möglichkeiten mittlerweile nicht mehr unter dem alten Begriff „Fehlerkonsole“, sondern „Web-Konsole“ geführt (was sich in kommenden Versionen aber möglicherweise wieder ändern kann). Exemplarisch soll der Weg zur Web-Konsole hier mit dem Firefox-Browser in der Version 73 gezeigt werden.

- ▶ Wenn das Menü des Browsers nicht sichtbar ist, blenden Sie es ein (z. B. mit dem Kontextmenü am oberen Rand des Browsers).
- ▶ Wählen Sie das Menü *Extras - Browser-Werkzeuge - Werkzeuge für Web-Entwickler*. Es sollte sich am unteren Rand des Browserfensters ein neuer Bereich öffnen, in dem die unterschiedlichsten Features zur Analyse einer Webseite bereitgestellt werden. Unter *Konsole* sollten die Meldungen des JavaScript-Interpreters angezeigt werden.



Geöffnete Web-Konsole im Firefox mit einer Fehlerbeschreibung

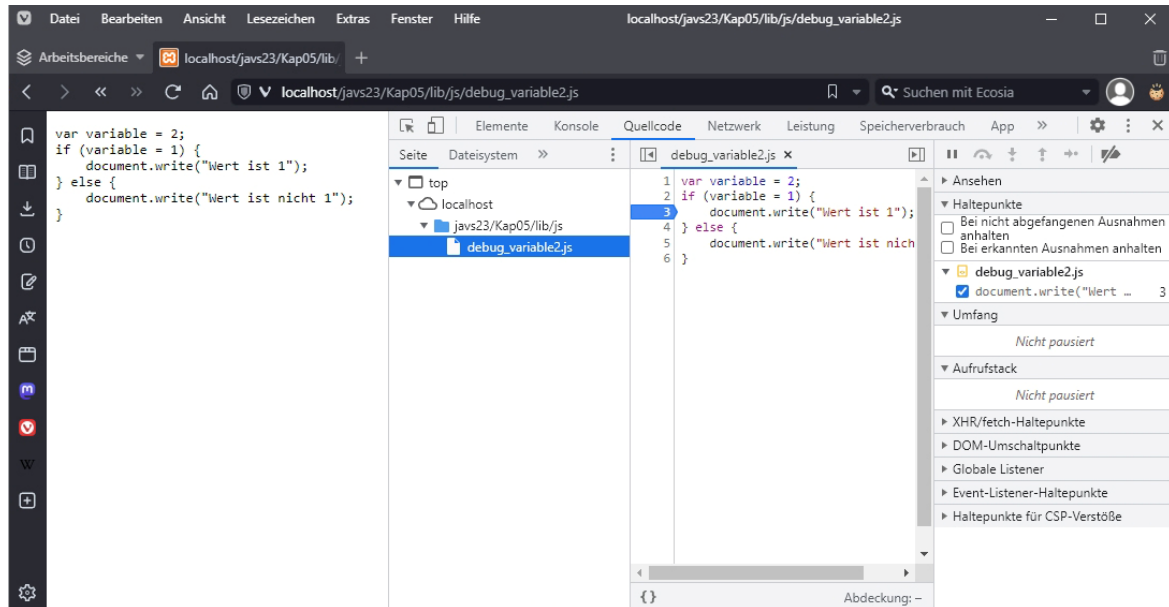
- ▶ Laden Sie die Datei *debug_variable.html* in den Browser.
- ▶ Wechseln Sie in die Web-Konsole, um den Fehler anzuzeigen.

Im Fenster wird die Meldung angezeigt, dass der Variablenname nicht angegeben wurde (missing variable name). Außerdem wird der Programmcode mit der entsprechenden Zeilennummer angezeigt. Zur besseren Lokalisierung des Fehlers wird die Stelle noch mit einem Pfeil markiert.

- ! In fast allen modernen Browsern kommen Sie mit der Taste **(F12)** an die Entwicklertools. Diese stellen im Wesentlichen die gleichen Möglichkeiten wie die Web-Konsole bei Firefox bereit. Für die meisten Browser gibt es auch Erweiterungen (Add-ons) und bei diesen weitere Tools zur Unterstützung der Web-Entwicklung im Allgemeinen und der Fehlersuche.

Debugger

Alle modernen Browser beinhalten in den Entwicklertools mittlerweile einen JavaScript-Debugger, der über die Entwicklertools bzw. die Web-Konsole zugänglich ist.



Moderne Browser haben alle einen integrierten Debugger – hier in Vivaldi.

Debugger in modernen Browsern bieten eine Vielzahl an Möglichkeiten zum professionellen Debuggen von JavaScript-Programmen. Alle Debugger funktionieren ähnlich. Sie können z. B. ...


- ✓ ein Skript schrittweise verfolgen (sogenanntes Steppen),
- ✓ es gezielt anhalten (mit Haltepunkten beziehungsweise Breakpoints) und
- ✓ einzelne Variablen und den Zustand des Programms oder Skripts auswerten.

In leistungsfähigen Tools können Sie auch, während das Skript oder Programm unterbrochen ist, einen Wert in einem Ausdruck manuell ändern. Innerhalb eines Debuggers können Sie in der Regel mithilfe von verschiedenen Schaltflächen bzw. Menübefehlen den Ablauf des Skripts verfolgen und kontrollieren. Als Werkzeuge bietet das Programm u. a. eine Übersicht der aktuell geöffneten Dokumente und eine Möglichkeit, Befehle einzugeben.

Die integrierten Debugger lassen sich allerdings meist nur in Verbindung mit externen Skripten sinnvoll nutzen. Das ist jedoch keine Einschränkung, weil man in der Praxis so gut wie ausschließlich externe Skripte verwendet und wir nur aus didaktischen Gründen in dem Buch die Skriptbereiche direkt in die Webseite notieren. Ebenso ist ein Debugger meist dafür gedacht, **logische** Probleme in einem Programm zu entdecken – weniger echte Syntaxprobleme. Diese führen in JavaScript ja zu einem Programmabbruch und die Stelle, wo sich der Fehler auswirkt, wird bereits in der Web-Konsole angezeigt und beschrieben.

Um einen Debugger praktisch auszuprobieren, können Sie so vorgehen:

- ▶ Laden Sie die Datei *debug_variable2.html* in den Browser. Diese verwendet eine externe JavaScript-Datei mit einem logischen Fehler. Es gibt in der Bedingung einer *if*-Anweisung eine Zuweisung statt eines Vergleichs und die Ausgabe ist damit immer *Wert ist 1*.
- ▶ Wechseln Sie in die Web-Konsole und dort zum Register *Debugger*.

- ▶ Im Bereich *Quellen* können Sie die externe JavaScript-Datei (in diesem Fall die Datei *debug_variable2.js* im Unterverzeichnis *lib/js*) auswählen, die Sie untersuchen wollen.
- ▶ Klicken Sie bei der damit ausgewählten Datei und deren rechts ausgezeigten Quellcode auf den linken Rand des Anzeigebereichs und dort die Zeilennummer 3. Die Stelle wird markiert. Damit wird ein Haltepunkt gesetzt.
- ▶ Laden Sie die Webseite mit **F5** oder  neu in den Browser. Die Programmausführung stoppt vor der Ausführung der Zeile 3. Sie können nun die Situation untersuchen und alle Schritte durchführen, die oben beschrieben wurden – also das Skript schrittweise verfolgen oder einzelne Variablen und den Zustand des Programms oder Skripts auswerten.

Typische Fehler

Folgende Fehlersituationen treten in der Praxis oft auf:

Zuweisung oder Vergleich?

Das einfache Gleichheitszeichen ist der Zuweisungsoperator. Um zwei Werte zu vergleichen, muss jedoch das Gleichheitszeichen zweimal angegeben werden. Das wird oft verwechselt, produziert aber keine Fehlermeldung, da es syntaktisch korrekt ist. Es ist nur logisch falsch und deshalb tritt dieser Fehler recht häufig auf.

```
var test = 0;
if (test = 1)
    document.write('Test war in Ordnung!');
```

Die fehlerhafte Verwendung des Zuweisungsoperators erzeugt keine Fehlermeldung. Der Variablen *test* wird der Wert 1 zugewiesen, d. h., es wird kein Vergleich, sondern eine Zuweisung durchgeführt. Die *if*-Anweisung wertet alle Werte ungleich 0 zu *true* aus, sodass die Anweisung ausgeführt wird. Richtig muss es heißen:

```
var test = 0;
if (test == 1)
    document.write('Test war in Ordnung!');
```

Groß oder klein?

JavaScript ist case-sensitive, d. h., die Groß- und Kleinschreibung wird in allen Variablennamen und Bezeichnern unterschieden. Bei größeren Skripten mit vielen Funktionen kann es passieren, dass z. B. ein Funktionsaufruf falsch geschrieben wird.

```
function AusgabeImBrowser() {
    // JavaScript-Anweisungen
}
document.write(ausgabeImBrowser());
```

Die Browser werden diesen Aufruf als Fehler ausgeben, da die aufgerufene Funktion nicht deklariert ist. Die Funktion wird in der Definition mit einem großen A geschrieben und hat somit einen anderen Bezeichner.

4.8 Übungen

Übung 1: Funktion zum Addieren zweier übergebener Werte

Übungsdatei: --

Ergebnisdatei: *kap04/uebung1.html*

1. Schreiben Sie eine JavaScript-Funktion `summe()`, die zwei eingegebene Werte addiert, so dass `document.write(summe(4, 5))`; die Ausgabe 9 ergibt. Die Funktionsdefinition soll im Kopf, der Aufruf im Rumpf des HTML-Dokuments erfolgen.

Übung 2: Zeichenketten in Form eines Absatzes ausgeben

Übungsdatei: --

Ergebnisdatei: *kap04/uebung2.html*

1. Entwickeln Sie eine Funktion `absatzAusgabe()`, die eine beliebige Anzahl von Zeichenketten als Parameter entgegennimmt und diese untereinander am Bildschirm ausgibt.

Übung 3: Arbeiten mit lokalen und globalen Variablen

Übungsdatei: --

Ergebnisdatei: *kap04/uebung3.html*

1. Eine Funktion soll die Werte der beiden globalen Variablen `BenutzerAnzahl` und `Versionsnummer` sowie der beiden lokalen Variablen `Kundennummer` und `Kundenname` ausgeben. Die Variablen sind von Ihnen entsprechend zu deklarieren.

5

Objekte

5.1 Grundlagen von Objekten

Das Konzept des objektorientierten Programmierens hat sich wegen seiner Vorteile schon lange durchgesetzt. Es bildet reale und abstrakte Objekte auf Codestrukturen ab, die leichter verständlich sind und zusammengehörende Daten und Funktionen kapseln. Objekte bieten dem Programmierer wiederverwendbaren Code und erlauben eine bessere Lokalisierung von Fehlern. Objekt-orientiertes Programmieren erfordert jedoch ein Verständnis der zugrunde liegenden Theorie.

Objekte sind in JavaScript Listen für logisch zusammengehörende Variablen und Funktionen, die als Eigenschaften und Methoden des Objekts bezeichnet werden. Die Eigenschaften sind eine Ansammlung von Werten, die das Objekt näher beschreiben. Methoden sind objekteigene Funktionen, welche die Eigenschaften oder andere Werte ändern und die Aktivität eines Objekts festlegen. Objekte haben insbesondere eine Identität: Sie sind nur sich selbst gleich.

In JavaScript gibt es eine gewisse Anzahl an direkt verfügbaren Objekten, aber Sie können auch selbst Objekte erzeugen. In JavaScript sind die folgenden Schritte zur Verwendung von Objekten notwendig:

- ✓ Erstellen der Objekte durch einen Konstruktor oder einen Objektinitialisierer,
- ✓ Hinzufügen von Eigenschaften,
- ✓ Hinzufügen von Methoden,
- ✓ Erzeugen von Objekten,
- ✓ Verwenden der Objekte.

Klassen, Prototypen und Erstellen der Objektdекlaration über einen Konstruktor

Ein Konstruktor (gelegentlich auch Konstruktor-Funktion genannt) enthält im Anweisungsblock die Deklaration der Eigenschaften und Methoden eines Objekts. Mithilfe des Konstruktors können dann Objekte erzeugt werden. Man kann sagen, dass alle Objekte, die mit demselben Konstruktor erzeugt wurden, einer Klasse von Objekten angehören. In vielen Sprachen ergibt sich der Bezeichner der Klasse aus dem Namen der Konstruktor-Funktion, wobei man in JavaScript hier etwas vorsichtig sein muss.

Denn JavaScript verwendete historisch statt Klassen nur sogenannte **Prototypen** – zumindest in älteren Versionen. Während eine Klasse eine Art abstrakte Bauvorschrift für neue Objekte gleichen Typs darstellt, stellt ein Prototyp bereits ein konkretes Objekt dar, nach dessen „Ebenbild“ neue Objekte zu erschaffen sind.

Ab ECMAScript 2015 (ES6) kann man in JavaScript mit dem Schlüsselwort `class` Klassen schreiben und alle modernen Browser unterstützen das, einschließlich Chrome, Firefox, Safari, Edge und Internet Explorer 11, Node.js ebenso. Das bedeutet, mittlerweile können Sie in JavaScript sowohl mit Prototypen als auch weitgehend zuverlässig mit Klassen arbeiten.

Wenn man sich dieser Details bewusst ist, kann man auch in JavaScript generell von Klassen sprechen, obgleich die Syntax als auch Konzepte zum Erzeugen eines Objekts sich, wie gesagt, unterscheiden. Die Anwendung des daraus erzeugten Objekts selbst ist hingegen unabhängig davon, wie es erzeugt wurde – aus einem Prototyp oder einer Klasse. Deshalb spricht man oft auch kurz und verallgemeinert vom **Typ** eines Objekts.

Beachten Sie, dass Klassen bzw. Prototypen (und damit Konstruktoren) nach Konvention **groß** und in **Einzahlform** geschrieben werden sollen, während enthaltene Methoden und Eigenschaften (verallgemeinert **Member** genannt) mit einem Kleinbuchstaben beginnen. Member sind nur spezielle Varianten von Variablen und Funktionen.

```
function Konstruktorname(Wert1, Wert2, ...) {  
    this.eigenschaft1 = Wert1;  
    this.eigenschaft2 = Wert2;  
}
```

So könnte ein konkreter Prototyp aussehen, mit dem eine Person mit typischen Eigenschaften beschrieben wird:

```
function Person(name, alter, geschlecht) {  
    this.name = name;  
    this.alter = alter;  
    this.geschlecht = geschlecht;  
}
```

Die Klasse mit identischer Beschreibung sieht dann so aus, wobei der Konstruktor in einer Klasse **intern** mit einer speziellen Methode mit Namen `constructor()` definiert wird:

```
class Person {  
    constructor(name, alter, geschlecht) {  
        this.name = name;  
        this.alter = alter;  
        this.geschlecht = geschlecht;  
    }  
}
```

Ein Objekt kann in jedem Fall mit `new` und dem Aufruf des Konstruktors (genau genommen des Bezeichners des Objekttyps – hier wird auch bei einer Klasse deren Name und **nicht** der interne Bezeichner `constructor()` verwendet) erzeugt werden.

Die dabei eventuell übergebenen Parameter werden von dem Konstruktor zur Initialisierung der Eigenschaften verwendet, wenn der Programmierer das so vorgesehen hat.

```
var objekt = new Typname(Wert1, Wert2, ...);
```

Der Konstruktor liefert eine Referenz auf das erzeugte Objekt. Die Variable mit der Objektreferenz, die nach Konvention ebenfalls klein zu schreiben ist (es handelt sich ja hier um eine Variable), ist also ein Zeiger auf den Speicherbereich des Objekts. Zusätzlich können im Konstruktor Methoden zur Bearbeitung des Objekts deklariert werden.

Um es noch einmal zu betonen – diese Instanziierung ist vollkommen unabhängig davon, ob Sie einen Prototyp oder eine Klasse verwenden. Dementsprechend führen die beiden folgenden JavaScripts zu dem gleichen Resultat:

Person { name: 'Max Mustermann', alter: 25, geschlecht: 'männlich' }

Person { name: 'Susi Sorglos', alter: 32, geschlecht: 'weiblich' }

Person { name: 'Susi Sorglos', alter: 32, geschlecht: 'weiblich' }

Beispiel: *klasse_person.js*

```
"use strict";
class Person {
  constructor(name, alter, geschlecht) {
    this.name = name;
    this.alter = alter;
    this.geschlecht = geschlecht;
  }
}
let person1 = new Person("Max Mustermann", 25, "männlich");
let person2 = new Person("Susi Sorglos", 32, "weiblich");
let person3 = new Person("Hans Hansen", 45, "männlich");
console.log(person1);
console.log(person2);
console.log(person2);
```

Beispiel: *prototype_person.js*

```
"use strict";
function Person(name, alter, geschlecht) {
  this.name = name;
  this.alter = alter;
  this.geschlecht = geschlecht;
}
let person1 = new Person("Max Mustermann", 25, "männlich");
let person2 = new Person("Susi Sorglos", 32, "weiblich");
let person3 = new Person("Hans Hansen", 45, "männlich");
console.log(person1);
console.log(person2);
console.log(person2);
```

Im Kontext der objektorientierten Programmierung werden die Bezeichnungen Objekt, Objektinstanz und Instanz oft synonym verwendet. Zusätzlich kann Objekt sich auf einen bestimmten Typ von gleichartigen Objekten mit den gleichen Eigenschaften und Methoden beziehen.

Erstellen der Objektdeklaration über einen Objektinitialisierer (JSON)

Während über einen Konstruktor bei Bedarf mehrere Objekte der gleichen Klasse erzeugt werden können, können Sie mit einem Objektinitialisierer genau **ein** Objekt erzeugen. Dazu wird einer Variablen eine Liste von Eigenschaft-Wert-Paaren, die durch einen Doppelpunkt getrennt sind, in geschweiften Klammern zugewiesen. Mehrere Paare werden durch Kommata getrennt. Sie können mit einem Objektinitialisierer auch Methoden mit entsprechenden Funktionen definieren. Dies erfolgt meist mit anonymen Funktionen.

Die Notation zur Erzeugung von Objekten mit einem Objektinitialisierer wird JavaScript Object Notation oder kurz **JSON** genannt. Über die letzten Jahre hat JSON eine Bedeutung gewonnen, die weit über JavaScript hinausgeht. JSON hat sich als das Standardformat zum Austausch von strukturierten Informationen im Internet schlechthin etabliert und dabei XML weitgehend verdrängt. JSON ist einfach, schlank und eindeutig und damit perfekt für so einen Datenaustausch geeignet. Aus nahezu allen Programmiersprachen wie auch Programmen mit der Verarbeitung von strukturierten Informationen können Sie JSON-Formate exportieren und importieren.

Beispiel

Der Variablen `objVar` werden die Eigenschaften `eigenschaft1`, `eigenschaft2` usw. zugewiesen, die bereits mit einem Wert vorbelegt sind. Auch Methoden werden als Eigenschaften verankert – als Zeiger bzw. Funktionsreferenzen.

```
var objVar = { eigenschaft1:Wert1,
               eigenschaft2:Wert2,
               methode1: function() {}
             };
document.write(objVar.eigenschaft1);
document.write(objVar.methode1);
```

So könnte die Deklaration einer Person mittels JSON aussehen (Beispiel: *json_person.js*):

```
"use strict";
let person1 = {
  "name": "Max Mustermann",
  "alter": 25,
  "geschlecht": "männlich",
  auskunft: function() {
    console.log("Ich heiße " + this.name + " und bin " +
      this.alter + " Jahre alt.")
  }
}
console.log(person1);
person1.auskunft();
```

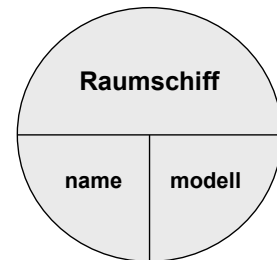
Die Ausgabe dieses Beispiels ist:

```
{  
  name: 'Max Mustermann',  
  alter: 25,  
  geschlecht: 'männlich',  
  auskunft: [Function: auskunft]  
}  
Ich heiße Max Mustermann und bin 25 Jahre alt.
```

Gehen wir nun die Details zu Eigenschaften und Methoden genauer an.

5.2 Eigenschaften

Während mehrere Objekte die gleichen Eigenschaften haben können, können die **Werte** dieser Eigenschaften in jedem Objekt verschieden sein. Jedes Objekt ist somit ein Vertreter seiner Klasse. So wäre ein Raumschiff mit dem Namen U.S.S. Enterprise ein Vertreter der Klasse `Raumschiff`, wenn so ein Beispiel die Basis sein soll.



Es soll nun ein einfacher Konstruktor für `Raumschiff` erstellt werden. Begonnen wird mit der Deklaration eines Konstruktors, womit die zwei Eigenschaften `name` und `modell` für alle Objekte dieses Typs festgelegt werden.

```
function Raumschiff(name, modell) {  
  this.name = name;  
  this.modell = modell;  
}
```

Über das Schlüsselwort `this` beziehen Sie sich innerhalb des Konstruktors immer auf das gerade erzeugte Objekt. In der Deklaration selbst ist es ein Stellvertreterbegriff dafür, da man in der Deklaration noch nicht den Namen für das erst später daraus erzeugte Objekt kennen kann. Mit dem vorangestellten `this` gibt man sogenannte **Instanzvariablen** an.

Die Eigenschaft `name` des Objekts wird im Beispiel als Instanzvariable auf den Wert des ersten übergebenen Parameters (`name`) gesetzt, die Eigenschaft `modell` auf den Wert des zweiten Parameters (`modell`).

Namensgleichheit von Parameter und Instanzvariable

Beachten Sie im Konstruktor die identischen Bezeichner für die Parameter und die Instanzvariablen. Das ist kein Zufall, Schludrigkeit oder gar ein Fehler, sondern ganz im Gegenteil gängige Praxis, von der man in der professionellen Programmierung niemals abweicht. Eine solche Identität der Bezeichner hat sich sogar so fest etabliert, dass viele Entwicklungstools für Sprachen wie Java, C# etc. bereits Befehle implementiert haben, die genauso eine Wahl der Bezeichner automatisch generieren. Insbesondere beim Konstruktor, aber auch sogenannten Gettern und Settern, die Werte von Instanzvariablen abfragen oder setzen.

Eine Verwechslungsgefahr besteht unter keinen Umständen. Mit dem vorangestellten `this` ist eindeutig die Instanzvariable gemeint und ohne das `this` der Parameter. Der entscheidende Vorteil ist, dass die notwendige Zuordnung damit eindeutig ist und es entsteht keine Unklarheit, welcher Parameterwert für welche Instanzvariablen gedacht ist. Ohne diese strenge Wahl der Bezeichner, wäre die Zuordnung dagegen erklärungsbedürftig.

Ansonsten gelten für Member über die technischen Zwänge hinaus die üblichen Konventionen der Groß- und Kleinschreibung sowie Camelnotation.

Mit der Anweisung `new` können Sie ab sofort über den Konstruktor `Raumschiff` Objekte, die Raumschiffe repräsentieren, erzeugen.

```
var enterprise = new Raumschiff("U.S.S. Enterprise", "NCC-1701");
```

Eine Variable mit Namen `enterprise` wird mit dieser Syntax deklariert. Das vorgestellte `new` erzeugt mit dem Konstruktor `Raumschiff` ein Objekt und übergibt diesem die zwei Werte `U.S.S. Enterprise` und `NCC-1701` als Parameter. Der deklarierte Konstruktor erzeugt daraufhin das Objekt im Speicher und belegt dessen Eigenschaften mit den übergebenen Werten. Er liefert auch eine Referenz auf die Speicherstelle zurück, die in der Variablen `enterprise` abgespeichert wird. Von nun an existiert ein `Raumschiff`-Objekt im Speicher und kann als Objekt über den Variablennamen `enterprise` angesprochen werden:

```
console.log(typeof(enterprise)); // liefert den Typ object
```

Um auf die Eigenschaften dieses Objekts zugreifen zu können, wird der Name der Variablen verwendet, der das Objekt zugewiesen wurde, gefolgt von einem Punkt und dem Namen der Eigenschaft. Alternativ können Sie in eckigen Klammern den Namen der Eigenschaft angeben, der in doppelten Anführungszeichen eingeschlossen ist.

```
Objekt.Eigenschaft;  
Objekt["Eigenschaft"];
```

In JavaScript wird die allgemein übliche Punktnotation (DOT-Notation) für den Zugriff auf Objekteigenschaften und -methoden genutzt. Sie wurde bereits von Ihnen in einigen Anwendungen verwendet. In der Anweisung `document.write()` ist z. B. `write()` die Methode des Objekts `document`, die Ausgaben in ein HTML-Dokument durchführen kann.

Beispiel: raumschiff.html

Es werden nun in dem Beispiel die beiden „Raumschiffe“ (Objekte) `enterprise` und `voyager` erzeugt. Danach wird auf deren Eigenschaften zugegriffen. Beachten Sie, dass wir mit Prototypen arbeiten (das ist besser abwärtskompatibel), aber mit Klassen geht das natürlich auch.

```

<body>
  <h3>Anlegen zweier Objekte des Konstruktors Raumschiff</h3>
  <script type="text/javascript">
    ①    function Raumschiff(name, modell)      {
          this.name = name;
          this.modell = modell;
        }
    ②    var enterprise = new Raumschiff("U.S.S. Enterprise",
                                         "NCC-1701");
          var voyager    = new Raumschiff("U.S.S. Voyager",
                                         "NCC-74656");
    ③    document.write("<p>", enterprise.name, ", Modell: ",
                      enterprise.modell);
    ④    document.write("<p>", voyager.name, ", Modell: ",
                      voyager.modell);

  </script>
</body>

```

- ① Im JavaScript-Bereich des HTML-Dokuments wird der Konstruktor `Raumschiff()` deklariert, welcher die beiden Eigenschaften `Name` und `Modell` besitzt.

Anlegen zweier Instanzen des Objekts `Raumschiff`

U.S.S. Enterprise, Modell: NCC-1701

U.S.S. Voyager, Modell: NCC-74656

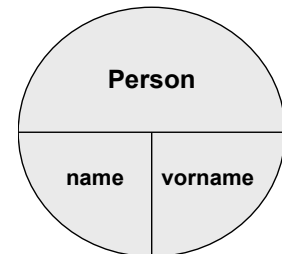
Eigenschaften der erzeugten Objekte ausgeben

- ② Über das Schlüsselwort `new` wird ein neues Objekt mit dem Konstruktor `Raumschiff` erzeugt. Die Angaben in Klammern beinhalten die Werte für die Objekteigenschaften. So wird zuerst das „Raumschiff“ (bedenken Sie, dass das im Grunde einfach ein beliebiges Objekt bzw. eine Objektreferenz ist) `enterprise` und dann das „Raumschiff“ `voyager` erzeugt.
- ③ Über die Variable `enterprise` werden der Name und das Modell des „Raumschiffs“ `enterprise` ausgegeben.
- ④ Ebenso werden die Eigenschaften des „Raumschiffs“ `voyager` angesprochen.

Die Eigenschaften (auch Felder genannt) beschreiben die Merkmale der Objekte. Die Eigenschaftswerte (auch Attribute genannt) eines Objekts können bei Objekten desselben Typs unterschiedlich sein. Der erste Schritt, um Eigenschaften zu definieren, besteht darin, im Anweisungsblock des Konstruktors die Eigenschaften zu deklarieren und zu initialisieren. Für jede Eigenschaft wird dazu eine eigene Variable angegeben. Die Anweisung zur Deklaration der Eigenschaft beginnt mit dem Schlüsselwort `this`, gefolgt von einem Punkt und dem Namen der Eigenschaft. Der Eigenschaft wird danach ein Wert zugewiesen, der z. B. als Parameter übergeben werden kann.

Für Eigenschaften gelten die gleichen Regeln wie für Variablen. Sie können alle einfachen Datentypen beinhalten, aber auch Objekte und Funktionen. Im Gegensatz zu Variablen erfolgt der Zugriff über die Eigenschaften und Methoden des Objekts. Eine Variable speichert lediglich eine Referenz auf den Speicherplatz des Objekts. Die Variable verweist dabei auf das entsprechende Objekt, der Eigenschaftsname auf die gewünschte Eigenschaft.

Im übernächsten Beispiel wird dem Konstruktor `Raumschiff` die weitere Eigenschaft `kapitaen` hinzugefügt. Der Unterschied zu den bisher verwendeten Eigenschaften liegt darin, dass diese Eigenschaft ebenfalls ein Objekt ist. Um einen „Kapitän“ erzeugen zu können, wird daher zuerst ein weiterer Konstruktor dafür benötigt.



```
function Person(name, vorname) {
  this.name = name;
  this.vorname = vorname;
}
```

Über den Konstruktor `Person` können Sie ein Objekt erzeugen, das die Eigenschaften `name` und `vorname` besitzt.

Beispiel: *raumschiff_person.html*

Nun können Sie die Besatzungsmitglieder der Raumschiffe inklusive des Kapitäns erzeugen und die entsprechenden Eigenschaften zuweisen.

```
<body>
  <h3>Die Personen der Raumschiffe</h3>
  <script type="text/javascript">
    ① function Raumschiff(name, modell)      {
        this.name = name;
        this.modell = modell;
      }
    ② function Person(name, vorname)      {
        this.name = name;
        this.vorname = vorname;
      }}
    ③ var enterprise = new Raumschiff("U.S.S. Enterprise",
                                     "NCC-1701");
    ④ var k1 = new Person("Kirk", "James Tiberius");
        document.write("<p>",enterprise.Name, " ",
                       enterprise.Modell);
    ⑤ document.write("<br />Person: ",k1.vorname, " ", k1.name);
```

```

⑥ var voyager = new Raumschiff("U.S.S. Voyager",
                                "NCC-74656");

var k2 = new Person("Janeway", "Kathryn");
document.write("<hr />",voyager.name, " ",
               voyager.modell);

document.write("<br />Person: ",k2.vorname, " ", k2.name);
</script>
</body>

```

- ① Der Konstruktor `Raumschiff` zum Erstellen eines Raumschiff-Objekts wird deklariert.
- ② Zum Erstellen der einzelnen Personen wird der Konstruktor `Person` eingefügt.
- ③ Der Variablen `enterprise` wird mit dem Konstruktor `Raumschiff` ein Objekt zugewiesen.
- ④ Das Objekt in der Variablen `k1` soll über die Eigenschaften `name` und `vorname` den entsprechenden Namen ausgeben und wird deshalb als Objekt des Konstruktors `Person` erstellt. Die Daten werden als Parameter übergeben und über den Konstruktor `Person` den einzelnen Eigenschaften zugewiesen.
- ⑤ Nach der Ausgabe des Namens des Raumschiffs wird der Name der Person `k1` ausgegeben.
- ⑥ Auf die gleiche Art und Weise werden die Daten der Person des zweiten Raumschiffs ausgegeben.

Die Personen der Raumschiffe

U.S.S. Enterprise NCC-1701
 Person: James Tiberius Kirk

U.S.S. Voyager NCC-74656
 Person: Kathryn Janeway

Personen werden hinzugefügt

Zu den deklarierten Eigenschaften lassen sich weitere hinzufügen. Sie könnten z. B. den Konstruktor `Person` um die Eigenschaften `geschlecht` und `lebensform` erweitern.

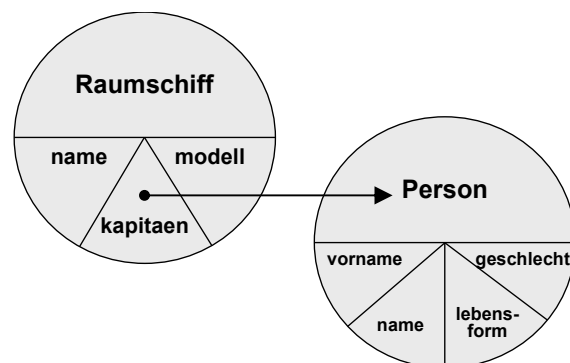
Nach der Deklaration des Konstruktors `Person` können Sie den jeweiligen Raumschiffen eine Person als Kapitän zuordnen. Dazu wird der Konstruktor `Raumschiff` um die Eigenschaft `kapitaen` erweitert.

```

function Raumschiff(name, modell, kapitaen) {
  this.name = name;
  this.modell = modell;
  this.kapitaen = kapitaen;
}

```

Der Konstruktor `Raumschiff` besitzt jetzt eine weitere Eigenschaft `kapitaen`. Als Kapitän des Raumschiffs soll eine Person angegeben werden, die mit dem Konstruktor `Person` erzeugt wurde. Beim Aufruf wird als Argument für die Funktion `Raumschiff()` die Variable `Person` vom Typ `object` übergeben.



Damit ist es möglich, den Vornamen des Raumschiffkapitäns über ein Objekt des Konstruktors `Raumschiff` und dessen Referenz `kapitaen` auf ein Objekt des Konstruktors `Person` und somit dessen Eigenschaft `Vorname` auszulesen, z. B. `enterprise.kapitaen.vorname`.

Durch dieses Prinzip können Sie Objekte beliebig tief verschachteln. So kann einer Person über die Eigenschaft `adresse` z. B. ein Adressobjekt zugewiesen werden. Auf den Wohnort, der von der Eigenschaft `ort` repräsentiert werden könnte, könnten Sie dann z. B. über `enterprise.kapitaen.adresse.ort` zugreifen.

Beispiel: *raumschiff_kapitaen.html*

Der Konstruktor `Raumschiff` wird nun wie beschrieben um die Eigenschaft `kapitaen` erweitert. Dazu wird ein `Person`-Konstruktor deklariert, der die Eigenschaften einer Person vereint. Die Eigenschaft `kapitaen` ist vom Typ `Person`. Um auf eine Eigenschaft des Kapitäns eines Raumschiffs außerhalb eines `Raumschiff`-Objekts zuzugreifen, verwenden Sie den Zugriffspfad wie oben behandelt.

```
<body>
  <h3>Die Kapitäne der Raumschiffe</h3>
  <script type="text/javascript">
    ① function Raumschiff(name, modell, kapitaen) {
        this.name = name;
        this.modell = modell;
        this.kapitaen = kapitaen;
      }
    ② function Person(name, vorname, geschlecht, lebensform) {
        this.name = name;
        this.vorname = vorname;
        this.geschlecht = geschlecht;
        this.lebensform = lebensform;
      }
    ③ var k1 = new Person("Kirk", "James Tiberius", "männlich",
                        "Mensch");
    ④ var enterprise = new Raumschiff("U.S.S. Enterprise",
                                    "NCC-1701", Kirk);
    ⑤ document.write("<br />", enterprise.name,
                    " ", voyager.modell);
    ⑥ document.write("<br />Kapitän: ",
                    enterprise.kapitaen.vorname, " ",
                    enterprise.kapitaen.name, " ",
                    enterprise.kapitaen.geschlecht);
    ⑦ document.write("<br />lebensform: ",
                    enterprise.kapitaen.lebensform);
    ⑧ var k2 = new Person("Janeway", "Kathryn", "weiblich",
                        "Mensch");
        var voyager = new Raumschiff("U.S.S. Voyager",
                                    "NCC-74656", k2);
```

```

⑨ document.write("<br />", voyager.name, " ", voyager.modell);
document.write("<br />Kapitän: ", voyager.kapitaen.vorname,
" ", voyager.kapitaen.name, " ",
voyager.kapitaen.geschlecht);
document.write("<br />lebensform: ",
voyager.kapitaen.lebensform);
</script>
</body>

```

- ① Es wird der Konstruktor `Raumschiff` mit den Eigenschaften `name`, `modell` und `kapitaen` deklariert.
- ② Zudem werden dem Konstruktor `Person` die beiden Eigenschaften `geschlecht` und `lebensform` hinzugefügt.
- ③ Ein neues Objekt wird in der Variablen `k1` erzeugt, indem der Konstruktor `Person` mit den entsprechenden Angaben aufgerufen wird. Die Angaben werden zur Initialisierung der Eigenschaften des Objekts benötigt.
- ④ Ein `Raumschiff`-Objekt `enterprise` wird erzeugt. Für die Angabe des Kapitäns wird das Objekt in der Variablen `k1` übergeben, sodass die Eigenschaft `enterprise.kapitaen` auf das Objekt vom Typ `Person` `k1` referenziert.
- ⑤ Wie bisher werden die Eigenschaften `name` und `modell` des Objekts in der Variablen `enterprise` ausgegeben.
- ⑥ Für die Angabe des Kapitäns der `Enterprise` wird über die Referenz `kapitaen` direkt auf die Eigenschaft `name` des `Person`-Objekts in der Variablen `k1` zugegriffen und deren Wert ausgegeben. Dies erfolgt ebenfalls für die beiden Eigenschaften `vorname` und `geschlecht`.
- ⑦ Bei der Ausgabe der Lebensform wird ebenso vorgegangen.
- ⑧ Ein zweites `Person`-Objekt wird in der Variablen `k2` erzeugt, und das `Raumschiff`-Objekt in der Variablen `voyager` wird mit dem Kapitän `Kathryn` erstellt.
- ⑨ Die Angaben zum `Raumschiff Voyager` und zu dessen Kapitän werden über die Variable `voyager` abgerufen.

Die Kapitäne der Raumschiffe

U.S.S. Enterprise NCC-1701
 Kapitän: James Tiberius Kirk, männlich
 Lebensform: Mensch

U.S.S. Voyager NCC-74656
 Kapitän: Kathryn Janeway, weiblich
 Lebensform: Mensch

*Geschlecht und Lebensform
werden festgelegt*

Werte überschreiben

Die Eigenschaft `kapitaen` des Objekts in der Variablen `enterprise` verweist auf dasselbe Objekt wie die Variable `k1`. Ändern Sie eine Eigenschaft über die Variable `k1`, ändert sich ebenfalls der Wert der Eigenschaft `kapitaen` in der Variablen `enterprise`. Die beiden folgenden Anweisungen führen dieselbe Operation aus:

```

k1.lebensform = "Nicht-Vulkanier";
enterprise.kapitaen.lebensform = "Nicht-Vulkanier";

```

Dynamisches Hinzufügen und Löschen von Eigenschaften und Methoden

Einem Objekt können in JavaScript wegen der losen Typisierung dynamisch Eigenschaften hinzugefügt oder daraus entfernt werden. Um einem Objekt eine Eigenschaft hinzuzufügen, geben Sie die gewünschte neue Eigenschaft durch einen Punkt getrennt hinter dem Namen der Variablen an und weisen ihr einen Wert zu. Damit wird die Liste, über die in JavaScript Objekte repräsentiert werden, automatisch um ein neues Schlüssel-Werte-Paar erweitert.

Es ist im JavaScript-Konzept auch vorgesehen, dass Sie über den Operator `delete` eine Eigenschaft wieder entfernen können. Wird nach dem Entfernen auf die Eigenschaft zugegriffen, wird der Wert `undefined` geliefert. Beachten Sie, dass Sie auf diese Weise keine privaten Eigenschaften löschen können. Da sich Klassen automatisch im strikten Modus befinden und private Eigenschaften nur in Klassenkörpern referenziert werden können, bedeutet dies, dass private Eigenschaften niemals gelöscht werden können. Aber es gibt noch weitere Einschränkungen hinsichtlich der Deklaration mit dem Schlüsselwort `var` oder `let`, während das Löschen von Bezeichnern funktionieren kann, wenn sich Bezeichner auf eine konfigurierbare Eigenschaft des globalen Objekts beziehen.

Auch sonst funktioniert dieses Löschen nicht in allen Browsern, nicht bei Standardeigenschaften und auch nicht bei Standardobjekten. Wegen der vielen Ausnahmesituationen sollten Sie besser eine Eigenschaft explizit auf den Wert `undefined` setzen, wenn Sie diese beseitigen wollen. Allerdings müssen Sie auch hier beachten, dass man nicht jeder Eigenschaft einen neuen Wert zuweisen darf.

Beispiel: *delete1.js*

```
function Kunde(name) {  
    this.name = name;  
}  
kd = new Kunde("Meier");  
kd.vorname = "Frank"; // neue Eigenschaft hinzufügen  
console.log(kd);  
delete kd.vorname; // die Eigenschaft wieder entfernen -  
                  // ein Weg, aber kritisch  
console.log(kd);  
kd.vorname = undefined; // die Eigenschaft wieder entfernen -  
                        // alternativer Weg  
console.log(kd);
```

5.3 Methoden

Funktionen, die einem Objekt zugeordnet sind, werden als Methoden bezeichnet. Sie werden einem Objekt als eine Eigenschaft zugewiesen. Der Wert der Eigenschaft ist eine **Funktionsreferenz**. Dies ist einfach der Name der Funktion oder der Rückgabewert einer anonymen Funktion bzw. einer sogenannten Callback-Funktion. Eine solche Callback-Funktion ist genau darüber definiert, dass sie eine Funktionsreferenz als Rückgabewert liefert.

Beachten Sie, dass bei einer Funktionsreferenz **niemals** ein Klammernpaar notiert wird (das wäre ein Funktionsaufruf, und der ist hier falsch).

```
function methodenName(Wert1, Wert2, ...) {  
    ...  
}  
function Konstruktorname(Wert1, Wert2, ...) {  
    this.eigenschaft1 = Wert1;  
    this.eigenschaft2 = methodenName;  
}
```

Über das Schlüsselwort `function` wird eine Funktion in JavaScript deklariert, die als Methode fungieren soll und sich erst einmal in nichts von einer normalen Funktion unterscheidet. Wird der Funktionsname in einem Konstruktor mit dem Schlüsselwort `this` zugewiesen, wird die Funktion jedoch als Methode des konstruierten Objekts interpretiert.

Innerhalb einer Funktion, die Sie als Methode bei einem Objekt verwenden wollen, können Sie mit dem Schlüsselwort `this` auf die Eigenschaften des aktuellen Objekts zugreifen. `this` enthält immer eine Referenz auf das aktuelle Objekt. Sie sollten diese Art der Notation von `this` (also in einer benannten Funktion) allerdings vermeiden, da es nicht ausgeschlossen ist, dass die Funktion auch direkt (also ohne vorangestelltes Objekt) aufgerufen wird und dann ist `this` nicht definiert. Methoden werden aber auch aus anderen Gründen (Datenkapselung) in der Regel anonym deklariert und dann befindet man sich innerhalb des Objekts und kann `this` gefahrlos verwenden. Aus objektorientierter Sicht ist diese Art der Programmierung, nämlich die Programmierung über Datenkapselung gut, da die Deklaration der Methode nicht von der Verwendung im Konstruktor getrennt ist. Die Datenkapselung ist ein grundlegendes Prinzip in der objektorientierten Programmierung.

Auch aus dem Grund sollten Sie eine Methode in einem Konstruktor mit einer **anonymen** Funktion deklarieren. Die anonyme Funktion liefert als Rückgabewert immer eine Funktionsreferenz auf sich selbst (wie oben erwähnt **Callback** genannt). Durch die Deklaration im Konstruktor wird die anonyme Funktion genau da deklariert, wo im Fall einer benannten Funktion der Bezeichner der Funktion steht.

Wir haben diese Verwendung einer anonymen Funktion schon eingangs gesehen, aber hier soll ein weiteres Beispiel folgen.

