

JAVS18

Autor: Heiko Schröder

Inhaltliches Lektorat: Andrea Weikert

1. Ausgabe, 2. Aktualisierung, Januar 2010

© HERDT-Verlag für Bildungsmedien GmbH, Bodenheim

Internet: www.herdt.com

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Herausgebers 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.

Sollte es uns trotz intensiver Recherchen nicht gelungen sein, alle Rechteinhaber der verwendeten Quellen und Abbildungen zu finden, bitten wir um kurze Nachricht an die Redaktion.

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. Übereinstimmungen oder Ähnlichkeiten mit lebenden oder toten Personen sowie tatsächlich existierenden Organisationen oder Informationen sind unbeabsichtigt und rein zufällig.

Die Bildungsmedien des HERDT-Verlags enthalten Links bzw. Verweise auf Internetseiten anderer Anbieter. Auf Inhalt und Gestaltung dieser Angebote hat der HERDT-Verlag keinerlei Einfluss. Hierfür sind alleine die jeweiligen Anbieter verantwortlich.

JavaScript 1.8

Grundlagen

JAVS18

1 Arbeiten mit diesem Buch	4	6 Objekte	58
1.1 Voraussetzungen und Ziele.....	4	6.1 Grundlagen von Objekten	58
1.2 Inhaltliche Konventionen.....	5	6.2 Eigenschaften	59
2 Einführung in JavaScript.....	6	6.3 Methoden	64
2.1 Entstehung von JavaScript	6	6.4 Anweisungen und Operatoren für Objekte	66
2.2 Grundlagen zu JavaScript.....	6	6.5 Übung	68
2.3 JavaScript-Versionen	8	7 Vordefinierte Objekte	70
2.4 JavaScript, JScript, VBScript und Java	9	7.1 Grundlagen zu vordefinierten Objekten.....	70
2.5 JavaScript im Browser aktivieren	10	7.2 Objekt <code>String</code> für Zeichenketten.....	70
2.6 JavaScript-Aktivierung im Browser testen	12	7.3 Objekt <code>Math</code> für mathematische Berechnungen	72
2.7 Nützliche Webseiten	12	7.4 Objekt <code>Number</code> für Zahlen	74
2.8 Übung	13	7.5 Objekt <code>Array</code> für Variablenlisten.....	75
3 Grundlegende Sprachelemente.....	14	7.6 Objekt <code>Date</code> für Zeitangaben.....	78
3.1 JavaScript in HTML verwenden	14	7.7 Objekt <code>RegExp</code> für reguläre Ausdrücke	82
3.2 Allgemeine Notationsregeln	19	7.8 Übung	86
3.3 Reservierte Wörter	19	8 Objektmodell	88
3.4 Bezeichner	20	8.1 Hierarchie der JavaScript-Objekte	88
3.5 Variablen.....	20	8.2 Objekt <code>navigator</code>	88
3.6 Konstanten	21	8.3 Objekt <code>window</code>	90
3.7 Datentypen	22	8.4 Meldungsfenster	95
3.8 Operatoren	24	8.5 Eingabefenster	95
3.9 Rangfolge der Operatoren.....	31	8.6 Bestätigungsfenster	96
3.10 Übung	32	8.7 Objekt <code>screen</code>	98
4 Kontrollstrukturen.....	34	8.8 Objekt <code>document</code>	100
4.1 Steuerung des Programmablaufs.....	34	8.9 Objekt <code>history</code>	102
4.2 Anweisungsblock.....	34	8.10 Objekt <code>location</code>	103
4.3 Auswahl	35	8.11 Objekt <code>frames</code>	105
4.4 Wiederholung.....	39	8.12 Objekt <code>event</code>	108
4.5 Übung	45	8.13 Übung	112
5 Funktionen.....	46	9 Zugriff auf HTML-Dokumente	114
5.1 Grundlagen zu Funktionen	46	9.1 Grundlagen zum <code>document</code> -Objekt	114
5.2 Funktionen mit Parametern.....	47	9.2 Unterobjekt <code>anchors</code>	115
5.3 Variable Parameterliste	48	9.3 Unterobjekt <code>applets</code>	115
5.4 Weitere Möglichkeiten für die Definition von Funktionen.....	50	9.4 Unterobjekt <code>forms</code>	115
5.5 Lokale und globale Variablen.....	51	9.5 Eingabefelder	117
5.6 Vordefinierte Funktionen in JavaScript.....	53	9.6 Kontroll- und Optionsfelder	118
5.7 Debuggen von Funktionen	54	9.7 Auswahllisten	118
5.8 Übung	57	9.8 Schaltflächen	120
		9.9 Eingaben prüfen.....	121
		9.10 Verschiedene Formulare	123
		9.11 Unterobjekt <code>images</code>	125
		9.12 Unterobjekt <code>links</code>	129
		9.13 Übung	129

10 Event-Handler.....	132	12 Komplexe Techniken.....	160
10.1 Grundlagen zu Events.....	132	12.1 Multimedia.....	160
10.2 Auf Ereignisse reagieren.....	132	12.2 JavaScript und Shockwave/Flash	160
10.3 Eine Webseite laden und verlassen.....	134	12.3 DOM - Document Object Model	165
10.4 Ereignisse bei Grafiken, Links und Ankern	135		
10.5 Ereignisse im Formular.....	137	13 Beispielanwendungen	168
10.6 Formulareingaben testen	140	13.1 Würfel mit grafischer Ausgabe	168
10.7 javascript:.....	145	13.2 Zwei Frames gleichzeitig ändern	169
10.8 Übung	145	13.3 Erzeugen eines Lauflichts.....	171
		13.4 Nachrichtenticker.....	172
		13.5 Gedächtnisspiel	174
11 Cookies	148		
11.1 Grundlagen zu Cookies.....	148	14 Anhang	182
11.2 Cookies akzeptieren.....	149	14.1 Schnellreferenz	182
11.3 Cookies setzen.....	150	14.2 Übersichten der Objektmodelle	183
11.4 Inhalt eines Cookies	152	14.3 Informationen im Internet	185
11.5 Cookies auslesen	154	14.4 Glossar	186
11.6 Cookies löschen.....	155		
11.7 Verfallsdatum angeben	156	Stichwortverzeichnis	188
11.8 Übung	158		

3 Grundlegende Sprachelemente

In diesem Kapitel erfahren Sie

- ▶ wie Sie JavaScript in ein HTML-Dokument einbinden
- ▶ welches die wichtigsten JavaScript-Sprachelemente sind
- ▶ was Bezeichner, Variablen und Konstanten sind
- ▶ welche Datentypen unterstützt werden
- ▶ welche Operatoren Sie nutzen können

Voraussetzungen

- ✓ HTML-Kenntnisse

3.1 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 Kopf als auch im 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. Skripts im `<body>`-Tag werden z. B. dazu genutzt, eine Ausgabe in das HTML-Dokument durchzuführen oder um auf ein Ereignis zu reagieren (z. B. auf das Klicken der Schaltfläche zum Absenden eines Formulars).

Skriptsprache angeben

Innerhalb des `<script>`-Tags wird über das Attribut `type` der MIME-Typ der Skriptsprache angegeben. Die Verwendung des Attributs `language` ist laut Standard veraltet und sollte daher nicht mehr verwendet werden. Allerdings haben Sie nur darüber die Möglichkeit, die vom Browser unterstützte JavaScript-Version bis zur Version 1.5 abzufragen.

Die Abfrage der höheren Version erhalten Sie über die nachfolgende Anweisung:

```
<script type="text/javascript; version=1.8">
  <!-- Hier werden die JavaScript-Anweisungen eingefügt -->
</script>
```



Diese Abfrage wird derzeit nur vom Browser Firefox 3.0 interpretiert, da der Browser Internet Explorer 7.0 derzeit nur JavaScript bis zur Version 1.3 und Opera 9.60 JavaScript bis zur Version 1.5 unterstützt.

Beispiel: kap03\javascript-test.html

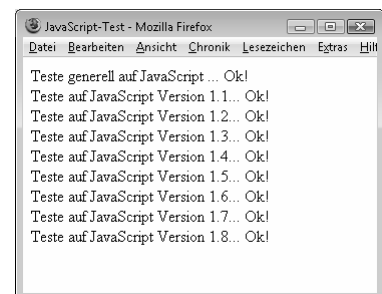
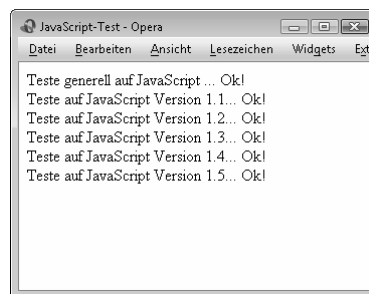
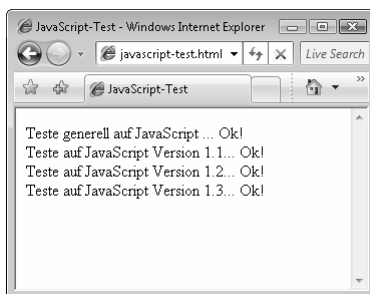
Im folgenden Beispiel wird die Funktion `document.write()` verwendet, um im HTML-Dokument eine Ausgabe zu erzeugen. Diese gibt an, welche JavaScript-Version durch Ihren Browser unterstützt wird.

```

<html>
<head>
  <title>JavaScript-Test</title>
</head>
<body>
  ① <script type="text/javascript">
  ②   document.write("Teste generell auf JavaScript ... Ok!<br>");
  ③ </script>
  ④ <script language="javascript1.1">
      document.write("Teste auf JavaScript Version 1.1... Ok!<br>");
  </script>
  <!-- Es folgen weitere Tests auf JavaScript 1.2, 1.3, 1.4 und 1.5 ... -->
  ⑤ <script type="text/javascript; version=1.6">
      document.write("Teste auf JavaScript Version 1.6... Ok!<br>");
  </script>
  <script type="text/javascript; version=1.7">
      document.write("Teste auf JavaScript Version 1.7... Ok!<br>");
  </script>
  <script type="text/javascript; version=1.8">
      document.write("Teste auf JavaScript Version 1.8... Ok!<br>");
  </script>
</body>
</html>