Beispiel: *raumschiff_bewegung.html*

Sie können einem Objekt des Konstruktors `Raumschiff` jetzt eine Methode hinzufügen, die eine Fortbewegung des Raumschiffs erlaubt. In dem Beispiel wurde der Konstruktor `Person` zugunsten der Übersicht weggelassen. Deshalb muss auch die Eigenschaft `raumschiff.kapitaen` entfernt werden.

```
<body>
  <h3>Und es bewegt sich doch</h3>
  <script type="text/javascript">
    ①  function Raumschiff(name, modell)      {
        this.name = name;
        this.modell = modell;
    ②  this.entfernung = 0;
    ③  this.entfernen = function (lichtjahre) {
    ④    this.entfernung += lichtjahre;
        document.write(this.name + " => Aktuelle Reise: " +
                        lichtjahre + " Lichtjahre ->
                        Erdentfernung: "+this.entfernung+"
                        Lichtjahre<br />");
    }
  }
  var enterprise = new Raumschiff("U.S.S. Enterprise",
                                  "NCC-1701");
  var voyager = new Raumschiff("U.S.S. Voyager",
                                "NCC-74656");

  document.write("<p>Beide Raumschiffe sind von der Erde
                  gestartet:<br />");

    ⑤  enterprise.entfernen(50);
        voyager.entfernen(180);
        enterprise.entfernen(200);
        voyager.entfernen(-50);
        enterprise.entfernen(-100);
        enterprise.entfernen(-150);
        voyager.entfernen(40);
        voyager.entfernen(-170);
  </script>
</body>
```

- ① Den Konstruktor `Raumschiff` wird erstellt.
- ② Für die Vorwärtsbewegung des Raumschiffs wird für das `Raumschiff`-Objekt eine neue Eigenschaft `entfernung` deklariert, die beim Erstellen eines Raumschiffs auf den Wert 0 gesetzt wird.
- ③ Weiterhin erhält jedes `Raumschiff`-Objekt die Methode `entfernen`, der eine anonyme Funktion zugewiesen wird. Als Übergabeparameter erwartet sie die für die Strecke benötigte Anzahl an Lichtjahren.

- ④ Über `this` sprechen Sie das gerade aktuelle Objekt an. Somit können Sie zu dessen Eigenschaft `entfernung` den aktuellen Wert des Parameters `lichtjahre` hinzuzählen. Es folgt die Ausgabe der aktuellen Werte.
- ⑤ Nach dem Erstellen der `Raumschiff`-Objekte erzeugen die Aufrufe der Methoden `voyager.entfernen()` und `enterprise.entfernen()` die unterschiedlichen Werte für die Eigenschaft `this.entfernung`. Die Funktionen erzeugen die folgende Ausgabe:

Und es bewegt sich doch

```
Beide Raumschiffe sind von der Erde gestartet:
U.S.S. Enterprise => Aktuelle Reise: 50 Lichtjahre -> Erdentfernung: 50 Lichtjahre
U.S.S. Voyager => Aktuelle Reise: 180 Lichtjahre -> Erdentfernung: 180 Lichtjahre
U.S.S. Enterprise => Aktuelle Reise: 200 Lichtjahre -> Erdentfernung: 250 Lichtjahre
U.S.S. Voyager => Aktuelle Reise: -50 Lichtjahre -> Erdentfernung: 130 Lichtjahre
U.S.S. Enterprise => Aktuelle Reise: -100 Lichtjahre -> Erdentfernung: 150 Lichtjahre
U.S.S. Enterprise => Aktuelle Reise: -150 Lichtjahre -> Erdentfernung: 0 Lichtjahre
U.S.S. Voyager => Aktuelle Reise: 40 Lichtjahre -> Erdentfernung: 170 Lichtjahre
U.S.S. Voyager => Aktuelle Reise: -170 Lichtjahre -> Erdentfernung: 0 Lichtjahre
```

Die Fortbewegung erfolgt über eine Methode.

Beachten Sie, dass die runden Klammern bei der Deklaration der anonymen Funktion dazugehören. Das ist ohne einen Namen **kein Aufruf**.

5.4 Vererbung

In modernen Versionen von JavaScript bzw. ECMAScript kann man das Konzept der Vererbung bei Klassen nutzen. Dies beschreibt ein Konzept der objektorientierten Programmierung, bei dem eine Klasse (die **Subklasse**) von einer anderen Klasse (der **Superklasse**) erbt. Die Subklasse erbt alle Eigenschaften und Methoden der Superklasse und kann diese überschreiben oder erweitern. Dies ermöglicht es der Subklasse, die Eigenschaften und Methoden der Superklasse zu nutzen, ohne sie neu schreiben zu müssen.

Generalisierung ist damit der logische Prozess, bei dem eine Superklasse erstellt wird, die allgemeine Eigenschaften und Methoden enthält, die von allen Subklassen verwendet werden können. **Spezialisierung** ist der entgegengesetzte Prozess, bei dem Subklassen erstellt werden, die spezifische Eigenschaften und Methoden enthalten, die nur von dieser Subklasse verwendet werden können.

Wenn Sie in JavaScript eine Subklasse von einer Superklasse erzeugen wollen, nutzen Sie dazu bei der Deklaration der Subklasse das Schlüsselwort `extends`, gefolgt von der Superklasse.

Im Konstruktor der Subklasse greifen Sie mit der Methode `super()` auf den Konstruktor der Superklasse zu und reichen damit die Werte aus der Subklasse weiter. Damit müssen Sie alle Schritte, die bereits in der Superklasse programmiert wurden, nicht noch einmal notieren.

Beispiel: *vererbung.js*

```
"use strict";
class Person {
  constructor(name, alter, geschlecht) {
    this.name = name;
    this.alter = alter;
    this.geschlecht = geschlecht;
  }
}
class Mitarbeiter extends Person {
  constructor(name, alter, geschlecht, id) {
    super(name, alter, geschlecht);
    this.id = id;
  }
}
let mitarbeiter1 = new Mitarbeiter("Max Mustermann", 25,
  "männlich", 4711);
console.log(mitarbeiter1);
```

Die Klasse `Person` deklariert bereits drei Eigenschaften. Die Subklasse `Mitarbeiter` ergänzt eine weitere Eigenschaft. Damit benötigt der Konstruktor von `Mitarbeiter` vier Angaben.

Das ist die Ausgabe von dem Quellcode:

```
Mitarbeiter {
  name: 'Max Mustermann',
  alter: 25,
  geschlecht: 'männlich',
  id: 4711
}
```

5.5 Anweisungen und Operatoren für Objekte

Im Folgenden lernen Sie Techniken kennen, die Ihnen im Zusammenhang mit Objekten nützlich sein können.

Test auf Existenz eines Objekts oder einer Eigenschaft

Über eine Anweisung, die eine Bedingung prüft – meist die `if`-Anweisung – können Sie testen, ob ein Objekt eine bestimmte Eigenschaft besitzt oder ob das Objekt vorhanden ist. Existiert die Eigenschaft bzw. das Objekt, erhalten Sie als Ergebnis `true`. Die nachfolgenden Beispiele finden Sie in der Datei *anweisung_operator.html*.

Beispiel

Der Konstruktor Kunde wird mit der Eigenschaft Name deklariert. In den beiden folgenden if-Anweisungen wird geprüft, ob das Objekt in kd die Eigenschaften Name und Vorname besitzt.

```
function Kunde(name) {  
    this.name = name;  
}  
var kd = new Kunde('Meier');  
if(kd.name)  
    document.write("Die Eigenschaft 'name' im Objekt 'Kunde'  
                    existiert.");  
if(!kd.vorname)  
    document.write("Die Eigenschaft 'vorname' im Objekt 'Kunde'  
                    existiert nicht.");
```

with-Anweisung

Müssen Sie hintereinander auf mehrere Eigenschaften und Methoden eines Objekts zugreifen, können Sie den Zugriff darauf über die with-Anweisung verkürzen. Dazu wird nach dem Schlüsselwort with der Name der Variablen, die das Objekt referenziert, in Klammern angefügt. Dadurch muss im folgenden Anweisungsblock nur der Name der Eigenschaft ohne den Variablennamen angegeben werden. Dabei wird für jeden Bezeichner (Variable) zuerst nach einer Eigenschaft in dem Objekt gesucht, dann nach einem lokalen und globalen Bezeichner.

Beispiel

Durch die Verwendung der with-Anweisung müssen Sie innerhalb der geschweiften Klammern `{ }` in den Methoden `document.write()` nicht mehr die Variable `kd` angeben.

```
function Kunde(name, vorname) {  
    this.name = name;  
    this.vorname = vorname;  
}  
var kd = new Kunde("Meier", "Hans");  
with(kd) {  
    document.write("Der Vorname lautet: " + vorname + "<br />");  
    document.write("Der Nachname lautet: " + name + "<br />");  
}
```

Beachten Sie, dass diese with-Anweisung die Ausführungsgeschwindigkeit eines Skripts verlangsamt. Sollte es auf Laufzeitoptimierung ankommen, verzichten Sie auf diese Anweisung. Der Vorteil ist die Reduzierung des Schreibaufwands und der Codegröße, die auch das Skript schneller übertragen lässt. Sie müssen bei einer Verwendung die Vor- und Nachteile abwägen.

for-in-Anweisung

Um über alle Eigenschaften eines Objekts zu iterieren, können Sie die `for-in`-Anweisung nutzen. Dazu wird nach der `for`-Anweisung in Klammern eine Laufvariable deklariert. Dieser folgt das Schlüsselwort `in` und ein Variablenname, der auf ein Objekt zeigt.

Beispiel

Die Laufvariable enthält in jedem Durchlauf den Bezeichner der nächsten Eigenschaft. Außerdem können Sie über den Variablennamen und die gleichzeitige Angabe des Eigenschaftsnamens in eckigen Klammern auf den Wert der Eigenschaft zugreifen. Die folgende Anweisung gibt die Namen der Eigenschaften des `Kunde`-Objekts in `kd` und dessen Werte aus.

```
function Kunde(name, vorname, alter, geschlecht) {  
    this.name = name;  
    this.vorname = vorname;  
    this.alter = alter;  
    this.geschlecht = geschlecht;  
}  
var kd = new Kunde('Meier', 'Hans', 65, 'männlich');  
for(var i in kd)  
    document.write("Name: " + i + ", Wert: " + kd[i]);
```

instanceof-Operator

Über diesen Operator können Sie prüfen, ob ein Objekt vom Typ eines bestimmten Konstruktors ist. In diesem Fall wird als Ergebniswert `true` geliefert.

Beispiel

```
function Kunde() {  
}  
var kd = new Kunde();  
if(kd instanceof Kunde)  
    document.write("kd ist eine Instanz vom Konstruktor 'Kunde'");
```

5.6 Übungen

Übung 1: Objekte erzeugen und auslesen

Übungsdatei: *kap05/raumschiff_bewegung.html*

Ergebnisdateien: *kap05/uebung1-1.html*,
kap05/uebung1-2.html

1. Erweitern Sie den Konstruktor `Raumschiff` um die Methode `zurueck`. Erzeugen Sie ein neues Objekt in der Variablen `enterprise`.
Der Methodenaufruf `enterprise.zurueck(lichtjahre)` soll die folgenden Ausgaben erzeugen:
Wenn die Entfernung > 0 Lichtjahre ist:
`"U.S.S. Enterprise kommt zurück zur Erde: nur noch x Lichtjahre;"`
Wenn die Entfernung = 0 Lichtjahre ist:
`"Willkommen zu Hause auf der Erde, U.S.S. Enterprise!"`
Ansonsten wird ausgegeben:
`"U.S.S. Enterprise ist bereits auf der Erde gelandet!"`

Und es bewegt sich doch

```
Das Raumschiff ist von der Erde gestartet  
U.S.S. Enterprise => Aktuelle Reise: 50 Lichtjahre -> Erdentfernung: 50 Lichtjahre  
U.S.S. Enterprise => Aktuelle Reise: 100 Lichtjahre -> Erdentfernung: 150 Lichtjahre  
U.S.S. Enterprise kommt zurück zur Erde: nur noch 100 Lichtjahre;  
Willkommen zu Hause auf der Erde, U.S.S. Enterprise!  
  
U.S.S. Enterprise ist bereits auf der Erde gelandet!
```

Diese Ausgabe wird generiert bei den Methodenaufrufen:

```
enterprise.entfernen(50);  
enterprise.entfernen(100);  
enterprise.zurueck(50);  
enterprise.zurueck(100);  
enterprise.zurueck(50);
```

2. Geben Sie über die `for-in`-Anweisung alle Eigenschaften des Objekts in der Variablen `voyager` aus.

Übung 2: Objekt-Methode mehrfach aufrufen

Übungsdatei: --

Ergebnisdatei: *kap05/uebung2.html*

1. Legen Sie über eine Schleifenfunktion mit dem Raumschiff `voyager` 10-mal die Entfernung von 20 Lichtjahren zurück.

6

Vordefinierte Objekte

6.1 Grundlagen zu vordefinierten Objekten

In JavaScript gibt es bereits einige vordefinierte Objekte bzw. Prototypen/Klassen, die Sie direkt verwenden können. Mathematische Methoden sind in diesen Objekten genauso vorhanden wie Methoden zur Berechnung und Verwendung von Uhrzeiten oder Kalendertagen. Zeichenketten werden grundsätzlich als `String`-Objekte angelegt, die dem Programmierer auch Methoden zur Zeichenkettenverarbeitung zur Verfügung stellen. Auch Zahlen und boolesche Werte werden als Objekte des jeweiligen Konstruktors abgelegt. Weiter gibt es in JavaScript die Klasse `RegExp`. Diese implementiert das Konzept der regulären Ausdrücke für JavaScript. Arrays sind bereits vom Konzept her in JavaScript Objekte. Auch zur Unterstützung von JSON gibt es eine Klasse `JSON`.

Beachten Sie, dass die eingebauten Objekte in JavaScript – im Gegensatz zu selbst erzeugten Objekten – immer mit einem Großbuchstaben beginnen. Das entspricht der weltweiten Konvention für Klassennamen und macht explizit deutlich, dass damit ein vordefiniertes Objekt in Form eines Prototyps vorliegt. Da diese Assoziation grundlegend ist, sollten Sie selbstdefinierte Objekte niemals mit einem Großbuchstaben beginnen. Aber um es noch einmal zu betonen: Das ist keine technische Notwendigkeit. Wie in der Geschichte die purpurne Farbe früher für Könige und Fürsten reserviert war, sollte man sich bei eigenen Objektvariablen keinen Missbrauch gestatten. Ärger ist sonst „vorprogrammiert“.

6.2 Das Objekt `String` für Zeichenketten

Das `String`-Objekt wurde bereits in den vorangegangenen Beispielen verwendet, da alle Zeichenkettenvariablen `String`-Objekte sind. Dies bedeutet, dass nicht nur der Inhalt der Zeichenkette durch den Variablennamen erreichbar ist, sondern es gibt auch Eigenschaften und Methoden. Die Eigenschaft `length` des `String`-Objekts liefert etwa die Länge einer Zeichenkette. Außerdem besitzt das Objekt Methoden, um z. B. einen Teil der Zeichenkette zurückzuliefern.

```
x = String.Eigenschaft;  
y = String.Methode(Übergabewert);
```

Zur Darstellung der verschiedenen Verarbeitungsmöglichkeiten soll als Beispiel die Variable `zitat` mit der Zeichenkette "Sein oder nicht sein," verwendet werden (`zitat = "Sein oder nicht sein, "`).

Der Zugriff auf das erste Zeichen erfolgt im Folgenden über den Index 0. Auf das zweite Zeichen greifen Sie mit dem Index 1 zu usw. Der Grund ist, dass ein String intern als Zeichenarray mit einer automatischen numerischen Indizierung aufgebaut ist und Arrays mit solch einer automatischen Indizierung nullindiziert sind.

Methode	Erläuterung	Ergebnis
<code>zitat.charAt(6);</code>	Mit <code>charAt(x)</code> können Sie das Zeichen auslesen, das sich an der <code>x+1</code> -ten Stelle befindet.	d
<code>zitat.charCodeAt(6);</code>	<code>charCodeAt(x)</code> liefert den ASCII-Code des <code>x+1</code> -ten Zeichens	100

Beispiel: *objekt_string1.html*

Zeichenketten-Verarbeitung

```
zitat1 = Sein oder nicht sein,
zitat2 = das ist hier die Frage.
zitat1.length = 21 Zeichen
zitat1.charAt(6) = d
zitat1.charCodeAt(6) = 100
zitat1.concat(zitat2) = Sein oder nicht sein, das ist hier die Frage.
zitat1.indexOf('c') = 12. Stelle
zitat1.lastIndexOf('e') = 17. Stelle
zitat1.slice(5, 9) = oder
zitat1.split(' ') = Sein,oder,nicht,sein,
zitat1.substr(5, 4) = oder
zitat1.substring(5, 9) = oder
zitat1.toLowerCase() = sein oder nicht sein,
zitat1.toUpperCase() = SEIN ODER NICHT SEIN,
```

Ausgabe der *String*-Eigenschaft und -Methoden im Browser

Ein String besitzt auch noch eine ganze Reihe an Methoden, die den String in bestimmte HTML-Tags wie `strong` oder `big` einschließen. Sie sind jedoch veraltet und werden in der Praxis nicht mehr eingesetzt. In den BuchPlus-Beispieldateien finden Sie unter *objekt_string2.html* einige Beispiele.

6.3 Math für mathematische Berechnungen

Einige Aufgabenstellungen benötigen mathematische Funktionen oder Konstanten zu ihrer Lösung. JavaScript stellt solche Funktionen bzw. Methoden und Konstanten über die Klasse `Math` zur Verfügung, ohne dass eine Instanz erzeugt werden muss. Das bedeutet, `Math` ist kein Objekt oder Prototyp im herkömmlichen Sinn, und die Konstanten und Methoden von `Math` sind das, was in echten objektorientierten Sprachen **Klassenelemente** bzw. statische Elemente sind. Aber das brauchen Sie bei der Anwendung nur insofern zu beachten, dass Sie immer `Math` direkt voranstellen und nicht vorher `Math` mit `new` instanziierten müssen.

Mathematische Konstanten

Diese Konstanten repräsentieren wichtige mathematische Werte wie die Eulersche Zahl, Pi oder die Wurzel aus der Zahl 2.

Beispiel: `objekt_math_const.html`

In der Tabelle sind die mathematischen Konstanten abgebildet, die Sie über die Angabe der Eigenschaft wie in der Spalte *Beispiel* abrufen können. Beachten Sie die konsequente Großschreibung.

Mathematische Konstanten			
Eigenschaft	Bedeutung	Beispiel	Ausgabe
E	Eulersche Zahl	Math.E	2.718281828459045
LN2	Natürlicher Logarithmus von 2	Math.LN2	0.6931471805599453
LN10	Natürlicher Logarithmus von 10	Math.LN10	2.302585092994046
LOG2E	Logarithmus von e zur Basis 2	Math.LOG2E	1.4426950408889633
LOG10E	Logarithmus von e zur Basis 10	Math.LOG10E	0.4342944819032518
PI	Zahl π	Math.PI	3.141592653589793
SQRT1_2	1 dividiert durch die Wurzel von 2	Math.SQRT1_2	0.7071067811865476
SQRT2	Wurzel aus 2	Math.SQRT2	1.4142135623730951

Mathematische Konstanten des `Math`-Objekts

Mathematische Methoden

JavaScript stellt wichtige mathematische Funktionalitäten über Methoden von `Math` bereit, um z. B. Winkel, Sinus, Cosinus oder Tangens zu berechnen. Wichtig sind die Methoden `round()` und `floor()` zum Runden und `random()` zur Generierung einer **Zufallszahl**.

Mathematische Funktionen				
Methode	Bedeutung	Wertebereich	Beispiel	Ausgabe
<code>abs(Zahl)</code>	Absolutwert	Reale Zahlen	<code>Math.abs(-9)</code>	9
<code>acos(Zahl)</code>	Arcuscosinus	$[-1;1]$	<code>Math.acos(-1)</code>	3.141592653589793
<code>asin(Zahl)</code>	Arcussinus	$[-1;1]$	<code>Math.asin(0.5)</code>	0.5235987755982989
<code>atan(Zahl)</code>	Arcustangens	$[-1;1]$	<code>Math.atan(-0.5)</code>	-0.4636476090008061
<code>ceil(Zahl)</code>	Aufrunden zur nächsten Ganzzahl	Reale Zahlen	<code>Math.ceil(9.3)</code>	10
<code>cos(Zahl)</code>	Cosinus	Reale Zahlen (Rad)	<code>Math.cos(Math.PI)</code>	-1
<code>exp(Zahl)</code>	Exponentialwert der Zahl auf Basis der Eulerschen Konstanten E	Reale Zahlen	<code>Math.exp(9.3)</code>	10938.019208165191
<code>floor(Zahl)</code>	Abrunden zur nächsten Ganzzahl	Reale Zahlen	<code>Math.floor(3.9)</code>	3
<code>log(Zahl)</code>	Natürlicher Logarithmus	Reale Zahlen > 0	<code>Math.log(3.9)</code>	1.3609765531356006
<code>max(Zahl1, Zahl2)</code>	Rückgabe der größeren Zahl	Reale Zahlen	<code>Math.max(9,3)</code>	9
<code>min(Zahl1, Zahl2)</code>	Rückgabe der kleineren Zahl	Reale Zahlen	<code>Math.min(9,3)</code>	3
<code>pow(Zahl1, Zahl2)</code>	Exponentialfunktion "Zahl1 hoch Zahl2"	Reale Zahlen	<code>Math.pow(9,3)</code>	729
<code>random()</code>	Zufallszahl zwischen 0 und 1		<code>Math.random()</code>	0.41002227039621724
<code>round(Zahl)</code>	Kaufmännisches Runden	Reale Zahlen	<code>Math.round(3.5)</code>	4
<code>sin(Zahl)</code>	Sinus	Reale Zahlen (Rad)	<code>Math.sin(Math.PI/2)</code>	1
<code>sqrt(Zahl)</code>	Quadratwurzel	Reale Zahlen > 0	<code>Math.sqrt(9)</code>	3
<code>tan(Zahl)</code>	Tangens	Reale Zahlen (Rad)	<code>Math.tan(Math.PI/4)</code>	0.9999999999999999

Mathematische Funktionen stehen als Methoden des `Math`-Objekts zur Verfügung

Beispiel: *objekt_math_random.html*

Die Methode `random()` liefert eine Zufallszahl zwischen 0 und 1. Das folgende Beispiel beinhaltet die Funktion `rolleWuerfel()`, die mit der `random()`-Methode zufällige Zahlen zwischen 1 und 6 erzeugt:

```
<body>
  <h2>Würfelergebnisse:</h2>
  <div id="wuerfel">-
    <script type="text/javascript">
      function rolleWuerfel() {
①      var zufallszahl = Math.random() * 6;
②      return (Math.floor(zufallszahl) + 1);
      }
      for(var i = 0; i < 10; i++)
③      document.write(rolleWuerfel() + " - ");
    </script>
  </div>
</body>
```

- ① Die Methode `Math.random()` erzeugt bei jedem Aufruf eine beliebig zufällige reelle Zahl zwischen 0 (inklusive) und 1 (nicht inklusiv). Um im ersten Schritt den Zahlenbereich 0 bis 5 zu erreichen, muss die Zufallszahl mit dem Wert 6 multipliziert werden. Das Ergebnis wird als Gleitkommazahl in der Variablen `zufallszahl` abgelegt.
- ② Vor der Rückgabe wird der Wert der Variablen `zufallszahl` mit der Methode `floor()` auf die nächste Ganzzahl abgerundet. Dies ergibt eine ganze Zahl zwischen 0 und 5. Die Addition von 1 ergibt das gewünschte Ergebnis einer ganzen Zahl zwischen 1 und 6.
- ③ Im Rumpf des HTML-Dokuments wird die Funktion `rolleWuerfel()` über eine `for`-Schleife zehnmal ausgeführt und das Ergebnis im Browser ausgegeben.

Würfelergebnisse:

- 1 - 4 - 4 - 6 - 4 - 2 - 4 - 5 - 2 - 6 -

Ausgabe von Zufallszahlen

Die Werte ändern sich jedes Mal, sobald Sie das HTML-Dokument über das entsprechende Menü im Browser aktualisieren. Es wird dadurch immer eine zufällige Zahlenfolge ausgegeben.

6.4 Number für Zahlen

In JavaScript gibt es für besondere Situationen bei Zahlen den `Number`-Typ. Der Typ `Number` besitzt Eigenschaften für numerische Konstanten. Die Werte dieser Eigenschaften können nicht geändert werden.

Eigenschaft	Erläuterung	Wert
<code>MAX_VALUE</code>	<code>MAX_VALUE</code> beinhaltet die größte Zahl, die verarbeitet werden kann.	<code>1.7976931348623157e+308</code>
<code>MIN_VALUE</code>	<code>MIN_VALUE</code> beinhaltet die kleinste Zahl, die verarbeitet werden kann.	<code>5e-324</code>
<code>NaN</code>	Der Wert <code>NaN</code> (not a number) kann über die Funktion <code>isNaN()</code> mit Ergebnissen von Operationen verglichen werden.	<code>NaN</code>
<code>NEGATIVE_INFINITY</code>	Der Wert einer Variablen ist <code>-Infinity</code> , wenn die Zahl kleiner als <code>MIN_VALUE</code> ist. Das steht also für negativ Unendlich.	<code>-Infinity</code>
<code>POSITIVE_INFINITY</code>	Der Wert einer Variablen ist <code>Infinity</code> , wenn die Zahl größer als <code>MAX_VALUE</code> ist. Das steht also für Unendlich.	<code>Infinity</code>

Beispiel

In den folgenden Anweisungen wird geprüft, ob die Multiplikation zweier Werte den maximalen Wertebereich von JavaScript überschreitet. Solange der Wert der Multiplikation kleiner als der Maximalwert ist, könnte in der ersten Funktion eine weitere Berechnung durchgeführt werden. Die zweite Funktion gibt stattdessen eine Meldung aus, in der auf den Überlauf des Wertebereichs hingewiesen wird.

```
if (Zahl1 * Zahl2 <= Number.MAX_VALUE)
    funktion1();
else
    funktion2();
```

6.5 Objekt vom Typ `Date` für Zeitangaben

Auf die Systemzeit wird in JavaScript über `Date`-Objekte zugegriffen. Diese Objekte geben aber nicht nur Auskunft über das aktuelle Datum, sondern es lassen sich auch für beliebige Kalendertage und Uhrzeiten Datumsobjekte erzeugen.



In JavaScript wird die Zeit in Millisekunden gespeichert, die seit dem 1. Januar 1970 um 0:00 Uhr vergangen ist.

Es gibt verschiedene Möglichkeiten, um ein Datumsobjekt zu erzeugen und zu initialisieren:

- ✓ Geben Sie bei der Instanziierung keine Parameter an, wird das Objekt mit dem aktuellen Datum und der aktuellen Uhrzeit initialisiert.

```
var d = new Date();
```

- ✓ Sie können aber auch ein bestimmtes Datum und eine Uhrzeit angeben. Dazu geben Sie z. B. die entsprechenden Werte in Form einer Zeichenkette an.

```
var d = new Date("Monatsname_englisch Tag, Jahr  
Stunde:Minute:Sekunde");
```

- ✓ Das Datum und die Uhrzeit lassen sich auch in numerischer Form angeben.

```
var d = new Date(Jahr, Monat, Tag, Stunde, Minute, Sekunde);
```

Beispiel

```
var zeit1 = new Date();    // aktuelles Datum und Uhrzeit wird  
                           übergeben  
var zeit2 = new Date("March 13, 2023 16:07:00"); // 13.03.2023  
                                                    um 16:07 Uhr  
var zeit3 = new Date(2022, 2, 13, 16, 07, 00);    // 13.03.2022  
                                                    um 16:07 Uhr
```



Bei der Initialisierung in numerischer Form ist zu beachten, dass die Monate im Wertebereich von 0 bis 11 liegen (0 = Januar, 1 = Februar ... 11 = Dezember). Deshalb steht der Wert 2 beim Monat für März.

Außerdem können Sie die Parameter für die Angabe der Uhrzeit einzeln oder auch alle weglassen. Diese Werte werden dann automatisch auf 0 gesetzt.

```
var zeit3 = new Date(2022, 3, 13); // 13.04.2022 um 0:00 Uhr
```

Beachten Sie, dass die gesetzten Werte nicht geprüft werden, ob sie in einem gewissen Wertebereich liegen. Wenn Sie für einen Monat den Wert 12 setzen, wird das einfach der Januar des Folgejahres. Oder der Wert 32 für einen Tag im Januar wird als 1. Februar interpretiert. Das ist auf den ersten Blick ungewöhnlich, aber im Grunde sehr logisch und erlaubt einen kompakten, intelligenten Umgang mit Datumsberechnungen in JavaScript.

Methoden eines Date-Objekts

Nach der Erzeugung eines Date-Objekts ist das in ihm gespeicherte Datum über seine Methoden zugänglich. Folgende Methoden stehen dafür zur Verfügung.

Methode	Erläuterung	Mögliche Rückgabewerte
<code>getDate()</code>	Tag im Monat	1 bis 31
<code>getDay()</code>	Nummer des Wochentages	0 bis 6
<code>getFullYear()</code>	Vierstellige Jahreszahl	Jahresangabe
<code>getHours()</code>	Stunde	0 bis 23
<code>getMinutes()</code>	Minuten	0 bis 59
<code>getMonth()</code>	Nummer des Monats	0 bis 11
<code>getSeconds()</code>	Sekunden	0 bis 59
<code>getTime()</code>	Millisekunden seit dem 1.1.1970, 0:00 Uhr	0 bis ...
<code>getTimezoneOffset()</code>	Abstand zwischen Lokalzeit und GMT in Minuten	-720 bis +720
<code>getYear()</code>	Die Anzahl der Jahre seit dem Jahr 1900	0 bis ...

Über die folgenden Methoden können Sie das Datum in andere Formate umwandeln.

Methode	Erläuterung	Mögliche Rückgabewerte
<code>toGMTString()</code>	Aktuelle Greenwichzeit	Tue, 10 Mar 2020 16:07:57 GMT
<code>toLocaleString()</code>	Landestypische Zeitausgabe	Dienstag, 10. März 2020 16:07:57

! Es existieren ebenfalls die beiden Methoden `parse()` und `UTC()`. Mit den Methoden `parse(Datumszeichenkette)` und `UTC(Jahr, Monat, Tag, Std., Min., Sek.)` können Sie Datumsangaben in Millisekunden ausgeben (*UTC = Universal Time Coordinate*). Beide Methoden stehen bei Date-Objekten nicht zur Verfügung, denn es sind statische Methoden. Um diese Methoden zu verwenden, muss die Schreibweise `Date.parse(Parameter)` bzw. `Date.UTC(Parameter)` verwendet (also die Klasse bzw. der Prototype `Date` vorangestellt) werden.

! Die Methode `getYear()` wird in verschiedenen Browsern unterschiedlich implementiert. Per Konvention soll die Methode die Anzahl der Jahre seit dem Jahr 1900 liefern (eine internationale Vereinbarung). Die alten Versionen des Internet Explorers halten sich nicht an diese Vereinbarung. Deshalb verwendete man längere Zeit meist die Methode `getFullYear()`, die in allen Browsern das gleiche Ergebnis liefert. Mittlerweile ist das aber nicht mehr notwendig, denn die modernen Browser verhalten sich identisch.

Datumsangaben verändern

Mit den folgenden Methoden ändern Sie die Datumsangaben, die in einem `Date`-Objekt gespeichert sind. Diese Änderungen sind nur innerhalb des JavaScripts gültig, die Systemzeit Ihres Computers wird dadurch nicht verändert.

Methode	Erläuterung
<code>setFullYear(Jahr)</code>	Ändert das Jahr
<code>setMonth(Monat)</code>	Ändert den Monat
<code>setDate(Tag)</code>	Ändert den Tag des Monats
<code>setHours(Stunde)</code>	Ändert die Stunde
<code>setMinutes(Minute)</code>	Ändert die Minute
<code>setSeconds(Sekunde)</code>	Ändert die Sekunde
<code>setTime(Millisekunden)</code>	Umrechnung in eine Datumsangabe

Beispiel: *objekt_date.html*

Das folgende Beispiel enthält einige Methoden zum Auslesen und Manipulieren von Datum und Uhrzeit mithilfe von JavaScript. Weiterhin wird das Datum ermittelt und ausgegeben, das in 150 Tagen erreicht wird.

	<code><script type="text/javascript"></code>
①	<code>var zeit = new Date(); // Variable mit aktuellem Datum initialisieren</code>
②	<code>document.write("Datum, Uhrzeit: " + zeit.toLocaleString() + " Minuten
"); document.write("Jahr: " + zeit.getFullYear() + "; "); document.write("Monat: " + zeit.getMonth() + "; "); // Monate 0...11 document.write("Tag: " + zeit.getDate() + "; "); document.write("Wochentag: " + zeit.getDay() + "; "); document.write("Stunden: " + zeit.getHours() + "; "); document.write("Minuten: " + zeit.getMinutes() + "; "); document.write("Sekunden: " + zeit.getSeconds() + "
 "); document.write("Greenwichzeit: " + zeit.toGMTString() + "
"); document.write("Lokale Ausgabe: " + zeit.toLocaleString() + "
"); document.write("Abweichung: " + zeit.getTimezoneOffset() + " Minuten
");</code>

```
③ var jahr = zeit.getFullYear();  
  var monat = zeit.getMonth() + 1;  
  var tag = zeit.getDate();  
  
  document.write("Heute ist der " + tag + "." + monat + "." +  
                 jahr + "<br />");  
④ dann = zeit.getTime() + (150*24*60*60*1000); // 150 Tage  
    in Millisekunden  
⑤ zeit.setTime(dann);  
⑥ document.write("In 150 Tagen ist " + zeit.toLocaleString() +  
                 "<br />");  
  
</script>
```

- ① Die Variable `zeit` wird mit dem aktuellen Datum initialisiert und für die weiteren Berechnungen genutzt.
- ② Es werden über die Methoden die verschiedenen Datums- und Zeitwerte ausgegeben. Zu beachten ist, dass beim Monat der Wert immer um 1 niedriger ist.
- ③ Für die Berechnung eines neuen Datums werden die notwendigen Variablen zum Speichern des Jahres, Monats und Tags erstellt und mit den aktuellen Werten initialisiert. Beachten Sie, dass der Wert des Monats von 0 bis 11 läuft und deshalb um 1 erhöht werden muss.
- ④ Die 150 Tage werden in Millisekunden umgerechnet, da nur damit eine Datumsberechnung durchgeführt werden kann.
- ⑤ Das Datum wird über die Methode `setTime` um 150 Tage vordatiert.
- ⑥ Über die Methode `toLocaleString` wird das zukünftige Datum im landestypischen Format ausgegeben.

6.6 RegExp für reguläre Ausdrücke

Reguläre Ausdrücke, auch Regular Expressions genannt, erlauben es, Zeichenketten zu vergleichen. Sie können verwendet werden, um zu überprüfen, ob z. B. die Eingabe einer E-Mail-Adresse in einem Formularfeld eine gültige Adresse ist oder ob eine Telefonnummer die korrekte Formatierung hat. Reguläre Ausdrücke beschreiben ein bestimmtes Textmuster.

Im Internet finden Sie zahlreiche Informationen zu regulären Ausdrücken. Geben Sie beispielsweise in der Suchmaschine <https://www.google.de> die Anfrage "reguläre +Ausdrücke +Tutorial" ein.

JavaScript verwendet einmal ganz klassisch die Objekterzeugung über einen Konstruktor mit `new RegExp`, um Objekte zu erzeugen, mit denen reguläre Ausdrücke verarbeitet werden können. Sie können damit auch Ergebnisse solcher Operationen überwachen. `RegExp`-Objekte speichern in ihren Eigenschaften Informationen über die zuletzt durchgeführte Operation mit einem regulären Ausdruck. Auch Methoden von `String`-Objekten wie `match()`, `replace()` und `search()` verwenden implizit reguläre Ausdrücke.

Auch literal kann man einen regulären Ausdruck erzeugen, was etwas weiter unten noch besprochen wird.

Aufbau eines regulären Ausdrucks

Reguläre Ausdrücke sind Muster (Pattern), die auf eine Zeichenkette angewendet werden. Ein gewöhnliches Muster besteht aus drei Teilen: dem Pattern, den Flags und den Begrenzern `/`.

```
/Pattern/Flag
```

Die Schrägstriche `/` sind sogenannte Delimiter (Begrenzer), die den Anfang und das Ende des Musters kennzeichnen. Ein Delimiter darf jedes beliebige nicht alphanumerische Zeichen außer dem Zeichen `\` sein. Das Pattern ist der Hauptteil, welches das Suchmuster beinhaltet. Der letzte Teil besteht aus den Flags, die das Suchmuster beeinflussen.

Instanzen aus regulären Ausdrücken erzeugen

```
var suche1 = /sel/
var suche2 = new RegExp("sel")
```

Der einfachste Weg ist es, einer Variablen einen regulären Ausdruck zuzuweisen.

- ✓ Die erste Instanziierung verwenden Sie, wenn das Suchmuster nicht verändert werden muss. Das ist eine literale Erzeugung.
- ✓ Die zweite Variante verwendet den Konstruktor. Diese ist flexibler und Sie sollten sie anwenden, wenn das Suchmuster innerhalb des Skripts geändert wird.

In beiden Fällen wird die erste Fundstelle gesucht, die die Zeichenkette `sel` enthält.

Flags

Einem Ausdruck können Sie die beiden Flags `g` und `i` anfügen:

- ✓ `g` bedeutet, dass die Suche nach dem Ausdruck global erfolgen soll. Nach dem ersten Treffer wird die Suche fortgesetzt, sodass mehrere Treffer gefunden werden können.
- ✓ `i` lässt bei der Suche die Groß- und Kleinschreibung unberücksichtigt.

```
var suche = /sel/gi
```

Metazeichen

Für die Definition des Musters können Sie Metazeichen verwenden, die bestimmte Sonderfälle markieren. Folgende Metazeichen werden von JavaScript interpretiert:

Metazeichen	Findet ...	Beispiel
<code>\b</code>	eine Wortgrenze	<code>/\bthe/</code> findet " <u>th</u> ematisch", " <u>th</u> eatralisch". <code>/the\b/</code> findet "Agat <u>h</u> e", "Mat <u>h</u> e". <code>/\bthe\b/</code> findet nur " <u>th</u> e".
<code>\B</code>	eine Nicht-Wortgrenze, also nicht am Beginn oder Ende eines Wortes	<code>/\Ber\B/i</code> findet "W <u>e</u> rt", aber nicht "Erde" oder "Heer".

Metazeichen	Findet ...	Beispiel
\d	eine Ziffer von 0 bis 9	/\d\d/ findet eine zweistellige Ziffer wie "42".
\D	eine Nicht-Ziffer	/\D\D/ findet alles außer Zahlen.
\s	ein Leerzeichen, einen Tabulator, einen Zeilenumbruch	/E-Mail\sAdresse/ findet "E-Mail Adresse".
\S	ein Nicht-Leerzeichen	/E-Mail\SAdresse/ findet "E-Mail-Adresse".
\w	einen Buchstaben, eine Ziffer oder einen Unterstrich	/\w1\w/ findet "A1A", "_12", "Z1_".
\W	keinen Buchstaben, keine Ziffer und keinen Unterstrich	/\W1\W/ findet " 1%", aber nicht "11%".
.	jedes Zeichen außer einem Zeilenumbruch	/ . . / findet zwei zusammenhängende Zeichen, wie z. B. "Z3".
^	den Beginn einer Zeichenkette	/^Frau/ findet " <u>Frau</u> Klößer...", aber nicht "Ich bin Frau Klößer."
\$	das Ende einer Zeichenkette	/sie\.\$/ findet "Alle schätzten <u>sie</u> ."
[...]	irgendein Zeichen, das in der Klammer aufgelistet ist	/W[eo a]rt/ findet "Wert", "Wort", "Wart", aber nicht "Wirt".
[^...]	keines der in Klammern angegebenen Zeichen	/W[^ au]rt/ findet "Wirt", "Wert", "Wort", aber nicht "Wart" und "Wurt".
(. . .)	eine Zeichenkette und legt sie in der Variablen \$1 ab	/(\d{1,}, \d{2,})€./ findet "Es kostet 49,90€." und legt "49,90€" in \$1 ab.

Mit den nachfolgenden Metazeichen geben Sie die Häufigkeit des Suchbegriffs an.

Metazeichen	Findet ...	Beispiel
+	ein- oder mehrmals	/\d+/ findet "3", "5678".
?	kein- oder einmal	/\d?/ findet "", "3" oder "9", aber nicht "11".
*	kein-, ein- oder mehrmals	/\d*/ findet "" und alle zusammenhängenden Zahlen.
{n}	genau n-mal	/\d{3}/ findet dreistellige Zahlen, wie "345".
{n, }	n- oder mehrmals	/\d{2,}/ findet zwei- und mehrstellige Zahlen.
{n, m}	mindestens n-, maximal m-mal	/\d{3, 5}/ findet die drei-, vier- und fünfstelligen Zahlen.

Wollen Sie nach den Zeichen `.` `[]*` `{}` `()` `^` oder `$` suchen, müssen diese mit einem Backslash `\` eingeleitet werden, da sie ansonsten als Metazeichen interpretiert werden. Das gilt auch für den Backslash selbst. Nach ihm wird mit einem doppelten Backslash gesucht `\\`, z. B.:

```
var suche = /\D:\\Daten\\Grafiken\\Apfel\\.GIF/i;
```

Dieses Suchmuster passt beispielweise auf die Zeichenkette `D:\DATEN\Grafiken\apfel.gif`, da die Groß- und Kleinschreibung nicht beachtet wird.

Methoden

Damit Sie eine Suche durchführen können, verwenden Sie die Methoden des regulären Ausdrucks.

Methode	Erklärung
<code>exec(Zeichenkette)</code>	Über diese Methode führen Sie eine Suche nach dem angegebenen Suchmuster in der Zeichenkette durch. Zurückgeliefert wird ein Array, das die gefundenen Stellen über die Eigenschaften <code>index</code> und <code>input</code> beschreibt. Die Eigenschaft <code>index</code> enthält die Position des Buchstabens, an dem das gefundene Muster beginnt. Die Eigenschaft <code>input</code> enthält die durchsuchte Zeichenkette. Der erste Feldinhalt enthält den Teilstring, der das Muster erfüllt hat. Die folgenden Felder enthalten die weiteren Suchergebnisse. Wird nichts gefunden, liefert <code>exec</code> den Wert <code>null</code> zurück.
<code>test(Zeichenkette)</code>	Es wird ein boolescher Wert <code>true</code> oder <code>false</code> zurückgeliefert, je nachdem, ob die Suche mit dem vorgegebenen Suchmuster erfolgreich war oder nicht.

Beispiel: [objekt_regexp.html](#)

Es wird eine vorgegebene Zeichenkette nach dem Auftreten bestimmter Suchmuster verarbeitet.

	<code><script type="text/javascript"></code>
①	<code>var text = "Frau Klößler hat Bücher im Wert von 49,90€ bestellt und muss diese innerhalb der nächsten 14 Tage bezahlen.";</code>
	<code>document.write("Vorgabewert:
" + Text);</code>
②	<code>var suche = new RegExp(/\d\d/);</code>
	<code>document.write("<hr />Suche nach der ersten zweistelligen Zahl
");</code>
③	<code>document.write(suche.test(Text) + "
");</code>
④	<code>document.write(suche.exec(Text));</code>
⑤	<code>var suche = new RegExp(/\d{1,}\sTage/);</code>
	<code>document.write("<hr />Suche nach dem Wort "Tage" mit vorangesetzter Zahl
");</code>
	<code>document.write(suche.test(Text) + "
");</code>
	<code>document.write(suche.exec(Text));</code>

```

⑥ var suche = new RegExp(/[MWG]ut/);
document.write("<hr />Suche nach \"Mut\", \"Wut\" oder  

    \"Gut\"<br />");
document.write(suche.test(Text) + "<br />");
document.write(suche.exec(Text));

⑦ var suche = new RegExp(/\d{1,}, \d{2}€/);
document.write("<hr />Suche nach einer Preisangabe mit zwei  

    Kommastellen<br />");
document.write(suche.test(Text) + "<br />");
document.write(suche.exec(Text));
</script>

```

① Der Variable `Text` wird die Zeichenkette zugewiesen, die dann am Bildschirm ausgegeben wird.

② Der Variable `Suche` wird ein `RegExp`-Objekt mit dem Suchmuster übergeben, das mit der Zeichenkette verglichen werden soll. Die beiden Metazeichen `\d\d` stehen hierbei für zwei Ziffern.