```

- ① Nach dem Tag `<body>` geben Sie das `<script>`-Tag zum Einfügen von JavaScript-Anweisungen an.
- ② Mit `document.write()` wird eine Ausgabe im HTML-Dokument erzeugt. Versteht der Browser JavaScript, wird er den angegebenen Text anzeigen.
- ③ Das Skript wird beendet.
- ④ Es folgt wieder ein `<script>`-Tag zum Ausführen von JavaScript. Dies ist noch die veraltete Abfrage für die Browser, die JavaScript bis Version 1.5 implementiert haben. Mit der Angabe der Versionsnummer können Sie testen, ob der Browser eine spezielle Version versteht. Dies ist beispielsweise nützlich, wenn Sie JavaScript-Funktionen nutzen, die erst ab einer bestimmten Version implementiert sind. Kann der Browser JavaScript 1.1 interpretieren, gibt er dies entsprechend über den nachfolgenden Befehl `document.write()` aus.
- ⑤ Dies ist die Abfrage, ob der Browser JavaScript 1.6 verarbeiten kann. Bei der Angabe des Typs geben Sie über `version` den Wert an, der nachfolgend berücksichtigt werden soll. Browser, die diese Abfrage nicht auswerten können, ignorieren die JavaScript-Ausführungen innerhalb des entsprechenden `<script>`-Tags.



JavaScript-Fähigkeiten der Browser Internet Explorer 7.0, Opera 9.60 und Mozilla Firefox 3.0

Wie Sie sehen, kann der Internet Explorer 7.0 nur JavaScript bis Version 1.3 interpretieren, und der Browser Opera 9.60 bis Version 1.5. Vorreiter ist der Mozilla Firefox 3.0, der JavaScript bis Version 1.8 umsetzen kann.



- ✓ Durch die Angabe des 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 geparkt werden muss, z. B. `<script type="..." defer>`.
- ✓ Wenn Sie über JavaScript Ausgaben im HTML-Dokument vornehmen, wird dieser Code erneut interpretiert. 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.

Standardskriptsprache festlegen

Innerhalb eines HTML-Dokuments können Sie verschiedene Skriptsprachen verwenden. Ob eine spezielle Skriptsprache unterstützt wird, ist browserabhängig. Über das `<script>`-Tag können Sie z. B. die folgenden Sprachen einbinden:

JavaScript	<code><script type="text/javascript"></code>
JScript	<code><script type="text/jscript"></code>
VBScript	<code><script type="text/vbscript"></code>

Zu Beginn eines HTML-Dokuments können Sie über das Tag `<meta>` die Standardskriptsprache festlegen. Geben Sie dann im `<script>`-Tag kein Attribut `type` an, wird die von `<meta>` angegebene Skriptsprache verwendet. Der MIME-Type der Sprache wird im Attribut `content` angegeben.

```
<meta http-equiv="Content-Script-Type" content="text/javascript">
```

Einbinden durch externe Dateien

JavaScript-Code können Sie auch über separate Dateien einbinden. Dies ist z. B. nützlich, wenn Sie Funktionen in mehreren HTML-Dokumenten verwenden möchten. Durch die Definition 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>
```

Durch die Verwendung des `<script>`-Tags mit dem Attribut `type` leiten Sie die Nutzung von JavaScript ein. Über das Attribut `src` 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`.



Eine JavaScript-Datei kann im Kopf oder Rumpf eines HTML-Dokuments eingebunden werden. Vorzugsweise sollten Sie diese jedoch immer im HTML-Kopf einbinden. Enthält die Datei Funktionen, die bereits beim Laden Ausgaben in das HTML-Dokument durchführen sollen, können Sie diese Funktionen innerhalb des `<body>`-Tags aufrufen.

Beispiel: *kap03\javascript.js*

In der Datei befindet sich lediglich eine Ausgabeanweisung, die einen Text in ein HTML-Dokument schreibt.

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



Beim Erstellen von JavaScript-Code in externen Dateien werden die Anweisungen `<script type="text/javascript">` weggelassen. Sie geben sofort die JavaScript-Anweisungen an.

Beispiel: *kap03\javascript-extern.html*

Binden Sie die JavaScript-Datei *javascript.js* im Kopf des HTML-Dokuments ein. Dadurch wird die Anweisung in der Datei ausgeführt und die betreffende Zeichenkette ausgegeben.

```
...
<head>
  <title>JavaScript-Test</title>
  <script type="text/javascript" src="javascript.js"></script>
</head>
...
```

Verwendung in Funktionen und HTML-Tags

Zur Behandlung von Ereignissen und bei der Verwendung in HTML-Tags können Sie JavaScript-Code auch ohne Angabe des `<script>`-Tags angeben.

Beispiel zur Behandlung von Ereignissen

Das Ereignis `onclick` wird 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 type="button" onclick="document.write('Sie haben mich gedr&uuml;ckt');">
  Klicken Sie auf mich
</button>
```

Beispiel zum Aufruf einer JavaScript-Funktion

Hier wird ein Hyperlink 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()">Zur&uuml;ck</a>
```

JavaScript für nicht scriptfähige Browser

Ist die Verwendung von JavaScript im Browser abgeschaltet oder versteht der Browser JavaScript generell nicht, können Sie eine entsprechende Information anzeigen. Diese wird zwischen die Tags `<noscript>...</noscript>` eingefügt und besteht aus üblichen HTML-Anweisungen. In den meisten Fällen wird dieser Bereich genutzt, um auf die fehlende JavaScript-Funktionalität des Browsers hinzuweisen. Die Tags können sich an einer beliebigen Stelle innerhalb des `<body>`-Tags befinden.

Unabhängig von der Ausgabe für nicht scriptfähige Browser im Tag `<noscript>` versuchen diese, den Inhalt des `<script>`-Tags zu interpretieren und anzuzeigen. Dies resultiert darin, dass der JavaScript-Code als Text im HTML-Dokument erscheint. Die Angabe der zusätzlichen Kommentarzeichen `<!--` und `-->` im `<script>`-Tag stellt sicher, dass die nachfolgenden Anweisungen von veralteten Browsern nicht als HTML-Code interpretiert werden. Dabei gelten folgende Konventionen:

- ✓ Die HTML-Kommentarzeichen schließen ein Skript vollständig ein.
- ✓ Das öffnende HTML-Kommentarzeichen `<!--` und der bis zum Zeilenende folgende Text werden vom JavaScript-Interpreter zu Beginn eines JavaScripts ignoriert.
- ✓ Der HTML-Kommentar wird am Ende des Skripts durch die Zeichenfolge `-->` geschlossen. Da ältere JavaScript-Interpreter diesen Kommentar jedoch als JavaScript-Anweisung interpretieren, muss vor dem HTML-Kommentar zusätzlich ein JavaScript-Kommentar, der mit der Zeichenfolge `//` eingeleitet wird, eingefügt werden. Dadurch entsteht die Zeichenfolge `//-->`, die sich auf einer eigenen Zeile befinden sollte.

Da mittlerweile kein Browser mehr verwendet wird, der JavaScript-Code innerhalb eines `<script>`-Tags als HTML-Code interpretieren würde, verzichtet man auf die Angabe der zusätzlichen Kommentare. In diesem Buch wird deshalb die Kommentar-Auszeichnung weggelassen.



Beispiel

Der Inhalt des Tags `<noscript>` wird angezeigt, wenn die Browser kein JavaScript verstehen bzw. die Unterstützung ausgeschaltet ist.

```
<script type="text/javascript">
  JavaScript-Anweisungen ...
</script>
<noscript>HTML-Code f&uuml;r Browser ohne JavaScript-Unterst&uuml;tzung</noscript>
```

Ausführungsreihenfolge

JavaScript-Anweisungen werden in der Reihenfolge ausgeführt, in der sie in einem HTML-Dokument verwendet werden. Dadurch werden JavaScript-Anweisungen, die im Kopfbereich eines HTML-Dokuments eingebunden werden, vor JavaScript-Anweisungen im Rumpfbereich 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.

Führen Sie JavaScript-Code aus, der z. B. beim Klick auf einen Link Ausgaben im Dokument durchführt, wird der Inhalt des alten Dokuments überschrieben.

Beispiel: *kap03\ausfuehrungsreihenfolge.html*

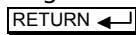

Beim Laden des HTML-Dokuments werden die JavaScript-Ausgaben aus dem Kopf- und Rumpfbereich sowie die Schaltfläche eines Formulars angezeigt. Aktivieren Sie den Hyperlink, wird der Inhalt des aktuellen HTML-Dokuments überschrieben.