③ Die Methode `test` des Objekts in der Variable `Suche` wird aufgerufen. Als Parameter erhält sie die zu untersuchende Zeichenkette. Der Rückgabewert wird am Bildschirm ausgegeben. Er kann die Werte `true` oder `false` besitzen.

④ Über die Methode `exec` wird ein Feld mit den Suchergebnissen zurückgeliefert. In diesem Fall wird als Ergebnis die Zahl 49 ausgegeben.

⑤ Über die Angabe von `/\d{1,}, \d{2}€/` wird nach einer Ziffer `\d` gesucht, die mindestens einmal vorkommen muss `{1,}`. Ihr muss die Zeichenfolge `Tage` mit einem vorangestellten Leerzeichen `\s` folgen. Mit der Methode `test` wird geprüft, ob die Suche erfolgreich war, und mit `exec` wird der gefundene Text ausgegeben.

⑥ Hier soll nach den Wörtern `Mut`, `Wut` oder `Gut` gesucht werden. Die Methode `test` wird in diesem Fall den Wert `false` zurückliefern, da keines der Wörter in der Zeichenkette `Text` vorkommt. Die Methode `exec` liefert dadurch den Wert `null` zurück.

⑦ Eine Suche nach einer Preisangabe mit dem Euro-Zeichen `€` gestalten Sie über den regulären Ausdruck `/\d{1,}, \d{2}€/`. Es wird nach einer Ziffer gesucht, die mindestens einmal vorhanden sein muss `\d{1,}`. Ihr müssen ein Komma und eine zweistellige Ziffer `\d{2}` folgen. Danach muss das Währungszeichen `€` stehen.

Methoden von `RegExp()`

Vorgabewert:

Frau Klöber hat Bücher im Wert von 49,90€ bestellt und muss diese innerhalb der nächsten 14 Tage bezahlen.

Suche nach der ersten zweistelligen Zahl

true
49

Suche nach dem Wort "Tage" mit vorangesetzter Zahl

true
14 Tage

Suche nach "Mut", "Wut" oder "Gut"

false
null

Suche nach einer Preisangabe mit zwei Kommastellen

true
49,90€

Ergebnisse regulärer Ausdrücke (*objekt_regexp.html*)

Die Instanz des regulären Ausdrucks besitzt weitere vier Eigenschaften, die bei der Verwendung der Flags gesetzt werden:

- ✓ **global** enthält `true` oder `false`, je nachdem, ob das Flag `g` gesetzt ist oder nicht.
- ✓ **ignoreCase** enthält `true` oder `false`, je nachdem, ob das Flag `i` gesetzt ist oder nicht.
- ✓ **lastIndex** enthält den Index des ersten Buchstabens, bei dem der nächste Suchdurchlauf des Musters beginnt. In einer neuen Suche ist der Wert 0. Ändern Sie diesen Wert, um die Suche an einer anderen Position zu beginnen.
- ✓ **source** enthält das Suchmuster als String ohne die beiden Schrägstriche.

Eigenschaften von **RegExp**

Jedes Mal, wenn Sie einen regulären Ausdruck anwenden, werden die Ergebnisse der Operation in den Eigenschaften des **RegExp**-Objekts abgelegt. Die Benennung der Eigenschaften erfolgt über die **\$**-Variablen oder über den entsprechenden ausgeschriebenen Namen.

Eigenschaft	Erläuterung
\$& oder lastMatch	Enthält den zuletzt gefundenen String
\$+ oder lastParen	Enthält die zuletzt gefundene Teilsuche
\$^ oder leftContext	Enthält den Teilstring links von der gefundenen Stelle
\$' oder rightContext	Enthält den Teilstring rechts von der gefundenen Stelle
\$* oder multiline	Enthält true , wenn die Suche über mehrere Zeilen stattgefunden hat
\$1, ..., \$9	Erlaubt den Zugriff auf die gespeicherten Teilsuchergebnisse 1 bis 9, die mit Klammern im regulären Ausdruck definiert worden sind

Die Zeichen **`** und **'** erhalten Sie durch die Tastenkombinationen **⌘** **⏏** und **⌘** **⏏**.

Reguläre Ausdrücke im **String**-Objekt

Das **String**-Objekt besitzt Methoden zum Durchsuchen und Verändern von Zeichenketten. Die Suchmuster lassen sich über reguläre Ausdrücke festlegen.



Bei der Angabe eines regulären Ausdrucks müssen Sie bei den Methoden des **String**-Objekts beachten, dass dieser im Gegensatz zum **RegExp**-Objekt als Parameter in Klammern angegeben wird.

Methode	Erklärung
match (Suchmuster)	Damit führen Sie eine Suche nach dem angegebenen Suchmuster in der Zeichenkette durch. Zurückgeliefert wird ein Array, das die gefundenen Stellen beschreibt. Wird nichts gefunden, liefert match den Wert null zurück.
search (Suchmuster)	Diese Methode liefert die Position zurück, an der das Suchmuster zutraf. Wurde nichts gefunden, wird der Wert -1 zurückgegeben.
replace (Suchmuster, Ersetzung)	Hiermit wird eine Suche nach dem Muster durchgeführt und der gefundene Teil der Zeichenkette wird entsprechend ersetzt.
split (Suchmuster)	Die Zeichenkette wird am Suchmuster aufgeteilt und die resultierenden Teile werden als Feld zurückgegeben.

Beispiel: *objekt_regexp_multi.html*

Über den regulären Ausdruck `/\d*\.\d{2}€/g` werden alle Zeichenfolgen gesucht, die den folgenden Aufbau haben: eine beliebige Anzahl Zahlen (`\d*`), einen Punkt (`\.`), zwei Zahlen (`\d{2}`) und das Eurozeichen. Die Zeichenketten werden in einem Feld zurückgegeben, das über die `for`-Anweisung durchlaufen wird.

```
<script type="text/javascript">
  var text = "Frau Klößler hat mehrere Buchbestellungen im Wert
              von 49.90€, " + "149.90€, 65.35€ und 400.00€
              aufgegeben.";
  document.write(text + "<br /><br />Die Bestellwerte im
                      Einzelnen:<br /><hr />");
  var suchmuster = new RegExp(/\d*\.\d{2}€/g);
  var bestellwerte = text.match(Suchmuster);
  for(var i = 0; i < bestellwerte.length; i++)
    document.write(bestellwerte[i] + "<br />");
</script>
```

6.7 Das Objekt **Image**

Für Grafiken, die Sie nachträglich mit JavaScript anzeigen möchten, können Sie mit Objekten vom Typ `Image` eigene Grafikobjekte erzeugen.

Objekterzeugung	<pre>bild = new Image(); bild.src = 'bild.jpg';</pre>
------------------------	---

Zum Erzeugen eines Objekts vergeben Sie einen Variablennamen, den Sie mit `new Image()` initialisieren. Dem erzeugten Objekt können Sie über die Eigenschaft `src` die entsprechende Grafikdatei zuweisen.

6.8 Arrays

Arrays haben eine große Bedeutung in der Programmierung, wobei in JavaScript die Bedeutung noch größer ist als in den meisten anderen Programmiersprachen. Ein Array (Feld) ist im Prinzip ein Speicherbereich, in dem Sie mehrere Werte speichern können. Der Zugriff auf die Werte erfolgt über einen Bezeichner (eine Variable, die auf das Array verweist) und einen Index (Schlüssel).



In JavaScript ist ein Array ein Objekt, aber **jedes** Objekt ist im Kern in JavaScript auch ein Array – nur eine Liste mit Schlüssel-Werte-Paaren (JSON) und das ist außergewöhnlich. Allerdings sind die verfügbaren Methoden zwischen JSON-Objekten und „normalen“ Arrays nicht identisch. Für die Struktur und den direkten Zugriff auf die enthaltenen Elemente ist das aber irrelevant.

Ein Array ist dann von Vorteil, wenn Sie eine Reihe von zusammengehörenden Informationen speichern wollen. Das könnten beispielsweise die Tage eines Monats sein. Sie können zwar für jeden Tag des Monats eine eigene Variable definieren, die dann die Nummer des Tags im Monat aufnimmt. Diese Methode wäre sehr umständlich. Durch die Verwendung eines Arrays genügt eine Variable und der Index. Mithilfe von Variable und Index können Sie auf mehrere Elemente zugreifen. Ein großer Vorteil von Arrays ist, dass Sie damit auch über Schleifen auf die Elemente zugreifen können.



Beachten Sie, dass durch die lose Typisierung in JavaScript auch unterschiedliche Datentypen in einem Array gespeichert werden können. Das ist in vielen anderen Sprachen nicht der Fall, die bei Array-Inhalten den gleichen Datentyp fordern.

Da Arrays in JavaScript auch als Objekte zu sehen sind, gibt es implementierte Methoden und Eigenschaften. Die für eine Iteration über das Array wichtige Eigenschaft ist z. B. `length`. Die Eigenschaft `length` enthält für **durchgängig numerisch indizierte** Arrays die Anzahl der Elemente im Array. Oder genauer: den letzten numerischen Index. Diese Information können Sie in einer Schleife über ein Array (oder auch die Anzahl der Elemente in einem Objekt) als Abbruchbedingung verwenden.

Ein Array erzeugen

Ein Array wird in JavaScript etwas anders erzeugt als eine normale Variable. Sie müssen neben dem Namen, über den das Array zugänglich ist, dem System auch kenntlich machen, dass eine ganze Sammlung an Werten vorliegt. Es gibt dazu verschiedene Wege in JavaScript.

Die Deklaration mit einem Konstruktor von **Array**

Zum Erzeugen von Arrays kann man den Array-Konstruktor verwenden. Beispiel:

```
var feldname = new Array(); // leeres Feld
feldname[1] = 100;
feldname[4] = "Test";      // die Feldlänge ist 5, da der Index
                           // bei 0 beginnt
```

Man kann in dem Konstruktor auch die Anzahl der Elemente angeben, die das Array enthalten soll. Das ist aber in JavaScript sowohl unnötig als auch wirkungslos. Zwar würde mit einem numerischen Parameter tatsächlich die Größe des Arrays vorgegeben, aber Sie können jederzeit einen größeren Index verwenden. Deshalb gibt man in der Praxis eigentlich nie eine Größe des Arrays bei der Deklaration an, da Array-Elemente sowieso „on the fly“ erzeugt werden, wenn sie benötigt werden.

```
var feldname = new Array(5); // Deklaration eines Feldes mit
                             // 5 Elementen
feldname[1] = 100;
feldname[42] = "Test"; // die originale Feldlänge wird nicht
                       // berücksichtigt und
                       // der Wert von length ist jetzt 43
```

Beachten Sie, dass die Eigenschaft `length` in vielen Situationen **nicht** die Größe eines Arrays bzw. die Anzahl der enthaltenen Elemente widerspiegelt, sondern nur den Wert des höchsten numerischen Index. Daraus kann man zwar in einigen Situationen auf die Anzahl der Elemente in einem Array schließen (wenn der Index, wie schon erwähnt, durchgängig hochgezählt wurde), aber man kann sich nicht darauf verlassen.

Deklaration mit einem Array-Literal

Arrays können Sie in JavaScript auch unter Verwendung von einem sogenannten **Array-Literal** erzeugen und damit auf die Notation von einem expliziten Konstruktor verzichten. Sie verwenden dann auch **nicht** das Schlüsselwort `new`.

In diesem Fall spricht man von einer deklarativen bzw. literalen Erzeugung.

Man muss dem System kenntlich machen, dass eine Deklaration nicht einfach eine Variable neu anlegt, sondern ein Datenfeld. Dazu verwendet man in JavaScript (und vielen anderen Sprachen) beim zugewiesenen Literal eckige Klammern – das kennzeichnet dann ein Array-Literal. Sie weisen bloß einer Variablen bei der Deklaration in eckigen Klammern eine Reihe an Werten zu. Die eckigen Klammern können auch leer bleiben.

```
// dem Feld werden vier Namen zugewiesen
var name = ["Müller", "Schmidt", "Schulze", "Lehmann"];
// oder
var name = new Array(4);
name[0] = "Müller";
name[1] = "Schmidt";
name[2] = "Schulze";
name[3] = "Lehmann";
```

Der Zugriff auf ein einzelnes Element des Arrays erfolgt über die Angabe eines Index, den Sie in eckigen Klammern angeben müssen. So erhalten Sie mit `name[0]` das erste Element des Feldes `name`, mit `name[1]` das zweite Element usw.

Die Anzahl der zugewiesenen Werte legt im ersten Fall indirekt die anfängliche Größe des Datenfeldes fest. Das Array wird dabei automatisch numerisch indiziert. Hier haben wir genau die oben angedeutete Situation – die Anzahl der zugewiesenen Werte wird durch die Eigenschaft `length` repräsentiert, sofern man ohne Lücken im Index arbeitet.

Die Array- bzw. Objekt-Notation – JSON

Objekte und Arrays werden in JavaScript identisch als Listen mit einer Auflistung über Schlüssel-Werte-Paare (sogenannte Hash-Listen) verwaltet. Das führt dazu, dass es auch eine deklarative Erzeugung von Arrays mit einem sogenannten Objekt-Literal gibt. Dazu notiert man in geschweiften Klammern eine Komma-getrennte Liste aus Schlüssel-Werte-Paaren als Literal, das einer Variablen zugewiesen wird. Das erlaubt sprechende Indizes – also Textindizes. Die Verwendung von Textindizes ist auch als **assoziierte Arrays** bekannt. Auch damit erzeugen Sie ein Array.

Beispiel

```
var kapitaen = {  
  name: "Kirk",  
  vorname: "James T"  
};
```

Für JSON gibt es einen speziellen Typ, der sehr interessante Methoden bereitstellt.

Die Klasse `JSON` stellt mit den Methoden `parse()` und `stringify()` zwei Möglichkeiten bereit, ein JSON-Objekt aus einem String zu erzeugen bzw. in einen String zu überführen (stringifizieren).

Sie können damit einen Objektzustand zur Laufzeit reproduzieren bzw. speichern (serialisieren). Wenn man ein Objekt auf diese Weise serialisiert, macht man es persistent (dauerhaft).

Beispiel: *serialisieren.js*

```
"use strict";  
var datum = {  
  jetzt: new Date()  
};  
console.log(datum);  
var datumString = JSON.stringify(datum);  
console.log(datumString);  
var jetztRepro = JSON.parse(datumString);  
console.log(jetztRepro);
```

- ✓ In dem JSON-Objekt `datum` wird eine Eigenschaft `jetzt` angelegt und mit dem aktuellen Systemdatum belegt.
- ✓ Die Ausgabe zeigt die JSON-Struktur des Objekts `datum` mit dem aktuellen Wert der Eigenschaft `jetzt`.
- ✓ Das ganze Objekt wird serialisiert mit `JSON.stringify(datum)`.
- ✓ Die Ausgabe zeigt die String-Struktur.
- ✓ Aus dem String mit der serialisierten JSON-Struktur wird mit `JSON.parse(datumString)` ein neues Objekt reproduziert, das genau die Werte enthält, die bei der Stringifizierung gültig waren. Beachten Sie, dass dies auch eine beliebige Zeit später erfolgen kann. Auch kann man den stringifizierte String über eine Datei oder das Netzwerk weitergeben, was große Möglichkeiten eröffnet.
- ✓ Das reproduzierte Objekt wird ausgegeben.

6.9 Zugriff auf Array-Elemente und der Index

Auf Array-Elemente greifen Sie in der Regel über den Bezeichner und den Index in eckigen Klammern zu. Einem Element eines Arrays weisen Sie einen Wert zu, indem Sie den Namen des Arrays gefolgt vom Index in eckigen Klammern angeben und dann einen Wert zuweisen. Wenn Sie den Wert auslesen wollen, der in einem Array-Element gespeichert ist, gehen Sie analog vor.

Beachten Sie, dass der Index bei Arrays mit numerischem Index mit 0 beginnt. Man nennt sie **nullindiziert**.

```
x[0] = 1;      Wert zuweisen
document.write(x[0]);  Wert auslesen
```

In der ersten Zeile weisen Sie einem Array-Element (`x[0]`) einen Wert zu, und in der zweiten Zeile lesen Sie den Wert eines Elements (das 1. Element) aus.

Bei einem assoziierten Array verwenden Sie als Index den Datentyp `String` als Schlüssel. Dementsprechend muss der Index in Hochkommata notiert werden. Der Zugriff auf ein Element des Datenfeldes erfolgt über einen Textindex.

Beispiel

```
var index = "Name"; // Anlegen einer Variablen, die den Index
                    // bilden soll
document.write (x[index]); // Ausgabe eines Elements des Arrays
                        // über die Indexvariable
document.write(x["vname"]); // Ausgabe eines Elements des Arrays
                        // über einen String als Index
```



In JavaScript gibt es grundsätzlich nur Textindizes. Wenn Sie numerische Indizes angeben oder diese automatisch generiert werden, wird der numerische Schlüssel im Hintergrund zum String umgewandelt. Deshalb sind auch scheinbar unlogische Array-Strukturen mit numerischen Indizes und Textindizes in einem Array gar nicht unlogisch.

```
var x = new Array ();
x[0] = 1;
x["name"] = "Meier";...
```

Methoden und Eigenschaften

Ein Array besitzt nur die Eigenschaft `length`, womit der Wert des höchsten numerischen Index in einem Array verfügbar ist. In dem Fall eines durchgängig numerisch indizierten Arrays ist dies die Anzahl der gespeicherten oder bei der Instanziierung definierten Werte.

Nach der Instanziierung `var feldname = new Array(4);` liefert `feldname.length` beispielsweise den Wert 4. Das letzte Element erhalten Sie in dem Fall direkt über die Angabe der Feldlänge, z. B. `feldname[feldname.length - 1]`.

Allerdings steht bei assoziierten Arrays die Anzahl der Elemente nicht über diese Eigenschaft zur Verfügung und auch dann nicht, wenn bei der Liste der „numerischen“ Schlüssel Lücken auftreten.

```
var feldname = new Array(); // Deklaration eines leeren Feldes -
                           length 0
feldname[99] = 100;         // length hat den Wert 100
```

In dem Beispiel hat die Eigenschaft `length` nach der Zuweisung den Wert 100. Die Anzahl der Elemente ist aber 2. Lassen Sie sich jetzt nicht irreleiten und glauben Sie nicht, dass es doch beispielsweise `feldname[40]` geben müsste. Das ist nicht der Fall, denn wie oben bereits erwähnt, gibt es gar keine numerischen Indizes in JavaScript. Die Suggestion, dass es die numerischen Indizes zwischen 0 und 99 in dem Beispiel geben müsste, ist verführerisch, aber ebenso falsch wie die Behauptung, es müsse doch `feldname["vorname"]` oder `feldname["irgendwas"]` geben. Wenn Sie diese Indizes verwenden, liefert JavaScript `undefined`. Um z. B. innerhalb einer Schleife auf alle Bezeichner und Werte eines assoziierten Arrays zugreifen zu können, können Sie die `for-in`-Anweisung verwenden.

Ein Objekt vom Typ `Array()` besitzt neben der Eigenschaft `length` auch mehrere Methoden, um auf die Feldelemente zuzugreifen.

Methode	Erläuterung
<code>concat(Array1, Array2, ...)</code>	Mit <code>concat()</code> können Sie dem ersten Array die Inhalte weiterer Arrays hinzufügen. Die neuen Inhalte werden hinten angehängt.
<code>join(Zeichen)</code>	Die einzelnen Elemente des Arrays werden durch das angegebene Zeichen miteinander verbunden und als Zeichenkette in einer Variablen abgelegt. Wird kein Verknüpfungszeichen angegeben, wird ein Komma verwendet. <pre>a = new Array("Wind", "Regen", "Feuer") Var1 = a.join() // Var1 = "Wind,Regen,Feuer" Var2 = a.join(" + ") // Var2 = "Wind + Regen + Feuer"</pre>
<code>pop()</code>	Liest das letzte Element im Array aus und löscht es danach (nur für indizierte Arrays).
<code>push(Wert1, Wert2, ...)</code>	An das Ende des Arrays werden weitere Elemente angefügt. Zurückgegeben wird die neue Länge des Feldes.
<code>reverse()</code>	Kehrt die Reihenfolge der Elemente innerhalb des Arrays um. Das letzte Element wird zum ersten Element usw.
<code>shift()</code>	Das erste Element wird ausgelesen und gelöscht.
<code>slice(Start, Ende)</code>	Diese Methode extrahiert einen Teil des Arrays, der mit dem Index <code>Start</code> beginnt und mit dem Index <code>Ende</code> endet. Ein negativer Endwert bedeutet, dass vom Ende des Arrays angefangen wird, zu zählen. Das Ergebnis ist ein neues Array.

Methode	Erläuterung
<code>splice(Start, Ende, Wert1, Wert2, ...)</code>	Die Elemente vom Index <code>Start</code> bis <code>Ende</code> werden durch die neuen Werte <code>Wert1, Wert2, ...</code> ersetzt. Als Rückgabewert erhalten Sie die Elemente von <code>Start</code> bis <code>Ende</code> . Werden keine Werte angegeben, werden die Elemente von <code>Start</code> bis <code>Ende</code> gelöscht.
<code>sort()</code>	Der Inhalt des Feldes wird alphabetisch sortiert. Enthalten die Elemente Zahlen, werden diese in Zeichenketten umgewandelt, sodass beispielsweise die Zahl 19 vor der Zahl 9 einsortiert wird. <code>name(1, 9, 19, 'Maria', 'Eva')</code> wird durch <code>name.sort();</code> sortiert zu <code>name(1, 19, 9, 'Eva', 'Maria')</code> .

Zusätzlich können Sie in der Methode `sort()` eine Funktion hinterlegen, mit der Sie die Sortierungslogik (d. h. den Vergleich zweier Elemente) selbst durchführen können.

Beispiel

```
function sortiere(a, b) {
    return a - b;
}
ArrayName.sort(sortiere);
```

Die Reihenfolge der Elemente wird gemäß der Sortierfunktion verändert. Die Sortierfunktion muss zwei Parameter entgegennehmen und eine Ganzzahl zurückliefern. Die Elemente des Arrays werden über diese Funktion verglichen und je nach Vorzeichen der Ganzzahl sortiert.

Ausgabe der Eigenschaften und Methoden

```
var name = new Array("Müller", "Schmidt", "Schulze", "Lehmann");
var vorname = new Array("Peter", "Frank", "Thomas", "Ingo");

name.concat(vorname): Müller,Schmidt,Schulze,Lehmann,Peter,Frank,Thomas,Ingo
name.join(" + "): Müller + Schmidt + Schulze + Lehmann
name.pop(): Lehmann
name: Müller,Schmidt,Schulze
name.push("Schröder"): 4
name: Müller,Schmidt,Schulze,Schröder
name.reverse(): Schröder,Schulze,Schmidt,Müller
name.slice(1,4): Schulze,Schmidt,Müller
name: Schröder,Schulze,Schmidt,Müller
vorname.shift(): Peter
vorname: Frank,Thomas,Ingo
vorname.splice(1,3,"Leon","Lucas","Julian"): Thomas,Ingo
vorname: Frank,Leon,Lucas,Julian
vorname.sort(): Frank,Julian,Leon,Lucas
vorname: Frank,Julian,Leon,Lucas
```

Methoden des Objekts Array

- ✓ `<0`: Der zweite Wert ist kleiner als der erste.
- ✓ `=0`: Beide Werte sind gleich.
- ✓ `>0`: Der erste Wert ist kleiner als der zweite.

Wird keine Funktion angegeben, werden die Elemente nach ihren ASCII-Codes sortiert (auch Zahlen).

In den JavaScript-Versionen nach 1.5 sind noch einige weitere Array-Methoden hinzugekommen, die aber auf absehbare Zeit in der Praxis nicht angewendet werden können.

6.10 Übungen

Übung 1: Suchen und Ersetzen

Übungsdatei: --

Ergebnisdateien: *kap06/uebung1-1.html*,
kap06/uebung1-2.html

1. Schreiben Sie eine Funktion, die alle Vorkommen der Buchstaben ä, ö, ü in die Zeichenketten ae, oe und ue umsetzt und die neue Zeichenkette als Ergebnis zurückliefert. Verwenden Sie dazu die Methoden des Objekts `String`.
2. Lösen Sie die Aufgabe 1 durch die Verwendung von regulären Ausdrücken.

Übung 2: Zufallswerte verwenden

Übungsdatei: *kap06/SpruchDesTages.js*

Ergebnisdateien: *kap06/uebung2.html*

1. Entwickeln Sie eine Funktion, die den Spruch des Tages zurückliefert. Die einzelnen Texte werden in einem Feld verwaltet, das über eine externe Datei eingebunden wird. Der Spruch des Tages wird über eine Zufallsfunktion ausgewählt.

Übung 3: Aktuelle Uhrzeit ausgeben

Übungsdatei: --

Ergebnisdateien: *kap06/uebung3.html*

1. Geben Sie die aktuelle Uhrzeit in der Titelleiste des Browsers aus. Durch das Zuweisen einer Zeichenkette an die Eigenschaft `document.title` können Sie in die Titelleiste schreiben. Über die folgenden Anweisungen rufen Sie die Funktion `zeigeZeit` jede Sekunde einmal auf. Auf diese Weise können Sie die Zeitanzeige in der Titelleiste aktualisieren.

```
function zeigeZeit()  
{  
    //... die Uhrzeit wird ermittelt und in der Statuszeile  
        ausgegeben  
    setTimeout("zeigeZeit()", 1000);  
}  
zeigeZeit();
```

7

Das DOM-Konzept

7.1 Objekte und Hierarchie des DOM

Was ist DOM?

In JavaScript werden Ihnen neben den vordefinierten Objekten vom Typ `String` und `Date` weitere Objekte zur Verfügung gestellt, über die Sie beispielsweise auf ein Element eines Formulars zugreifen und dieses ändern können, sofern das JavaScript in eine Webseite eingebunden ist. Diese Objekte gehören nämlich explizit nicht zu JavaScript, sondern zu der Objektschnittstelle **DOM** (**D**ocument **O**bject **M**odel). Diese Objektschnittstelle wird vom W3C (World Wide Web Consortium) standardisiert (<https://www.w3c.org>) und ist aus beliebigen Sprachen zugänglich, wenn diese im Rahmen eines Browsers verfügbar sind und von der Webseite aus referenziert werden (beispielsweise Java, JScript, ECMAScript, JavaScript).

Diese Objekte des DOM sind in mehreren Hierarchiestrukturen geordnet. Die hierarchiehöchsten Objekte sind ...

- ✓ das Fensterobjekt `window`,
- ✓ das Navigatorobjekt `navigator` und
- ✓ das Bildschirmobjekt `screen`.

Entwicklung von DOM

Das DOM-Konzept wurde ursprünglich von Netscape entwickelt, aber bereits 1997 vom W3C übernommen und standardisiert. Es gibt verschiedene Level (Versionen), die jedoch unterschiedlich weit in Browsern implementiert sind. Moderne Browser unterstützen allerdings DOM 3 und das DOM-Konzept, das mit HTML5 verbunden ist.

Mit der Entwicklung von XML (**e**Xtensible **M**arkup **L**anguage) hat das W3-Konsortium das ursprüngliche DOM-Konzept modifiziert und universell auf baumartig strukturierte Dokumente anwendbar gemacht. Dieses Objektmodell bildet die Struktur eines baumartig strukturierten Dokuments im Arbeitsspeicher ab und erlaubt den Zugriff auf die Elementstruktur des Dokuments über eine API-Schnittstelle (API = **A**pplication **P**rogramming **I**nterface).

Über bestimmte Methoden und Eigenschaften können somit die einzelnen Elemente, Attribute und Inhalte angesprochen, verarbeitet oder verändert werden. Sie können die Elemente, die Attribute und den Inhalt jeder HTML-Datei auslesen und löschen, aber auch neue Elemente erstellen und in das Dokument einfügen. Dies ist möglich, da das Document Object Model den Zugriff auf jedes Element eines HTML-Dokuments erlaubt. Somit haben Sie die Möglichkeit, Dokumente dynamisch zu generieren. Dazu wird die geladene Webseite im DOM in einzelnen identifizierbare Elemente zerlegt und nach HTML-Tags, Attributen, Typen und der Position innerhalb der Webseite indiziert. Über diese Baumstruktur können Sie mit JavaScript Elemente ansprechen und deren Werte ändern.

Die hierarchiehöchsten Objekte und ihre Unterobjekte

Der Inhalt eines Fensters und somit das HTML-Dokument wird durch das `document`-Objekt repräsentiert. Dieses Objekt ist dem `window`-Objekt untergeordnet. Das HTML-Dokument enthält Elemente wie z. B. Grafiken, Verweise und Formulare.



Für diese Elemente gibt es teils weitere Unterobjekte, beispielsweise das Objekt `forms` für Formulare. Um auf die Elemente des Formulars (Eingabefelder, Auswahllisten oder Schaltflächen) zuzugreifen, existiert das Unterobjekt `elements`, mit dem Sie einzelne Felder und andere Elemente innerhalb eines Formulars ansprechen können.

Für das `navigator`-Objekt und das `screen`-Objekt existieren Hierarchien, die unabhängig von der Hierarchie des `window`-Objekts sind. Diese Objekte geben Auskunft über den Browser, in dem das Skript läuft, bzw. über die Einstellungen des Computerbildschirms eines Besuchers.

Aufgrund der großen Menge an Objekten im DOM und deren zahlreicher Eigenschaften und Methoden werden Ihnen in diesem Kapitel wie auch in den folgenden Kapiteln anhand von ausgesuchten Beispielen die wichtigsten Objekte und deren Eigenschaften und Methoden erläutert.



Ergänzende Lerninhalte: *Übersicht über die Objektmodelle.pdf*
JavaScript-Techniken für ältere Skripte.pdf

Hier finden Sie z. B. weitere Ausführungen zu DOM-Techniken, die man früher oft verwendet hat, die aber mittlerweile an Bedeutung verloren haben oder nicht mehr zu empfehlen sind. Wenn Sie allerdings alte JavaScripts sehen, kann es sein, dass ein Verständnis dieser alten Vorgehensweisen nützlich ist.

7.2 Das Objekt `window`

Das Objekt `window` (Fenster) steht in der Objekthierarchie des DOM an oberster Stelle. Alle anderen Objekte, die das im Fenster geladene Dokument sowie die Bestandteile des Browserfensters beschreiben, sind Eigenschaften des `window`-Objekts. Es erlaubt Ihnen somit Abfragen zu einzelnen Dokumentfenstern und ermöglicht Ihnen, neue Fenster zu öffnen sowie deren Eigenschaften (Name, Höhe, Breite usw.) festzulegen. Auch das Schließen von Fenstern ist hiermit möglich.

- ✓ Ist das Browserfenster in sogenannte Frames unterteilt, befinden sich die Objekte der HTML-Dokumente in einer eigenen Hierarchie unterhalb eines `frames`-Objekts. Frames sind allerdings veraltet und werden – bis auf `IFrames` – so gut wie gar nicht mehr eingesetzt.
- ✓ Das `document`-Objekt enthält alle wesentlichen HTML-Elemente wie Links, Anker, Bilder und Formulare des betreffenden HTML-Dokuments.
- ✓ Über das `event`-Objekt können Sie Ereignisse im Browser abfragen (Klicken, Auswählen usw.) und darauf reagieren.
- ✓ Das `history`-Objekt enthält die Liste der zuletzt geladenen URLs.
- ✓ Das `location`-Objekt enthält Informationen über die URL des aktuellen Dokuments.

Zusätzlich existieren weitere untergeordnete Objekte mit Eigenschaften, die in den folgenden Abschnitten im Einzelnen besprochen werden.

Methoden des `window`-Objekts

Das `window`-Objekt besitzt eine ganze Reihe an Methoden, von denen in der folgenden Tabelle die wichtigsten angegeben werden sollen.

Methoden	Bedeutung
<code>alert()</code>	Ein kleines Meldungsfenster. Es wurde früher oft verwendet, wird aber mittlerweile in Webseiten nicht mehr eingesetzt. Programmierer nutzen es jedoch immer noch gerne zur Fehlersuche ohne Debugger.
<code>blur()</code>	Diese Methode deaktiviert das betreffende Browserfenster. Es wird in den Hintergrund verschoben.
<code>clearTimeout(TimeoutID)</code>	Der Zeitgeber, der mit <code>TimeoutID = setTimeout()</code> gestartet wurde, wird angehalten.
<code>close()</code>	Damit schließen Sie ein Browserfenster.
<code>confirm()</code>	Es wird ein Bestätigungsfenster angezeigt.
<code>focus()</code>	Hiermit aktivieren Sie das Browserfenster. Es wird in den Vordergrund gebracht.
<code>open()</code>	Mit der Methode <code>open()</code> öffnen Sie ein neues Browserfenster.
<code>prompt()</code>	Zur Eingabe eines Textes wird ein Eingabefenster angezeigt. Es wurde früher oft verwendet, wird aber mittlerweile in der Praxis nicht mehr eingesetzt. Eingaben werden nur noch über Formulare vorgenommen.
<code>setTimeout()</code>	Nach der in Millisekunden angegebenen Zeit wird ein festgelegter JavaScript-Ausdruck ausgeführt. Als Rückgabewert der Methode wird eine <code>TimeoutID</code> geliefert, mit der sich der laufende Zeitgeber referenzieren lässt.

Eigenschaften des **window**-Objekts

Die Eigenschaften des `window`-Objekts besitzen in der Praxis so gut wie keine Bedeutung mehr, sofern es nicht die Unterobjekte sind, die oben angesprochen wurden.

Standardmäßig müssen Sie zum Aufruf einer Methode oder Eigenschaft eines Objekts den Objektnamen voranstellen, z. B. `window.alert()` oder `window.document`. Im Falle des `window`-Objekts können Sie den Namen `window` meist weglassen. Dieser wird automatisch von JavaScript vorangesetzt.

Fenster öffnen und schließen

Beim Browsen im Internet wurden in der Vergangenheit bei vielen Webseiten automatisch weitere Fenster, meist Werbeeinblendungen, geöffnet (sogenannte **Pop-ups**). Dazu wurde meist die Methode `open()` verwendet.

```
window.open("URL", "Fenstername", [Optionen])
```

- ✓ Die angegebene URL wird in einem Fenster mit den angegebenen Fenstername geöffnet. Geben Sie eine leere Zeichenkette an, wird eine leere Seite angezeigt.
- ✓ Damit Sie die Inhalte von geöffneten Fenstern ansprechen können, müssen Sie dem Fenster einen eindeutigen Namen geben.
- ✓ Über weitere Optionen können Sie die Anzeige des Fensters konfigurieren.
- ✓ Als Rückgabewert wird eine Referenz auf das `window`-Objekt des erstellten Fensters geliefert.

Durch den starken Missbrauch dieser Techniken hat sich der Einsatz von Pop-up-Blockern durchgesetzt, und moderne Browser verhindern in der Grundeinstellung zudem das Öffnen und Manipulieren von Fenstern durch Skripte. Die Anwendung der Methode `open()` und weiterer Methoden zur Manipulation von Fenstern ist daher in der Praxis so gut wie gar nicht mehr zu finden und grundsätzlich nicht mehr zu empfehlen.

Den Zeitgeber (Timer) verwenden

Mit der Methode `setTimeout()` haben Sie die Möglichkeit, eine Anweisung oder Funktion nach einer bestimmten Verzögerungszeit auszuführen. Dazu geben Sie innerhalb der Klammern den entsprechenden JavaScript-Code an. Als zweite Angabe folgt die Verzögerungszeit in Millisekunden.

```
window.setTimeout("Anweisung/Funktion/Funktionsreferenz",  
                  Millisekunden)
```


Beispiel: *objekt_window_settimeout.html*

Durch die Anweisung `setTimeout("zeitVorbei()", 5000)` wird nach einer Zeit von 5 Sekunden (5.000 Millisekunden) die Funktion `zeitVorbei()` aufgerufen.

```
<body>
  <h3>Bitte warten, der Countdown läuft</h3>
  <script type="text/javascript">
    function zeitVorbei() {
      alert("Die Zeit ist abgelaufen...");
      return;
    }
    setTimeout(zeitVorbei, 5000);
  </script>
</body>
```

Beachten Sie, dass der erste Parameter eine Funktionsreferenz (also ohne runde Klammern) ist und kein Funktionsaufruf. Sie können hier auch einen Funktionsaufruf notieren (dann aber in Hochkommata eingeschlossen), aber die Funktionsreferenz ist besser für die Ressourcen und zudem sauberer in das Konzept der allgemeinen Ereignisbehandlung in JavaScript integriert.

Ein Beispiel für ein Meldungsfenster

Obwohl diese Art der Meldungsfenster veraltet ist, soll ein kurzes Beispiel die grundsätzliche Verwendung von `alert()` (Alarm) zeigen, da man dies zur Fehlersuche verwenden kann. Es zeigt eine Information an, die der Benutzer mit einem Klick auf die Schaltfläche *OK* bestätigen muss. Als Parameter können Sie eine Zeichenkette oder Variable angeben, die als Meldung angezeigt werden soll.

```
alert("Anzeige");
```

Ein Beispiel für ein Bestätigungsfenster

Über das `confirm()`-Dialogfenster (Bestätigung) können Sie eine Frage anzeigen, die der Benutzer über die Schaltflächen *OK* und *Abbrechen* beantworten kann. Als Parameter geben Sie die anzuzeigende Frage an.

```
confirm("Frage");
```

Über den Rückgabewert können Sie die betätigte Schaltfläche auswerten. Es werden die folgenden Rückgabewerte geliefert:

- ✓ `true`, wenn der Benutzer die Frage mit *OK* bestätigt hat,
- ✓ `false`, wenn er die Frage verneint, d. h., die Schaltfläche *Abbrechen* betätigt.

Wichtig ist, dass der Anwender das Dialogfenster erst bestätigen muss, bevor das Skript fortgesetzt wird.

Beispiel: *interaktion_confirm.html*

In Abhängigkeit von der Beantwortung der Frage soll eine entsprechende Meldung angezeigt werden.

```
<body>
  <script type="text/javascript">
    wahl = confirm("Sind Sie das erste Mal hier?");
    if(wahl == true)
      alert("Sie haben die Frage mit OK bestätigt!");
    else
      alert("Sie haben die Frage mit ABBRECHEN beantwortet!");
  </script>
</body>
```

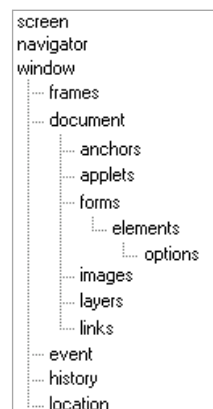


Ein Bestätigungsdialog

7.3 Grundsätzliches zur Struktur des DOM einer Webseite

Die Struktur eines **Dokuments** im Browser ist aus Sicht des DOM-Konzepts eine Art **Baum**, der beim Laden eines Dokuments durch den Browser im Speicher aufgebaut wird. Beim Verlassen der Seite wird der Baum wieder gelöscht.

Ähnliche Elemente werden beim Aufbau des Baums bei der Indizierung auf gemeinsame dynamische Datenstapel (so genannte Stacks) für das Dokument abgelegt, die aus JavaScript über (besondere) Arrays bereitstehen. Auf diese Weise hat ein Browser nach dem Laden der Webseite genaue Kenntnis über alle relevanten Daten sämtlicher eigenständig für ihn ansprechbarer Elemente (Objekte) in der Webseite. Welche das sind und was er damit anstellen kann, das hat sich in der Vergangenheit je nach Browser erheblich unterschieden. Diese Unterschiede nehmen aber stark ab.



Knoten, Verwandtschaftsbeziehungen und das `node`-Objekt

Die im Baum einer Webseite enthaltenen Elemente, aber auch andere Bestandteile eines Dokuments wie Attribute, Kommentare oder Textinhalte werden **Knoten (nodes)** genannt. Knotenstrukturen in einem Baum erlauben auch Arten von Beziehungen zu anderen Knoten zu definieren.

- ✓ Jeder Baum entspringt einem **Wurzelknoten (root)**. Bei einer Webseite ist das das Element `html`.
- ✓ Jeder Teilbaum hat für seine spezielle Struktur einen entsprechenden Wurzelknoten, etwa bei einer Tabelle das `table`-Element oder bei einer Aufzählungsliste das Element `ul`.
- ✓ Untergeordnete Knoten werden in einem Baum als **Kinder (children)** bzw. Kindelemente oder – in tieferen Ebenen – als Nachfolger bezeichnet. Knoten auf derselben Ebene sind **Geschwister (siblings)** und übergeordnete Knoten **Eltern (parent)** bzw. Vorfahren, wenn es um mehrere Ebenen geht.
- ✓ Ausgehend vom Wurzelknoten, ist der Weg zu jedem anderen Knoten im Baum über diese Verwandtschaftsbeziehungen beschreibbar.

Wenn ein Browser die Webseite als ein strukturiertes Dokument aufgebaut hat, steht Ihnen damit ein sogenanntes `node`-Objekt zur Verfügung. Über dessen Eigenschaften und Methoden können Sie auf den Elementen im Baum navigieren und Teile einer Webseite dynamisch auswerten und ergänzen.

Dazu benötigen Sie Zugriff auf den jeweiligen Knoten, der Ihnen im DOM-Konzept **indirekt** bereitgestellt wird. Sie können **nicht** `node` als Objektnamen nehmen, sondern verwenden den Knoten, den die nachfolgend genauer vorgestellten Methoden `document.getElementById()` und `document.getElementsByName()` sowie `document.getElementsByTagName()` liefern. Zusätzlich werden die Knoten von Elementen in **Objektfeldern** bereitgestellt. Sie erhalten auf jedem Weg des Zugriffs das gleiche Objekt. Damit können Sie in jeder Webseite die Strukturen abfragen und gezielt verwenden.

Wichtige Knotentypen bei Webseiten

Das DOM-Konzept stellt eine ganze Reihe an Knotentypen zur Verfügung, die nur teilweise in Webseiten eine Rolle spielen (dafür aber etwa unter XML). Die wichtigsten Knotentypen für Webseiten im DOM-Konzept sind folgende:

- ✓ Ein **Dokumentknoten** repräsentiert die gesamte Baumstruktur einer Webseite.
- ✓ Ein **Elementknoten** entspricht exakt einem Element in HTML.
- ✓ Ein **Attributknoten** entspricht einem Attribut in HTML.
- ✓ Ein **Textknoten** repräsentiert den textuellen Inhalt eines Elements, wenn dieses einen solchen haben kann (etwa eine Überschrift oder ein `div`-Bereich).
- ✓ Ein **Kommentarknoten** repräsentiert jeden Bereich mit einem HTML-Kommentar.

Die meisten Elemente einer Webseite können bei Bedarf während der „Lebenszeit“ der Webseite aktualisiert werden (ein `
`-Element z. B. wäre nicht aktualisierbar). Etwa wenn mittels eines Skripts die Position eines Elementes in der Webseite oder dessen Inhalt verändert oder über Style Sheets nach dem vollständigen Laden der Webseite das Layout eines Elements dynamisch umgewandelt wird.

Das DOM besteht also aus Knoten, die Referenzen auf ihre über- und untergeordneten Knoten haben. Jeder Knoten repräsentiert genau ein Element oder Attribut in der Baumstruktur des Dokuments.

- ✓ Das aktuelle HTML-Dokument ist ein Dokumentknoten.
- ✓ Jedes HTML-Element ist ein Elementknoten.
- ✓ Texte in HTML-Elementen sind Textknoten.
- ✓ Jedes HTML-Attribut ist ein Attributknoten.
- ✓ Kommentare sind Kommentarknoten.

So besteht beispielsweise die HTML-Angabe

```
<div>Inhalt des Blocks</div>
```

aus zwei Knoten, dem umschließenden Elementknoten `div` und dem Textknoten "Inhalt des Blocks". Da sich der Textknoten innerhalb des Elements `div` befindet, wird er auch als Kindknoten (child node) des Elements `div` bezeichnet. Das umschließende Element wird Elternknoten (parent node) genannt. Befindet sich innerhalb des Elements `div` ein weiteres Element, wie beispielsweise

```
<div>Inhalt <i>des Blocks</i></div>
```

enthält das Element `div` zwei Kindknoten, das Wort `Inhalt` sowie das Element `i`. Der Text "des Blocks" ist nun nicht mehr das Kind des Elements `div`, sondern das Kind des Elements `i`.