```
<html>
<head>
  <script type="text/javascript">
①    document.write("Ausgabe aus dem Kopfbereich!<br>");
②    function zeigeInfo()
    {
③      document.write("Sie haben die Funktion zeigeInfo aufgerufen.");
    }
  </script>
</head>
<body>
  <script type="text/javascript">
④    document.write("Ausgabe aus dem Rumpfbereich!<br>");
  </script>
⑤  <a href="javascript:zeigeInfo();">Information zeigen</a>
</body>
</html>
```

- ① Da sich diese Ausgabe nicht innerhalb einer Funktion befindet, wird sie sofort beim Laden des HTML-Dokuments ausgeführt.
- ② Die Funktion `zeigeInfo()` wird definiert, aber noch nicht ausgeführt. Dies ist erst durch einen entsprechenden Aufruf wie in ⑤ möglich.
- ③ Da sich diese Ausgabeanweisung in einer Funktion befindet, wird sie nicht sofort beim Laden des Dokuments ausgeführt.
- ④ Diese Ausgabe wird beim Laden des HTML-Dokuments eingefügt.
- ⑤ Beim Aktivieren des Hyperlinks wird die Funktion `zeigeInfo()` ausgeführt. Da die Textausgabe in das HTML-Dokument nach dem Ladevorgang erfolgt, wird der bisherige Inhalt überschrieben.

3.2 Allgemeine Notationsregeln

Anweisungen

Über Anweisungen beschreiben Sie die Funktionalität eines Programms. Die Anweisungen werden vom Browser interpretiert und ausgeführt. Eine Anweisung wird durch einen Zeilenumbruch  oder die Angabe eines Semikolons  beendet.

Zur besseren Lesbarkeit sollten Sie eine Anweisung immer mit einem Semikolon beenden. Wenn sich mehrere Anweisungen in einer Zeile befinden, muss generell ein Semikolon gesetzt werden.



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. Dabei sollten Sie einerseits sparsam mit Kommentaren umgehen, da zu viele Kommentare nicht unbedingt zum besseren Verständnis (und der Lesbarkeit) beitragen. Andererseits sollten kompliziertere Code-Stellen durch Kommentare erläutert werden, insbesondere wenn noch andere Personen den JavaScript-Quelltext bearbeiten sollen.

//	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>");
```

3.3 Reservierte Wörter

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

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

3.4 Bezeichner

Bezeichner benennen unter anderem Konstanten, Variablen und Funktionen in JavaScript-Programmen. Bei der Programmierung können Sie über einen Bezeichner auf diese Elemente zugreifen. Ein Bezeichner wird vom Programmierer festgelegt. Dabei gelten 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.
- ✓ Namen 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
```

3.5 Variablen

Variablen können während der Programmausführung unterschiedliche, veränderbare Werte annehmen, wie z. B. Zwischen- oder Endergebnisse aus Berechnungen. 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:

- ✓ Der Name einer Variablen unterliegt den Vorgaben an Bezeichner. Ist dies nicht der Fall, bricht der JavaScript-Interpreter die weitere Ausführung des Codes ab.
- ✓ Variablen können an einer beliebigen Stelle in einem JavaScript-Programm durch die Angabe des Variablennamens und der Zuweisung eines Wertes definiert werden. Obwohl es nicht zwingend notwendig ist, Variablen mit dem einleitenden Schlüsselwort `var` (variable) zu definieren, dient es der besseren Lesbarkeit.
- ✓ Variablen, die Sie über die Angabe `var` definieren und denen Sie keinen Wert zuweisen, haben den Zustand undefiniert (`undefined`).
- ✓ Variablen, die weder über `var` noch durch eine Wertzuweisung definiert wurden, erzeugen beim Lesen einen Laufzeitfehler. Die Interpretation des JavaScript-Codes wird abgebrochen.
- ✓ Haben Sie eine Variable nicht explizit definiert, erfolgt dies automatisch bei ihrer ersten Verwendung.
- ✓ Mehrere Variablen werden durch Kommata getrennt definiert. Dabei können Sie auch unterschiedliche Datentypen verwenden.
- ✓ Mehreren Variablen kann auch gemeinsam ein Wert zugewiesen werden.
- ✓ Einer Variablen können im Programmverlauf Werte verschiedener Datentypen zugewiesen werden.

Beispiel für die Definition von Variablen

Im Folgenden werden die unterschiedlichen Möglichkeiten der Definition einer Variablen gezeigt. Ein Programm wird verständlicher, wenn Variablen schon vor ihrer Verwendung definiert und initialisiert werden. Initialisieren bedeutet, dass eine Variable mit einem Anfangswert belegt wird.

```
k = 10;           // die Variable k ist vom Typ Zahl und besitzt den Wert 10
var k;           // die Variable k ist noch undefiniert
var i, k = 10;    // die Variablen i und k sind vom Typ Zahl und besitzen
                  // den Wert 10
var k = 10, i = "Text"; // mehrere Definitionen durch Kommata trennen
```

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.



Beispiel: `kap03\variablen.html`

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

```
<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>
```

Wertzuweisungen

Variablen müssen vor ihrer ersten Verwendung mit einem Wert belegt werden. Diese Wertzuweisung erfolgt über das Gleichheitszeichen `=`. Diese Werte gelten lediglich zu Anfang eines Programms und können im Programmverlauf jederzeit geändert werden. Sie erreichen dadurch, dass alle Variablen vor der ersten Verwendung gültige Werte besitzen.

Beispiel

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

```
var summand1 = 1;
var summand2 = 8;
var summe = 0;
document.write(summe); // Diese Anweisung löst jetzt keinen Fehler aus!
summe = summand1 + summand2;
```

3.6 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. Jede Konstante, die Sie in Ihrem Programm verwenden, muss vorher innerhalb einer Anweisung deklariert und initialisiert werden.

- ✓ Konstanten werden über das Schlüsselwort `const` deklariert. Sie werden wie schreibgeschützte Variablen eingesetzt.
- ✓ Bei der Konstantendefinition muss sofort ein Wert zugewiesen werden. Dieser Wert kann später nicht mehr geändert werden.
- ✓ Konstanten vereinfachen die Lesbarkeit und Wartung eines Programms.

Beispiel

Zum Beispiel ist der Name `MWST` in Berechnungen besser lesbar als der nichtssagende Wert `0.19`.

```
const MWST = 0.19;
```



Konstanten sind in JavaScript 1.5 definiert. Der Internet Explorer 7.0 kann aufgrund seiner eingeschränkten JavaScript-Funktionalität nicht mit Konstanten umgehen und ignoriert diese Angabe.

3.7 Datentypen

Ganze Zahlen

Ganze Zahlen besitzen keine Nachkommastellen und können negative sowie positive Werte und den Wert null annehmen. In JavaScript können sie 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.

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



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.

Gleitkommazahlen

Gleitkommazahlen können Nachkommastellen besitzen. Das Kennzeichen ist entweder ein Dezimalpunkt, ein Exponent oder beides.

In JavaScript wird die amerikanische Notation, also ein Punkt `.`, verwendet, um die Dezimalstellen einer Zahl kenntlich zu machen.

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

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

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 Rundungsfehlern. Beachten Sie dies bei Berechnungen mit großen Zahlen.

```
k = 12345678901234567890;  
document.write(k); // liefert 12345678901234567000, da die Zahl zuerst in eine  
// Gleitkommazahl konvertiert und danach gerundet wird !
```

Zeichenketten

Eine Zeichenkette (String) ist eine Folge von Zeichen, die in einfache oder doppelte Anführungszeichen eingeschlossen ist. Es ist jedoch zu beachten, dass gleiche Arten von Anführungszeichen verwendet werden. Beginnen Sie eine Zeichenkette mit einem doppelten Anführungszeichen `"`, so müssen Sie diese auch mit einem weiteren 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.

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.

Der wichtigste 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. 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 Zahl = 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 auch als Escape-Sequenzen oder Steuerzeichen bezeichnet.

Die Wirkung der Steuerzeichen ist nur innerhalb eines ausgegebenen Textes zu erkennen, z. B. innerhalb einer Meldung über die Funktion `alert()`. Bei der Ausgabe von Text im Browser über die Funktion `document.write()` müssen Sie stattdessen 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.
<code>\t</code>	Tabulator
<code>\f</code>	form feed: Seitenvorschub
<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> .

Objekte

Objekte sind Datenelemente, die Eigenschaften und objektgebundene Funktionen (Methoden) besitzen können. In JavaScript werden Ihnen vordefinierte Objekte angeboten, um auf verschiedene Elemente einer Webseite zuzugreifen, z. B. auf ein Eingabefeld eines Formulars.

Jedes einzelne Objekt enthält Eigenschaften, die Sie auslesen können, z. B. die Adresse der geladenen Webseite oder den Namen des Fensters. Außerdem erlauben Ihnen die Objektmethoden das Ändern der Objekteigenschaften. So können Sie beispielsweise den Inhalt eines Eingabefeldes ändern.

3.8 Operatoren

Operatoren sind Zeichen, die verwendet werden, um Berechnungen durchzuführen oder Verknüpfungen und Vergleiche zwischen Variablen herzustellen.

Eine Operation arbeitet mit einem oder zwei Operanden. Entsprechend wird von unären oder binären Operatoren gesprochen. Ein Operator erzeugt immer einen Ergebniswert. Manche Operatoren können nur in Verbindung mit Variablen eines bestimmten Datentyps eingesetzt werden.

Operator	Datentyp
Arithmetischer Operator	Zahlen
Vergleichsoperator	Zahlen, Strings (Zeichenketten), boolesche Werte
Verknüpfungsoperator, Konkatenationsoperator	Strings (Zusammenfügen von Zeichenketten)
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

Name	Operator	Syntax	Beispiel	Wert
Modulo	%	Dividend % Divisor	7 % 4	3
Negation	-	-Operand	-(2 + 5)	-7
Inkrement	++	++Variable;	x = 10; ++x;	11
		Variable++	y = 135; y++	136
Dekrement	--	--Variable;	x = 10; --x;	9
		Variable--	y = 135; y--	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
```