Attributknoten (attribute nodes) entstehen, wenn bei einem Element ein Attribut angegeben wird.

```
<div style="text-align:center;">Inhalt <i>des Blocks</i></div>
```

Das Element `div` enthält nun drei Knoten, denn das Attribut `style` ist ein Attributknoten.

Eine Webseite dynamisch aus Knoten aufbauen

Eine Webseite kann per JavaScript dynamisch aus Knoten aufgebaut werden. Alle `node`-Objekte besitzen dazu geeignete Methoden und Eigenschaften, von denen nur die wichtigsten hier vorgestellt werden:

- ✓ Um Knoten an bestehende Knoten dynamisch anzufügen, besitzen alle Dokument- und Elementknoten eine Methode `appendChild()`. Der Name ergibt sich aus oben beschriebener Verwandtschaftsbeziehung, die man in einem Knotenbaum verwendet.
- ✓ Attributknoten werden mit `setAttributeNode()` einem Elementknoten angefügt.
- ✓ Den Zugriff auf den Wert eines Attributknotens können Sie mit der Eigenschaft `nodeValue` durchführen.

7.4 Das Objekt `document`

Nachfolgend finden Sie die Eigenschaften und Methoden des `document`-Objekts, das die Webseite selbst repräsentiert und in der Praxis eines der wichtigsten Objekte im DOM-Konzept ist.

Die Eigenschaften

Das `document`-Objekt enthält eine Vielzahl von Eigenschaften, über die Sie auf die Elemente eines HTML-Dokuments zugreifen. Viele Eigenschaften des Objekts sind veraltet, weil Sie HTML-Attribute repräsentieren, die heutzutage nicht mehr verwendet werden, etwa Eigenschaften für das Setzen von Farben über generierte HTML-Attribute – stattdessen verwendet man CSS. Diese Eigenschaften werden hier nicht mehr betrachtet. Die Tabelle enthält nur Eigenschaften, die in der Praxis noch eingesetzt werden.

Eigenschaft	Bedeutung
<code>title</code>	Mit der Eigenschaft <code>title</code> lässt sich der Dokumenttitel auslesen bzw. setzen.
<code>referrer</code>	Die URL des Dokuments, das per Hyperlink auf das aktuelle Dokument verwiesen hat. Der Referrer wird nur übermittelt, wenn sich die Webseiten auf einem Webserver befinden und der Browser das nicht deaktiviert hat.
<code>lastModified</code>	Damit erhalten Sie das Datum, an dem das Dokument zuletzt verändert wurde.
<code>cookie</code>	Zugriff auf Cookies, die lokal abgespeichert werden und Parameter-Paare in Form von Zeichenketten enthalten können. Cookies können gelesen und verändert werden.
URL	Hiermit erhalten Sie die URL des geladenen Dokuments.

Die Eigenschaft `URL` ist ein Synonym für die Eigenschaft `document.location`, um bei der Anwendung eine Verwechslung mit `window.location` zu vermeiden. Denn `window.location` beinhaltet die vom Nutzer angeforderte URL, und `document.URL` beinhaltet die an ihn zurückgelieferte URL. Diese können sich durch Weiterleitungen (Redirects) voneinander unterscheiden.

Beispiel: *objekt_document.html*

In dem folgenden Beispiel werden die Werte von Eigenschaften von `document` ausgegeben. Dazu kommt die Methode `document.write()` zum Einsatz.

```
...
<body>
  <h3>Eigenschaften des document-Objekts</h3>
  <script type="text/javascript">
    document.write("Titel der Webseite: mit document.title = " +
      document.title + "<br />");
    document.write("Adresse der Webseite: mit document.location = " +
      document.location + "<br />");
    document.write("Adresse der Webseite: mit document.URL = " +
      document.URL + "<br />");
    document.write("Seite wurde von Adresse aufgerufen:
      mit document.referrer = " +
      document.referrer + "<br />");
    document.write("Letzte Änderung der Seite:
      mit document.lastModified = " +
      document.lastModified + "<br />");
  </script>
</body>
</html>
```

Eigenschaften des document-Objekts

Titel der Webseite: mit `document.title` = Objekt: `document`
 Adresse der Webseite: mit `document.location` = `file:///F:/Users/ralph/Documents/B%C3%BCcher/herdt/JavaScript2023/Beispiele/Kap08/objekt_document.html`
 Adresse der Webseite: mit `document.URL` = `file:///F:/Users/ralph/Documents/B%C3%BCcher/herdt/JavaScript2023/Beispiele/Kap08/objekt_document.html`
 Seite wurde von Adresse aufgerufen: mit `document.referrer` =
 Letzte Änderung der Seite: mit `document.lastModified` = 05/18/2023 21:31:06

Ausgabe von verschiedenen document-Eigenschaften

Die Methoden

Das `document`-Objekt enthält verschiedene Methoden, die in der Praxis teilweise nur selten eingesetzt werden. Andere Methoden sind dagegen unabdingbar für jede moderne Webseite. Hier folgt eine Auswahl wichtiger Methoden – ggf. mit einem Hinweis, ob diese wichtig sind oder nicht.

Methode	Bedeutung
<code>clear()</code>	Hiermit löschen Sie den Inhalt der aktuellen Webseite. Die Methode wird sehr selten eingesetzt.
<code>close()</code>	Die Methode <code>close()</code> schließt das Dokument, sodass keine weiteren Inhalte eingefügt werden können. Die Methode wird sehr selten eingesetzt.
<code>createAttribute()</code>	Erzeugen eines neuen Attributknotens für ein HTML-Element
<code>createElement()</code>	Erzeugen eines neuen Elementknotens im DOM-Baum eines Dokuments
<code>createTextNode()</code>	Mit der Methode können Sie im DOM-Baum eines Dokuments einen Textknoten erzeugen.
<code>getElementById()</code>	Damit können Sie auf ein HTML-Element mit einem im Attribut <code>id</code> festgelegten Namen zugreifen. Sie erhalten als Ergebnis den selektierten Elementknoten . Die Methode ist sehr wichtig und Bestandteil fast jeder modernen Webseite.
<code>getElementsByName()</code>	Die Methode funktioniert wie <code>getElementByTagName()</code> , nur dass hier das Attribut <code>name</code> eines HTML-Elements ausgewertet wird und alle Elemente als Array zurückgeliefert werden. Die Methode funktioniert nur bei einigen HTML-Elementen. Zuverlässig in allen Browsern ist nur der Zugriff auf die HTML-Elemente, die auch in den standardisierten Objektfeldern von <code>document</code> zur Verfügung stehen.

Methode	Bedeutung
getElementsByName()	<p>Alternativ kann man bei solchen Elementen auch direkt den Namen als Eigenschaft notieren, obwohl das nicht empfohlen wird und man in dem Fall die Methode getElementByTagName() verwenden sollte.</p> <p>Nachfolgend sehen Sie ein Beispiel mit gleichwertigen Zugriffen auf ein Formularfeld:</p> <pre> <form name="form1"> ... <input name="feld1" /> ... </form> <script type="text/javascript"> document.write(document.form1.feld1.value + "
"); document.write(document.getElementsByName("form1")[0].feld1.value + "
"); document.write(document.getElementsByName("feld1")[0].value + "
"); </script> </pre>
getElementsByTagName()	Hiermit erhalten Sie ein Array der HTML-Elemente als Knoten, deren Namen Sie als Parameter angegeben haben, z. B. h1, p. Die Methode ist sehr wichtig und Bestandteil fast jeder modernen Webseite.
open()	Damit löschen Sie die aktuelle Webseite und öffnen das Dokument für neue Inhalte. Die Methode wird sehr selten eingesetzt.
write(Text) und writeln(Text)	Über diese Methoden schreiben Sie Text in das HTML-Dokument. Dieser wird an der Stelle eingefügt, an der die Methode aufgerufen wird. Bei writeln(Text) wird zusätzlich ein Zeilenumbruch eingefügt. Dieser Umbruch ist nur im Quelltext der Webseite sichtbar und entspricht nicht dem HTML-Tag . Die Methoden werden heutzutage nur noch beim Laden einer Webseite eingesetzt. Die nachträgliche Veränderung einer Webseite, was bei dynamischen Webseiten ständig erfolgt, ist damit nicht adäquat möglich.

Beispiel: *elemente_erstellen.html*

In dem folgenden Beispiel werden Elemente der Webseite mit den Methoden von document und node dynamisch erstellt und einer Webseite hinzugefügt.

	<pre> <body> ① <h1>Erstellen von DOM-Elementen</h1><hr /> <script type="text/javascript"> ② var bildquelle = document.createAttribute("src"); </pre>
--	--

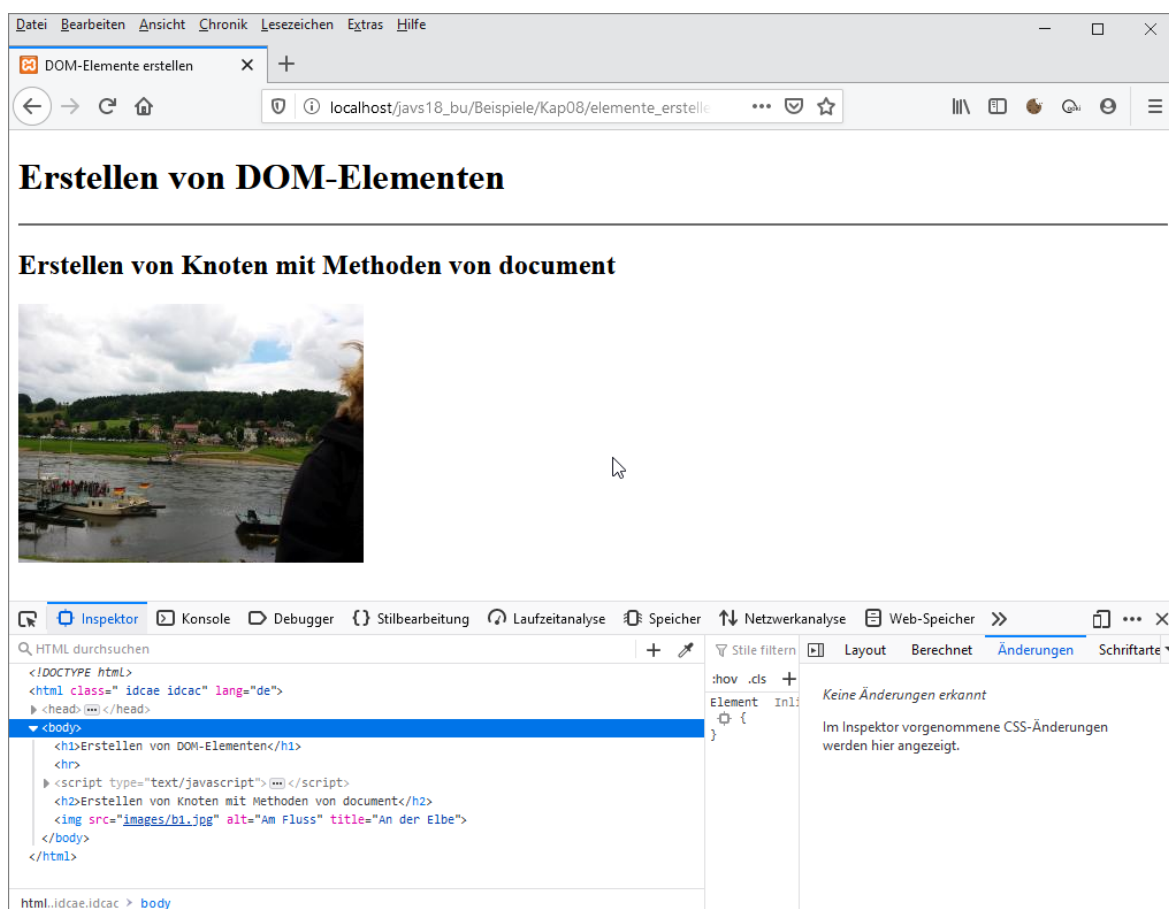
```

③ bildquelle.nodeValue = "images/b1.jpg";
④ var alternativtext = document.createAttribute("alt");
  alternativtext.nodeValue = "Am Fluss";
  var titel = document.createAttribute("title");
  titel.nodeValue = "An der Elbe";
⑤ var b1 = document.createElement("img");
⑥ b1.setAttributeNode(bildquelle);
  b1.setAttributeNode(alternativtext);
  b1.setAttributeNode(titel);
⑦ var u2 = document.createElement("h2");
  var text = document.createTextNode(
    "Erstellen von Knoten mit Methoden von document");
⑧ u2.appendChild(text);
⑨ document.getElementsByTagName("body")[0].appendChild(u2);
  document.getElementsByTagName("body")[0].appendChild(b1);
</script>
</body>

```

- ① Zuerst sehen Sie in dem Quelltext der Webseite die HTML-Tags für eine Überschrift der Ordnung 1 und eine Trennlinie. Ansonsten enthält die Webseite keine weiteren Tags außer dem folgenden Skriptbereich.
- ② Mit der Methode `createAttribute()` wird ein Attributknoten erzeugt. Der Wert des Parameters ist `"src"`. Das ist der Name des Attributs für die Quelle eines Bildes bei einem `img`-Tag. Der erzeugte Attributknoten soll einem Bildobjekt hinzugefügt werden. Das Bildobjekt muss aber noch erzeugt werden. Der Attributknoten kann zu dem Zeitpunkt noch nicht dem Elementknoten für das noch zu erzeugende Bild angehängt oder sonst in die Webseite eingefügt worden sein. Die Referenz auf den Attributknoten wird in der Variablen `bildquelle` gespeichert und steht somit für die spätere Verwendung zur Verfügung.
- ③ Mit der Eigenschaft `nodeValue` wird der Wert des Attributknotens `bildquelle` gesetzt. Der Wert ist die relative Adresse einer Bilddatei, die in der Webseite angezeigt werden soll.
- ④ Mit der Methode `createAttribute()` werden zwei weitere Attributknoten erzeugt, denen dann jeweils mit `nodeValue` Werte zugewiesen werden. Die Attribute stehen für den Alternativtext und den Titel bei einem Bild.
- ⑤ Mit der Methode `createElement()` wird jetzt ein Elementknoten vom Typ `img` erstellt. Auch dieser Knoten ist danach noch nicht der Webseite hinzugefügt worden, die Referenz darauf steht in der Variablen `b1` zur Verfügung.
- ⑥ Bis zu dem Zeitpunkt gibt es noch keine Beziehung zwischen dem Elementknoten `b1` und den drei Attributknoten. Mit der Methode `setAttributeNode()` werden dem Elementknoten `b1` jetzt alle drei vorher erstellten Attributknoten zugewiesen. Die dynamisch generierte HTML-Struktur des Elementknotens `b1` sieht danach so aus:
``
 Der Knoten ist danach noch nicht der Webseite hinzugefügt worden und damit noch nicht sichtbar in der Webseite.

- ⑦ Mit der Methode `createElement()` wird ein Elementknoten vom Typ `h2` erstellt. Auch dieser Knoten ist nach dem Erstellen noch nicht der Webseite hinzugefügt worden, aber die Referenz darauf steht in der Variablen `u2` zur Verfügung. Dieses Element ist eine Überschrift der Ordnung 2, und die hat einen Textinhalt. Das bedeutet, dass das Kindelement dieses Elementknotens ein Textknoten ist, und so ein Textknoten wird im folgenden Schritt mit der Methode `createTextNode()` erstellt. Es gibt zu dem Zeitpunkt noch keine Verbindung zwischen dem Elementknoten `h2` und dem Textknoten `text`. In dem DOM der Webseite ist damit noch keine Überschrift der Ordnung 2 vorhanden.
- ⑧ In diesem Schritt wird der Textknoten `text` dem Elementknoten `u2` hinzugefügt. Dazu kommt die Methode `appendChild()` zu Einsatz. Die dynamisch generierte HTML-Struktur des Elementknotens sieht so aus:
`<h2>Erstellen von Knoten mit Methoden von document</h2>`
 Der Knoten ist danach noch nicht der Webseite hinzugefügt worden und damit nicht sichtbar in der Webseite.
- ⑨ Mit der Methode `appendChild()` werden die Elementknoten `b1` und `u2` nacheinander der Webseite hinzugefügt. Die Webseite wird dazu mit `document.getElementsByTagName("body")[0]` selektiert. Beachten Sie, dass die Methode `document.getElementsByTagName()` ein Array zurückliefert und man mit dem Index unbedingt arbeiten muss. Beim `body`-Element, das in jeder Webseite nur einmal vorkommt, erscheint das überflüssig, aber da die Methode `document.getElementsByTagName()` **immer** ein Array liefert, muss mit dem Index gearbeitet werden. Die Knoten sind nun dynamisch der Webseite hinzugefügt worden und damit sichtbar.



Die dynamisch aufgebaute Webseite – Struktur, wie der Browser sie verwendet – Anzeige in der Web-Konsole/Inspektor

7.5 Zugriff auf Inhalte von Elementen in der Webseite

Der Zugriff auf den Inhalt von Elementen in einer Webseite per JavaScript lässt sich im Wesentlichen in drei Kategorien unterscheiden:

- ✓ Zugriff auf Textinhalte
- ✓ Zugriff auf Werte von Formularfeldern
- ✓ Zugriff auf klassische HTML-Attribute über deren DOM-Repräsentation

Zugriff auf Textinhalte mit `innerHTML`

Sie können bei jedem Element in der Webseite, das einen Text als Inhalt hat, diesen Textinhalt per JavaScript ansprechen und austauschen, sofern Sie den zugehörigen Knoten selektiert haben. Sie haben schon die Eigenschaft `nodeValue` gesehen, aber es gibt mit `innerHTML` eine einfachere Möglichkeit. Als Alternative steht `innerText` zur Verfügung.

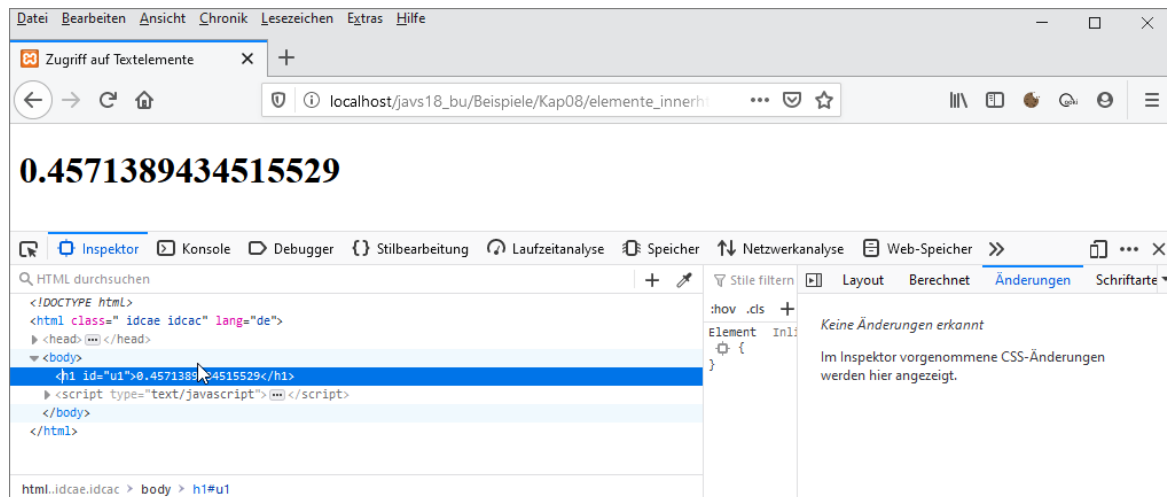
Beide Eigenschaften gehörten schon lange zum Werkzeugkasten von JavaScript-Programmierern, obwohl sie nie zum offiziellen DOM-Standard gezählt haben. Dennoch funktioniert der Zugriff über `innerHTML` in allen neuen Browser-Generationen. In HTML5 wurde `innerHTML` dann auch offiziell eingeführt.

Beispiel: *elemente_innerhtml.html*

In dem folgenden Beispiel wird der Inhalt einer Überschrift über `innerHTML` dynamisch gefüllt.

```
<body>
① <h1 id="u1"></h1>
   <script type="text/javascript">
②   document.getElementById("u1").innerHTML=Math.random();
   </script>
</body>
```

- ① Zuerst sehen Sie in der Webseite die HTML-Tags für eine leere Überschrift vom Typ `h1`. Diese Überschrift hat eine ID `u1`.
- ② Im Skript wird mit der Methode `document.getElementById()` die Überschrift selektiert und mit `innerHTML` der Inhalt für diesen Elementknoten gesetzt. Mit `Math.random()` wird ein Zufallswert zwischen 0 und 1 in die Webseite geschrieben.



Die dynamisch über `innerHTML` gefüllte Überschrift

Der Unterschied zwischen `innerHTML` und `innerText` ist gering. Über `innerHTML` wie auch `innerText` haben Sie Zugang zum Textinhalt eines HTML-Elements. Wenn Sie beim dynamischen Ändern des gespeicherten Inhalts bei `innerHTML` jedoch HTML-Tags notieren, werden diese bei der Aktualisierung des Elementinhalts interpretiert. Das ist bei `innerText` nicht der Fall. Allerdings wirkt sich das bei vielen Browsern nicht aus. Einige Browser sperren sogar die Verwendung von `innerText`. Auch sonst waren in der Vergangenheit diverse Probleme im Zusammenhang mit `innerText` bekannt, weshalb Sie in der Praxis `innerText` nicht verwenden sollten.

Der Zugriff auf `innerHTML` wird normalerweise nicht beim Laden einer Webseite erfolgen, sondern nach dem Laden der Webseite dynamisch aufgrund eines Ereignisses (vgl. Kapitel 8).

Zugriff auf klassische HTML-Attribute

Die per DOM verfügbaren Elemente einer Webseite verfügen in der Regel über die gleichen Eigenschaften, die als Attribute bei den zugrunde liegenden HTML-Tags vorhanden sind. Sie können also Kenntnisse aus HTML übernehmen und die Namen der bekannten Attribute von Elementen als Eigenschaften des zugehörigen Elementknotens verwenden. Indem Sie die Namen der Eigenschaften angeben, können Sie lesend und meist auch schreibend darauf zugreifen.

Beispiel: `elemente_klassisch.html`

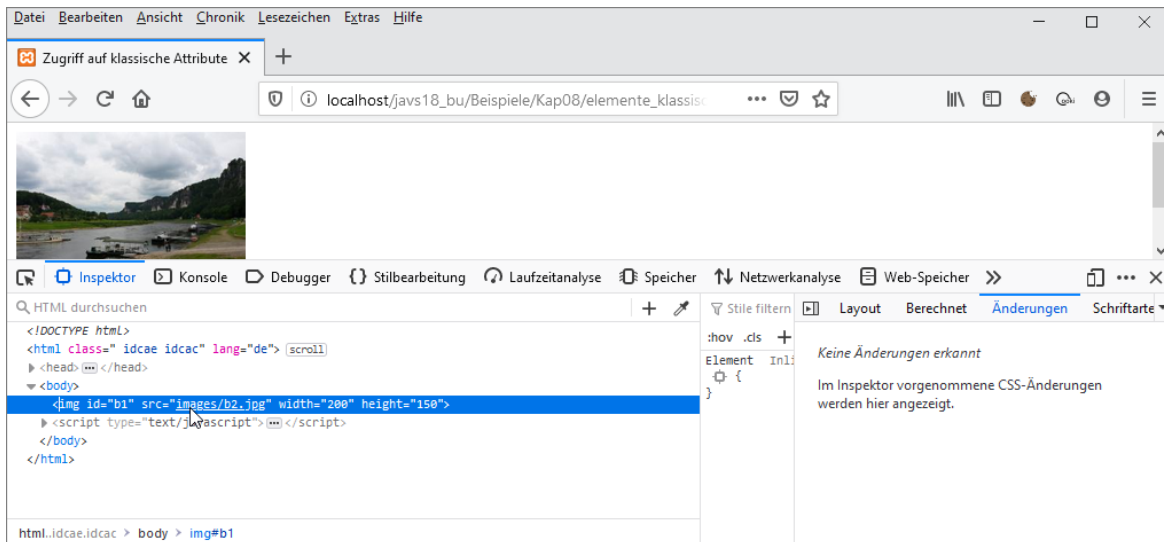
In dem folgenden Beispiel werden mehrere Attribute eines Bildes angesprochen.

```

<body>
① <img id="b1" />
  <script type="text/javascript">
②   document.getElementById("b1").width = "200";
     document.getElementById("b1").height = "150";
     document.getElementById("b1").src = "images/b2.jpg";
  </script>
</body>

```

- ① Zuerst sehen Sie in der Webseite den HTML-Tag für eine leere Bildreferenz. Der Tag hat eine ID `b1`.
- ② Im Skript wird in den drei folgenden Anweisungen mit der Methode `document.getElementById()` das Bild selektiert. Die verwendeten Eigenschaften entsprechen den Namen klassischer HTML-Attribute bei diesem Tag.



Zugriff auf klassische HTML-Attribute

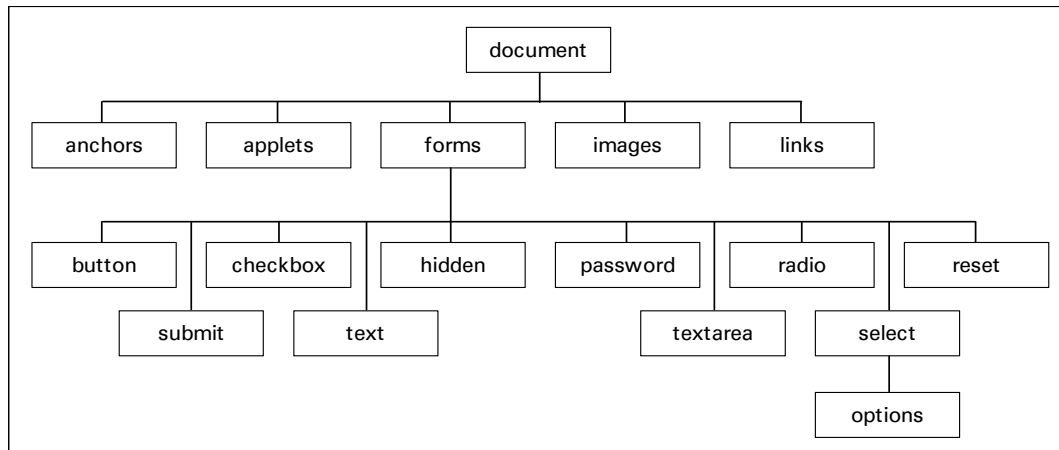
Zugriff auf Formularinhalte

Der Inhalt von Formulareingabefeldern steht über die Eigenschaft `value` zur Verfügung. Diese Eigenschaft ist in der Regel sowohl zu lesen als auch zu schreiben (vgl. Kapitel 9).

7.6 HTML-Elemente als Unterobjekte von `document`

Nun soll noch ein Blick auf die Unterobjekte von `document` aus Sicht der **Objektfelder** erfolgen. Ein `document`-Objekt besitzt Unterobjekte in Form standardisierter Arrays (Objektfelder), worüber auch ein Zugriff auf Elemente einer Webseite möglich ist. Wie bei sämtlichen generierten Arrays in JavaScript erhalten Sie über die Eigenschaft `length` die Anzahl der jeweils enthaltenen Objekte.

So können Sie über das Objektfeld `forms` auf Formulareigenschaften und -elemente zugreifen, sobald sich ein Formular (`<form> . . . </form>`) in einer Webseite befindet. Diese Art des Zugriffs war historisch die erste Möglichkeit, um Bestandteile einer Webseite aus JavaScript heraus anzusprechen. Heutzutage nutzt man sie nur noch sehr selten, aber grundsätzlich funktioniert diese Alttechnik immer noch in allen modernen Browsern, und es gibt einige Situationen, in denen diese historische Vorgehensweise sogar noch ihre Berechtigung hat. Ebenso ist wichtig, dass ein Zugriff über den Namen auf DOM-Elemente auf die Elemente Objektfelder in allen Browsern zuverlässig ist. Andere Elemente können nur in einigen Browsern so angesprochen werden.



Hierarchie des *document*-Objekts

Der Aufbau der Webseite bestimmt dabei das Vorhandensein von diesen Objektfeldern. Enthält beispielsweise ein HTML-Dokument kein Formular, besitzt das *document*-Objekt auch keine Objektreferenzen auf Formulare über *forms*.

- ✓ `window.document.images []` enthält alle Grafiken in der Reihenfolge, in der die ``-Tags im HTML-Quelltext verwendet werden.
- ✓ `window.document.anchors []` enthält alle Anker entsprechend den Tags ``.
- ✓ `window.document.links []` enthält alle Links entsprechend den Tags ``.
- ✓ `window.document.forms []` enthält alle Formulare, die durch `<form>`-Tags definiert wurden.

Das Objektfeld **anchors**

Mit diesem Objektfeld haben Sie Zugriff auf die Verweisanker einer Webseite. Über die folgende HTML-Anweisung wird z. B. ein Verweisanker definiert:

```
<a name="absatz2">Überschrift zum 2. Absatz</a>
```

Eigenschaft	Beschreibung
<code>name</code>	Erlaubt den Zugriff auf den Namen des Ankers
<code>text</code>	Erlaubt den Zugriff auf den Text des Ankers

Das Unterobjekt **applets**

Dieses Objektfeld erlaubt den Zugriff auf Java-Programme, die speziell für den Gebrauch im Browser erstellt wurden, sogenannte Java-Applets.

Java-Applets werden allerdings heutzutage so gut wie gar nicht mehr verwendet.

Das Unterobjekt `links`

Über das Objektfeld `links` haben Sie per JavaScript Zugriff auf die Verweise bzw. Hyperlinks in einem HTML-Dokument. Die Eigenschaften eines enthaltenen Hyperlinkobjekts entsprechen den klassischen HTML-Attributen:

Überschrift	Überschrift
<code>href</code>	Damit erhalten Sie die URL, auf die verwiesen wird.
<code>name</code>	Erlaubt den Zugriff auf den Namen des Verweises
<code>target</code>	Über diese Eigenschaft greifen Sie auf das Zielframe des Verweises zu.
<code>text</code>	Erlaubt den Zugriff auf den Text des Verweises. Diese Eigenschaft wird vom Internet Explorer nicht interpretiert.

Beispiel: `links.html`

Zu Beginn werden im `<body>`-Tag zwei Links definiert. Im folgenden Skript werden die Informationen für jeden Link mit einer Schleife über das Array `links` in einem Meldungsbereich der Webseite per `innerHTML` ausgegeben. Beachten Sie den Operator `+=`, mit dem der Meldungsbereich in der Webseite so gefüllt wird, dass bei jedem Schleifendurchlauf der neue Inhalt an den bestehenden Inhalt angehängt wird.

```
<a href="http://www.google.de/" name="link1">Google</a> |
<a href="http://www.bing.com" target="neu">Bing</a>
<div id="info"></div>
<script type="text/javascript">
  for(var i = 0; i < document.links.length; i++)
  {
    document.getElementById("info").innerHTML += ((i + 1) +
      ". Link: " + "<br />" +
      "Name: " + document.links[i].name + "<br />" +
      "Ziel: " + document.links[i].target + "<br />" +
      "Verweis: " + document.links[i] + "<br />" +
      "Text: " + document.links[i].text + "<hr />");
  }
</script>
```

```
Google | Bing
1. Link:
Name: link1
Ziel:
Verweis: http://www.google.de/
Text: Google

2. Link:
Name:
Ziel: neu
Verweis: http://www.bing.com/
Text: Bing
```

Auswertung von Links

Das Objektfeld `images`

Bilder in Webseiten werden durch das Objektfeld `images` bereitgestellt. Die Eigenschaften eines enthaltenen Bildobjekts in so einem Objektfeld entsprechen den klassischen HTML-Attributen eines `img`-Tags.

Das Objektfeld `forms`

Formulare können u. a. in dem Objektfeld `forms` verfügbar gemacht werden.

7.7 Das Objekt `history`

Eine weitere Eigenschaft des `window`-Objekts ist das `history`-Objekt (`history` = Geschichte). Laden Sie eine Webseite in Ihren Browser, wird deren URL im Verlauf des Browsers gespeichert (im Firefox als Chronik bezeichnet).

Mit diesem Objekt erhalten Sie einen bedingten Zugriff auf diese gespeicherten URLs. Bedingt heißt, dass Sie nur vorwärts und rückwärts zu den Einträgen springen, jedoch nicht direkt deren Inhalt auslesen können. Dies dient dem Schutz der Privatsphäre des Nutzers.

Eigenschaft	Bedeutung
<code>length</code>	Diese Eigenschaft gibt die Anzahl der Einträge in der History des Browsers wieder.

Methode	Bedeutung
<code>back()</code>	Die Methode lädt das zuvor besuchte Dokument.
<code>forward()</code>	Die Methode lädt das nächste Dokument innerhalb der History.
<code>go(Schritte)</code>	Bei der Methode <code>go()</code> können Sie eine bestimmte Anzahl an Seiten überspringen. Eine negative Zahlenangabe (<code>go(-2)</code>) lädt die vorletzte Seite, eine positive Angabe (<code>go(2)</code>) lädt die übernächste Seite.

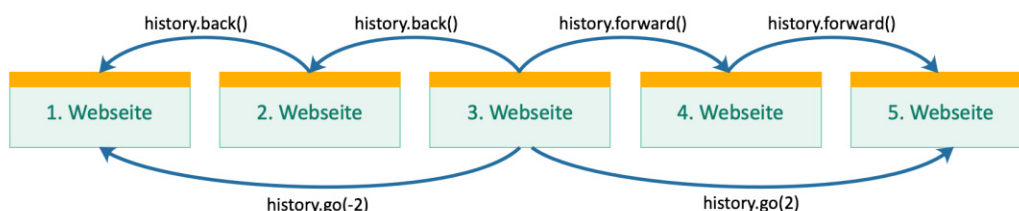
Wird `back()` beim ersten und `forward()` beim letzten Eintrag in der History aufgerufen, haben diese Methoden keine Auswirkung.

Beispiel: *objekt_history.html*

Auf einer Webseite fügen Sie vier Hyperlinks ein, mit denen Sie in der History des Browsers navigieren können. Sie müssen dazu im Browserfenster bereits einige Webseiten besucht haben, um diese Funktionalität zu nutzen.

	<pre> <body> <h3>Eigenschaften des history-Objekts</h3> ①
 Eine Seite zurückblättern ②
 Eine Seite vorwärtsblättern ③
 Zwei Seiten zurückblättern
 Zwei Seiten vorwärtsblättern </body> </pre>
--	--


- ① Ausgehend von der aktuellen Seite, wird mit der Methode `history.back()` im Verlauf des Browsers eine Webseite zurückgeblättert. Dies ist die auf Webseiten am häufigsten eingesetzte Methode.
- ② Mit `history.forward()` navigieren Sie eine Seite vorwärts. Dazu müssen Sie sich im mittleren Teil der History befinden, sodass Sie vorwärts navigieren können.
- ③ Das Blättern über mehrere Webseiten realisieren Sie über `history.go()`.



Mit den Methoden des `history`-Objekts können Sie durch die History der besuchten Webseiten navigieren. Dies ist jedoch erst möglich, wenn Sie bereits einige Webseiten mit dem Browser besucht haben und dann die betreffende Webseite im Browser laden. Zur Ausführung von `history.go(2)` müssen Sie mindestens zwei weitere Webseiten besuchen und sich wieder zur Webseite des Beispiels in der History zurückbewegen.

7.8 Das Objekt `location`

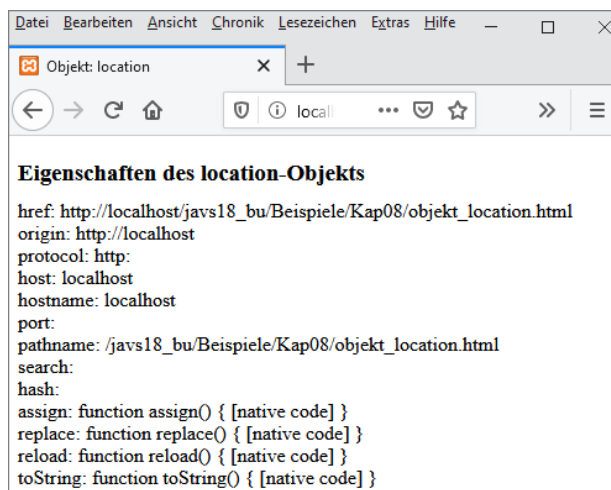
Das `location`-Objekt (`location` = Ort) ist eine weitere Eigenschaft des `window`-Objekts. Mit diesem Objekt können Sie auf die URL des aktuell dargestellten Dokuments lesend und schreibend zugreifen. Sie können dabei die gesamte URL oder auch einzelne Teile davon auslesen und verarbeiten.

Eigenschaft/ Methode	Bedeutung
protocol	Mit der Eigenschaft erhalten Sie das verwendete Protokoll, mit dem die Webseite aufgerufen wurde, z. B. <code>http:</code> oder <code>https:</code> .
hostname	Den Host- und Domain-Namen können Sie mit dieser Eigenschaft auslesen, z. B. <code>www.example.com</code> .
port	Die Port-Nummer des Servers wird über die Eigenschaft <code>port</code> zurückgeliefert.
host	Diese Eigenschaft entspricht den Angaben von <code>hostname:port</code> .
pathname	Die Pfadangabe des Dokuments wird über <code>pathname</code> ausgelesen, z. B. <code>docs/index.html</code> .
search	Es wird die Zeichenkette der an die Webseite übergebenen Parameter geliefert, z. B. <code>?suche=test</code> .
hash	Falls in der URL ein Anker angegeben wurde, können Sie dies über diese Eigenschaft ermitteln, z. B. <code>#anker</code> .
href	Die vollständige URL wird ausgelesen.
reload()	Die Methode lädt das Dokument erneut, wie beim Klick auf das Symbol  zum Aktualisieren der Webseite.
replace()	Die Methode lädt eine neue Seite.

Beispiel: *objekt_location.html*

In diesem Beispiel werden die Eigenschaften des `location`-Objekts in einer Tabelle ausgegeben. Damit die meisten Eigenschaften des Objekts `location` einen Wert zurückliefern, laden Sie die Webseite über einen Webserver und rufen Sie diese dort auf. Zusätzlich geben Sie in der URL einen Anker sowie Parameter an.

```
<h3>Eigenschaften des location-Objekts</h3>
<script type="text/javascript">
  for(var eig in location)
    document.write("<br />" + eig + ": " + location[eig]);
</script>
```



Übersicht und Ergebnis der Eigenschaften des `location`-Objekts

Beispiel: *objekt_location_redirect.html*

Durch das Setzen der Eigenschaft `href` kann der Browser über JavaScript veranlasst werden, eine neue Seite zu laden. Ein Anwendungsbeispiel für das Objekt `location` findet sich in einer URL-Weiterleitung. Beispielsweise können Sie den Besuchern eine bestimmte Webseite anzeigen, wenn kein JavaScript aktiviert ist. Den Benutzern mit aktiviertem JavaScript bieten Sie eine separate Webseite an, auf die automatisch weitergeleitet wird, z. B. *objekt_location_mit_js.html*.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Objekt: location</title>
  ① <script type="text/javascript">
      window.location.href='objekt_location_mit_js.html';
  </script>
</head>
<body>
  ② <h3>Bei Ihnen ist JavaScript deaktiviert.</h3>
</body>
</html>

```

- ① Im Kopfbereich der Seite notieren Sie einen Skriptbereich und geben dort die Weiterleitung an. Durch das Zuweisen eines Wertes an die Eigenschaft `window.location.href` wird eine andere Webseite in das Browserfenster geladen.
- ② Ist JavaScript nicht aktiv, wird der Inhalt dieser Webseite angezeigt.

7.9 Das Objektfeld `frames`

Früher hatte man Webseiten oft mit Segmenten unterteilt, die **Frames** genannt wurden. In einem Fenster können mehrere Frames enthalten sein. In jeden Frame kann wiederum ein HTML-Dokument geladen werden, in dem erneut Frames definiert sind. Frames gelten etwa seit dem Jahr 2000 als veraltet, wurden aber teils noch bis zum Jahr 2005 eingesetzt. Heutzutage findet man Frames fast nur noch in Form sogenannter **Inline Frames** (IFrames). Bei anspruchsvolleren RIAs, wie sie Google z. B. bereitstellt, werden diese IFrames meist auch noch versteckt und als unsichtbare Datenspeicher genutzt.

Das `frames`-Objekt ist auf der einen Seite eine Eigenschaft des `window`-Objekts, das selbst die gleichen Eigenschaften und Methoden wie das `window`-Objekt selbst besitzt. Auf der anderen Seite ist `frames` ein Array, und die einzelnen Unterframes können Sie über das `frames`-Objektfeld zugreifen.

Um einen Frame ansprechen zu können, geben Sie entweder den Namen des Frames aus `window.FrameName` an oder verwenden Sie einen Index. Beachten Sie, dass der Zähler bei null beginnt, d. h., das erste Frame-Fenster sprechen Sie mit `frames[0]` an, das zweite Frame-Fenster mit `frames[1]` usw. Beim Zählen gilt die Reihenfolge, in der die Frames im Frameset definiert sind.

Da Frames im Web nur noch sehr selten zu finden sind (und dann meist bei RIAs, die nur noch IFrames als Datenspeicher einsetzen), werden Frames hier nicht weiter behandelt.



Ergänzende Lerninhalte: *JavaScript-Techniken für ältere Skripte.pdf*

7.10 Das Objekt `screen`

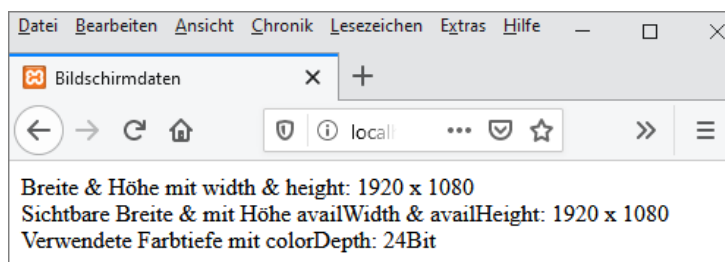
Es kann vorkommen, dass Sie die Bildschirmdaten eines Besuchers auswerten wollen. Dazu gibt es das `screen`-Objekt, wobei dessen Eigenschaften nicht immer in allen Browsern implementiert sind.

Eigenschaft(en)	Bedeutung
<code>height, width</code>	Über diese Eigenschaften lassen sich die Höhe und Breite der eingestellten Bildschirmauflösung in Pixeln bestimmen.
<code>availHeight, availWidth</code>	Die Ausmaße der wirklich zur Verfügung stehenden Fläche auf dem Bildschirm, ohne die Windows-Taskleiste oder die Macintosh-Menüleiste, erhalten Sie über diese Eigenschaften.
<code>colorDepth</code>	Die verwendete Farbtiefe wird in Bits pro Pixel ausgelesen.

Beispiel: *screen.html*

Im folgenden Beispiel werden die ermittelten Bildschirmwerte ausgegeben.