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. Der Operand besitzt zwar nach der Operation denselben Wert, 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: *kap03\inkdek.html*

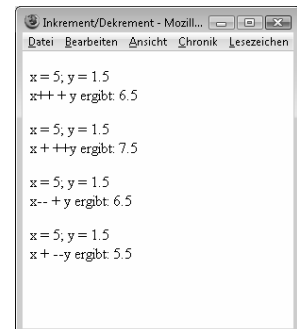
In dem Beispiel werden Variablen mit denselben Werten unterschiedlich inkrementiert und dekrementiert. Beachten Sie die daraus resultierenden unterschiedlichen Ergebnisse.

```

<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, dann wird `x` um den Wert 1 erhöht.
- ③ Die Variable `y` wird 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 Variablen addiert.



Verschiedene Inkrement- und Dekrementberechnungen

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
: ; < = > ? @
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[ \ ] ^ _ `
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

Name	Operator	Syntax	Beispiel	Wert
größer als	>	Operand1 > Operand2	"abd" > "abc" "X" > 10	true false
kleiner/gleich	<=	Operand1 <= Operand2	6 <= 7 "abc" <= "abc"	true true
größer/gleich	>=	Operand1 >= Operand2	"6" >= "6" "Buch" >= "buch"	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.

```
document.write(2 == "2"); // liefert nach der Typumwandlung von "2" nach 2 true
document.write(2 === "2"); // liefert false, da unterschiedliche Typen vorliegen
```

Verknüpfungsoperator

Dieser Operator, der auch Konkatenationsoperator (engl.: concatenate = verknüpfen) 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 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 `+` verwendet, wird er als Konkatenationsoperator 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.



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	true
			false && true	false
			true && false	false
			false && false	false
ODER		Operand1 Operand2	true true	true
			false true	true
			true false	true
			false false	false
NICHT	!	!Operand	!true	false
			!false	true

- ✓ Das logische UND ist ein binärer Operator. UND muss zwei Operanden verknüpfen 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 immer dann 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.

Short-Circuit-Auswertung

Ausdrücke werden immer 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 bezeichnet.

Beispiel

Bei der Verwendung des ODER-Operators wird die Auswertung abgebrochen, wenn ein Teilausdruck bereits 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 ein 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 nämlich 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.

In den folgenden Beispielen werden die Operatoren auf Dezimalzahlen angewendet.

Name	Operator	Syntax	Beispiel	Ergebnis
Bitweises NICHT	<code>~</code>	<code>~Operand</code>	<code>~12</code>	<div>0000...00001100 12</div> <div>1111...11110011 = -13</div>
Bitweises ODER	<code> </code>	<code>Operand1 Operand2</code>	<code>12 6</code>	<div>0000...00001100 12</div> <div>0000...00000110 6</div> <div>0000...00001110 = 14</div>
Bitweises UND	<code>&</code>	<code>Operand1 & Operand2</code>	<code>12 & 6</code>	<div>0000...00001100 12</div> <div>0000...00000110 6</div> <div>0000...00000100 = 4</div>
Bitweises XODER (Exklusiv-ODER)	<code>^</code>	<code>Operand1 ^ Operand2</code>	<code>12 ^ 6</code>	<div>0000...00001100 12</div> <div>0000...00000110 6</div> <div>0000...00001010 = 10</div>
Arithmetische Linksverschiebung	<code><<</code>	<code>Operand << Anzahl</code>	<code>12 << 2</code>	<div>0000...00001100 12</div> <div>0000...00110000 = 48</div>

Name	Operator	Syntax	Beispiel	Ergebnis
Arithmetische Rechtsverschiebung	>>	Operand >> Anzahl	13 >> 2	0000...00001101 13 0000...00000011 = 3
			-13 >> 2	1111...11110011 -13 1111...11111100 = -4
			13 >>> 2	0000...00001101 13 0000...00000011 = 3
			-13 >>> 2	1111...11110011 -13 0011...11111100 = 1073741820

Anmerkung zur Verschiebung

- ✓ Die arithmetische Linksverschiebung liefert den Zahlenwert von Operand um Anzahl Bit-Stellen nach links verschoben. Mathematisch gesehen handelt es sich hierbei um eine Multiplikation mit 2^{Anzahl} .