```
<body>
<script type="text/javascript">
  document.write("Breite & Höhe mit width & height: " +
    screen.width + " x " + screen.height + "<br />");
  document.write("Sichtbare Breite & mit Höhe availWidth &
    availHeight: " + screen.availWidth + " x " +
    screen.availHeight + "<br />");
  document.write("Verwendete Farbtiefe mit colorDepth: " +
    screen.colorDepth + "Bit");
</script>
</body>
```



Ausgabe der Bildschirmeinstellungen

7.11 Das Objekt `navigator`

Um die Eigenschaften eines Browsers auszulesen, stellt das DOM-Konzept das Objekt `navigator` zur Verfügung. Es besitzt einige interessante Eigenschaften, die aber zum Teil browserabhängig sind. Mit dem DOM-Konzept, das mit HTML5 verbunden ist, wurde das `navigator`-Objekt um mehrere Eigenschaften erweitert. Die nachfolgende Tabelle gibt die Eigenschaften an, die auch in alten Browsern funktionieren, aber teils browserabhängig sind.

Eigenschaft	Bedeutung
<code>appName</code>	Der Arbeitsname des Browsers, wie ihn der Browserhersteller bekannt gibt
<code>appVersion</code>	Die Version des Browsers inklusive der Plattform und des Länderkennzeichens
<code>userAgent</code>	Hiermit erhalten Sie die vollständige Browserbezeichnung.
<code>platform</code>	Dahinter verbirgt sich die verwendete Computer-Plattform.
<code>language</code>	Damit können Sie die Spracheinstellung des Client-Computers auslesen.
<code>plugins</code>	Diese Eigenschaft, die nicht alle Browser unterstützen, liefert ein Feld zurück, mit dem alle installierten Plug-ins aufgelistet werden können.
<code>mimeType</code>	Das Feld beinhaltet alle MIME-Typen, die vom Browser verarbeitet werden können.

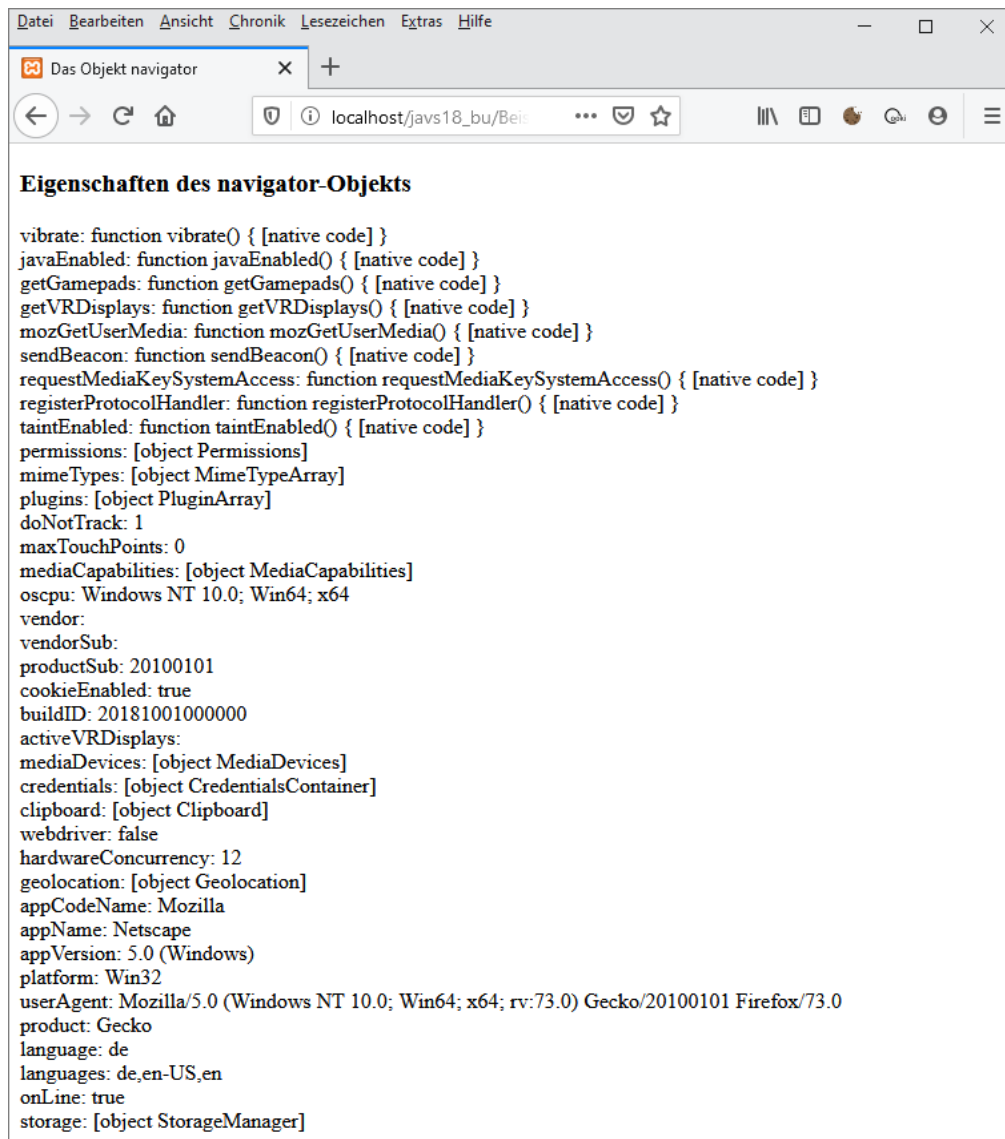
Diese Informationen über einen Browser waren lange Zeit wichtig, wenn Verzweigungen für browserspezifische JavaScript-Anweisungen benötigt wurden (sogenannte **Browserweichen**). Die alten Varianten vom Internet Explorer wiesen große Unterschiede in ihren Implementierungen der HTML- und JavaScript-Standards zu allen anderen Browsern auf. Somit war oft das Verzweigen in unterschiedliche Codesegmente nötig, um nicht eine Fehlermeldung in dem einen oder anderen Browser zu erhalten.

Beispiel: *objekt_navigator.html*

Dieses Beispiel liefert Angaben zu den Eigenschaften des `navigator`-Objekts des verwendeten Browsers.

```
<body>
  <h3>Eigenschaften des navigator-Objekts</h3>
  <script type="text/javascript">
    for(var eig in navigator)
      document.write(eig + ": " + navigator[eig] + "<br />");
  </script>
</body>
```

Im Firefox werden zum Beispiel für die Version 73 unter Windows die folgenden Ergebnisse angezeigt:



navigator-Eigenschaften

Über das `navigator`-Objekt gibt es auch einige wenige Methoden, die aber keine Bedeutung in der Praxis haben.

- ✓ Um zu ermitteln, welchen Browser der Besucher der Webseite verwendet, wurde früher oft der Wert der Eigenschaft `appName` überprüft. Durch die Vereinheitlichung der Angaben für diese Eigenschaft kann man in neuen Browsern diesen Weg nicht mehr zuverlässig gehen. Stattdessen nutzt man `userAgent` und/oder die weiteren Eigenschaften, die aber sehr aufwändig ausgewertet werden müssen, um einen Browser wirklich zu erkennen. Deshalb nutzt man in der Praxis durchaus noch die alten Browsererkennungen und nimmt in Kauf, dass man beispielsweise den neuen Internet Explorer als Mozilla-Browser einordnet. Da dieser aber standardkonform ist, ist das kein wirkliches Problem. Man will damit in der Regel nur alte Versionen vom Internet Explorer identifizieren, um diesen eine individuelle Webseite vorzusetzen. In den BuchPlus-Beispieldateien finden Sie eine alte Browserweiche unter `objekt_navigator_weiche.html`.

- ✓ Die Browser besitzen meist noch weitere individuelle Eigenschaften und Methoden, die Sie durch das Ausführen des letzten Beispiels anzeigen können. Diese sind – bei entsprechender Kenntnis – in Kombination mit der Auswertung der Standardeigenschaften oft sehr gut geeignet, um einen bestimmten Browser dann genau zu erkennen.

7.12 Übungen

Übung 1: Neue Fenster öffnen und schließen

Übungsdatei: --

Ergebnisdatei: *kap07/uebung1.html*

1. Erstellen Sie eine Webseite, die in einer Tabelle einige Bilder verkleinert als Vorschau enthält. Nach einem Klick auf das Bild zeigen Sie in einem `img`-Tag unterhalb der Tabelle eine große Version des Bildes an. Verwenden Sie den folgenden Code in jedem ``-Tag in der Tabelle zum Aufruf der Funktion (veraltete HTML-Eventhandler, aber sie funktionieren immer noch in allen Browsern):
``
2. Sprechen Sie den Tag, in dem Sie die große Version des Bildes anzeigen wollen, über die ID an.

Übung 2: Eingabefenster und Zeitgeber verwenden

Übungsdatei: --

Ergebnisdatei: *kap07/uebung2.html*

1. Erstellen Sie analog der letzten Übung eine Webseite, die in einer Tabelle einige Bilder enthält. Nach einem Klick auf das Bild zeigen Sie in einem `div`-Tag unterhalb der Tabelle eine Zusatzinformation zu dem Bild an (etwa die URL). Verwenden Sie wieder einen `onclick`-HTML-Eventhandler beim `img`-Tag. Sprechen Sie den Tag, in dem Sie die Zusatzinformation des Bildes anzeigen wollen, über die ID an.
2. Lesen Sie über ein Eingabefenster (`prompt()`) die Anzahl von Sekunden ein, die für ein Bild die Zusatzinformation angezeigt bleiben soll, bevor sie wieder gelöscht wird. Zeigen Sie eine Meldung an, wenn die eingegebene Zeit nicht zwischen 1 und 20 Sekunden liegt. Implementieren Sie die Funktionalität zum Löschen der Information nach der angegebenen Zeit.

Übung 3: Dokumenteigenschaften auslesen und ändern

Übungsdatei: --

Ergebnisdateien: *kap07/uebung3.html,*
kap07/uebung3_1.html

1. Setzen Sie nach dem Laden einer Webseite deren Hintergrundfarbe auf Hellblau (das geht mit der Eigenschaft `document.bgColor`, wobei dies nur noch aus Übungsgründen gemacht werden sollte), und zeigen Sie die URL der Webseite an, von der Sie zum aktuellen Dokument gewechselt sind.
 Letzteres wird nur übermittelt, wenn sich Ihre Webseite auf einem Webserver befindet und durch einen Klick auf einen Link aufgerufen wurde.

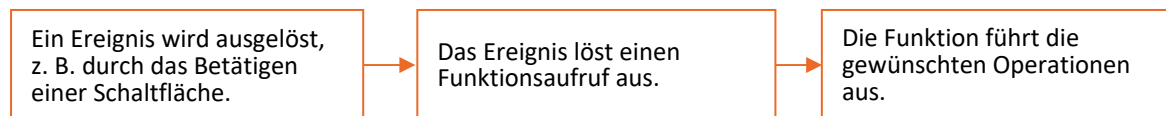
8

Ereignisse

8.1 Grundlagen zu Ereignissen

Es gibt die Möglichkeit, dass JavaScript-Anweisungen sofort nach dem Laden einer Webseite ausgeführt werden. Der Aufruf dieser Anweisungen wird direkt in den Skript-Bereich geschrieben. In der Praxis wird diese Möglichkeit nur für Initialisierungen und Deklarationen genutzt. Der Regelfall ist, dass Anweisungen erst nach Eintritt eines sogenannten **Ereignisses** abgearbeitet werden. Ein Ereignis erfolgt beispielsweise bei einem gezielten Aufruf einer Funktion durch einen Anwender (z. B. Anwender klickt eine Schaltfläche) oder einer maschinellen Aktion (z. B. Server schickt Daten an Browser). In JavaScript können Sie gezielt auf zahlreiche Ereignisse reagieren.

Tritt ein bestimmtes Ereignis ein, wird die angegebene Funktion aufgerufen, und die enthaltenen Anweisungen werden abgearbeitet.



Eine Webseite interaktiv zu gestalten, bedeutet also, auf die Aktivitäten des Benutzers und gewisser maschineller Ereignisse zu reagieren. Ein typisches Beispiel ist das Auswerten eines HTML-Formulars. JavaScript ermöglicht dabei das Abfangen von fehlerhaften Eingaben, bevor der Formularinhalt an den Webserver geschickt wird. Dies hilft, unnötigen Datentransfer im Internet zu vermeiden, die Server nicht durch Fehleingaben zu belasten und dem Benutzer Wartezeiten zu ersparen. Andere Aktivitäten sind z. B. vom Benutzer ausgeführte Mausbewegungen und -klicks.

Im **Document Object Modell** (DOM) besitzen alle sichtbaren HTML-Objekte Ereignisse (engl.: events) als Eigenschaften, auf die der Browser mit der Ausführung von JavaScript-Anweisungen reagieren kann.

8.2 Ereignisbehandlung

Zu einer Reaktion auf die verschiedenen Ereignisse, die in einem Browser verwertet werden können, verwenden Sie **Eventhandler**. Das Verfahren heißt **Ereignisbehandlung**.

Eventhandler

Ein Eventhandler-Ereignis wird in JavaScript als Eigenschaft von allen Objekten zur Verfügung gestellt, für die ein Ereignis überwacht werden kann. Das sind einmal die DOM-Objekte, aber es können auch unsichtbare Objekte sein, die etwa den Datenaustausch mit einem Server im Hintergrund verwalten (was man unter Ajax und manchen neuen HTML5-Techniken macht).

Die Eventhandler, die Sie für Knoten einer Webseite unter JavaScript verwenden, gehören direkt zum DOM-Konzept und nicht zu JavaScript selbst. Sie werden oft auch JavaScript-Eventhandler genannt und im Buch unter diesem Begriff verwendet. Für Objekte, die **nicht** zur DOM-Schnittstelle zählen, gehören die Eventhandler zu JavaScript (oder genau genommen dem jeweiligen Objekt, was in JavaScript bereitgestellt wird). Als Programmierer brauchen Sie diese Details nicht zu berücksichtigen, denn der Zugriff auf einen Eventhandler erfolgt in jedem Fall wie üblich über die Punktnotation.

```
document.onclick=meinefunktion;
```

Im Beispiel sehen Sie den Zugriff auf eine Eigenschaft `onclick` eines Objekts mit Namen `document`. Die Eigenschaft ist ein Eventhandler – eine Zuweisung einer Funktionsreferenz zu einer passenden Eigenschaft des Objekts, bei dem das Auslösen von dem Ereignis überwacht werden soll. Alternativ können Sie auch eine anonyme Funktion verwenden, zum Beispiel:

```
document.onclick=function() { ... };
```

In der Praxis werden in der Regel anonyme Funktionen verwendet.

Über die verschiedenen Versionen des DOM-Konzepts wurden immer wieder neue Eventhandler hinzugefügt. Die Eventhandler der neueren DOM-Versionen können in der Praxis mit etwas Vorsicht mittlerweile verwendet werden. Nachfolgend werden nur die Eventhandler angegeben, die in der Praxis sinnvoll sind.

HTML-Eventhandler

Früher hat man oft in Webseiten Eventhandler als Attribute in einen HTML-Tag notiert. Beispiel:

```
<h3 onclick='MeineFunktion()'>Überschrift</h3>
```

Diese Möglichkeit besteht weiterhin, sollte aber nicht verwendet werden. Nur mit JavaScript-Eventhandlern können Sie eine **saubere Trennung von Struktur und Funktionalität** gewährleisten. Die Vermischung von JavaScript-Aufrufen direkt in HTML-Tags verhindert diese Trennung. JavaScript-Eventhandler können sowohl in einen internen Skriptcontainer als auch in externe JavaScript-Dateien ausgelagert werden. Dazu kommt, dass es für jeden HTML-Eventhandler ein JavaScript-Gegenstück gibt. Es gibt jedoch Ereignisse, für die kein HTML-Eventhandler zur Verfügung steht.

8.3 Auf Ereignisse reagieren

Die verfügbaren Ereignisse, auf die Sie unter JavaScript reagieren können, kann man nach Kategorien unterteilen.

Fensterereignisse und Dokumentereignisse

Ein aktives Browserfenster stellt Ereignisse bereit, die zum `window`-Objekt gehören. Hier finden Sie eine Auflistung der wichtigsten Ereignisse:

Ereignis	Wird ausgelöst ...
<code>onbeforeload</code>	vor dem Laden der Webseite
<code>onblur</code>	wenn das Fenster den Fokus verliert
<code>onfocus</code>	wenn das Fenster den Fokus erhält
<code>onload</code>	beim Laden einer Webseite
<code>onunload</code>	beim Verlassen einer Webseite

Die wichtigen Ereignisse sind die Reaktion auf das Laden und Verlassen der Webseite. Sie werden mit `onload` und `onunload` behandelt. Die Eventhandler werden direkt beim `window`-Objekt notiert.

Beispiel

```
window.onload=Funktionsname;
```



Die Eventhandler könnten theoretisch auch bei anderen Elementen wie Grafiken verwendet werden. Diese Möglichkeit wird in der Praxis nicht genutzt. Es gibt im aktuellen DOM-Konzept ebenso einige weitere Dokumentereignisse (`onabort`, `onerror`, `onresize` oder `onscroll`), die bei vielen noch in der Praxis eingesetzten Browsern nicht funktionieren und deren Verwendung nicht zu empfehlen ist.

Fehler beim Zugriff auf den DOM

Der Eventhandler `onload` wird zum Schutz des DOM-Baums genutzt. Dieser Schutz ist besonders wichtig. Es ist in JavaScript ein großes Problem, wenn Sie auf Bestandteile des DOM-Baums zugreifen, bevor der DOM vollständig aufgebaut ist.

Beispiel: *falscher_dom_zugriff.html*

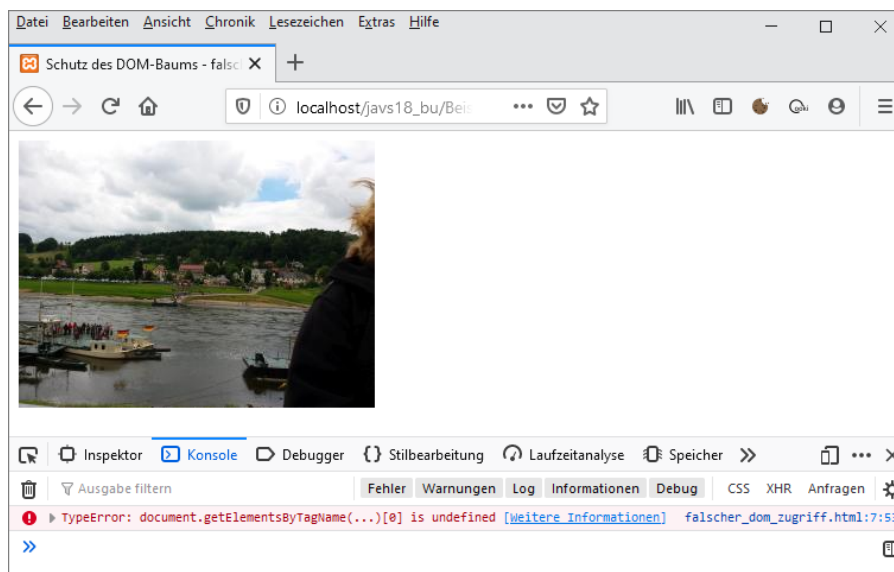
Angenommen, Sie wollen eine Grafik manipulieren, bevor der Browser deren DOM-Objekt erstellt hat. Der folgende Code macht den Fehler deutlich.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Schutz des DOM-Baums - falscher Zugriff</title>
  <script type="text/javascript">
    ① document.getElementById("info").innerHTML =
    ② document.getElementsByTagName("img")[0].alt;
  </script>
</head>
<body>
    ③ 
    ④ <div id="info"></div>
</body>
</html>

```

- ① Mit `document.getElementById("info")` wird in dem Skript beim Laden der Webseite auf das DOM-Objekt zugegriffen, das den DIV-Container repräsentiert, der erst später in der Webseite definiert wird. Das kann zu dem Zeitpunkt nicht funktionieren, und das Skript bricht mit einem Fehler ab.
- ② Mit `document.getElementsByTagName("img")[0]` wird auf das erste Bild in der Webseite zugegriffen. Zu dem Zeitpunkt gibt es im DOM noch kein Objekt, das dieses Bild repräsentiert. Auch das kann nicht funktionieren! Es gibt somit zwei fehlerhafte DOM-Zugriffe in dem Beispiel.
- ③ Hier wird erst das Bild per HTML definiert, und der Browser kann das DOM-Objekt aufbauen.
- ④ Nun wird erst das DOM-Objekt für den DIV-Container erzeugt. Die JavaScript-Aufrufe, die auf diese DOM-Objekte zugreifen, könnten erst jetzt durchgeführt werden.



Fehlermeldung in der Web-Konsole bei falschen DOM-Zugriffen

Eine Initialisierungsfunktion mit `onload`

DOM-Zugriffe dürfen nicht zu früh verwendet werden. Früher wurden deshalb der Skriptbereich bzw. die Skriptreferenz mit den Zugriffen auf DOM-Objekte erst **am Ende der Webseite eingebunden**, zum Beispiel:

```
<!DOCTYPE html>
<html>
<head>
...
</head>
<body>
  // normale Webseite
① <script type="text/javascript" src="..."></script>
</body>
</html>
```

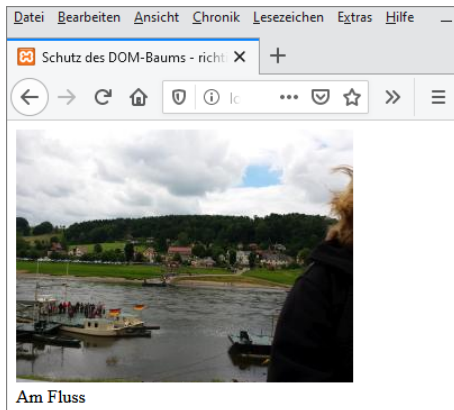
- ① Die Referenz auf die externe Skript-Datei oder den Skript-Container mit dem DOM-Zugriff ist am Ende des Body-Bereichs notiert.

Diese Möglichkeit besteht heute immer noch. Eleganter können Sie das Problem mit der Initialisierungsfunktion des Eventhandlers `onload` lösen. Der Eventhandler `onload` erlaubt eine Initialisierungsfunktion zu erstellen, wobei der Eventhandler in einigen alten Browsern (insbesondere in älteren Versionen des Internet Explorers) fehlerhaft implementiert ist. Die Probleme sind in neueren Browsern nicht mehr vorhanden. Die Notation von Skripten am Ende der Webseite findet man derzeit dennoch wieder häufiger – wenngleich aus anderen Gründen. Man kann damit unter gewissen Konstellationen das Laden relevanter Bestandteile einer Webseite vorziehen gegenüber später benötigten Bestandteilen.

Die folgende Abwandlung des letzten Beispiels zeigt den Weg, wie Sie heutzutage beim Laden einer Webseite auf den DOM zugreifen.

```
<script type="text/javascript">
...
① window.onload=function() {
②   document.getElementById("info").innerHTML =
      document.getElementsByTagName("img")[0].alt;
    }
  </script>
</head>
...
```

- ① **Alle** DOM-Zugriffe, die beim Laden der Webseite ausgeführt werden, werden in einer Funktion durchgeführt, die mit `onload` erst nach dem Fertigstellen des DOM-Baums aufgerufen wird. Das gewährleistet den Schutz des DOM-Baums.
- ② Im Inneren der Funktion kann man nun gefahrlos mit `document.getElementById("info")` auf den DIV-Container und mit `document.getElementsByTagName("img")[0]` auf das erste Bild in der Webseite zugreifen.



Nun funktioniert der Zugriff.

Mausereignisse

Klicks auf Referenzen mit der Maus sowie das Überstreichen eines Elements einer Webseite mit dem Mauszeiger sind sehr oft genutzte Ereignisse, um ein JavaScript aufzurufen. In der nachfolgenden Tabelle sehen Sie Ereignisse, auf die Sie reagieren können und die in der Praxis breite Unterstützung in den aktuellen Browsern haben:

Event	Wird ausgelöst, wenn ...
onclick	ein Klick in den Bereich eines Elements erfolgt
ondblclick	ein Doppelklick in den Bereich eines Elements erfolgt
onmousedown	eine Maustaste in dem Bereich eines Elements gedrückt wird
onmouseenter	die Maus in den Bereich eines Elements eintritt
onmouseleave	die Maus den Bereich eines Elements verlässt
onmousemove	der Mauszeiger bewegt wird, solange er sich im Bereich eines Elements befindet
onmouseout	die Maus den Bereich eines Elements oder eines enthaltenen Kindelements verlässt
onmouseover	der Mauszeiger bewegt wird, solange er sich im Bereich eines Elements oder eines enthaltenen Kindelements befindet
onmouseup	eine Maustaste in dem Bereich eines Elements wieder losgelassen wird

Beispiel

```
document.getElementsByTagName("h1")[0].onclick=Funktionsname;
```

Es gibt eine Reihe von Eventhandlern, die auf Drag & Drop-Aktionen (Verschieben und Loslassen von Elementen) reagieren (ondragstart, ondrag, ondragenter, ondragleave, ondragover, ondrop, ondragend). Allerdings ist die eigentliche Programmierung von Drag & Drop-Aktionen nicht trivial, und deshalb werden Sie vermutlich selten mit diesen Eventhandlern in Kontakt kommen.

Beispiel für einen Rollover-Effekt: *bildwechsel.html*

Ein Einsatzgebiet der beiden Ereignisse `onmouseout` und `onmouseover` ist die gemeinsame Verwendung bei einem Rollover-Effekt. Rollover bedeutet, dass eine Grafik sich in dem Moment ändert, in dem die Maus über sie bewegt wird. Verlässt die Maus die Grafik, kehrt die Grafik zur ursprünglichen Anzeige zurück. Diesen Effekt finden Sie in den Navigationsbereichen von Webseiten.

Das Beispiel zeigt, wie Sie einen einfachen Rollover-Effekt erzeugen. Dabei werden zwei gleich große Grafiken gegeneinander ausgetauscht.

```
1 <script type="text/javascript"><...
2   window.onload=function() {
3     document.images[0].src="images/b3.jpg";
4     document.getElementById("b1").onmouseover=function() {
5       document.images[0].src="images/b1.jpg";
6     }
7     document.getElementById("b1").onmouseout=function() {
8       document.images[0].src="images/b2.jpg";
9     }
10  }
11  </script>
12 </head>
13 <body>
14   <h3>onmouseover und onmouseout</h3>
15   <img alt="Grafikwechsel" title="Grafikwechsel" id="b1" />
16 </body>
17 </html>
```

- ① Da auf den DOM zugegriffen werden soll, wird mit `window.onload` eine anonyme Initialisierungsfunktion erstellt. Das Bild selbst ist mit HTML in der Webseite vorhanden, die `src`-Eigenschaft wurde explizit **nicht** mit HTML gesetzt.
- ② Das Bild wird per JavaScript gefüllt, indem die Bildeigenschaft `src` mit dem Wert `"images/b1.jpg"` gefüllt wird.

onmouseover und onmouseout



Die Webseite nach dem Laden

- ③ Das Bildobjekt, das eine ID `b1` hat, wird mit `document.getElementById()` angesprochen, und beim Eventhandler `onmouseover` wird eine anonyme Funktion registriert.
- ④ Das Bild auf der Webseite wird durch das neue Bild ersetzt.
- ⑤ Analog wird für den Eventhandler `onmouseout` vorgegangen. Die für den Rollover-Effekt verantwortlichen `onmouseover`- und `onmouseout`-Handler werden gemeinsam beim Bildobjekt registriert. Beim Überfahren mit der Maus (`onmouseover`) wird die eine Grafik angezeigt, beim Verlassen mit der Maus (`onmouseout`) die andere Grafik.

onmouseover und onmouseout



Die Grafik bei `onmouseout`

onmouseover und onmouseout



Die Grafik bei `onmouseover`

Tastaturereignisse

Die Reaktion auf das Betätigen einer (beliebigen) Taste auf der Tastatur ist eine weitere Situation, auf die Sie reagieren müssen. Man unterscheidet drei verschiedene Ereignisse.

Event	Wird ausgelöst, wenn ...
<code>onkeydown</code>	eine Taste gedrückt wird
<code>onkeyup</code>	eine Taste losgelassen wird
<code>onkeypress</code>	eine Taste gedrückt gehalten wird. Nach dem Drücken einer Taste steht der Tastaturcode in einem Tastaturpuffer zur Verfügung

Formularereignisse

Es gibt eine Reihe von Ereignissen, die im Zusammenhang mit Webformularen sinnvoll sind. Das sind zum einen die Änderung, die Aktivierung und das Verlassen eines Elements in einer Webseite. Diese Aktionen sind bei Elementen zur Entgegennahme von Benutzereingaben sinnvoll. Zum anderen geht es um das Verschicken von Formulareingaben bzw. das Zurücksetzen des Formulars.

Event	Wird ausgelöst, wenn ...
<code>onblur</code>	ein Element den Fokus verliert
<code>onchange</code>	ein Element geändert wurde

Event	Wird ausgelöst, wenn ...
onfocus	ein Element den Fokus bekommt
onreset	ein Formular zurückgesetzt wird
onselect	ein Element selektiert wird
onsubmit	ein Formular abgeschickt wurde

8.4 Das Ereignisobjekt `event`

Eine Eigenschaft des `window`-Objekts ist das `event`-Objekt (`event` = Ereignis). Mit diesem Objekt können Sie auf Ereignisse wie Mausklicks oder Tastatureingaben reagieren und JavaScript-Code ausführen, der bestimmte Informationen in dem Ereignisobjekt verwendet. Beispielsweise können Sie bei einem Mausklick ermitteln, ob die linke oder rechte Maustaste gedrückt wurde, oder bei einem Tastendruck die betätigte Taste ermitteln.

Das Ereignisobjekt bzw. `event`-Objekt, das hinter der Ereignisbehandlung in JavaScript liegt, ist ein Mitteilungsobjekt mit Informationen über die Art des ausgelösten Ereignisses. Solche Ereignisobjekte werden vom Browser unablässig erzeugt, und ein solches Objekt kann bei der Reaktion auf Ereignisse gezielt verwertet werden. Dabei dürfen Sie nicht nur an die wenigen offensichtlichen Ereignisse wie den Klick des Besuchers mit der Maus oder eine Tastatureingabe denken. Das einfache Verschieben des Mauszeigers über den Bereich des Browsers findet quasi permanent statt.

Dabei können in kürzester Zeit Tausende von Ereignisobjekten entstehen. Jedes Verschieben einer Mauszeigerposition um wenige Pixel bewirkt in der Regel bereits das Erzeugen von einem eigenständigen Ereignisobjekt. So kann beispielsweise das Scrollen eines Textes um wenige Millimeter bereits mehrere Ereignisobjekte generieren.

Das Ereignisobjekt stellt zur gezielten Reaktion eine Reihe interessanter Eigenschaften mit spezifischen Informationen wie auch diverse Methoden bereit. Betrachten Sie als Beispiel die Situation, wenn ein Anwender mit der Maus in irgendeinen Bereich der Webseite klickt. Bei einem Mausklick erzeugt der Browser ein Ereignisobjekt, das u. a. folgende Informationen enthält:

- ✓ die verwendete Maustaste
- ✓ eventuell gedrückte Zusatz Tasten
- ✓ die Koordinaten des Klicks

Andere Ereignisobjekte, die bei weiteren Ereignissen erzeugt werden, beinhalten natürlich andere Informationen, die dem Ereignis angepasst sind. So steht beispielsweise bei einem Tastendruck die gedrückte Taste als abzufragende Information bereit. Allgemein beinhaltet ein Ereignisobjekt zahlreiche sinnvolle Informationen, die Sie zur Erstellung gut angepasster Applikationen nutzen können.

Event-Bubbling

Es gibt im Zusammenhang mit der Behandlung des Ereignisobjekts den Begriff der **Bubble-Events** bzw. der Bubble-Phase. Wenn ein Ereignis bei einem Knoten in einer Baumstruktur wie dem DOM-Baum auftritt, stellt sich die Frage, welches der Objekte im Baum nun für ein aufgetretenes Ereignis zuständig ist. Knoten sind im Baum sehr tief ineinander verschachtelt. Wenn ein Anwender auf ein Bild in einer Webseite klickt, hat er ja auch gleichzeitig auf die Webseite selbst geklickt. Ist die Webseite oder das Bild für die Behandlung zuständig? Und wie erfährt der Knoten der Webseite ggf. davon, dass auf den untergeordneten Knoten des Bildes ein Klick erfolgt ist, der dort nicht behandelt wurde? Oder was soll passieren, wenn mehrere ineinander geschachtelte Objekte den gleichen Eventhandler besitzen und deshalb auf das Ereignis reagieren könnten?

Die Probleme werden über das sogenannte **Event-Bubbling** gelöst. Ein Ereignis wird bei diesem Konzept immer zuerst im **innersten** Element behandelt, bei dem es aufgetreten ist – sofern es dort einen geeigneten Eventhandler gibt! Ein behandeltes Ereignisobjekt wird nach der Handhabung in einem Eventhandler sofort vernichtet.

Das innerste Element ist in der Hierarchie des DOM-Baums am weitesten von der Wurzel entfernt. Hat dieses Element keinen geeigneten Eventhandler, wird das Ereignisobjekt an das nächsthöhere Objekt im DOM-Baum (dem direkten Vorfahren) weitergeleitet usw. Es steigt wie eine Blase (Bubble) über alle Vorfahren hinauf bis zur Wurzel. Es „blubbert“ durch den Baum, bis es behandelt wird.

Wenn es bis zur Wurzel keine Behandlung von dem Ereignisobjekt gab, wird das Ereignisobjekt bei einer Variante des Event-Bubbings dort vernichtet. Allerdings gibt es für das Event-Bubbling auch eine zweite Variante, bei der ein unbehandeltes Ereignisobjekt nach Erreichen der Wurzel wieder bis zum auslösenden Element zurückblubbert und auf dem Weg zurück behandelt werden kann. Dazu werden Sie eine geeignete Methode kennenlernen.

Dieses Bubbling der Ereignisse können Sie in alten Internet Explorern mit der Eigenschaft `cancelBubble` unterbinden, indem Sie den Wert auf `true` setzen. In allen anderen Browsern wird zum Unterbinden von Bubbling die Methode `stopPropagation()` aufgerufen.

8.5 Das Ereignisobjekt verwenden

Um das Ereignisobjekt zu verwenden, mussten Sie bis vor einigen Jahren zwei inkompatible Modelle zur Ereignisbehandlung unterscheiden:

- ✓ die alte Ereignisbehandlung von Microsoft
- ✓ die vom W3C standardisierte Ereignisbehandlung

Das standardisierte Konzept für die Reaktion per JavaScript auf das Auftreten von einem Mitteilungsobjekt innerhalb des Browsers geht in vielen Aspekten auf die Firma Netscape zurück. Das Modell wird mittlerweile von allen modernen Browsern unterstützt und das proprietäre Microsoft-Modell spielt in der Praxis keine Rolle mehr.

Der entscheidende Unterschied beider Modelle ist, dass im Standardmodell der **erste Parameter** einer Ereignisfunktion immer das `event`-Objekt repräsentiert.

Beispiel für die anonyme Behandlung:

```
document.getElementsByTagName("h1")[0].onclick = function(evt) {
// evt ist das event-Objekt in der Funktion als lokale Variable
};
```

Beispiel für die benannte Behandlung mit einer Funktionsreferenz:

```
document.getElementsByTagName("h1")[0].onclick =
    benannteEreignisfunktion;

...
function benannteEreignisfunktion (evt) {
    // evt ist das event-Objekt in der Funktion
};
```



Beide Ereignismodelle beinhalten einige veraltete Eigenschaften und Methoden, die in modernen Browsern entweder nicht mehr unterstützt werden oder nicht eingesetzt werden sollten, z. B. die Methode `captureEvents()`. Die Verwendung dieser alten Techniken ist unnötig, denn die hier vorgestellten Techniken beinhalten alles, was man braucht.

Das standardisierte Ereignismodell

Im standardisierten Ereignismodell gibt es folgende Eigenschaften:

Eigenschaft	Bedeutung
offsetX, offsetY	Koordinaten des Ereignisses bezogen auf das auslösende Objekt oder Element
pageX, pageY	Koordinaten des Ereignisses bezogen auf den Dokumentbereich des Browsers
screenX, screenY	Koordinaten des Ereignisses bezogen auf den gesamten Bildschirm
target	Referenz auf das eigentliche Zielobjekt des Ereignisses
type	Zeichenkette mit dem Namen des Ereignisses. Der Name des Ereignisses entspricht dem Namen des auslösenden Eventhandlers, wenn Sie das „on“ weglassen, z. B. "click" bei einem onclick-EventHandler.
which	Ganzzahl, die bei Maus-Ereignissen den Button, bei Tasten den ASCII-Code der Taste repräsentiert. Der Wert 1 steht bei einem Mausereignis für die linke Maustaste, 2 für die mittlere und 3 für die rechte Maustaste.

Beispiel: *tastaturauswertung_standard.html*

Hier folgt ein Beispiel zur Auswertung des Tastaturcodes. In diesem Beispiel soll die folgende einfache Situation realisiert werden: Der Besucher drückt eine beliebige Taste, und der Tastaturcode der gedrückten Taste wird mittels `getElementById().innerHTML` in der Webseite angezeigt.

```

<script type="text/javascript">
① window.onkeypress = function(ev) {
②   document.getElementById("antwort").innerHTML =
      "Gedrückter Tastencode: " + ev.which;
    }
</script>
</head>
<body>
  <h1>Drücken Sie eine beliebige Taste!
  </h1><div id="antwort"></div>
</body>

```

- ① Im Skript wird das Ereignis `onkeypress` beim `window`-Objekt überwacht (Drücken irgendeiner Taste). Wenn der Anwender eine Taste drückt, wird die anonyme Callback-Funktion aufgerufen. Nach dem Niederdrücken einer Taste wird der Tastaturcode in dem Tastaturpuffer zur Verfügung gestellt.
- ② Der erste Parameter, der an die Ereignisfunktion übergeben wird, ist in dem Standardereignismodell per Vorstellung **immer** eine Referenz auf das Ereignisobjekt. Die Eigenschaft `which` des Ereignis-Objekts enthält den Tastaturcode. In der Funktion wird der aktuelle Wert bei jedem Tastendruck in dem `<div>`-Element ausgegeben (der numerische Code, der über die Eigenschaft `which` verfügbar ist).



Eine Taste wurde gedrückt.

Die Methode `addEventListener()` als Alternative zu Eventhandlern

Es gibt eine zentrale Funktion, mit der man auf Ereignisobjekte reagieren kann – ein sogenannter **Event Listener**, wie er in Sprachen wie Java oder C# üblich ist. Die Methode heißt `addEventListener()`. Die Methode erwartet drei Parameter:

- ✓ Der erste Parameter gibt an, welcher Ereignistyp überwacht werden soll. Diese werden im Wesentlichen wie die Eventhandler benannt, nur entfällt das vorangestellte „on“, z. B. `click`, `mouseover`, `mousedown`, `mouseup`, `mousemove` oder `keyup`. Die Ereignisse werden als Strings notiert.
- ✓ Der zweite Parameter ist die Funktion, die beim Eintreten des Events aufgerufen werden soll.
- ✓ Der dritte Parameter gibt an, ob das Ereignis beim Aufsteigen in der Bubble-Phase aufgefangen werden soll (`true`) oder erst beim Weg zurück.

Beispiel: *tastaturauswertung_addeventlistener.html*

Hier ist ein Beispiel mit der Methode `addEventListener()`, in der der Tastaturcode ausgewertet und der Anzeigebereich direkt nach dem Loslassen der Taste wieder gelöscht wird. Sie müssen somit auf zwei Ereignisse reagieren.

```
<script type="text/javascript">...
① window.addEventListener("keypress",function(ev) {
    document.getElementById("antwort").innerHTML =
        "Gedrückter Tastencode: " + ev.which;
    },true);
② window.addEventListener("keyup",function(ev) {
    document.getElementById("antwort").innerHTML = "";
    },true);
</script>
</head>
<body>
    <h1>Drücken Sie eine beliebige Taste!
    </h1><div id="antwort"></div>
</body>
```

- ① Die Ereignismethode wird beim `keypress`-Event für das gesamte Browserfenster registriert. Nach dem Drücken einer Taste wird der Tastaturcode im Tastaturpuffer zur Verfügung gestellt und in der Ereignisfunktion verwertet, die als zweiter Parameter angegeben wird.
- ② Eine weitere Ereignismethode wird registriert, allerdings beim `keyup`-Event, das beim Loslassen einer Taste ausgelöst wird. Wenn das Ereignis ausgelöst wird, wird in der Ereignisfunktion der Anzeigebereich im Browser wieder geleert.

 **Ergänzende Lerninhalte:** *Eventhandler.pdf*

8.6 Übungen

Übung 1: Theoriefrage zu Events

Übungsdatei: --

Ergebnisdatei: *kap08/uebung1.pdf*

1. Erklären Sie, welche Benutzeraktionen mit den folgenden Eventhandlern verbunden sind:

- | | | |
|------------|--------------|------------|
| ✓ onabort | ✓ onmouseout | ✓ onload |
| ✓ onchange | ✓ onfocus | ✓ onunload |
| ✓ onclick | | |

Übung 2: Reagieren auf Benutzeraktionen

Übungsdatei: --

Ergebnisdatei: *kap08/uebung2.html*

1. In dieser Übung sollen Sie Informationen zu einem Mausklick nach dem Standardereignismodell auswerten. Die Ereignisbehandlung wird per Funktionsreferenz an das `document`-Objekt gebunden. Auf das Loslassen der Maustaste soll reagiert werden (`document.onmouseup`). Zu dem Zeitpunkt stehen spezifische Informationen über den Mausklick zur Verfügung.
2. Bei dem überwachten Eventhandler `onmouseup` soll eine Funktionsreferenz die Funktion `pos()` referenzieren. In der Funktion setzen Sie einen Antworttext mit diversen Werten aus dem Ereignisobjekt nach dem Standardereignismodell zusammen. Diese sollen in einem DIV-Container in der Webseite mit `innerHTML` ausgegeben werden.

9

Formulare

9.1 Grundlagen zu Formularen

Ein Formular wird zur Eingabe von Daten verwendet, die verarbeitet werden sollen. Die Auswertung kann dabei im Client per JavaScript oder nach dem Absenden auf dem Server erfolgen. Für die Auswertung auf dem Server werden die Daten an ein Skript oder Programm auf dem Server geschickt, das meist in PHP, ASP.NET, Java oder auch serverseitigem JavaScript erstellt wurde. In diesem Fall müssen die Adresse des Programms (`action`), die für die Übertragung zu verwendende Codierung der Daten (`encoding`) und das zu verwendende Übertragungsverfahren (`method`) bekannt sein.

Struktur eines Formulars

Wie die Daten eingegeben werden, ist von den Eingabeelementen im Dokument abhängig. Im Zusammenhang mit den Formularen in einem HTML-Dokument besitzen die Elemente des `forms`-Arrays weitere Unterobjekte, die den Unterelementen des `<form>`-Tags in HTML entsprechen.

Den Aufbau des Formulars im HTML-Quelltext finden Sie in den Eigenschaften und Methoden der Elemente in der Objektrepräsentation des Formulars wieder. Die folgende Tabelle listet die Eigenschaften und Methoden eines einzelnen Formularobjekts zur Rekapitulation auf.

Eigenschaft	Bedeutung
<code>action</code>	Die URL des Skripts, das die Daten verarbeiten soll
<code>elements</code>	Ein Objektfeld der Elemente des Formulars. Sie werden in der Reihenfolge abgelegt, wie sie im Formular erscheinen.
<code>encoding</code>	Die Kodierung der Daten für die Übertragung (z. B. <code>"text/plain"</code>)
<code>method</code>	Die Art der Datenübertragung (z. B. <code>"post"</code> oder <code>"get"</code>)
<code>name</code>	Der Namen des Formulars, wenn die Eigenschaft <code>name</code> gesetzt wurde
<code>target</code>	Das Zielfenster, in dem die Rückmeldung des Servers erscheinen soll

Methode	Bedeutung
reset()	Damit werden alle Einträge des Benutzers im Formular auf den Anfangswert zurückgesetzt. Der Aufruf der Methode entspricht dem Klick des Benutzers auf die <i>Abbrechen</i> -Schaltfläche.
submit()	Mit der Methode werden die Daten des Formulars gesendet. Der Aufruf der Methode entspricht dem Klick des Benutzers auf die <i>Absenden</i> -Schaltfläche.

Beispiel: *formular_setzen.html*

Im folgenden Beispiel sollen die Einstellungen für ein Formular mit JavaScript gesetzt werden.