- ✓ Die arithmetische Rechtsverschiebung liefert den Zahlenwert von Operand um Anzahl Bit-Stellen nach rechts verschoben, wobei das Vorzeichen erhalten bleibt. Mathematisch gesehen handelt es sich hierbei um eine ganzzahlige Division mit 2^{Anzahl} .
- ✓ Die logische Rechtsverschiebung liefert den Zahlenwert von Operand um Anzahl Bit-Stellen nach rechts verschoben, wobei mit Nullen aufgefüllt wird. Diese Verschiebung wird auch Null-Fill-Rechtsverschiebung genannt.

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	a = 5;	a = 5
+=	Addieren und Zuweisen	a = 5; a += 5;	5 + 5 a = 10
+=	Anfügen und Zuweisen	a = 'Ei'; a += 'dotter';	'Ei' + 'dotter' a = 'Eidotter'
-=	Subtrahieren und Zuweisen	a = 5; a -= 2;	5 - 2 a = 3
*=	Multiplizieren und Zuweisen	a = 5; a *= 2	5 * 2 a = 10
/=	Dividieren und Zuweisen	a = 20; a /= 4	20 / 4 a = 5

Operator	Wirkung	Beispiel	Ergebnis
<code>%=</code>	Modulo-Operation und Zuweisen	<code>a = 5;</code> <code>a %= 3</code>	<code>5 % 3</code> <code>a = 2</code>
<code>&=</code>	Bitweises UND und Zuweisen	<code>a = 5;</code> <code>a &= 3</code>	<code>0101 & 0011</code> <code>a = 1</code>
<code>^=</code>	Bitweises XOR und Zuweisen	<code>a = 5;</code> <code>a ^= 3;</code>	<code>0101 ^ 0011</code> <code>a = 6</code>
<code> =</code>	Bitweises ODER und Zuweisen	<code>a = 5;</code> <code>a = 3;</code>	<code>0101 0011</code> <code>a = 7</code>
<code><<=</code>	Bitweises Linksverschieben und Zuweisen	<code>a = 5;</code> <code>a <<= 2</code>	<code>0101</code> <code>a = 20</code>
<code>>>=</code>	Bitweises Rechtsverschieben und Zuweisen	<code>a = 40;</code> <code>a >>= 2;</code>	<code>101000</code> <code>a = 10</code>
<code>>>>=</code>	Null-Fill-Rechtsverschiebung und Zuweisen	<code>a = 100;</code> <code>a >>>= 4;</code>	<code>1100100</code> <code>a = 6</code>

Bedingungsoperator

Mithilfe des Bedingungsoperators können einige `if-else`-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 oder JavaScript-Codes, der zurückgegeben wird, wenn der Ausdruck den Wert `true` liefert (`if`-Zweig)
- ✓ einen Rückgabewert eines beliebigen Datentyps oder JavaScript-Codes, 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 keine streng typisierte Sprache. Der Interpreter wandelt Operanden automatisch in die entsprechenden Datentypen um, damit eine Operation richtig ausgeführt wird.

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
<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>	<code>49</code>	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 Werte:

- ✓ `number` für Zahlen
- ✓ `string` für Zeichenketten

- ✓ `boolean` für Wahrheitswerte
- ✓ `undefined` für nicht initialisierte Variable

Beispiel	Rückgabewert als Typ
<code>var Variable = 5;</code> <code>Typ = typeof Variable;</code>	number
<code>var Variable = "Hallo";</code> <code>Typ = typeof Variable;</code>	string
<code>var Variable = true;</code> <code>Typ = typeof Variable;</code>	boolean
<code>Typ = typeof Variable1;</code>	undefined

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");</code> <code>Typ = typeof Variable;</code>	function
<code>var Variable = [1, 2, 3, 4];</code> <code>Typ = typeof Variable;</code>	object

3.9 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 Null-Fill-Rechtsverschiebung	Bit-Operatoren

Rangstufe (1=höchste Priorität)	Operatorzeichen	Operatorname	Operatortyp
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.

```
a = 8 + 2 * 7 + 3 // a = 8 + 14 + 3 = 25
a = (8 + 2) * (7 + 3) // a = 10 * 10 = 100
```

3.10 Übung

Verschiedene Datentypen

Übungsdatei: --

Ergebnisdatei: *kap03\uebung1.html*

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

Boolesche Werte

Übungsdatei: --

Ergebnisdatei: *kap03\uebung2.html*

- ① Welchen Wert haben die folgenden Ausdrücke?

- ✓ `3 < 8`
- ✓ `true && false`
- ✓ `6 + 4 * 2 - 1`
- ✓ `"Hund" != "Katze"`

Steuerzeichen in Zeichenketten

Übungsdatei: --

Ergebnisdatei: *kap03\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"