	<pre> <body> <h3>Auslesen der Formulardaten</h3> ① <form id="form1"> <input name="vname"/> Vorname
 <input name="nname"/> Nachname
 <input type="submit"/> </form> <script type="text/javascript"> ② with(document.getElementById("form1")) { ③ action="test.php"; method="get"; enctype="text/plain"; } </script> </body> </pre>
--	---

- ① Die Webseite definiert mit HTML ein Formular-Tag, dessen Attribute nicht per HTML gesetzt werden.
- ② Mit `document.getElementById("form1")` wird das Formular selektiert. Über die `with()`-Anweisung sparen Sie in dem nachfolgenden Block mit dem Setzen der Eigenschaften, dass Sie bei den verwendeten Eigenschaften jedes Mal `document.getElementById("form1")` voranstellen müssen.
- ③ Die drei Eigenschaften für das Ziel der Daten, die Methode und die Kodierung werden mit JavaScript gesetzt.

Auslesen der Formulardaten

Hans Vorname

Dampf Nachname

Das Formular vor dem Versenden

Array ([vname] => Hans [nname] => Dampf)

Das Ziel der Formulardaten, ein PHP-Skript, hat eine Antwort zum Client geschickt.

9.2 Gemeinsame Methoden und Eigenschaften von Formularelementen

Die Funktionalität und Interaktivität eines Formulars verbirgt sich in den einzelnen Elementen, die im `elements`-Array bzw. dem entsprechenden `node`-Knoten im DOM bereitstehen. Der Zugriff erfolgt über die Array-Notation (z. B. das vierte Element über `document . Form-Name . elements [3] . Eigenschaft`) oder in der Regel über die ID oder den Tag-Namen. Die meisten Formularelemente besitzen eine Reihe an gemeinsamen Methoden.

Die Tabelle zeigt eine Aufstellung wichtiger Methoden, die bei (fast) allen Formularelementen vorkommen.

Methode	Bedeutung
<code>blur()</code>	Verlassen eines Feldes
<code>focus()</code>	Diese Methode setzt den Fokus in das entsprechende Feld.
<code>select()</code>	Mit <code>select()</code> können Sie den Inhalt des Feldes auswählen.
<code>click()</code>	Auslösen eines Klicks auf einem Element

Als gemeinsame Eigenschaft besitzen Formularelemente fast immer die Eigenschaft `value`. Hiermit greifen Sie auf den Inhalt des Eingabefeldes zu. Darüber hinaus sind in der Praxis die folgenden Eigenschaften von Bedeutung.

Eigenschaft	Bedeutung
<code>enabled</code>	Die Angabe, ob das Eingabefeld aktivierbar ist
<code>form</code>	Zugriff auf das umgebende Formular-Objekt, zu dem das Element gehört
<code>type</code>	Der Typ des Formularelements – etwa <code>text</code> , <code>textarea</code> , <code>hidden</code> , <code>button</code> oder <code>password</code>

9.3 Eingabefelder und Schaltflächen

Alle Standardeingabefelder (Texteingabe, versteckte Eingabefelder, Passwortfelder) und Schaltflächen (`type`-Angabe `submit`, `reset` und `button`) besitzen nur die gemeinsamen Eigenschaften und Methoden. So wichtig diese Elemente auch sind, so einfach sind sie.

9.4 Kontroll- und Optionsfelder

Besondere Formularelemente sind Kontrollkästchen (engl.: checkbox) und Optionsfelder (engl.: radio buttons) zur Auswahl einer vorgegebenen Möglichkeit. Der Unterschied in der Verwendung der beiden Felder besteht darin, dass über Kontrollfelder genau eine Einstellung vorgenommen werden soll (ja/nein). Optionsfelder werden innerhalb einer Gruppe zusammengefasst, und es kann nur eine Option der Gruppe gewählt werden (z. B. eine Auswahl der Pizza: entweder groß, klein oder mittelgroß).

Kontroll- und Optionsfelder

Zueinander gehörige Kontroll- oder Optionsfelder werden intern in einem jeweiligen Array verwaltet. Den Zugriff auf diese Elemente erhalten Sie über die entsprechenden Objekte, die neben den Standardeigenschaften und -methoden die folgenden speziellen Eigenschaften besitzen.

Eigenschaft	Bedeutung
checked	Festlegung, ob ein Kontroll- oder Optionsfeld markiert ist (<code>true</code> = ja, <code>false</code> = nein)
defaultChecked	Festlegung, ob das Kontrollfeld beim Laden der Webseite bereits aktiviert ist
defaultSelected	Festlegung, ob eine Option beim Laden der Webseite standardmäßig ausgewählt ist

9.5 Auswahllisten

Auswahllisten sind ebenso spezielle Formularelemente und ermöglichen eine Selektion von einer oder mehreren Optionen (HTML-Schlüsselwort `multiple`). Diese Listen werden in HTML mit dem HTML-Tag `<select>` erzeugt und in JavaScript durch gleichnamige `select`-Objekte repräsentiert, die neben den Standardeigenschaften und -methoden die folgenden speziellen Eigenschaften besitzen.

Eigenschaft	Bedeutung
options	Ein untergeordnetes Objektfeld, das die einzelnen Optionen der Auswahlliste in der Reihenfolge des Auftretens beinhaltet. Der Zugriff auf die Elemente wird im nächsten Abschnitt erklärt.
selectedIndex	Der Index der ausgewählten Option – bei <code>type= "select-multiple"</code> wird der erste ausgewählte Eintrag zurückgegeben.

Das **options**-Array

Die Einträge der Auswahlliste sind über das Objektfeld `options` erreichbar. Die Listenelemente besitzen folgende spezielle Eigenschaften:

Eigenschaft	Bedeutung
<code>defaultSelected</code>	Festlegung, ob eine Auswahl voreingestellt ist
<code>index</code>	Die Position des Eintrags in der Liste. Die erste Position ist 0.
<code>selected</code>	Die vom Benutzer vorgenommene Auswahl. Als Wert wird <code>true</code> oder <code>false</code> zurückgeliefert.
<code>selectedIndex</code>	Sie erhalten den Indexwert der aktuell ausgewählten Option.
<code>text</code>	Der Text, der als Auswahl angezeigt wird
<code>value</code>	Man kann bei einer Auswahl zu dem Text noch einen zusätzlichen Wert angeben, der dann über diese Eigenschaft <code>value</code> bereitsteht und auch an den Server übermittelt wird. Wird diese Eigenschaft nicht explizit gesetzt, wird sie aus dem Text des <code>option</code> -Containers bzw. der Eigenschaft <code>text</code> vorbelegt.

Beispiel: *auswahlliste.html*

Die Funktion `durchlauf()` zeigt alle Eigenschaften eines `option`-Elements an.

```

<script type="text/javascript"><...
⑥ window.onload = function() {
⑤   durchlauf();
⑦   document.getElementsByName("auswahl")[0].onchange =
    durchlauf;
};
function durchlauf() {
    var text = "";
    var liste = document.getElementsByName("auswahl")[0];
①   for ( i = 0; i < liste.length; i++) {
        text += 'index: ' + liste.options[i].index + '<br />' +
            'defaultSelected: ' + liste.options[i].defaultSelected
            + '<br />' + 'selected: ' + liste.options[i].selected
            + '<br />' + 'text: ' +
            liste.options[i].text + '<br />' + 'value: ' +
            liste.options[i].value + '<br />' + '<hr />';
    }
③   document.getElementById("ausgabe").innerHTML = text;
}
</script>
</head>
<body>
    <h3>Anzeige der Eigenschaften der ausgewählten Elemente</h3>

```

```

④ <form id="form1">
    <select name="auswahl" size="4" multiple>
        <option value="Hahn">Der laute Hahn</option>
        <option value="Katze" selected>Die sanfte Katze</option>
        <option value="Hund">Der treue Hund</option>
        <option>Der schlaue Esel</option>
    </select><hr />
</form>
② <div id="ausgabe"></div>

```

JavaScript-Quellcode zur Ausgabe der Eigenschaften der einzelnen Listeneinträge

- ① Dazu werden über eine `for`-Schleife die Eigenschaften jedes Eintrags des Formularelements mit dem Namen `auswahl` ausgelesen und eine Zeichenkette erzeugt.
- ② Diese wird in dem Div-Container mit der ID `ausgabe` über `innerHTML` zugewiesen und ausgegeben ③.
- ④ Die Auswahlliste befindet sich in einem HTML-Formular mit der ID `form1`.
- ⑤ Die Funktion `durchlauf()` wird nach dem Laden der Webseite aufgerufen, da in der Initialisierungsfunktion ⑥ der Funktionsaufruf als erste Anweisung notiert wird.
- ⑦ Nach jeder geänderten Auswahl wird die Anzeige ebenfalls aktualisiert, da in dieser Initialisierungsfunktion ein Callback auf die Funktion erfolgt, bei der die Liste über eine Funktionsreferenz registriert wird.

Beachten Sie die vierte Auswahlmöglichkeit in der Liste. Hier wurde in HTML explizit **kein** `value`-Attribut angegeben. In dem Fall wird der Text im Inneren des `option`-Containers als Wert der Eigenschaft `value` übernommen.

Anzeige der Eigenschaften der ausgewählten Elemente

index: 0
 defaultSelected: false
 selected: true
 text: Der laute Hahn
 value: Hahn

index: 1
 defaultSelected: true
 selected: false
 text: Die sanfte Katze
 value: Katze

index: 2
 defaultSelected: false
 selected: false
 text: Der treue Hund
 value: Hund

index: 3
 defaultSelected: false
 selected: false
 text: Der schlaue Esel
 value: Der schlaue Esel

Angabe der Eigenschaften (auswahlliste.html)

9.6 Eingaben prüfen

Sehr oft wird man mit JavaScript prüfen, ob die Felder eines Formulars korrekt ausgefüllt wurden.

Ein Pflichtfeld überprüfen

Ein erstes einfaches Beispiel zeigt eine erste Anwendung der Überprüfung von Formularfeldern. Sie erstellen dort ein Formular mit nur einem Eingabefeld und einer Schaltfläche zum Abschicken der Daten. Vor dem Abschicken der Formulardaten wird das Ereignis `onsubmit` ausgelöst und dadurch die Funktion `testen()` ausgeführt, die die Eingaben überprüft. Das Feld wird als **Pflichtfeld** verstanden, und ein Anwender muss zwingend etwas eingeben, bevor die Daten verschickt werden können.

Beispiel: *form_tester.html*

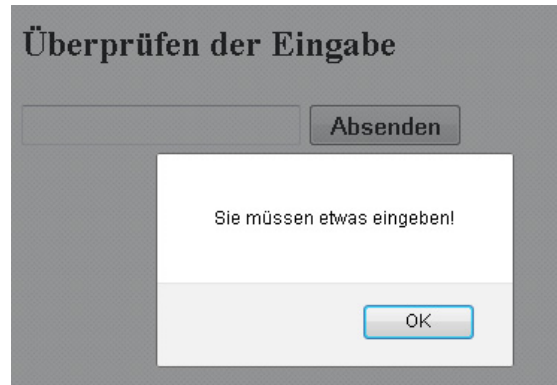
Zur Überprüfung der Eingabe wird das Ereignis `onsubmit` verwendet.

```

<script type="text/javascript"><...
  window.onload = function() {
    ① document.getElementById("formular").onsubmit = testen;
  };
  function testen() {
    ③ if (document.getElementsByName("Feld")[0].value == "") {
      alert("Sie müssen etwas eingeben!");
      ④ document.getElementsByName("Feld")[0].focus();
      ② return false;
    } else {
      alert("Vielen Dank!");
    }
  }
</script>
</head>
<body>
  <h3>Überprüfen der Eingabe</h3>
  <form action="test.php" id="formular">
    <input type="text" name="Feld" />
    <input type="submit" value="Absenden" />
  </form>
</body>
</html>
```

- ① Beim Betätigen der Schaltfläche absenden wird die Funktion `testen()` aufgerufen.
- ② Um das Versenden der Daten bei einer fehlerhaften Eingabe zu verhindern (die Eigenschaft `value` von dem Eingabefeld ist ein Leerstring ③), gibt `return` den Rückgabewert `false` zurück.

Ist die Eingabe korrekt, braucht die Funktion keinen Rückgabewert zu liefern (default ist immer `true`) und die Aktion wird ausgeführt. Vorher wird noch mit der Methode zum Setzen des Cursors (`focus()` ④) der Cursor in das Eingabefeld gesetzt. Bei umfangreichen Formularen mit vielen Eingabefeldern erleichtern Sie somit dem Anwender die Suche nach dem fehlerhaften Eingabefeld.



Die Meldung, wenn nichts eingegeben wurde (*form_tester.html*)

Als noch mit HTML-Eventhandlern gearbeitet wurde, musste man den `onsubmit`-Eventhandler beim `form`-Tag als Attribut notieren und dem Aufruf der überprüfenden JavaScript-Funktion ein `return` voranstellen.

```
<form onsubmit="return Test();" >
...
</form>
```

Nur dann wurde der Rückgabewert der Funktion an den Formularhandler des Browsers weitergereicht und bei einem `false` die Versendung des Formulars abgebrochen. Mit modernen JavaScript-Eventhandlern ist diese Vorgehensweise nicht mehr notwendig.

Bevor Sie ein Formular zurücksetzen, ist eine Sicherheitsabfrage sinnvoll. Die folgende Anweisung ermittelt über ein Bestätigungsfenster, ob die Daten wirklich gelöscht werden sollen.

```
return confirm("Wollen Sie wirklich alle Eingaben zurücksetzen!");
```

Mehrere Formulareingaben testen

Im folgenden Beispiel sollen mehrere Eingaben eines Anwenders überprüft werden. Somit ist schon eine Kontrolle der eingegebenen Daten auf dem Clientrechner möglich, um bei Fehleingaben das Versenden des Formulars zu verhindern.

Beispiel: *kontaktformular.html*

Das folgende umfangreichere Beispiel zur Prüfung der Formulareingaben zeigt ein Anmeldeformular, das Adressdaten des Benutzers aufnehmen und versenden soll. Das zugrunde liegende Skript ...

- ✓ gibt bei der Eingabe dem Anwender bei einigen Feldern Hilfestellungen in einen Div-Container in der Webseite,
- ✓ führt vor dem Versenden eine Überprüfung der Formularfelder durch (z. B. dürfen die Telefonnummer und die Postleitzahl keine Buchstaben beinhalten, und die anderen Eingabefelder sind Pflichtfelder),

- ✓ fragt den Anwender, ob er die Daten wirklich versenden will, und
- ✓ fragt den Anwender, ob die Felder beim Betätigen der Schaltfläche *Zurücksetzen* auf die Vorgabewerte gesetzt werden sollen.

In dem Beispiel kommt eine externe JavaScript-Datei zum Einsatz, da das Beispiel recht umfangreich ist.

Der HTML-Quellcode

Hier ist zuerst die Webseite *kontaktformular.html* mit den einzelnen Formularelementen und der Referenz auf die externe JavaScript-Datei *kontaktformular.js*.

```

① <script type="text/javascript"
src="lib/js/kontaktformular.js"></script>
</head>
<body>
  <h3>Anmeldeformular</h3>
② <form id="anmeldeform" action="test.php" method="get">
③ <table>
  <tr><td width="100">Anrede</td>
    <td>
      <input type="radio" name="anrede" value="Frau" checked />
      Frau
      <input type="radio" name="anrede" value="Herr" />
      Herr</td></tr>
  <tr><td>Nachname:</td>
    <td><input name="name" size="50" maxlength="70"
      /></td></tr>
  <tr><td>Vorname:</td>
    <td><input name="vorname" size="50" maxlength="70"
      /></td></tr>
  <tr><td>Telefonnummer:</td>
    <td><input name="telefon" size="15" maxlength="15"
      /></td></tr>
  <tr><td>Straße</td>
    <td><input name="anschrift" size="50" maxlength="70"
      /></td></tr>
  <tr><td>Postleitzahl</td>
    <td><input name="plz" size="5" maxlength="5" /></td></tr>
  <tr><td>Ort</td>
    <td><input name="ort" size="50" maxlength="70"
      /></td></tr>
  <tr><td>E-Mail-Adresse</td>
    <td><input name="mail" size="50" maxlength="50"
      /></td></tr>
</table>
  <input type="submit" value="Anmelden" />
  <input type="reset" value="Zurücksetzen" />
</form>
④ <div id="status"></div>
</body>
</html>

```

- ① Im Kopfbereich der Webseite wird eine Referenz auf die externe JavaScript-Datei *kontaktformular.js* im Unterverzeichnis *lib/js* notiert. Sie erkennen keinerlei JavaScript-Funktionalität oder Eventhandler in dem HTML-Code, was die saubere Trennung von Struktur und Funktionalität bedeutet und in allen modernen und insbesondere etwas komplexeren Web-Applikationen zwingend ist.
- ② Das Formular wird mit der ID `anmeldeform` angelegt. Die Daten werden an die PHP-Skriptdatei *test.php* im selben Verzeichnis gesendet, die aber für die zu zeigende Funktionalität nicht von Bedeutung ist und deshalb nicht weiter beachtet wird. Die Datenübermittlung erfolgt über die Methode `GET`.
- ③ Das Layout des Formulars wird über eine HTML-Tabelle realisiert. Zwar sind HTML-Tabellen im Web nicht mehr gewünscht, aber um CSS-Formatierungen im Beispiel zu vermeiden, soll eine HTML-Tabelle zu Übungszwecken verwendet werden. Die Formularelemente werden in die Zellen eingefügt. Die Anrede (Frau oder Herr) wird über Kontrollfelder realisiert, die anderen Eingaben erfolgen mit einfachen Eingabefeldern.
- ④ Der Div-Bereich erhält über das Attribut `id` eine eindeutige Kennung, auf die im weiteren Verlauf zugegriffen werden kann. Hier sollen Statusmeldungen erscheinen.

Anmeldeformular

Anrede: ☒ Frau ☐ Herr

Nachname:

Vorname:

Telefonnummer:

Straße:

Postleitzahl:

Ort:

E-Mail-Adresse:

Grundaufbau des Anmeldeformulars
(*kontaktformular.html*)

Die JavaScript-Datei ist etwas umfangreicher und soll in einzelnen Teilen betrachtet werden. In der Datei *kontaktformular.js* sind alle Teile enthalten, die Sie benötigen.

Die JavaScript-Datei *kontaktformular.js* – die zentrale Initialisierungsfunktion

Im Quelltext der JavaScript-Datei *kontaktformular.js* ist sämtliche Logik zum Aufruf der Funktionen enthalten. Dies ist die Initialisierungsfunktion, die nach dem Laden der Webseite mit `window.onload` ausgeführt wird und alle notwendigen Eventhandler einrichtet.

```

window.onload = function() {
  ① document.getElementById("anmeldeform").onsubmit = checkform;
  ② document.getElementById("anmeldeform").onreset = checkreset;
  ③ document.getElementsByName("telefon")[0].onfocus = function() {
    putstatus('Bitte geben Sie nur Zahlen ein.');
```

```

  };
  document.getElementsByName("telefon")[0].onblur = function() {
    clearstatus();
  };
  document.getElementsByName("anschrift")[0].onfocus = function() {
    putstatus('Bitte geben Sie die Straße und Hausnummer ein.');
```

```

  };
  document.getElementsByName("anschrift")[0].onblur = function() {
    clearstatus();
  };
  document.getElementsByName("plz")[0].onfocus = function() {
    putstatus('Bitte geben Sie nur Zahlen ein.');
```

```

  };
  document.getElementsByName("plz")[0].onblur = function() {
    clearstatus();
  };
};

```

- ① Beim Verschicken des Formulars wird die Funktion `checkform()` aufgerufen. Dazu wird das Formular über seine ID selektiert und beim Eventhandler `onsubmit` eine Funktionsreferenz auf die Funktion `checkform()` notiert. Anhand des Rückgabewertes der Funktion legen Sie fest, ob das Absenden durchgeführt wird.
- ② Das Zurücksetzen des Formulars steuern Sie über das Ereignis `onreset` und das Ausführen der Funktion `checkreset()`, die mit dem Eventhandler `onreset` an das Formular gebunden wird.

The screenshot shows a web form titled "Anmeldeformular". It contains several input fields: "Anrede" with radio buttons for "Frau" (selected) and "Herr"; "Nachname:" with the value "Meier"; "Vorname:" with the value "Hans"; "Telefonnummer:" with the value "98765"; "Straße" with the value "Milchstraße"; "Postleitzahl" with the value "12345"; "Ort" with the value "Irgendwo"; and "E-Mail-Adresse" with the value "mhans@...". At the bottom are two buttons: "Anmelden" and "Zurücksetzen". A modal dialog box is open in the center, asking "Wollen Sie das Formular wirklich zurücksetzen?" (Do you really want to reset the form?). It has a checkbox labeled "Diese Seiten daran hindern, weitere Dialoge zu öffnen" (Prevent these pages from opening further dialogs) and two buttons: "OK" and "Abbrechen" (Cancel).

Der Anwender muss bestätigen, dass das Formular zurückgesetzt wird.

- ③ Die folgenden anonymen Funktionen steuern das Anzeigen der Hilfe beim Fokussieren von drei Eingabefeldern, die jeweils über den Namen (mit `document.getElementById()` und dem Index 0) ausgewählt werden. Es gibt eine Hilfe für die Telefonnummer und für die Eingabefelder `anschrift` und `plz`. Mit der Funktion `putstatus()` zeigen Sie den übergebenen Text als Hilfetext direkt in der Webseite an, wenn ein Eingabefeld den Fokus erhält (Eventhandler `onfocus`). Verlässt der Benutzer mit dem Cursor ein Eingabefeld (Eventhandler `onblur`), wird die Funktion `clearstatus()` ausgeführt, und der bisherige Hilfetext wird gelöscht.

Telefonnummer:	<input type="text"/>
Straße	<input type="text"/>
Postleitzahl	<input type="text"/>
Ort	<input type="text"/>
E-Mail-Adresse	<input type="text"/>
<input type="button" value="Anmelden"/> <input type="button" value="Zurücksetzen"/>	
Bitte geben Sie nur Zahlen ein.	

Hilfe bei der Eingabe
der Telefonnummer

Die JavaScript-Datei – die Testfunktionen zur Überprüfung der Eingaben

In dem Beispiel kommen zwei Testfunktionen zum Einsatz, die etwas komplexere Bedingungen bei der Eingabe testen.

```

① function testeZeichen(testString, erlaubteZeichen) {
    var allezeichenok = true;
    for (var i = 0; i < testString.length; i++)
        if (erlaubteZeichen.indexOf(testString.charAt(i)) == -1) {
            allezeichenok = false;
            break;
        }
    return allezeichenok;
}

② function testemailadresse(testString) {
    var suche = /^[\\w\\.\\-]{2,}\\@[\\äöüa-z0-9\\-\\.]{1,}\\.[a-z]{2,4}$/i;
    return suche.test(testString);
}

```

- ① Die Funktion `testeZeichen()` prüft die übergebene Zeichenkette `testString` darauf, ob jedes ihrer Zeichen ein erlaubtes Zeichen ist. Welche Zeichen erlaubt sind, legt die übergebene Zeichenkette `erlaubteZeichen` fest. Über eine `for`-Schleife wird jeder Buchstabe der Zeichenkette `testString` mit allen Zeichen der Zeichenkette der erlaubten Zeichen verglichen. Wird ein Zeichen der Zeichenkette `testString` nicht in der Zeichenkette `erlaubteZeichen` gefunden, liefert die String-Methode `indexOf()` den Wert -1 zurück. Daraufhin wird die Variable `allezeichenok` auf `false` gesetzt und über `return` zurückgegeben.
- ② Über diese etwas komplexere Funktion prüfen Sie für die übergebene Zeichenkette `testString`, ob sie eine gültige E-Mail-Adresse enthält. Die Gültigkeitsprüfung realisieren Sie über einen **regulären Ausdruck**. Vor dem Zeichen `@` dürfen sich nur Buchstaben, Ziffern, Unterstriche, Punkte oder Bindestriche befinden. Die Länge muss mindestens zwei Zeichen betragen. Ebenso ist es nach dem Zeichen `@`. Hier darf kein Punkt `.` und kein Unterstrich `_` im Namen vorkommen. Die Top-Level-Domains, also `de`, `at`, `com`, `info` usw., dürfen nur zwei bis vier Zeichen lang sein und Buchstaben enthalten. Die Groß- und Kleinschreibung wird aufgrund der Angabe des Flags `i` nicht berücksichtigt.

Die Methode `test()` überprüft den Inhalt der Zeichenkette und liefert `true` zurück, wenn die Eingabe korrekt war. Ansonsten wird `false` zurückgegeben, und das Absenden des Formulars wird aufgrund einer falschen E-Mail-Adresse verhindert.

Die JavaScript-Datei – die Hilfsfunktionen

Sie sehen nun drei einfache Hilfsfunktionen, die in dem Formular zum Einsatz kommen.

```
① function checkreset(myform) {  
    return confirm("Wollen Sie das Formular wirklich  
        zurücksetzen?");  
}  
② function putstatus(mytext) {  
    document.getElementById('status').innerHTML = mytext;  
}  
③ function clearstatus() {  
    document.getElementById('status').innerHTML = '';  
}
```

- ① Klickt der Benutzer im Formular auf die Schaltfläche *Zurücksetzen*, wird er über die Funktion `checkreset()` gefragt, ob die Eingaben tatsächlich zurückgesetzt werden sollen. Dazu wird die window-Methode `confirm()` verwendet.
- ② Mit der Funktion `putstatus()` zeigen Sie den übergebenen Text als Hilfetext in der Webseite an. Hierzu verwenden Sie die Methode `getElementById()`, um das HTML-Element mit der `id='status'` anzusprechen und über die Eigenschaft `innerHTML` einen neuen Text anzugeben. Dieser Text wird direkt an dem selektierten HTML-Element `status` ausgegeben.
- ③ Verlässt der Benutzer mit dem Cursor ein Eingabefeld, wird diese Funktion ausgeführt, und der bisherige Hilfetext in dem HTML-Element mit der `id='status'` wird gelöscht.

Die JavaScript-Datei – die eigentliche Überprüfungsfunktion

Hier ist nun die zentrale Überprüfungsfunktion `checkform()`. Diese verwendet die vorangegangenen Funktionen, um zu testen, ob alle Felder des Formulars korrekt ausgefüllt wurden.

```

function checkform() {
  ① var fehlermeldung = "";
  ② with (document.getElementById("anmeldeform")) {
    ③ if (name.value == "") {
      fehlermeldung += "Bitte geben Sie Ihren Nachname an!<br />";
    }
    if (vorname.value == "") {
      fehlermeldung += "Bitte geben Sie Ihren Vornamen an!<br />";
    }
    if (telefon.value == "") {
      fehlermeldung += "Bitte geben Sie Ihre Telefonnummer an!<br />";
    }
    if (!testeZeichen(telefon.value, "1234567890-/")) {
      fehlermeldung +=
        "Geben Sie für die Telefonnummer bitte nur Zahlen ein!<br />";
    }
    if (anschrift.value == "") {
      fehlermeldung += "Bitte geben Sie Ihre Anschrift an!<br />";
    }
    if (!testeZeichen(plz.value, "1234567890")) {
      fehlermeldung +=
        "Geben Sie für die Postleitzahl bitte nur Zahlen ein!<br />";
    }
    if (plz.value.length != 5) {
      fehlermeldung +=
        "Fünf Stellen sollte die Postleitzahl schon haben!<br />";
    }
    if (ort.value == "") {
      fehlermeldung += "Bitte geben Sie Ihren Wohnort an!<br />";
    }
    if (!testemailadresse(mail.value)) {
      fehlermeldung +=
        "Die eingegebene E-Mail-Adresse ist nicht korrekt!<br />";
    }
  }
  ④ if (fehlermeldung != "") {
    putstatus(fehlermeldung);
    return false;
  } else {
    ⑤ return confirm("Möchten Sie die Daten jetzt absenden?");
  }
}

```

- ① In der Testfunktion wird eine Stringvariable `fehlermeldung` deklariert, über die bei Fehlern des Anwenders eine Meldung zusammengesetzt werden soll, die diesem dann angezeigt wird. Ebenso wird über diese Variable entschieden, ob das Formular abgesendet werden kann.

- ② Über die `with`-Anweisung und den Selektor `document.getElementById("anmeldeform")` wird das Formular ausgewählt.
- ③ Im Inneren der `with`-Anweisung können direkt die Namen der Formularfelder notiert werden. Jedes einzelne Eingabefeld wird auf gewisse Voraussetzungen geprüft. Dabei ist wichtig, dass **keine** mehrstufige Bedingung (etwa mit `if-else`) verwendet wird, sondern jeder einzelne Test auch unabhängig von den anderen Tests durchgeführt wird. Damit können Sie dem Anwender eine zusammenfassende Meldung aller Fehleingaben präsentieren.

Wichtig ist auch, dass nach einem Fehler die Funktion nicht sofort über die Anweisung `return false` verlassen wird, sondern die Testroutine bis zum Ende (④) durchläuft. Die Funktion `checkform()` verwendet zur Plausibilisierung die beiden Funktionen `testeZeichen()` und `testemailadresse()`, um zu testen, ob alle Felder des Formulars korrekt ausgefüllt wurden. Mithilfe der Funktion `testeZeichen()` überprüfen Sie, ob die Eingabe der Telefonnummer außer Zahlen noch Schräg- oder Bindestriche enthält. Neben der Prüfung der Zahlenangabe kontrollieren Sie, ob die Postleitzahl fünfstellig ist. Über die Funktion `testemailadresse()` überprüfen Sie die angegebene E-Mail-Adresse.

- ④ Wenn die Variable zum Sammeln der Fehlermeldungen nach der Prüfung kein Leerstring mehr ist, wird das Versenden des Formulars abgebrochen und die Fehlermeldung angezeigt.

Anmeldeformular

Anrede: ☒ Frau ☐ Herr

Nachname:

Vorname:

Telefonnummer:

Straße:

Postleitzahl:

Ort:

E-Mail-Adresse:

Bitte geben Sie Ihren Nachnamen an!
 Bitte geben Sie Ihren Vornamen an!
 Bitte geben Sie Ihre Telefonnummer an!
 Bitte geben Sie Ihre Anschrift an!
 Fünf Stellen sollte die Postleitzahl schon haben!
 Bitte geben Sie Ihren Wohnort an!
 Die eingegebene E-Mail-Adresse ist nicht korrekt!

Das Formular darf keine Fehler enthalten.

- ⑤ Wenn alle Eingaben des Formulars in Ordnung sind, wird der Benutzer über `confirm()` gefragt, ob das Formular abgeschickt werden soll. Je nachdem, wie der Benutzer auf die Frage antwortet, wird die Funktion mit dem entsprechenden Wahrheitswert verlassen. Die Formulardaten werden nur beim Bestätigen des Dialogfensters versendet.

Anmeldeformular

Anrede: ☒ Frau ☐ Herr

Nachname:

Vorname:

Telefonnummer:

Straße:

Postleitzahl:

Ort:

E-Mail-Adresse:

Möchten Sie die Daten jetzt absenden?

Das Formular wurde korrekt ausgefüllt und der Anwender muss das Versenden bestätigen.

9.7 Formulareingaben direkt in JavaScript verwerten

Es kommt auch oft vor, dass Formulareingaben in einer Webseite gar nicht (direkt) an den Webserver geschickt werden. Sie können auch mit JavaScript im Client selbst verwertet werden.

Beispiel: *volumen_rechner.html*

Das folgende Beispiel zeigt Ihnen, wie Sie nach der Eingabe eines Wertes und der Auswahl einer Maßeinheit die Umrechnungen in andere Maßeinheiten durchführen und die Werte ausgeben. Auch in dem Beispiel wird, wie in der Praxis üblich, explizit eine externe JavaScript-Datei eingesetzt.

Der HTML-Quellcode

Die Webseite *volumen_rechner.html* umfasst die einzelnen Formularelemente und die Referenz auf die externe JavaScript-Datei *volumen_rechner.js*.

```

① <script type="text/javascript"
    src="lib/js/volumen_rechner.js"></script>
</head>
<body>
  <h3>Volumenmaße berechnen</h3>
  ② <form id="Volumen">
    ③ <input name="eingabewert" size="6" maxlength="5" />
    ④ <select name="einheit">
      <option value="cmm">mm3</option>
      <option value="ccm">cm3</option>
      <option value="cdm">dm3</option>
      <option value="cm">m3</option>
      <option value="l" selected>l (Liter)</option>
      <option value="hl">hl (Hektoliter)</option>
    </select>
    ⑤ <hr /><input type="button" id="los" value="Umrechnen"
      /><hr />
      <input name="cmm" readonly />mm3<br />
      <input type="text" name="ccm" readonly />cm3<br />
      <input type="text" name="cdm" readonly />dm3<br />
      <input type="text" name="cm" readonly />m3<br />
      <input type="text" name="liter" readonly
        />l (Liter)<br />
      <input type="text" name="hektoliter" readonly
        />hl (Hektoliter)
    </form>
  </body>
</html>

```

- ① Im Kopfbereich der Webseite wird eine Referenz auf die externe JavaScript-Datei im Unterverzeichnis *lib/js* notiert. Sie erkennen in der restlichen HTML-Datei keinerlei JavaScript-Funktionalität oder Eventhandler, womit wieder die gewünschte saubere Trennung von Struktur und Funktionalität erreicht wird.
- ② Das Formular mit der ID `Volumen` wird angelegt. Da es nicht abgesendet werden soll, werden keine weiteren Attribute angegeben.
- ③ In das Eingabefeld `eingabewert` kann der Anwender eine Zahl eingeben.
- ④ Über die Liste `einheit` hat der Benutzer die Möglichkeit, die Maßeinheit auszuwählen. Die Angabe des HTML-Codes `³`; ermöglicht die Darstellung des Zeichens ³.
- ⑤ Das Formular erhält eine Schaltfläche vom Typ `button`. Ihr wird in der JavaScript-Datei der Event-Handler `onclick` zugewiesen. Damit wird nach dem Klick auf die Schaltfläche die Funktion `umrechnen()` aufgerufen und ausgeführt. In den unteren Eingabefeldern, die aufgrund des Attributs `readonly` nur gelesen werden können, sollen die berechneten Werte angezeigt werden.

Die JavaScript-Datei

Sie sehen nachfolgend den Quelltext der JavaScript-Datei *volumen_rechner.js*. Wie üblich finden Sie am Beginn die Initialisierungsfunktion, die nach dem Laden der Webseite mit `window.onload` ausgeführt wird und alle notwendigen Eventhandler einrichtet. Es folgt die JavaScript-Funktion `umrechnen()`, die die eigentliche Berechnung durchführt und die bei einem Klick auf die Schaltfläche aufgerufen wird.

```
window.onload = function() {  
  ① document.getElementById("los").onclick = umrechnen;  
};  
② function umrechnen()  
③ var menge =  
document.getElementById("Volumen").eingabewert.value;  
  with (document.getElementById("Volumen")) {  
    switch (einheit.options.selectedIndex) {  
      case 0:  
        tausender = 1;  
        break;  
      case 1:  
        tausender = 1000;  
        break;  
      case 2:  
        tausender = 1000 * 1000;  
        break;  
      case 3:  
        tausender = 1000 * 1000 * 1000;  
        break;  
    }
```

```

        case 4:
            tausender = 1000 * 1000;
            break;
        case 5:
            tausender = 1000 * 1000 * 100;
    }
    cmm.value = menge * tausender;
    ccm.value = menge * tausender / 1000;
    cdm.value = menge * tausender / 1000 / 1000;
    cm.value = menge * tausender / 1000 / 1000 / 1000;
    liter.value = menge * tausender / 1000 / 1000;
    hektoliter.value = menge * tausender / 1000 / 1000 / 100;
}

```

- ① Beim Klick auf die Schaltfläche wird die Funktion `umrechnen()` aufgerufen. Dazu wird die Schaltfläche in der Initialisierungsfunktion über die ID selektiert und beim Eventhandler `onclick` eine Funktionsreferenz auf die Funktion `umrechnen()` notiert.
- ② Die Variable `menge` erhält den Wert (`value`) des Eingabefeldes `eingabewert` des Formulars `Volumen`.
- ③ Über die `with`-Anweisung wird das Formular ausgewählt. Mit `switch` leiten Sie eine mehrseitige Fallauswahl ein. Der zu vergleichende Selektor wird über den Index der ausgewählten Maßeinheit ermittelt. Je nach übergebenem Wert wird über die `case`-Anweisung die Variable `tausender` ausgehend von der kleinsten Maßeinheit berechnet. Sie ermöglicht das einfache Umrechnen in die anderen Maßeinheiten.
- ④ Die Eingabefelder werden mit den Werten für die einzelnen Maßeinheiten ausgefüllt.

Volumenmaße berechnen	
2	dm³
<input type="button" value="Umrechnen"/>	
2000000	mm³
2000	cm³
2	dm³
0.002	m³
2	l (Liter)
0.02	hl (Hektoliter)

Umrechnen in verschiedene Maßeinheiten (volumen_rechner.html)

9.8 Übungen

Übung 1: Eingabefelder fokussieren und Statustext anzeigen

Übungsdatei: --

Ergebnisdatei: *kap09/uebung1.html*

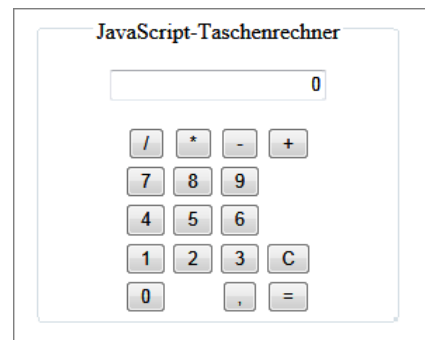
1. Erstellen Sie ein Formular mit den Eingabefeldern für Name, Vorname, Ort, Postleitzahl, Größe, Gewicht und Alter.
2. Überprüfen Sie, ob die Pflichtfelder für Name, Vorname, Ort und Postleitzahl ausgefüllt sind.
3. Wurden Angaben bei Größe, Gewicht und Alter gemacht, dann kontrollieren Sie, ob sinnvolle Angaben gemacht wurden. Nutzen Sie dazu den Zugriff über Namen, um auf das Formular und die Formularelemente zuzugreifen. Setzen Sie im Falle einer fehlerhaften Eingabe den Fokus in das entsprechende Eingabefeld, und geben Sie in einem Div-Bereich in der Webseite eine entsprechende Fehlermeldung aus.

Übung 2: Mathematische Berechnungen in JavaScript

Übungsdatei: --

Ergebnisdatei: *kap09/uebung2.html*

1. Erstellen Sie über ein Formular einen JavaScript-Taschenrechner, mit dem Sie die vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division durchführen können. Wenn Sie wollen, nutzen Sie zu Übungszwecken bei den Tags für die Schaltflächen explizit die (veralteten) HTML-Eventhandler, da diese im Buch sonst nicht verwendet werden. Sie werden erkennen, dass damit die Struktur und Funktionalität stark vermischt wird.



Simulation eines Taschenrechners

10

Ajax

10.1 Grundlagen zu Ajax

Ajax bedeutet Aynchronous JavaScript and XML und ist eine Möglichkeit, Informationen im Browser auszutauschen, ohne die Webseite neu laden zu müssen. Fast alle interaktiven Webseiten nutzen mittlerweile Ajax. Einsatzgebiete sind Vorschläge bei Benutzereingaben, Anzeige von Zusatzinformationen bei Bedarf oder das Übermitteln von Mausdaten an den Server.

Ajax ist eigentlich nur ein Begriff, der im Jahr 2005 in einem Artikel eines amerikanischen Firmenberaters für eine Zusammenfassung bereits bekannter Techniken genannt wurde. Die zentrale Technik selbst, das Laden von Informationen im Hintergrund, wurde bereits von der Firma Microsoft 1998 im Internet Explorer 4 implementiert. Mittlerweile stellen alle Browser ein Objekt bereit, mit dem man diese Technik nutzen kann. Dabei haben die Browserhersteller allerdings nicht das von Microsoft schon lange in seinem Internet Explorer vorhandene ActiveX-Objekt implementiert, sondern eine andere Technik verwendet. Microsoft hat mittlerweile ebenfalls dieses alternative Objekt implementiert, das `XMLHttpRequest` heißt.

10.2 Das `XMLHttpRequest`-Objekt

Das `XMLHttpRequest`-Objekt (kurz **XHR**) ist der Kern der gesamten Ajax-Technologie. Es sorgt für den Datenaustausch im Hintergrund.

Das XHR-Objekt erstellen

Um die Funktionalität des XHR-Objekts nutzen zu können, müssen Sie zuerst über den Konstruktor `XMLHttpRequest()` eine Instanz erzeugen.

```
<script type="text/javascript">
  var ajaxhttp = new XMLHttpRequest();
</script>
```

Damit haben Sie die Instanz eines `XMLHttpRequest`-Objekts erstellt und können über die Variable `ajaxhttp` auf die Eigenschaften und Methoden zum asynchronen Informationsaustausch zugreifen.

- Ältere Versionen des Internet Explorer kommen mit der Syntax nicht zurecht, sind aber mittlerweile in der Praxis so gut wie gar nicht mehr vorzufinden.

10.3 Eine HTTP-Anfrage erstellen

Nachdem Sie die Objektinstanz erstellt haben, können Sie eine Verbindung zu der Datei oder dem Skript auf dem Server aufbauen, woher die Daten geliefert werden sollen. Die Antwort des Servers können Sie im Client verarbeiten.

Initialisieren der Verbindung – die Methode `open()`

Die Verbindung zum Server definieren Sie über die Methode `open()`. Das ist aber nur eine Initialisierung und noch keine Datenanforderung

<code>open(Methode, URL, [asynchron])</code>	Neben der aufzurufenden URL geben Sie als Methode <code>POST</code> oder <code>GET</code> an, mit der die Daten übertragen werden sollen. Wie die Kommunikation zwischen Browser und Webserver ablaufen soll, legen Sie mit einem Wert <code>true</code> oder <code>false</code> fest. Standardmäßig wird <code>asynchron (true)</code> kommuniziert. Dies hat den Vorteil, dass der Browser nicht blockiert wird, weil auf die Antwort gewartet wird.
--	---

<pre>ajaxhttp.open("POST", "datei.txt", true); ajaxhttp.open("GET", "datei.php", false);</pre>
--

- Ajax-Anfragen dürfen aus Sicherheitsgründen nur Daten von der gleichen Domain nachfordern, von der die anfordernde Webseite stammt (**Same Origin Policy**). Die Vorgehensweise wird Sandkastenprinzip (**Sandbox**) genannt. Andernfalls würde man einen sogenannten **Cross-Domain-Zugriff** durchführen und damit Missbrauch und Manipulation riskieren. Allerdings ist die Einschränkung erheblich, und es gibt einige Anstrengungen, diese Beschränkungen aufzuheben, ohne die Sicherheit zu gefährden.

Der Ajax-Eventhandler `onreadystatechange`

Da bei einer asynchronen Kommunikation nicht blockierend auf die Antwort gewartet wird, gibt es die Eigenschaft `onreadystatechange`. Dies ist ein Eventhandler des XHR-Objekts. An diesen binden Sie, wie beim Eventhandling üblich, eine beliebige Callback-Funktion.

```
/* Variante 1 */
ajaxhttp.onreadystatechange = function() { }

/* oder Variante 2 */
ajaxhttp.onreadystatechange = meineFunktion;
function meineFunktion {
}
```

Die Zustände und die Eigenschaft `readyState`

Die Callback-Funktion wird immer dann aufgerufen, wenn sich der Zustand des XHR-Objekts ändert. Folgende Zustände gibt es:

Zustand	Beschreibung
0	Das Objekt ist nicht initialisiert.
1	Das Objekt baut eine Verbindung auf.
2	Die Anfrage wurde erfolgreich gesendet.
3	Das Objekt wartet auf die Rückantwort.
4	Das Ergebnis wurde zurückgeliefert.

Eine HTTP-Anfrage durchläuft hierbei immer alle Zustände von 0 bis 4. Im Normalfall müssen Sie nur den Zustand 4 beachten, da erst zu dem Zeitpunkt die zurückgelieferten Daten über verschiedene Eigenschaften abgefragt werden können. Die Zustände stehen in der Eigenschaft `readyState` bereit.

Die Antworteigenschaften

Für die Verwertung der Antwort des Servers gibt es die folgenden Eigenschaften:

<code>responseText</code>	Die Antwort des Servers wird als Zeichenkette bereitgestellt.
<code>responseXML</code>	Die Antwort des Servers wird als XML-Objekt mit DOM-Struktur zurückgeliefert.
<code>status</code>	Der HTTP-Statuscode
<code>statusText</code>	Der Beschreibungstext des Statuscodes

Die eigentliche HTTP-Anfrage senden

Mit der Methode `send()` senden Sie die definierte Anfrage an den Webserver und starten damit die Anforderung der Daten.

```
ajaxhttp.send(null);
```

Der Parameter der Methode ist bei der Methode `GET` immer `null` oder leer. Bei `POST` werden darin eventuell an den Server zu übertragende Daten (etwa Formulareingaben) notiert.

10.4 Das Datenformat

Bei der Datenanforderung mit Ajax fordert man in der Praxis zwei Arten an Klartextdaten an:

- ✓ Klartext, der mit HTML-Tags durchzogen ist, also HTML-Fragmente oder reiner Klartext ohne Struktur
- ✓ strukturierte Daten mit XML oder JSON

HTML-Fragmente und reiner Klartext

Im Fall von Ajax-Datenanforderungen werden sehr oft HTML-Fragmente vom Webserver angefordert und diese in der Regel im Client einfach per DHTML in der Webseite eingebaut. Hierbei werden einfach unter Verzicht auf eine vollständige Webseitenstruktur Texte mit HTML-Tags gemischt. Meistens erfolgt der Einbau dieser Texte dann ohne eine weitere Verarbeitung der HTML-Fragmente im Client durch JavaScript oder eine andere clientseitige Programmieretechnik. Die Antwort des Webserver wird also vorwiegend unverändert angezeigt und muss vorher auf dem Server bereits in die endgültige Form gebracht werden, die im Client dann verwendet werden soll. Wenn Sie auf alle HTML-Tags in den Daten verzichten, fordern Sie reinen Klartext an. Das ist ebenfalls möglich.



Auch wenn Klartext HTML-Tags enthält, sollte die angeforderte Datei nicht die Datei-erweiterung `.html` haben. Die meisten Browser kommen damit zwar zurecht, aber diese Dateierweiterung steht eigentlich für eine vollständige Webseite. Eine vollständige Webseite wird mit Ajax nicht nachgefordert, denn das neue Grundgerüst könnte nicht ohne Probleme in das vorhandene Grundgerüst der Webseite integriert werden. Zwar kommen auch damit die modernen Browser zurecht, aber so ganz vorhersehbar ist nicht immer, wie die Browser die Fehler korrigieren. Sie sollten immer nur Fragmente ohne Body oder `html`-Elemente per Ajax nachfordern.

Klartext mit Struktur

Wenn Sie anspruchsvollere Applikationen erstellen wollen, können Sie die Antwort des Servers um ein strukturiertes Format erweitern, das im Client gezielt verwertet werden kann. Die Strukturierung der übertragenen Information lässt Ihnen weit mehr Möglichkeiten, um Geschäftsprozesse zu verteilen. Als Strukturierungsformat kommen JSON oder XML infrage. Schickt der Webserver also XML oder JSON, können Sie im Client auf der Antwort operieren und damit Geschäftslogik zum Client verlagern. Allerdings ist das hauptsächlich bei Ajax eine sehr interessante Möglichkeit. Das direkte Laden dieser Datenstrukturen ist selten sinnvoll.

- Als sich Ajax 2005 etabliert hatte, wurde oft XML für strukturierte Daten verwendet (XML gibt ja Ajax auch einen Teil des Namens). Mittlerweile nutzt man überwiegend JSON, denn die Verarbeitung von JSON im Browser ist konsistenter und einfacher, und das Datenformat ist schlanker als XML. In JavaScript erleichtern die Klasse mit Namen `JSON` den Umgang mit JSON erheblich.

10.5 Daten per Ajax zum Server schicken

Bei Ajax werden auch sehr oft gezielt im Hintergrund Daten vom Client an den Server geschickt, um darauf dann eine individuelle Antwort des Servers zu generieren. Dies bedeutet die Übergabe zusätzlicher Parameter an die Skript-Dateien.

Je nach verwendeter Methode `GET` oder `POST` sind die Angaben im JavaScript dann unterschiedlich. Bei der Methode `GET` brauchen Sie die Parameter einfach nur an den angeforderten Dateinamen anzuhängen.

```
ajaxhttp.open("GET", "gettext.php?param1=wert1&param2=wert2", true);
```

Bei der Methode `POST` können die Parameter nicht an den Dateinamen angehängt werden. Hier übergeben Sie die Parameter mit der Methode `send()`.

```
ajaxhttp.open("POST", "gettext.php", true);  
/* ... */  
ajaxhttp.send("param1=wert1&param2=wert2");
```

Möchten Sie Daten über die Methode `POST` an den Server schicken, müssen Sie zusätzlich den Inhaltstyp über die Methode `setRequestHeader()` definieren. Die erforderliche Angabe sieht folgendermaßen aus:

```
ajaxhttp.setRequestHeader("Content-Type",  
                           "application/x-www-form-urlencoded");
```

Die Methode `GET` wird verwendet, wenn Sie Daten als Name-Wert-Paare – durch ein Fragezeichen an die URL angehängt – übertragen wollen. Die Menge der Daten ist beschränkt und vom jeweils verwendeten Browser abhängig. Mit der Methode `POST` können Sie auch binäre Daten in beliebiger Menge übertragen. Dazu werden die Daten codiert und im Nachrichtenrumpf übertragen.

10.6 Praktische Beispiele

Sie sehen nun ein paar praktische Beispiele zum Umgang mit Ajax.

Beachten Sie, dass Sie die HTML-Datei, aus der Sie die Ajax-Anfrage aufrufen, unbedingt von einem Server aufrufen müssen. Etwa über localhost von Ihrem lokalen Webserver. Wenn Sie die HTML-Datei direkt im Browser laden, wird Ajax normalerweise nicht funktionieren. Ajax ist ja explizit nur dafür da, Daten vom Webserver nachzufordern, von dem die Webseite geladen wurde.

Klartext nachfordern – Beispiel: *ajax-get.html*

In diesem Beispiel soll eine Webseite erstellt werden, die bei einem Klick auf eine Schaltfläche im Hintergrund eine einfache Textdatei lädt und deren Inhalt über die Eigenschaft `innerHTML` an einer bestimmten Stelle innerhalb der Webseite einfügt. Außerdem soll der Status ausgegeben werden, den der Server nach der Anfrage zurücksendet.

AJAX - Text laden
Der nachfolgende Text wird dynamisch nach dem Laden der Webseite geladen und angezeigt.
Dieser Text wird ausgetauscht.. Statusausgabe

Die Webseite vor der Ajax-Anforderung

```

<script type="text/javascript"><...
①   var ajaxhttp = new XMLHttpRequest();
②   window.onload = function() {
③       document.getElementById("laden").onclick = function() {
④           if (ajaxhttp != null) {
⑤               ajaxhttp.open("GET", "text.txt", true);
⑥               ajaxhttp.onreadystatechange = function() {
⑦                   if (ajaxhttp.readyState == 4) {
⑧                       document.getElementById("textstelle").innerHTML =
                           ajaxhttp.responseText;
                           document.getElementById("status").innerHTML =
                           'Status: ' + ajaxhttp.status +
                           ' (' + ajaxhttp.statusText + ')';
                           }
                           }
                           };

```

```

⑨      ajaxhttp.send(null);
        }
      };
    };
  </script>
</head>
<body>
  <button id="laden">AJAX - Text laden</button><hr/>
  Der nachfolgende Text wird dynamisch nach dem Laden der
  Webseite geladen und angezeigt.<hr/>
  <div id="textstelle">Dieser Text wird ausgetauscht...</div>
  <div id="status">Statusausgabe</div>
</body>
</html>

```

- ① Zu Beginn definieren Sie die Variable `ajaxhttp`, die auf XMLHttpRequest-Objekt verweist.
- ② Eine Initialisierungsfunktion zum Binden des Eventhandlers an die Schaltfläche wird deklariert.
- ③ Bei der Schaltfläche wird der `onclick`-Eventhandler mit einer anonymen Callback-Funktion registriert.
- ④ War das Zuweisen des Objekts erfolgreich, also `!= null`, wird in diesen Block verzweigt, in dem die Eigenschaften des XHR-Objekts festgelegt werden.
- ⑤ Über die GET-Methode soll die Datei `text.txt`, die einen beliebigen Inhalt haben kann, angefordert werden. Es soll eine asynchrone Verbindung aufgebaut und somit nicht blockierend auf die Antwort gewartet werden.
- ⑥ In der Funktion wird überprüft, ob ein Ergebnis zurückgeliefert wurde (`readyState == 4`).
- ⑦ Ist dies der Fall, wird in der Webseite das Element mit der ID `textstelle` ermittelt. Über die Eigenschaft `innerHTML` können Sie beliebigen Text übergeben, der dann vom Browser angezeigt wird. In diesem Fall ist es der Inhalt der Rückgabe (`responseText`).
- ⑧ Am Element `status` werden der Status sowie die entsprechende Erklärung ausgegeben. Beachten Sie, dass der Status nur von einem Server zurückgeliefert wird und nicht lokal auf dem Rechner funktioniert. Sie müssen also die entsprechenden Dateien auf einen Webserver laden und von dort abrufen.
- ⑨ Die Anfrage wird gesendet.

AJAX - Text laden
Der nachfolgende Text wird dynamisch nach dem Laden der Webseite geladen und angezeigt.
Text, der per AJAX über eine HTTP-Anfrage in diese Webseite nachträglich eingefügt wurde. Status: 200 (OK)

Die Antwort des Servers ist da und wird angezeigt.

JSON nachfordern – Beispiel: *ajax-getjson.html*

Wie erwähnt, bietet JSON die gleiche Funktionalität und Flexibilität wie XML, ohne dessen Komplexität und problematische Auswertung in verschiedenen Browsern im Weg zu stehen. Aus diesem Grund hat sich JSON als Standardformat bei RIAs durchgesetzt, wenn tatsächlich eine gewisse Funktionalität zur Auswertung der Antwort im Client gefordert wird.

Die JSON-Datei

Die JSON-Datei hat folgende Struktur (*ajax.json* – die Dateierendung *.json* ist üblich, aber für die Datennachforderung per Ajax irrelevant):

```
① {  
②   "webseiten": {  
③     "seite": [  
④       {  
        "titel": "Google",  
        "url": "http://www.google.de",  
        "bemerkung": "Suchmaschine von Google"  
      }, {  
        "titel": "Bing",  
        "url": "http://www.bing.de",  
        "bemerkung": "Suchmaschine von Microsoft"  
      }, {  
        "titel": "Yahoo",  
        "url": "http://www.yahoo.de",  
        "bemerkung": "Suchmaschine von Yahoo"  
      }  
    ]  
  }  
}
```

- ① Eine JSON-Struktur wird in geschweifte Klammern eingeschlossen. Dieser Block umschließt den gesamten Inhalt und damit den gesamten Dateiinhalt.
- ② Das Element `webseiten` ist das Wurzelement der Baumstruktur, die von der JSON-Datei abgebildet wird. Diese entspricht einem DOM. Der Wert des Elements `webseiten` ist ein Array, das in JSON-Notation notiert wurde. Es enthält genau ein Schlüssel-Wert-Paar, dessen Schlüssel `seite` lautet.
- ③ Das Element `seite` hat als Wert ein Array, das aber nicht in JSON-Form, sondern in Array-Form (automatisch numerisch indiziert) notiert wird. Es hat drei weitere Arrays als Elemente, die wieder in JSON-Form notiert werden. Jedes Element in diesem Array, das über den Schlüssel `seite` verfügbar ist, besteht aus den Elementen `titel`, `url` und `bemerkung` (④).

Die Webseite

Hier ist die Webseite, die die JSON-Datei anfordert und dann gezielt verarbeitet.

```

<script type="text/javascript">
①   var ajaxhttp = new XMLHttpRequest();
②   window.onload = function() {
③       document.getElementById("laden").onclick = function() {
④           if (ajaxhttp != null) {
⑤               ajaxhttp.open("GET", "ajax.json", true);
⑥               ajaxhttp.onreadystatechange = function() {
⑦                   if (ajaxhttp.readyState == 4) {
⑧                       var obj = JSON.parse(ajaxhttp.responseText);
                           obj.webseiten.seite[0].titel + "<br/>" +
                           obj.webseiten.seite[0].url + "<hr/>" +
                           obj.webseiten.seite[1].titel + "<br/>" +
                           obj.webseiten.seite[1].url + "<hr/>" +
                           obj.webseiten.seite[2].titel + "<br/>" +
                           obj.webseiten.seite[2].url;
                           document.getElementById("status").innerHTML =
                               'Status: ' + ajaxhttp.status + '
                               (' + ajaxhttp.statusText + ')';
                           }
                           };
                           ajaxhttp.send(null);
                           }
                           };
                           };
</script>

```

- ① Die Variable `ajaxhttp` wird definiert, die dann auf das `XMLHttpRequest`-Objekt verweist.
- ② Eine Initialisierungsfunktion zum Binden des Eventhandlers an die Schaltfläche wird deklariert.
- ③ Bei der Schaltfläche wird der `onclick`-Eventhandler mit einer anonymen Callback-Funktion registriert.
- ④ War das Zuweisen des Objekts erfolgreich, wird in den Block verzweigt, in dem die Eigenschaften des XHR-Objekts festgelegt werden.
- ⑤ Über die `GET`-Methode wird asynchron die Datei `ajax.json` angefordert.
- ⑥ In der Funktion wird überprüft, ob ein Ergebnis zurückgeliefert wurde (`readyState == 4`).
- ⑦ Ist dies der Fall, wird die Antwort des Servers mit der Methode `parse()` der Klasse `JSON` in ein Objekt überführt.
- ⑧ Auf dem so erhaltenen Objekt kann man wie auf jedem JavaScript- oder DOM-Objekt navigieren. Die verschachtelten JSON-Strukturen sind im Objekt über die übliche Punktnotation zugänglich. Sie gehen vom Root-Element `webseiten` zum entsprechenden Array-Element (über den Index) und dann zu dessen Inhaltselement, das Sie interessiert.

AJAX - Text laden
Der nachfolgende Text wird dynamisch nach dem Laden der Webseite geladen und angezeigt.
Google http://www.google.de
Bing http://www.bing.de
Yahoo http://www.yahoo.de Status: 200 (OK)

Die JSON-Antwort ist da und wurde gezielt im Client ausgewertet.

Benutzerdaten per Ajax an den Server senden – Beispiel: *ajax-senden.html*

Hier ein Beispiel, bei dem Daten bei einer Ajax-Anfrage an den Webserver übergeben werden und die Antwort sich erst aufgrund der übergebenen Daten ergibt. Die Antwort soll zusätzlich HTML-Fragmente enthalten. Konkret wird eine Eingabe aus einem Webformular und an ein PHP-Skript weitergereicht. Dieses wird eine Antwort schicken, die sich für den Client in nichts von dem unterscheiden wird, was eine statische Textdatei liefert. Dazu erstellen Sie wieder eine HTML-Datei, die einfach ein Webformular und einen leeren DIV-Container als Bereich für die Antwort der Ajax-Anfrage bereitstellt.

- ! Sie benötigen zur Ausführung vom PHP-Skript einen Webserver, der damit umgehen kann (etwa Apache). Sie können auch eine beliebige andere serverseitige Programmiersprache oder Skriptsprache verwenden. Nur um die Daten vom Browser auf dem Server entgegennehmen zu können, müssen Sie in irgendeiner Form dort programmieren. Aber wie erwähnt, benötigen Sie ja für Ajax sowieso einen Webserver.

```

<script type="text/javascript">
    var ajaxhttp = new XMLHttpRequest();
    window.onload = function() {
        document.getElementById("frage").onclick = function() {
            ① if (ajaxhttp != null) {
                ajaxhttp.open("GET", 'ajax.php?n=' +
                    encodeURIComponent(document.getElementById
                        ("n").value), true);
                ajaxhttp.onreadystatechange = function() {
                    if (ajaxhttp.readyState == 4) {
                        document.getElementById("antwort").innerHTML =
                            ajaxhttp.responseText;
                    }
                };
                ajaxhttp.send(null);
            }
        };
    };
</script>
</head>
<body>
    <h1>Geben Sie Ihren Namen ein</h1>
    ② <form>Name: <input type="text" id="n" /></form><hr />
        <button id="frage">OK</button><div id="antwort"></div>
    </body>
</html>

```

- ① In der `open()`-Methode wird ein Pseudo-URL aus einem Namen für das Eingabefeld, über den der Wert auf dem Server entgegengenommen werden kann (`n`), und dem Wert zusammengesetzt (über die ID abgefragt). Zusätzlich wird der Wert mit `encodeURIComponent()` umgewandelt, um Sonderzeichen zu maskieren.
- ② Das Formular, das die Benutzerdaten entgegennimmt.

Obwohl es irrelevant für Ajax ist, sehen Sie hier das Listing der aufgerufenen PHP-Datei:

```

<?php
    echo "Hallo <b>" . $_GET['n'] . "</b>.";
?>

```

Beachten Sie, dass HTML-Tags zum Client geschickt werden, die dort interpretiert werden (wegen der Verwendung von `innerHTML`).

Geben Sie Ihren Namen ein

Name:

Hallo **Hans**

Die Antwort des Servers mit individueller Anpassung und der Interpretation der Tags im Client

10.7 Übung

Anfordern von XML-Daten

Übungsdatei: --

Ergebnisdateien: *kap10/uebung1.html*,
kap10/uebung1.xml

1. Mit Ajax können Sie auch XML vom Server als Antwort anfordern. In der `open()`-Methode muss dazu eine Referenz auf der URL der Datei notiert werden, die die Antwort generiert, oder eine Referenz auf der URL der zum Download verfügbaren statischen XML-Datei. Wenn Sie die Daten in XML-Form erhalten, können Sie diese im Client entgegennehmen und adäquat verarbeiten, denn die Daten bilden einen DOM, wenn Sie die Eigenschaft `responseXML` verwenden. Nehmen Sie als Basis folgende XML-Datei.

```
<?xml version="1.0" encoding="UTF-8" ?>
<webseiten>
  <seite>
    <titel>Google</titel>
    <url>http://www.google.de</url>
    <bemerkung>Suchmaschine von Google</bemerkung>
  </seite>
  <seite>
    <titel>Bing</titel>
    <url>http://www.bing.de</url>
    <bemerkung>Suchmaschine von Microsoft</bemerkung>
  </seite>
</webseiten>
```

2. Geben Sie jeweils den Titel und die URL der beiden Seiten aus.

11

Erweiterte JavaScript-Techniken und Ausblick

11.1 Hinweise zu JavaScript-Techniken

In diesem Kapitel werden Ihnen verschiedene JavaScript-Techniken vorgestellt, die zum Teil schon recht lange in Browsern genutzt werden können, zum Teil aber auch erst mit HTML5 Einzug gehalten haben. HTML5 ist bereits geraume Zeit verfügbar, aber immer noch können nicht alle HTML5-Techniken wirklich zuverlässig in allen verbreiteten Browsern genutzt werden. Aber die meisten Techniken aus HTML5 funktionieren in modernen Browsern.

11.2 DHTML

Dynamic HTML (dynamisches HTML), kurz DHTML, ist eine Kombination aus HTML, CSS (Cascading Style Sheets), JavaScript und dem DOM, die ursprünglich auf die Firma Netscape zurückgeht. Der Begriff DHTML ist jedoch weder eindeutig, standardisiert oder geschützt. Darunter versteht man im weitesten Sinn die Veränderung einer Webseite, nachdem sie bereits in den Browser des Anwenders geladen wurde. Die konkrete Technik, wie die Veränderung dabei realisiert wird, ist irrelevant, und es brauchen nicht zwingend alle oben genannten Techniken beteiligt zu sein. Auch Animationen kann man im Grunde zu DHTML zählen, wenn man für Animation die gleichen Techniken (HTML, CSS und JavaScript) verwendet. Wenn man überhaupt eine Differenzierung zwischen DHTML und Animationen vornehmen will, dann geht das über den Zeitfaktor. Denn Animationen laufen über einen längeren Zeitraum ab und erzielen ihre Wirkung durch diese zeitorientierten Veränderungen. Allgemein zählt die dynamische Veränderung einer Webseite zu den eindrucksvollsten Anwendungen von JavaScript, rein von der Programmierung her zu den einfachsten. Nur die hohe Inkompatibilität verschiedener Browsermodelle machte in der Vergangenheit Probleme. Aber diese Probleme nehmen in modernen Browsern immer mehr ab.

DHTML mit reinem JavaScript

Viele Effekte, die DHTML zuzuordnen sind, lassen sich rein mit JavaScript durchführen, z. B. der Austausch und die Größenveränderung von Bildern. Sie können beispielsweise die klassischen HTML-Attribute des ``-Tags manipulieren und mit geeigneten Eventhandlern – etwa für die Reaktion auf das Überstreichen mit dem Mauszeiger – koppeln. Das wurde bereits mit einem Bildwechsel in Kapitel 8 gezeigt.

DHTML und Animation mit reinem CSS

Auch CSS selbst bietet einige dynamische Effekte an, die man als DHTML oder gar Animation bezeichnen kann:

- ✓ Der bekannteste und häufig eingesetzte Effekt dürfte der Hover-Effekt sein. In so einem Fall ergibt sich eine Änderung eines Bereichs in der Webseite, wenn dieser Bereich mit dem Mauszeiger überstrichen wird. Ganz einfach kann man den Hover-Effekt für Hyperlinks mit der CSS-Pseudo-Klasse `:hover` erzeugen. Meist verwendet man diesen Effekt bei Hyperlinks über `a:hover`, aber im Prinzip kann man ihn auch bei anderen Elementen verwenden (etwa `h1:hover`), wobei das nicht alle älteren Browser unterstützen. Der Hover-Effekt wirkt sich aber immer nur auf das Element aus, das gerade mit dem Mauszeiger überstrichen wird. Deshalb ist er bedeutend unflexibler als eine Programmierung mit JavaScript.
- ✓ Allgemein ändert sich das Aussehen vom Mauszeiger bei einer grafischen Oberfläche automatisch, um dem Anwender Rückmeldung darüber zu geben, was in einer bestimmten Situation auf eine gewisse Mausektion durch den Anwender passieren wird. Im Fall von Hyperlinks wird aus dem Mauszeiger etwa eine Hand. Allerdings gewinnen in modernen RIAs auch immer mehr andere Elemente als sensitive Bereiche an Bedeutung. Aber hier sieht der Anwender erst einmal nicht (zumindest nicht am Mauszeiger), dass er etwa mit der Maus auf ein Element klicken kann und damit eine Aktion auslöst. Das Aussehen des Mauszeigers können Sie mit DHTML gezielt steuern, sogar nur mit CSS und dem Hover-Effekt. Dazu geben Sie im Hover-Fall über die Eigenschaft `cursor` einen bestimmten Wert vor. Als Wertzuweisung für `cursor` sind etwa `crosshair`, `e-resize`, `hand` oder `help` als Werte möglich.

DHTML über die Manipulation des `style`-Objekts

Im Document Object Model gibt es seit der Version 2.0 das `style`-Objekt, worüber alle Stilinformationen von allen Elementen bzw. Elementknoten mit einer sichtbaren Repräsentation in der Webseite abgefragt und meist auch geändert werden können. Unabhängig vom konkreten Zugriff auf ein Element werden Sie in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit dem Element zur Verfügung haben und dementsprechend auch auf `style` zugreifen können.

```
document.getElementById("id").style.backgroundColor = yellow;
```

Die Stileigenschaften vom `style`-Objekt unterscheiden sich teilweise in der Schreibweise ein wenig von der Schreibweise bei den CSS-Regeln. Dies führt einerseits leicht zu Fehlern, aber andererseits lassen sich viele Namen der Eigenschaften gut aus CSS herleiten.

Im Allgemeinen gilt, dass der Bindestrich bei CSS-Attributen bei den Eigenschaften des `style`-Objekts entfällt und stattdessen der nachfolgende Buchstabe groß zu schreiben ist (Groß- und Kleinschreibung ist zu beachten – die Camel-Notation). Beispiele:

- ✓ Aus `font-size` bei CSS wird `fontSize` beim `style`-Objekt.
- ✓ Aus `background-color` bei CSS wird `backgroundColor` bei der Benennung der identischen Eigenschaft des `style`-Objekts.
- ✓ Aus `text-align` bei CSS wird `textAlign` beim `style`-Objekt.

Wenn man diese Regel einhält, werden in den meisten Fällen die Eigenschaften des `style`-Objekts direkt aus CSS-Attributen herzuleiten sein (und umgekehrt).

Achten Sie beim Umgang mit den Werten von Eigenschaften des `style`-Objekts darauf, dass viele Werte mit Einheiten (etwa px oder pt) angegeben werden müssen.

Beispiel: *changeColor.html*

Das nächste Beispiel zeigt folgende Situation: Wird auf ein farbig gekennzeichnetes Feld geklickt, sollen alle Absätze der Webseite entsprechend hervorgehoben werden.

Die Webseite

```

① <link type="text/css" href="lib/changeColor.css"
    rel="stylesheet" />
② <script type="text/javascript"
    src="lib/changeColor.js"></script>
</head>
<body>
③ <h1>Hintergrundfarbe ändern</h1>
    <p>Fortune plango vulnera stilantibus ocellis. Mihi ad
    enarrandum hoc argumentum comit, si ad ausculandum vostra
    erit benignitas. Qui autem auscultare nolet, exsurgat
    foras, ut sit, ubi sedeat ile qui auscultare vult. Nunc
    qua ad enarrandum hoc argumentums comit, si ad
    auscultandum vostra erit benignitas.</p>
    <p>Fortune plango vulnera stilantibus ocellis. Mihi ad
    enarrandum hoc argumentum comit, si ad ausculandum vostra
    erit benignitas. Qui autem auscultare nolet, exsurgat
    foras, ut sit, ubi sedeat ile qui auscultare vult.
    Nunc qua ad enarrandum hoc argumentums comit, si ad
    auscultandum vostra erit benignitas.</p>
④ <p>Fortune plango vulnera stilantibus ocellis. Mihi ad
    enarrandum hoc argumentum 190ommit, si ad ausculandum vostra
    erit benignitas. Qui autem auscultare nolet, exsurgat
    foras, ut sit, ubi sedeat ile qui auscultare vult.
    Nunc qua ad enarrandum hoc argumentums 190ommit, si ad
    auscultandum vostra erit benignitas.</p>
    <ul id="farbliste">
        <li id="farbe1" /><li id="farbe2" /><li id="farbe3" />
        <li id="farbe4" />
    </ul>
</body>
</html>

```

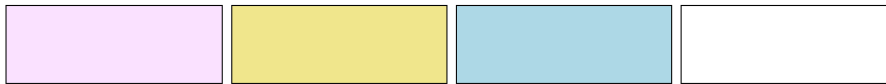
- ① Eine Referenz auf eine externe Style-Sheet-Datei wird eingebunden.
- ② Eine Referenz auf eine externe JavaScript-Datei wird eingebunden.
- ③ Verschiedene Absätze werden mit einem beliebigen Text gefüllt.
- ④ Die Liste wird mit der ID `farbliste` gekennzeichnet. Es gibt vier Listeneinträge, die eine eindeutige ID besitzen. Jeder Listeneintrag erhält über DHTML (in der JavaScript-Datei) eine andere Hintergrundfarbe. Beim Klick auf einen Listeneintrag soll die Funktion `changeColor()` aufgerufen werden.

Hintergrundfarbe ändern

Fortune plango vulnere stilandibus ocellis. Mihi ad enarrandum hoc argumentum comit, si ad ausculandum vostra erit benignitas. Qui autem auscultare nolet, exurgat foras, ut sit, ubi sedeat ile qui auscultare vult. Nunc qua ad enarrandum hoc argumentums comit, si ad auscultandum vostra erit benignitas.

Fortune plango vulnere stilandibus ocellis. Mihi ad enarrandum hoc argumentum comit, si ad ausculandum vostra erit benignitas. Qui autem auscultare nolet, exurgat foras, ut sit, ubi sedeat ile qui auscultare vult. Nunc qua ad enarrandum hoc argumentums comit, si ad auscultandum vostra erit benignitas.

Fortune plango vulnere stilandibus ocellis. Mihi ad enarrandum hoc argumentum comit, si ad ausculandum vostra erit benignitas. Qui autem auscultare nolet, exurgat foras, ut sit, ubi sedeat ile qui auscultare vult. Nunc qua ad enarrandum hoc argumentums comit, si ad auscultandum vostra erit benignitas.



Die Webseite vor einem Klick

Die CSS-Datei

```
① p {
  padding: 5px;
}
#farbliste {
  list-style-type: none; margin: 0; padding: 0; width: 800px;
}
#farbliste li {
  float: left; border: 1px solid black; width: 150px;
  height: 50px; margin: 3px; padding: 3px;
}
```

- ① Im Kopf der Webseite wird die externe Style-Sheet-Datei eingebunden, die die Listeneinträge festlegt. Diese sollen nicht mehr als Listeneinträge erkennbar sein, sondern als vier Farbboxen nebeneinander abgebildet werden.

Die JavaScript-Datei

```

window.onload = function() {
  ① document.getElementById("farbe1").style.backgroundColor =
    "#FAE1FF";
    document.getElementById("farbe2").style.backgroundColor =
    "#F0E68C";
    document.getElementById("farbe3").style.backgroundColor =
    "#ADD8E6";
    document.getElementById("farbe4").style.backgroundColor =
    "transparent";
    var liste = document.getElementsByTagName("li");
    ② for ( i = 0; i < liste.length; i++) {
      liste[i].onclick = function() {
        changeColor(this.id);
      };
    }
  };
  ③ function changeColor(id) {
    var farbe = document.getElementById(id).style.backgroundColor;
    var ps = document.getElementsByTagName("p");
    for (var i = 0; i < ps.length; i++) {
      ps[i].style.backgroundColor = farbe;
    }
  }
}

```

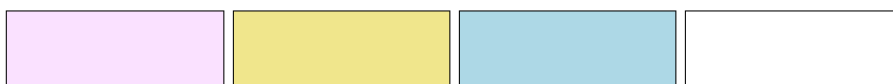
- ① In der Initialisierungsfunktion werden die Farben der Listeneinträge festgelegt. Dazu wird das `style`-Objekt verwendet. Jeder Listenpunkt wird über seine ID angesprochen. Beachten Sie, dass eine Formatierung von Elementen über die Angabe eines ID-Selektors in der externen CSS-Datei in den meisten Browsern nicht ausreicht, damit `style.backgroundColor` einen Wert hat. Der Wert soll in der Funktion `changeColor()` abgefragt werden, und deshalb muss er zwingend vorher gesetzt werden. Das kann man mit JavaScript machen (wie in dem Beispiel) oder mit einem Inline-Style (was eines der wenigen Argumente für den Einsatz eines Inline-Styles ist).
- ② Für jeden Listenpunkt wird eine Callback-Funktion registriert, die auf einen Klick reagiert. Beachten Sie, dass im Inneren der Callback-Funktion das DOM-Objekt des Listenpunkts über `this` bereitsteht. Damit kann man mit `this.id` auf den Wert des ID-Attributs des Listenpunkts zugreifen und diese ID der JavaScript-Funktion `changeColor()` als Parameter übergeben.
- ③ In der JavaScript-Funktion `changeColor()` wird die Hintergrundfarbe des angeklickten Elements bestimmt. Dazu wird die an die Funktion übergebene `id` verwendet, um mittels `getElementById()` das gewünschte Element zu selektieren. Die Farbe wird über das Objekt `style` und die Eigenschaft `backgroundColor` ermittelt. Damit alle Absätze der Webseite entsprechend angesprochen werden können, werden sie über `getElementsByTagName("p")` ermittelt und in der Variablen `ps` abgelegt. Mithilfe der `for`-Schleife wird jedem `p`-Element die gespeicherte Hintergrundfarbe übergeben.

Hintergrundfarbe ändern

Fortune plango vulnera stilantibus ocellis. Mihi ad enarrandum hoc argumentum comit, si ad ausculandum vostra erit benignitas. Qui autem auscultare nolet, exurgat foras, ut sit, ubi sedeat ile qui auscultare vult. Nunc qua ad enarrandum hoc argumentums comit, si ad auscultandum vostra erit benignitas.

Fortune plango vulnera stilantibus ocellis. Mihi ad enarrandum hoc argumentum comit, si ad ausculandum vostra erit benignitas. Qui autem auscultare nolet, exurgat foras, ut sit, ubi sedeat ile qui auscultare vult. Nunc qua ad enarrandum hoc argumentums comit, si ad auscultandum vostra erit benignitas.

Fortune plango vulnera stilantibus ocellis. Mihi ad enarrandum hoc argumentum comit, si ad ausculandum vostra erit benignitas. Qui autem auscultare nolet, exurgat foras, ut sit, ubi sedeat ile qui auscultare vult. Nunc qua ad enarrandum hoc argumentums comit, si ad auscultandum vostra erit benignitas.



Aussehen der Absätze nach dem Klick auf eine Farbauswahl

11.3 Umgang mit Multimedia

Webseiten, die möglichst viele Besucher anziehen sollen, verwenden immer mehr multimediale Technologien. Früher waren es Java-Applets oder Flash, heute sind es Streaming Videos oder Musik. Über JavaScript können Sie mit solchen Elementen kommunizieren und diese steuern. Da im WWW HTML5 immer mehr an Bedeutung gewinnt, wird in den folgenden Abschnitten gerade auf die Steuerung von Audio- und Videodateien eingegangen, die in HTML5 über neue Tags bzw. DOM-Elemente `<audio>` und `<video>` eingeführt werden.

Mit diesen Elementen ist es nicht mehr notwendig, dass Anwender zusätzliche Plug-ins installieren, um Musik hören oder ein Video anschauen zu können. Die HTML5-kompatiblen Browser haben jeweils einen integrierten Player, den Sie per JavaScript steuern können.

Für die Steuerung der Audio-Elemente steht das `HTMLAudioElement` und für Videos das `HTMLVideoElement` zur Verfügung.

Den Audio- und den Videoplayer sprechen Sie in JavaScript über dessen ID an und steuern ihn über die nachfolgenden Methoden und Eigenschaften.

Methode/Eigenschaft	Bedeutung
<code>autoplay</code>	Soll das Musikstück sofort nach dem Laden bzw. dem Liedwechsel starten, geben Sie hier den Wert <code>true</code> an.
<code>currentsrc</code>	Das aktuell gespielte Musikstück lässt sich über diese Methode abfragen.
<code>currentTime</code>	Hiermit können Sie die aktuelle Spielzeit auslesen und setzen.
<code>duration</code>	Mit <code>duration</code> ermitteln Sie die Gesamtlänge des geladenen Musikstücks.

Methode/Eigenschaft	Bedeutung
loop	Mit <code>true</code> oder <code>false</code> legen Sie fest, ob das Musikstück wiederholt werden soll oder nicht.
muted	Hiermit können Sie den Ton abschalten (<code>true</code>) oder wieder einschalten (<code>false</code>).
paused	Damit lesen Sie aus, ob das Abspielen eines Liedes angehalten wurde.
play()	Dies startet das Abspielen des Liedes von der aktuellen Stelle.
pause()	Mit dieser Methode können Sie das Abspielen eines Liedes anhalten.
src	Das Musikstück, das abgespielt werden soll, wird hiermit übergeben.
volume()	Damit können Sie die Lautstärke steuern. Dabei sind Werte von 0.0 (aus) bis 1.0 (maximale Lautstärke) möglich.

Beispiel: *audio-js.html*

In dem nachfolgenden Beispiel steuern Sie den im Browser integrierten Audioplayer.

Die Webseite

①	<code><script type="text/javascript" src="lib/auto-js.js"></code> <code></script></code>
	<code></head></code> <code><body></code>
②	<code><audio id="myaudio" controls="controls"></code>
③	<code><source src="song.ogg" type="audio/ogg" /></code> <code><source src="song.mp3" type="audio/mpeg" /></code> HTML5 audio wird nicht unterstützt
④	<code></audio>
</code> <code><button>Abspielen</button><button>&lt;&lt;</code> <code>-30Sek.</button></code> <code><button>&lt;&lt; -10Sek.</button><button>+10Sek &gt;</code> <code></button></code> <code><button>+30Sek &gt;&gt;</button><button>Neustart</button></code> <code></body></code> <code></html></code>

- ① Eine Referenz auf eine externe JavaScript-Datei wird eingebunden.
- ② Im Dokumentkörper geben Sie über das HTML-Tag `<audio>` an, dass der Audioplayer mit seinen Kontrollelementen (`controls`) angezeigt werden soll. Der Player kann per JavaScript über die entsprechende ID `myaudio` angesprochen werden.
- ③ Die zu ladende Musikdatei wird über das Tag `<source>` definiert. Da nicht jeder Browser die gleichen Musikformate unterstützt, werden zwei Musikformate angeboten, `audio/ogg` und `audio/mpeg`.
- ④ Zur Steuerung per JavaScript werden einige Schaltflächen festgelegt, die bei einem Klick eine entsprechende Funktion aufrufen.

Die JavaScript-Datei

```
① window.onload = function() {
②   var b = document.getElementsByTagName("button");
③   b[0].onclick = playAudio;
   b[1].onclick = function() {
       rewindAudio(30.0);
   };
   b[2].onclick = function() {
       rewindAudio(10.0);
   };
   b[3].onclick = function() {
       forwardAudio(10.0);
   };
   b[4].onclick = function() {
       forwardAudio(30.0);
   };
   b[5].onclick = restartAudio;
};
④ function getAudioElement() {
   var oAudio = null;
   if (window.HTMLAudioElement) {
       oAudio = document.getElementById('myaudio');
   }
⑤   return oAudio;
}
⑥ function playAudio() {
   oAudio = getAudioElement();
   if (oAudio != null){
⑦       var btn = document.getElementById('play');
       if (oAudio.paused) {
           oAudio.play();
           btn.textContent = "Pause";
       } else {
           oAudio.pause();
           btn.textContent = "Abspielen";
       }
   }
}
⑧ function rewindAudio(sek) {
   oAudio = getAudioElement();
   if (oAudio != null){
       oAudio.currentTime -= sek;
   }
}
```

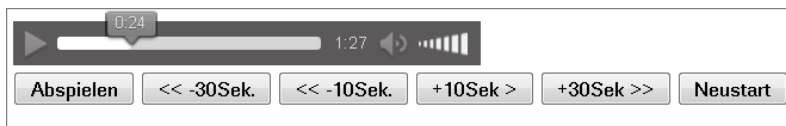
```

⑨ function forwardAudio(sek) {
    oAudio = getAudioElement();
    if (oAudio != null){
        oAudio.currentTime += sek;
    }
}

⑩ function restartAudio() {
    oAudio = getAudioElement();
    if (oAudio != null){
        oAudio.currentTime = 0;
    }
}

```

- ① Für alle Schaltflächen wird eine `onclick`-Reaktion aufgebaut.
- ② Wenn kein Parameter übergeben wird, wird eine Funktionsreferenz benutzt.
- ③ Oder eine anonyme Callback-Funktion wird registriert, wenn beim Aufruf einer Funktion ein Parameter zu übergeben ist.
- ④ In der Funktion `getAudioElement()` wird die Variable `oAudio` auf `null` gesetzt und dann überprüft, ob der Browser das Objekt *HTMLAudioElement* unterstützt. Ist dies der Fall, wird der Variablen `oAudio` mithilfe der Methode `getElementById` das Element mit der ID `myaudio` zugewiesen.
- ⑤ Die Funktion gibt über das Schlüsselwort `return` die Variable `oAudio` zurück, mit der Sie den Audioplayer ansprechen können.
- ⑥ Über die Funktion `playAudio()` steuern Sie das Anhalten und Abspielen des Musikstücks. Über die Funktion `getAudioElement()` wird das *HTMLAudioObjekt* zurückgeliefert. Hat `oAudio` nicht den Wert `null` zurückgeliefert, weisen Sie der Variablen `btn` das Element mit der ID `play` zu.
- ⑦ Über die Eigenschaft `paused` kontrollieren Sie, ob sich der Player im Pause-Zustand befindet. Ist dies der Fall, wird er über `play()` wieder gestartet. Über `btn.textContent` erhält die entsprechende Schaltfläche die Aufschrift *Pause*. Im anderen Fall wird das Musikstück mit `pause()` angehalten und die Schaltfläche mit *Abspielen* beschriftet.
- ⑧ Mit der Funktion `rewindAuto()` wird von der aktuellen Spielzeit `currentTime` der übergebene Sekundenwert `sek` abgezogen. Der Player springt automatisch an die entsprechende Stelle.
- ⑨ Ebenso funktioniert die Funktion `forwardAudio()`. Hier wird die Zeit hinzuaddiert.
- ⑩ Über die Funktion `restartAudio()` und die Übergabe des Wertes 0 an `currentTime` wird der Player an den Anfang zurückgesetzt.



Der Mediaplayer

Die Videosteuerung funktioniert analog der Audiosteuerung. Über das Tag `<video>` definieren Sie das anzuzeigende Video, wobei hier die beiden Formate `video/ogg` bzw. `video/mpeg` verwendet werden sollten. Dies setzt voraus, dass Sie das Video im OGG-Format (Theora Video Codec und Vorbis Audio Codec) und im MPG-4-Format (H264 Video Codec und AAC Audio Codec) vorliegen haben. Dieses Format wird derzeit von den meisten Browsern unterstützt, und es ist zu erwarten, dass bald alle Browser damit zurechtkommen. Möglicherweise werden in Zukunft auch andere Formate unterstützt.

11.4 Datenspeicherung im Client

In der Vergangenheit konnte man auf dem Rechner eines Besuchers nur über Cookies Daten speichern. **Cookies** (deutsch: Kekse) sind Informationen, die vom Browser auf der Festplatte des Anwenders gespeichert werden und zu einem späteren Zeitpunkt wieder abgefragt werden können. Cookies sind einfach (reiner Text) und sehr beschränkt. Obwohl man auch heute noch gelegentlich Cookies einsetzt, hat die mangelnde Leistungsfähigkeit und Zuverlässigkeit (jeder Anwender kann Cookies auf seinem Rechner löschen) zu diversen Alternativen geführt, die insbesondere mit HTML5 einen neuen Standard einführen. In dem Abschnitt sehen Sie kurz die Anwendung von einem Cookie und vor allen Dingen die neue Technik des Local Storage.

Cookies mit JavaScript

Cookies werden in JavaScript über die Eigenschaft `cookie` des `document`-Objekts erstellt und ausgelesen. Als Wert wird beim Schreiben nur eine beliebige Zeichenkette erwartet. Um allerdings die Daten wieder sinnvoll auslesen zu können, muss ein Cookie einen gewissen Aufbau haben. Die Zeichenkette, die Sie im Cookie speichern können, hat den folgenden Aufbau:

```
"name=Wert; expires=Verfallszeit; path=Pfad; domain=Domainname; secure"
```

- ✓ Bis auf den Parameter `name` sind alle Angaben optional.
- ✓ Mit dem Parameter `Wert` legen Sie fest, welcher Inhalt im Cookie gespeichert werden soll.
- ✓ Mit der Angabe einer Verfallszeit können Sie festlegen, wie lange die Information verfügbar sein soll. Die Zeitangabe erfolgt im UNIX-Zeitstempel-Format. Dies bedeutet, wenn das Cookie zwei Tage gespeichert werden soll, geben Sie die aktuelle Zeit in Sekunden an zuzüglich 172.800 Sekunden (86.400 Sekunden pro Tag). Geben Sie keine Verfallszeit an, wird das Cookie gelöscht, sobald der Nutzer den Browser schließt. Wollen Sie ein vorhandenes Cookie löschen, geben Sie einfach einen Wert in der Vergangenheit an. Eine explizite Löschfunktion gibt es nicht.
- ✓ Ein Cookie kann standardmäßig nur von Dokumenten ausgelesen werden, die unterhalb des Verzeichnisses liegen, in dem sich die Cookie setzende Webseite befindet. Wird ein Cookie beispielsweise durch das HTML-Dokument `http://www.example.com/test/Setze.html` gesetzt, kann es von `http://www.example.com/test/Cookies/Lese.html`, jedoch nicht von `http://www.example.com/Lese.html` gelesen werden. Sollen alle HTML-Dokumente eines Servers Zugriff auf die Informationen haben, geben Sie als Pfad das Hauptverzeichnis `/` der Homepage an.

- ✓ Statt von der eigenen Domain kann ein Cookie auch von Webseiten einer anderen Domain, z. B. *www.IhreFirma.de*, ausgelesen werden. Dies sind sogenannte Cookies für Drittanbieter.
- ✓ Der Parameter `secure` legt fest, dass der Zugriff auf das Cookie nur über eine gesicherte SSL-Verbindung erfolgen darf.

Bei der Angabe von Parametern müssen Sie darauf achten, dass alle zuvor erwarteten Cookie-Parameter ebenfalls angegeben werden. Werden diese nicht benötigt, sind sie als leere Zeichenkette in Anführungszeichen anzugeben.

Zum Umgang mit Cookies gibt es diverse Standardfunktionen, die zwar nicht zu JavaScript gehören, aber im Internet frei verfügbar sind. In den BuchPlus-Beispieldateien finden Sie die Datei *cookie.js* mit den Funktionen `writeCookie()`, `readCookie()` und `deleteCookie()`.

Local Storage und Session Storage

Mit HTML5 wurde die lokale Speicherung von Daten persistenter (dauerhafter), mächtiger, leichter zu handhaben und schneller abrufbar als mit Cookies. Vor allen Dingen ist die Menge der speicherbaren Daten erheblich größer geworden. Denn Anwendungen von lokalen Speichervorgängen sind – trotz Ajax – immer noch vorhanden, z. B. To-Do-Listen, Einkaufszettel, Notizbücher etc. Im Konzept von HTML5 gibt es u.a. drei Möglichkeiten, um Daten beim Benutzer zu speichern:

- ✓ Local Data Storage oder kurz Local Storage: Grundsätzlich sollten diese Daten permanent im Speicher des Browsers bleiben, bis sie mit speziellen Methoden oder automatischen Bereinigungen vom Browser beseitigt werden.
- ✓ Session Data Storage oder kurz Session Storage: Hiermit speichern Sie Informationen nur so lange, wie eine Sitzung im Browser dauert. Und da in den Browsern auch jeder Tab bzw. jedes Browserfenster als eigene Sitzung verwaltet wird, gibt es auch jeweils einen eigenen Sitzungsspeicher. Deshalb kann man von einem Tab/Fenster nicht auf die Daten in einem anderen Tab/Fenster zugreifen, was beim Local Data Storage möglich ist.
- ✓ Web SQL Database (worauf hier nicht eingegangen wird)

Das Konzept des Local Storage ist wirklich lokal. Das bedeutet, serverseitige Applikationen können diese gespeicherten Daten nicht direkt auslesen bzw. verändern oder erstellen. Die gespeicherten Daten werden nicht wie Cookies zum Server geschickt. Der Weg des Zugriffs führt immer über JavaScript und damit über eine rein clientseitige Programmierung. Zudem sind gespeicherte Daten allgemein nur für jeweils eine spezifische Domain und die eigene Browserinstanz zugänglich, wobei sich Browser hier in den genauen Details unterscheiden.

Die Objekte `localStorage` und `sessionStorage` sowie deren Methoden und Eigenschaft

Die eigentliche Verwendung von lokalen Speichern ist einfach. Es gibt ein neues Objekt `localStorage`, das im Zentrum des gesamten Verfahrens steht. Bei der sitzungsbezogenen Speicherung arbeiten Sie mit dem Objekt `sessionStorage`. Beide stellen einige identische Methoden und eine Eigenschaft zur Verfügung. Die einzige Eigenschaft ist `length` mit der Anzahl der gespeicherten Elemente.

Die Methoden eines Objekts vom Typ `localStorage` bzw. `sessionStorage` sind folgende:

Methode	Bedeutung
<code>clear()</code>	Damit wird der gesamte Speicher gelöscht.
<code>getItem()</code>	Damit erhalten Sie den Wert, der hinter dem als Parameter angegebenen Schlüssel abgelegt ist. Gibt es den Schlüssel nicht, erhalten Sie den Wert <code>null</code> .
<code>key()</code>	Damit erhalten Sie den Namen des Schlüssels an der als numerischen Parameter angegebenen Position. Ist der Wert des Parameters größer als die Anzahl der Einträge im Speicher, erhalten Sie den Wert <code>null</code> .
<code>removeItem()</code>	Die Methode löscht das spezifizierte Schlüssel-Werte-Paar aus dem Storage. Der Parameter ist der Schlüssel.
<code>setItem()</code>	Die Methode verwendet zwei Parameter, die ein klassisches Schlüssel-Werte-Paar darstellen. Man speichert damit eine neue Information (2. Parameter), die über den Schlüssel (1. Parameter) adressiert wird. Gibt es den Schlüssel schon, wird der vorhandene Wert überschrieben. Als Werte können Sie Strings speichern, aber auch andere Formate. JSON-Objekte bieten sich für komplexere Strukturen geradezu an.

Ein Beispiel, um einfache Daten in einem Local Storage abzulegen – *storage.html*

In der Webseite werden mit den Schaltflächen Daten gespeichert, ausgelesen und gelöscht.

Die Webseite

①	<code><script type="text/javascript"</code> <code>src="lib/storage.js"></script></code>
	<code></head></code> <code><body></code>
	<code><h1>Local Data Storage mit HTML5</h1></code>
②	<code><h1>Local Data Storage mit HTML5</h1></code> <code><button id="b1">Speichere Zufallswert</button></code> <code><button id="b2">Lese lokalen Speicher aus</button></code> <code><button id="b3">Lösche lokalen Speicher</button></code>
③	<code><div id="ausgabe"></div></code>
	<code></body></code> <code></html></code>

- ① Eine Referenz auf eine externe JavaScript-Datei wird eingebunden.
- ② Im Dokumentkörper werden drei Schaltflächen definiert.
- ③ Der Ausgabebereich wird deklariert.

Die JavaScript-Datei

```

① var counter = 0;
window.onload = function() {
②   document.getElementById("b1").onclick = function() {
       localStorage.setItem("key" + counter++, Math.random());
   };
   document.getElementById("b2").onclick = function() {
③     document.getElementById("ausgabe").innerHTML = „“;
       var j = 0;
④     for (var i in localStorage) {
         if (localStorage.key(j) != null){
           document.getElementById("ausgabe").innerHTML +=
             localStorage.key(j++) + ": " +
             localStorage.getItem(i) + "<br />";
         }
       }
   };
⑤   document.getElementById("b3").onclick = function() {
       counter = 0;
       localStorage.clear();
       document.getElementById("ausgabe").innerHTML = "";
   };
};

```

- ① Eine Zählvariable wird definiert, über die die Anzahl der Einträge im Speicher gezählt wird.
- ② Für alle Schaltflächen wird eine `onclick`-Reaktion aufgebaut.
- ③ Mit dem Klick auf die erste Schaltfläche speichern Sie eine Zufallszahl. Deren Schlüssel ergibt sich aus dem Token "key" und dem Wert einer globalen Variablen `counter`, die am Beginn des Skripts angelegt wurde und nach jedem Speichern eines Werts mittels `setItem()` um den Wert 1 erhöht wird. Mit jedem Klick eines Anwenders wird also ein neuer Wert im lokalen Speicher angelegt.
- ④ Die Funktionalität zum Auslesen des Inhalts vom lokalen Speicher ist etwas umfangreicher. Alle gespeicherten Werte inklusive der Namen des jeweiligen Schlüssels werden ausgelesen und ausgegeben. Dazu arbeiten Sie mit einer `for...in`-Schleife, bei der `i` den Schlüssel repräsentiert. Synchron zum jeweiligen Schleifendurchlauf zählen wir die lokale Variable `j` hoch, die Sie als Index in der Methode `key()` verwenden, um darüber den Namen des Schlüssels abzufragen. Mit `getItem(i)` ermitteln Sie den jeweils gespeicherten Wert.
- ⑤ Mit dem Klick auf die dritte Schaltfläche wird der lokale Speicher mittels `clear()` gelöscht und der Zähler zurückgesetzt, damit über die erste Schaltfläche neu angelegte Schlüssel-Werte-Paare wieder mit dem Schlüssel "key0" beginnen werden.

Local Data Storage mit HTML5

key0: 0.23903527312349804
 key1: 0.013798495603277616
 key2: 0.27103190853634085

Die Daten im Speicher

Die Zukunft von JavaScript

Obwohl die Unterstützung der offiziellen Versionen von JavaScript sehr eingeschränkt ist, haben Ajax, HTML5 und die weiteren Versionen der Mozilla Foundation bzw. ECMA über die Zeit verschiedene Erweiterungen von JavaScript bewirkt, deren Einsatz bei der Einführung allerdings nur in einigen Browsern möglich ist. Allerdings hat man mit HTML5 gesehen, dass sich neue Entwicklungen über die Jahre etablieren.

JavaScript-Erweiterungen, die unter HTML5 gefasst werden

- ✓ Session Storage und Local Storage
- ✓ Zeichnen von grafischen Formen mit Canvas
- ✓ Lokalisierung von Besuchern einer Webseite mit Geolocation
- ✓ Web Worker (Hintergrundprozesse, die den Browser nicht blockieren)
- ✓ Push Services
- ✓ Web-Sockets

11.5 Übung

Cookies

Übungsdatei: *kap11/cookies.js* **Ergebnisdateien:** *kap11/uebung1.html, kap11/uebung2.html, kap11/uebung3.html, kap11/uebung4.html*

1. In der JavaScript-Datei *kap11/cookies.js* finden Sie die Standardfunktionen zum Setzen, Löschen und Auslesen von Cookies. Erstellen Sie Anwendungen von diesen Möglichkeiten.

12

Frameworks

12.1 Was sind Frameworks?

RIAs mit ihren reichhaltigen Möglichkeiten verändern die Art der Nutzung des WWW. Ebenso wird die Bedeutung von klassischen Desktop-Applikationen neu positioniert. Viele früher nur als Desktop-Applikation genutzte Programmtypen finden sich plötzlich im Web und werden mit dem Browser ausgeführt. Seien es persönliche Kalender, vollständige Office-Programme, Spiele, Routenplaner, ganz integrierte Entwicklungsumgebungen oder Kommunikationsprogramme. Auch mobile Webseiten oder Anwendungen für Tablets oder Smartphones basieren mehr und mehr auf Web-Technologie. Dies verändert nicht zuletzt das Anwenderverhalten wie auch die Anwendererwartung bei Internet-Applikationen im Allgemeinen sowie der Verfügbarkeit von Leistungen. RIAs stehen auf der einen Seite als klassische Web-Applikationen (aber mit einem gewissen Mehrwert) immer zur Verfügung, wenn man einen halbwegs schnellen Internetzugang und einen modernen Browser hat. Andererseits sind sie teilweise von der Bedienung, der Performance wie auch Optik mittlerweile fast gar nicht mehr von klassischen Desktop- oder Mobil-Applikationen zu unterscheiden.

Frameworks und Toolkits

Die Erstellung von RIAs ist kaum ohne den Einsatz von Web-Frameworks oder Toolkits zu bewerkstelligen, zumindest nicht mit einem adäquaten Aufwand. Bedenken Sie, dass Sie sich beim Einsatz eines Frameworks oder Toolkits in Abhängigkeit von einem Hersteller respektive einem Projekt begeben. Ebenso erfordert der Einsatz von Frameworks/Toolkits die Einarbeitung in die jeweiligen Funktionsbibliotheken und Arbeitsweisen dieses Systems.

Was genau ein Framework ist und wie es sich von einem Toolkit unterscheidet, ist nicht standardisiert. Sowohl eine verbindliche Definition als auch eine Abgrenzung ist nicht einfach. Allgemein versteht man unter einem Framework ein Programmiergerüst, das bestimmte Funktionalitäten bereitstellt. Ein Framework ist selbst kein fertiges Programm, sondern stellt den Rahmen (Frame) zur Verfügung, innerhalb dessen ein oder mehrere Programmierer eine Anwendung erstellen. Ein Framework beinhaltet in der Regel eine Bibliothek mit nützlichen vorgegebenen Codestrukturen (meist auf Basis einer spezifischen Sprache wie JavaScript), legt aber – im Gegensatz zu einer reinen Bibliothek – auch eine Steuerung von Verhaltensweisen bei der Verwendung fest (z. B. eine Syntax bzw. Grammatik). Frameworks können auf verschiedene Weise arbeiten. Beliebte sind JavaScript-Frameworks, die dem Programmierer im Browser Unterstützung bei JavaScript-Aktionen liefern.

Aber auch auf Seiten des Webserver gibt es Frameworks und im Browser bekommen Besucher meist gar nicht mit, dass Webseiten im Hintergrund von einem Framework generiert wurden. Dazu gibt es Frameworks, die bestimmte Entwurfsmuster bereitstellen sowie spezielle Techniken auf dem Server fordern.

Bei einem Toolkit steht dagegen mehr die Sammlung an Programmen (Tools) im Fokus, die aber durchaus auch auf spezifischen Bibliotheken und einem Syntaxkonzept aufsetzen können. Sowohl ein Framework, aber vor allen Dingen Toolkits stellen vielfach sogenannte **Widgets** beziehungsweise Komponenten zur Verfügung. Dabei handelt es sich in der Regel um Elemente, aus denen eine grafische Benutzerschnittstelle (UI – User Interface oder GUI – Graphical User Interface) zusammengesetzt wird.

12.2 Einsatz von reinen JavaScript-Frameworks anhand von jQuery

jQuery ist ein beliebtes JavaScript-Framework und soll den Einsatz von clientseitigen Frameworks als Stellvertreter demonstrieren. Das Kern-Framework selbst unterstützt beim Programmieren von JavaScript durch eine darauf aufbauende zusätzliche Syntax und stellt komfortable Funktionalitäten zur Verfügung. Zum Beispiel:

- ✓ sichere und einfache Techniken zur Manipulation des DOM,
- ✓ erweiterte Ajax-Funktionalität,
- ✓ Effekte und Animationen,
- ✓ universelle CSS-Unterstützung und
- ✓ eine erweiterte Ereignisbehandlung.

Es gibt auch eine einfache Funktionserweiterung durch frei verfügbare Plug-ins, beispielsweise jQuery UI. jQuery ist die Basis, die beim Quellcode und der Programmierung unterstützt, während jQuery UI darauf als eigenständiges Framework aufbaut und optisch ausgereifte Oberflächenkomponenten und ein CSS-Themeframework zur Verfügung stellt. Das Mobile Framework, das wie jQuery UI auf nativem jQuery unmittelbar aufsetzt, stellt Unterstützung für mobile Webseiten bereit. Auch Bootstrap basiert auf jQuery, vereinheitlicht aber den mobilen und den Desktop-Ansatz.

Herunterladen können Sie das kostenlose Framework unter <https://www.jquery.com>. Dieses besteht aus einer JavaScript-Datei, die Sie in einem Verzeichnis ablegen und über das Skript-Tag im Kopfbereich Ihrer Webseite einbinden. Oder Sie laden die Datei direkt vom Server:

```
<script src="https://code.jquery.com/jquery-3.7.0.min.js"
  integrity="sha256-2Pmvv0kuTBOenSvLm6bvfbSSHrUU+3A7x6P5Ebd07/g="
  crossorigin="anonymous"></script>
```

In diesem Beispiel handelt es sich um die Version 3.7. Die Angabe von `min` im Dateinamen weist darauf hin, dass dies eine minimierte Version von jQuery ist, in der alle überflüssigen Zeichen entfernt worden sind. Die Datei ist daher sehr kompakt und auf die schnelle Übertragung optimiert, aber in einem Texteditor nur schwer zu lesen. Die Integritäts- und Crossorigin-Attribute werden für die SRI-Prüfung (Subresource Integrity) verwendet.

Dadurch können Browser sicherstellen, dass auf Servern Dritter gehostete Ressourcen nicht manipuliert wurden. Die Verwendung von SRI wird als Best Practice empfohlen, wenn Bibliotheken von einer Drittanbieterquelle geladen werden.

Wenn Sie mit jQuery arbeiten, können Sie im JavaScript die nachfolgende Funktion verwenden:

```
$(document).ready(function() {
});
```

Mit diesen Zeilen geben Sie an, dass das darin auszuführende JavaScript erst gestartet werden soll, wenn das komplette HTML-Dokument geladen wurde und der DOM-Baum fertig ist. Von Hand wurde das im Buch immer mit `window.onload=function() { ...}` erledigt, das Framework kümmert sich jedoch im Hintergrund noch um diverse Korrekturen und Anpassungen.

Selektoren

Die grundlegende Schreibweise in jQuery sieht folgendermaßen aus:

```
$(Selektor).Methode
```

Das `$`-Zeichen kennzeichnet den jQuery-Namensraum (engl.: namespace) zum Ansprechen der Elemente. Über den Selektor geben Sie das Element an, das angesprochen werden soll. Mit der Methode legen Sie fest, was mit dem Element passieren soll.

Die Selektoren in jQuery entsprechen im Wesentlichen denen von CSS3. Sie können dort beispielsweise den Namen eines Elements, die ID oder den Klassennamen angeben. Sie können aber auch Elemente über die von CSS bekannten Selektoren ansprechen (Nachfahre-Selektor, Kind-Selektor, Attribut-Selektor).

Selektion	Schreibweise
Ein HTML-Element	<code>\$('div')</code>
Mehrere HTML-Elemente	<code>\$('div, h1, p')</code>
Ein mit <code>id</code> definiertes HTML-Element	<code>\$('#id')</code>
Ein HTML-Element mit bestimmtem CSS-Klassennamen	<code>\$('div.Klassename')</code>
Nachfahre eines Elements	<code>\$('div a')</code>
Kind des Elements	<code>\$('h1 > div')</code>
Attribut eines Elements	<code>\$('[align]')</code>
Attributswert	<code>\$('[align=right]')</code>
HTML-Element und Attribut	<code>\$('p[align=right]')</code>

Ereignisbehandlung

Nach der Selektion eines Elements in einer Webseite können Sie mit jQuery angeben, auf welches Ereignis reagiert werden soll. Dazu gibt es diverse Methoden, die an den klassischen Eventhandlern orientiert sind.

```
$(Selektor).click( function() {  
});
```

Beispiel: *jquery-start.html*

In diesem Beispiel sollen die Absätze eines HTML-Dokuments über drei verschiedene Schaltflächen angesprochen werden. Das Aussehen der Absätze soll entsprechend per CSS verändert werden.

```
<!DOCTYPE html>  
<html>  
<head>  
  <meta charset="utf-8">  
  <title>Mein erstes jQuery Dokument</title>  
① <script src="https://code.jquery.com/jquery-3.7.0.min.js"  
  integrity=  
  "sha256-2Pmvv0kuTBOenSvLm6bvfBSSHrUJ+3A7x6P5Ebd07/g="&br/>  crossorigin="anonymous"></script>  
</head>  
<body>  
  <h3>Ansprechen der HTML-Elemente</h3>  
② <p>Absatz ohne Attribut und ID: Fortune plango vulnera  
  stilantibus ...</p>  
  <p align="right">rechtsbündiger Absatz: Fortune plango vulnera  
  ...</p>  
  <p id="absatz3">Absatz mit der ID "absatz3":  
  Fortune plango vulnera ...</p>  
  <div id="buttons">  
③ <button id="btn1">Alle Absätze grün einfärben</button>  
  <button id="btn2">nur rechtsbündige Absätze rot einfärben  
  </button>  
  <button id="btn3">Absatz mit ID rechtsbündig ausrichten  
  </button>  
</div>  
  <script type="text/javascript">  
④ $(document).ready(function() {  
⑤   $('#btn1').click(function() {  
⑥     $('p').css('background-color', '#47b300' );  
    });  
⑦   $('#btn2').click(function() {  
    $('p[align=right]').css('background-color', '#b30047' );  
  });  
}
```

```

⑧      $('#btn3').click(function(){
        $('#absatz3').css('text-align', 'right' );
      });
    });
</script>
</body>
</html>

```

- ① Als Erstes binden Sie die jQuery-Bibliothek in die Webseite ein. In diesem Fall befindet sich die Bibliotheksversion 3.4.1 im Unterverzeichnis *jquery*.
- ② Im nächsten Schritt geben Sie drei Absätze an, wobei der zweite rechts ausgerichtet wird und der dritte Absatz die ID `absatz3` erhält.
- ③ Über die drei Schaltflächen, denen jeweils eine andere ID zugewiesen wird, sollen die drei Absätze verändert werden. Das Ereignis `onclick` wird hier nicht angegeben.
- ④ Mit `$(document).ready(function() { });` leiten Sie die Funktion zum Ansprechen der jQuery-Funktionen ein. Diese sollen überwacht und ausgeführt werden, sobald das Dokument komplett geladen wurde.
- ⑤ In dieser Zeile wird das HTML-Element mit der ID `btn1` selektiert. Mit der Methode `click()` leiten Sie die Methode ein, die aufgerufen werden soll, wenn auf das Element mit der ID `btn1` geklickt wird.
- ⑥ Ist dies der Fall, werden über `$('p')` alle Absätze selektiert und ihnen wird über die Methode `css()` die entsprechende CSS-Formatierung zugewiesen. Hier wird z. B. die Hintergrundfarbe Grün gesetzt.
- ⑦ Wird auf das Element mit der ID `btn2` geklickt, werden mit `$('p[align=right]')` nur die rechtsbündig ausgerichteten Absätze selektiert und der Hintergrund wird entsprechend rot eingefärbt.
- ⑧ Hier legen Sie das Klickereignis für Element `btn3` fest, bei dem der Absatz mit der ID `absatz3` rechtsbündig ausgerichtet werden soll.

Wie Sie im Quelltext sehen können, werden die Ereignisse, die abgefragt werden sollen, nicht direkt am Element angegeben, sondern über den Selektor innerhalb der jQuery-Routine definiert.

Weitere Features von jQuery

Mit jQuery können Sie Elemente animiert ein- und ausblenden lassen. Bekannt sind diese Effekte z. B. in FAQs, in denen man auf eine Frage klickt und die Antwort animiert ein- und ausgeblendet wird. Dies ist in jQuery über die Methode `slideToggle()` zu realisieren.

Ein weiteres Feature von jQuery ist die Integration der Ajax-Funktionalität. Hierfür wird in der JavaScript-Bibliothek eine erweiterte Form des `XMLHttpRequest`-Objekts zur Verfügung gestellt.

jQuery ist so aufgebaut, dass Erweiterungen erstellt und in eigene Skripte eingebunden werden können. Das bekannteste Plug-In ist das jQuery User Interface, kurz jQuery UI. Neben einer Reihe von Interaktionen, wie Drag & Drop und Sortieren, oder erweiterten Effekten, wie Farbanimationen und Überblendeffekten, stellt das jQuery UI auch eine Reihe visueller Steuerelemente zur Verfügung, die unter dem Begriff **Widgets** zusammengefasst sind.

12.3 Einsatz von JavaScript-Frameworks mit Entwurfsmuster anhand von Vue.js

Die Bedeutung von rein syntaktisch konzipierten JavaScript-Frameworks war lange Zeit sehr groß, geht aber mittlerweile etwas zurück. Denn andere Formen von Frameworks werden immer bedeutender. Diese verwenden über die reine Unterstützung bei der Syntax hinaus oft ein Design-Pattern wie **MVC** oder **MVVC** samt teils spezieller Laufzeitumgebungen auf dem Server.

MVC (Model View Controller) ist ein Softwareentwurfsmuster, das die Logik der Anwendung von der Benutzeroberfläche trennt. Es besteht aus drei Komponenten: dem Model, der View (Ansicht) und dem Controller.

- ✓ Das Modell ist für die Datenverwaltung zuständig.
- ✓ Die View ist für die Darstellung der Daten zuständig.
- ✓ Der Controller ist für die Verarbeitung der Benutzereingaben zuständig.

MVVC (Model View ViewModel) ist ein verwandtes Softwareentwurfsmuster, das aber mehr Funktionen bietet. Es besteht aus vier Komponenten: dem Modell, der Ansicht, dem ViewModel und dem Controller.

Die drei Bestandteile von MVC sind weitgehend gleich, das ViewModel ist jedoch zusätzlich für die Verarbeitung der Benutzereingaben zuständig und kann die Ansicht aktualisieren, ohne dass der Controller eingreifen muss.

Beispiele für Web-Frameworks, die mit MVC oder MVVC arbeiten, sind ASP.NET MVC, Ruby on Rails, AngularJS, React und Vue.js (<https://vuejs.org>).

Ein typischer Vertreter des MVVC-Konzepts ist das immer populärer werdende Vue.js, das zur Erstellung von Benutzeroberflächen verwendet wird. Es ist ein leichtgewichtiges, modulares Framework, das eine einfache Syntax und eine intuitive Struktur bietet. Vue.js ermöglicht es, komplexe Benutzeroberflächen zu erstellen, indem HTML, CSS und JavaScript kombiniert werden. Es kann für die Erstellung von Single-Page-Anwendungen, Desktop-Anwendungen, mobile Anwendungen und mehr verwendet werden. Beispiele für den Einsatz von Vue.js sind die Erstellung von Webseiten, die Erstellung von Benutzeroberflächen für mobile Apps, die Erstellung von Dashboards und mehr.

Allerdings erfordern Frameworks dieser Art eine etwas aufwändigere Infrastruktur, bei der diverse Komponenten installiert und teils auch zur Bereitstellung der damit erzeugten Webanwendungen zur Laufzeit bereitgestellt werden müssen. Ebenso gibt es meist eine nicht ganz triviale Syntax und Denkweise, in die man sich einarbeiten muss. Solche Frameworks sind deshalb meist professionellen Entwicklern vorbehalten.

<

<noscript> 17
<script> 8

A

addEventListener() 155
Addition 30
Ajax 10, 176, 180
Ajax, jQuery 206
alert() 121
Alternativen 41
anchors 133
Anfügeoperator 33
Anonyme Funktionen 56, 61, 63
Anweisungen 17
Anweisungen, Reihenfolge 41, 44
Anweisungsblöcke 40, 43
appendChild() 124
applets 133
appName 141
arguments 57
Array, Eigenschaft 113
Array-Literal 111
Arrays 109
Arrays, assoziierte 111
Attributknoten 123, 126
audio, HTML5 193
Ausdrücke, reguläre 103, 104, 107
Auswahlliste 160

B

Baumstruktur DOM 122
Bedingungsoperator 36
Berechnungsoperator 29
Bestätigungsfenster 121
Bezeichner 19
Bezeichner, Regeln 19
Binär 29
Bit-Operator 29, 35
Blöcke 40
Blocklokal 66
Boolesche Werte 28, 35, 36, 95
Bootstrap 203
break 50
break, benanntes 52
Browser 6
Browsereigenschaft 140
Browserweichen 140
Bubble-Events 152
Bubbling 152

C

Callback 88
Callback-Funktion 88, 178

Camelnotation 20
cancelBubble 152
Canvas 201
captureEvents() 153
case 44, 174
case-sensitive 19, 22
case-sensitive, Fehler 74
Casting 24
Checkbox 160
child node 124
Children 123
class 77
clearTimeout() 119
Closures 62
concat() 114
confirm() 121, 169
console.log() 23
const 24
constructor() 77
continue 50
continue, benanntes 52
Cookie 197
Cookie löschen 197
Cookie, Verfallszeit 197
Cosinus 97
createElement() 126
createTextNode() 126
Cross-Domain-Zugriff 177

D

Datenkapselung 63, 88
Datentypen 24, 26, 29
Datentypkonvertierung 36, 37
Date-Objekt 100, 101
Debuggen 71
decodeURI 69
defer 14
Deklaration 19
Dekrement 30
delete 87
Design-Pattern 207
Dezimalzahlen 26
DHTML 188
Division 30
Document Object Model (DOM) 117
document.getElementById() 123
document.getElements-
 ByName() 123
document.getElements-
 ByTagName() 123
document.write() 125
Document-Object-Modell 143
document-Objekt 81, 119,
 124, 125, 126
Dokumentereignisse 145
Dokumentknoten 123
DOM 117, 143, 188

DOM-Baum 122
Don't repeat yourself 53
DOT-Notation 81
do-while-Schleifen 49
Drag & Drop 148
Dynamic HTML 188

E

ECMA 6
ECMAScript 6
Eigenschaften, Referenzierung 83
Einbettung in HTML 8
Eingabefeld überprüfen 163
Eingabefelder 159
Elementknoten 123, 126
else, Anweisung 43, 44
Elternelement 123
encodeURIComponent 69
encodeURIComponent() 69, 186
Endlosschleife 51
Entwicklertools 72
Ereignisabfrage 151
Ereignisbehandlung 144
Ereignisobjekt 151
Ereignisse 143
escape 69
Escape-Sequenzen 27
Eulersche Zahl 97
eval 69
Event Listener 155
Event-Bubbling 152
EventHandler 144
EventHandler, HTML 16
event-Objekt 119
Events 143
extends 91

F

Fallauswahl, mehrseitige 45
Fall-Through 44
false 25, 28
Fehlerkonsole 72
Felder 82
Fenster öffnen 120
Fenster schließen 120
First-Class Citizens 63
floor() 98
for-in-Anweisung 93
forms 135
Formulareingaben testen 164
Formularereignisse 150
for-Schleife 46
Frames 138
frames-Objekt 119
Frameworks 202
Funktion 55

| | | | | | |
|--------------------------------|------------|-------------------------------|---------|---------------------------------|------------|
| Funktionen referenzieren | 56 | innerHTML | 130 | length, Array | 111 |
| Funktionen, anonyme | 56, 61, 63 | innerText | 130 | let | 21, 66 |
| Funktionen, innere | 62 | instanceof-Operator | 93 | links | 134 |
| Funktionen, mathematische | 96 | Instanzen | 79, 82 | Literale | 21 |
| Funktionen, Parameterübergabe | 57 | Instanzen, reguläre Ausdrücke | 104 | LiveScript | 6 |
| Funktionen, vordefinierte | 69 | Instanzvariablen | 80 | Local Storage, HTML5 | 198 |
| Funktionsaufruf | 56, 74 | Interaktion | 143 | location-Objekt | 119, 136 |
| Funktionsdeklaration | 60 | Interpreter | 36, 72 | Lose Typisierung | 21, 24, 25 |
| Funktionsreferenz | 56, 88 | isFinite() | 70 | | |
| Funktionszeiger | 56 | isNaN() | 69, 99 | M | |
| FXML | 7 | Iterationen | 46 | Matht | 96 |
| G | | J | | Mausereignisse | 148 |
| Ganzzahlen | 26 | Java | 6 | Maustaste | 153 |
| Generalisierung | 90 | Java-Applets | 133 | Member | 77 |
| Geolocation | 201 | JavaFX | 7 | Metazeichen | 104, 106 |
| Geschwister | 123 | JavaScript als externe Datei | | Methoden | 28, 88 |
| getElementById | 126 | einbinden | 15 | Methoden, mathematische | 97 |
| getElementById() | 123 | JavaScript einbinden | 14 | MIME | 8 |
| getElementsByName | 126, 127 | JavaScript Object Notation | | MIN_VALUE | 99 |
| getElementsByName() | 123 | (JSON) | 79, 112 | Model View Controller | 207 |
| getElementsByTagName | 127 | JavaScript testen | 11 | Model View ViewModel | 207 |
| getElementsByTagName() | 123 | JavaScript, clientseitiges | 8 | Modulo-Operator | 30 |
| Getter | 81 | JavaScript, Entstehung | 6 | Mozilla Foundation | 6 |
| Gleitkommazahlen | 26 | JavaScript, Objekte | 95 | MP3-Format | 194 |
| Grundrechenarten | 30 | JavaScript, serverseitiges | 8 | MPG-4-Format | 197 |
| Gültigkeitsbereich | 21, 65 | JavaScript, Sonderzeichen | 27 | Multimedia | 193 |
| H | | JavaScript, Versionen | 10 | Multiplikation | 30 |
| Hexadezimalzahlen | 26 | JavaScript, Versionen angeben | 11 | Multipurpose Internet Mail | |
| hidden | 159 | JavaScript-Debugger | 73 | Extensions, MIME | 8 |
| History | 135 | join() | 114 | muted, Audio und Video | 194 |
| history-Objekt | 119 | jQuery | 203 | MVC | 207 |
| Höckernotation | 20 | jQuery UI | 206 | MVVC | 207 |
| Hover-Effekt | 189 | JScript | 10 | N | |
| HTML-Eventhandler | 16, 144 | JSON | 95 | Nachfolger | 123 |
| HTTP-Anfrage | 177 | K | | Name, Variable | 19 |
| Hyperlinks | 134 | Kindelement | 123 | Namenskonventionen | 19 |
| I | | KISS-Prinzip | 53 | Namensraum | 15 |
| Identität | 33 | Klasse | 76 | namespace | 15 |
| if-Anweisung | 41 | Knotentypen | 123 | NaN | 27, 70 |
| if-Anweisung, verschachtelte | 42 | Kommentar, JavaScript | 18 | Netscape | 6 |
| if-else-Anweisung | 43 | Kommentarknoten | 123 | Netscape Navigator | 6 |
| if-else-Anweisung, mehrstufige | 43 | Konkatenationsoperator | 29, 33 | new | 77 |
| IFrames | 119, 138 | Konsistenzüberprüfung | 103 | Node.js | 9 |
| Image-Objekt | 109 | Konstanten | 24 | node-Objekt | 123 |
| images | 135 | Konstruktor | 76 | nodeValue | 124 |
| indexOf() | 168 | Konstruktor-Funktion | 82 | Notationsregeln | 17 |
| Initialisieren | 21 | Kontrollkästchen | 160 | Notationsregeln, | |
| Initialisierung | 23 | Konventionen für Bezeichner | 19 | Anweisungsblock | 40 |
| Inkrement | 30 | L | | Notationsregeln, break/continue | 50 |
| Inline Frames | 138 | Lambda-Ausdrücke | 63 | Notationsregeln, | |
| Inline-Referenz | 16 | lang | 14 | do-while-Schleifen | 49 |
| Innere Funktionen | 62 | Lazy Evaluation | 34 | Notationsregeln, Eigenschaften | 82 |
| | | | | Notationsregeln, for-Schleifen | 46 |
| | | | | Notationsregeln, Funktionen | 56 |
| | | | | Notationsregeln, Objektinstanz | 77 |

| | | | | | |
|---|---------------|--------------------------------|-----------|--------------------------------------|------------|
| Notationsregeln, verschachtelte Bedingungen | 42 | password | 159 | shift() | 114 |
| Number | 70 | Pattern | 104 | Short-Circuit-Auswertung | 34 |
| Number-Typ | 99 | pause(), Audio und Video | 194 | Siblings | 123 |
| O | | Persistent | 112 | Sicherheit | 9 |
| Objekt | 93 | Pfeiloperator, Lambda-Ausdruck | 63 | Sicherheitsabfrage beim Zurücksetzen | 164 |
| Objekt document | 124, 125, 126 | Pflichtfeld | 163 | Sinus | 97 |
| Objekt event | 151 | PI | 97 | slice() | 114 |
| Objekt frames | 138 | play(), Audio und Video | 194 | Softwareentwurfsmuster | 207 |
| Objekt history | 135 | pop() | 114 | Sonderzeichen | 27 |
| Objekt location | 136 | Pop-ups | 120 | sort() | 115 |
| Objekt navigator | 140 | Postfixoperator | 30 | Spaghetticode | 52 |
| Objekt screen | 139 | Präfixoperator | 34 | Spezialisierung | 90 |
| Objekt window | 118 | Prototypen | 77 | splice() | 115 |
| Objekte | 28, 118 | Punktnotation | 81 | Sprunganweisungen | 50 |
| Objekte referenzieren | 88 | push() | 114 | Sprungmarke | 52 |
| Objekte, Attribute | 82 | R | | SRI | 204 |
| Objekte, Eigenschaften | 76, 82 | Radio Button | 160 | status | 178 |
| Objekte, Eigenschaften hinzufügen | 87 | random() | 98 | Steuerzeichen | 27 |
| Objekte, Eigenschaften löschen | 87 | Rangordnung Operatoren | 38 | stopPropagation() | 152 |
| Objekte, eingebaute | 95 | readyState | 178 | Stringifizieren | 112 |
| Objekte, Instanzen | 82 | RegExp | 104 | stringify(), JSON | 112 |
| Objekte, Konzept | 76 | RegExp() | 103, 108 | String-Objekt | 95 |
| Objekte, Methoden | 76 | reguläre Ausdrücke | 95 | Strings | 27, 70 |
| Objekte, Referenzierung | 88 | Reguläre Ausdrücke | 103, 168 | Stringverkettung | 27 |
| Objekteigenschaften | 93 | Reservierte Wörter | 18 | Stringverkettungsoperator | 33 |
| Objektfelder | 123, 132 | Reset, Sicherheitsabfrage | 164 | style DOM-Objekt | 189 |
| Objektinitialisierer | 79 | responseXML | 187 | Subklasse | 90 |
| Objekt-Literal | 111 | return | 56, 60 | Subtraktion | 30 |
| OGG-Format | 194, 197 | reverse() | 114 | Sun Microsystems | 6 |
| onclick | 16 | RIA | 6, 202 | super() | 90 |
| onkeypress | 154 | Rich Internet Application, RIA | 6 | Superklasse | 90 |
| onload | 145 | Rollover-Effekt | 149 | switch | 44, 174 |
| onunload | 145 | Root | 123 | Systemuhrzeit | 100 |
| open() | 120, 177 | round() | 97 | T | |
| Operator, arithmetischer | 29 | Rückgabewert | 55, 59 | Tangens | 97 |
| Operator, binärer | 29, 35 | Runden | 97 | Tastatur | 153 |
| Operator, logischer | 29 | S | | Tastaturereignisse | 150 |
| Operator, ternärer | 29, 36 | Same Origin Policy | 177 | Tastaturpuffer | 150 |
| Operator, unärer | 29 | Sandbox | 8, 9, 177 | text | 159 |
| Operatoren | 29 | Schaltflächen | 159 | textarea | 159 |
| Operatoren, Datentypen | 35, 36 | Schleifen | 40, 46 | Textknoten | 123, 126 |
| Operatoren, Rangordnung | 38 | Schleifenkörper | 46 | this | 80, 88 |
| options | 161 | Schleifensteuerung | 46 | Timer | 120 |
| Optionsfeld | 160 | Schlüsselwörter | 18 | true | 25, 28 |
| P | | Scope | 21, 65 | type | 14 |
| Parameterübergabe | 57 | Screen-Objekt | 139 | typeof | 36, 37 |
| Parent | 123 | select() | 159 | TypeScript | 10 |
| parent node | 124 | Selektion | 45 | Typisierung, lose | 21, 24, 25 |
| parse() | 184 | Semikolon | 17 | Typumwandeln | 24 |
| parse(), JSON | 112 | Serialisieren | 112 | U | |
| parseFloat | 70 | Session Storage, HTML5 | 198 | Unär | 29 |
| parseInt | 70 | setAttributeNode() | 124 | undefined | 21 |
| | | Setter | 81 | Unendlich | 99 |
| | | setTimeout() | 120 | | |

| | | | | | |
|-------------------------------|------------|--------------------------------|-----|----------------------------------|----------------|
| unescape | 69 | Vordefinierte Funktionen | 69 | write() document | 125 |
| Unterprogramm | 55 | Vorfahren | 123 | Wurzelknoten | 123 |
| use strict | 21 | Vorgabeparameter, Funktionen | 61 | | |
| | | Vorgabewerte, Parameter | 61 | X | |
| V | | Vue.js | 207 | | |
| | | W | | | |
| var | 21, 65 | W3C | 117 | XHR | 176 |
| Variable, Wertzuweisung | 23 | Wahrheitswert | 28 | XML Ajax | 180 |
| Variablen | 21 | Web Worker | 201 | XMLHttpRequest | 176 |
| Variablen, lokale und globale | 65 | Web-Konsole | 72 | Z | |
| Variablen, Operatoren | 29, 35 | Webseite laden | 145 | | |
| Variablenname | 19 | Wertrückgabe | 55 | Zahlen | 26, 95 |
| Vererbung | 90 | Wertzuweisung | 23 | Zahlenvariablen | 27 |
| Vergleichsoperator | 29, 32, 74 | while | 48 | Zeichenketten | 27 |
| Verknüpfungsoperator | 29 | Widgets | 206 | Zeitangaben | 101 |
| Verschachtelte Funktionen | 62 | Wiederholung, fußgesteuerte | 49 | Zeitgeber | 120 |
| Verweis | 134 | Wiederholung, zählergesteuerte | 46 | Zufallsgenerator | 98 |
| Verweisanker | 133 | window-Objekt | 119 | Zufallszahl | 97 |
| Verzögerungszeit | 120 | with() | 158 | Zuweisung oder Vergleich, Fehler | 74 |
| Video | 197 | with-Anweisung | 92 | Zuweisungsoperator | 23, 29, 35, 74 |
| video, HTML5 | 193 | | | | |
| volume(), Audio und Video | 194 | | | | |

Impressum

Matchcode: JAVS_2023

Autor: Ralph Steyer

1. Ausgabe, Juni 2023

© HERDT-Verlag für Bildungsmedien GmbH, Bodenheim

Druck und Bindearbeiten: Esser printSolutions GmbH, D-75015 Bretten
Edubook AG, CH-5634 Merenschwand

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlags reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit großer Sorgfalt erstellt und geprüft. Trotzdem können Fehler nicht vollkommen ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Wenn nicht explizit an anderer Stelle des Werkes aufgeführt, liegen die Copyrights an allen Screenshots beim HERDT-Verlag. Sollte es trotz intensiver Recherche nicht gelungen sein, alle weiteren Rechteinhaber der verwendeten Quellen und Abbildungen zu finden, bitten wir um kurze Nachricht an die Redaktion.

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) verzichtet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

Die in diesem Buch und in den abgebildeten bzw. zum Download angebotenen Dateien genannten Personen und Organisationen, Adress- und Telekommunikationsangaben, Bankverbindungen etc. sind frei erfunden. Eventuelle Übereinstimmungen oder Ähnlichkeiten sind unbeabsichtigt und rein zufällig.

Die Bildungsmedien des HERDT-Verlags enthalten Verweise auf Webseiten Dritter. Diese Webseiten unterliegen der Haftung der jeweiligen Betreiber, wir haben keinerlei Einfluss auf die Gestaltung und die Inhalte dieser Webseiten. Bei der Bucherstellung haben wir die fremden Inhalte daraufhin überprüft, ob etwaige Rechtsverstöße bestehen. Zu diesem Zeitpunkt waren keine Rechtsverstöße ersichtlich. Wir werden bei Kenntnis von Rechtsverstößen jedoch umgehend die entsprechenden Internetadressen aus dem Buch entfernen.

Die in den Bildungsmedien des HERDT-Verlags vorhandenen Internetadressen, Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen waren zum Zeitpunkt der Erstellung der jeweiligen Produkte aktuell und gültig. Sollten Sie die Webseiten nicht mehr unter den angegebenen Adressen finden, sind diese eventuell inzwischen komplett aus dem Internet genommen worden oder unter einer neuen Adresse zu finden. Sollten im vorliegenden Produkt vorhandene Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen nicht mehr der beschriebenen Software entsprechen, hat der Hersteller der jeweiligen Software nach Drucklegung Änderungen vorgenommen oder vorhandene Funktionen geändert oder entfernt.