3 Syntax und Variablen

Am Anfang und im Zentrum der JavaScript-Programmierung stehen Daten und Werte, unabhängig davon, ob Sie nur einen einfachen Text ausgeben möchten oder komplexe Algorithmen schreiben. Es handelt sich immer darum, Werte zu behandeln und zu ändern.

Daten werden in JavaScript – wie auch in anderen Programmiersprachen – in Variablen gespeichert, wenn sie weiterverwendet und verändert werden sollen. Daten, die direkt in Anweisungen geschrieben werden, nennt man dagegen Literale. Sie sind nicht veränderbar und treten nur einmal auf.

Ein Literal ist beispielsweise der Text »Alles Gute!« in der folgenden Textausgabe:

```
document.write("Alles Gute!");
```

In Kapitel 3.2 »Variablen deklarieren« zeigen wir Ihnen, wie Sie Variablen erzeugen und mit ihnen arbeiten.

3.1 Datentypen

Daten treten nicht einfach nur als Literale oder Variablen auf, sie unterscheiden sich auch aufgrund ihrer Art. Folgende Zeilen verdeutlichen das:

```
document.write(4);
document.write("Text");
```

Die erste Zeile gibt 4 aus. 4 ist eine ganze Zahl (`Number`). Die zweite Zeile dagegen enthält Text. Der zugehörige Datentyp heißt `String` und wird in JavaScript immer in Anführungszeichen geschrieben.

Im Folgenden erhalten Sie eine Übersicht über alle Datentypen, die JavaScript unterscheidet.

In Kapitel 3.3 im Abschnitt »Typkonvertierung« wird erklärt, wie JavaScript die verschiedenen Datentypen intern behandelt und wie Sie den Datentyp eines Wertes ändern.

Integer

Integer sind ganze Zahlen ohne Nachkommastellen, die sowohl positive als auch negative Werte annehmen können, also beispielsweise:

```
-1
20
0
```

Die einzige Frage ist, wie klein oder groß die Zahlen werden dürfen. JavaScript stellt Integer von -2^{53} (-9 007 199 254 740 992) bis $+2^{53}$ (+9 007 199 254 740 992) exakt dar. Kleinere oder größere Zahlen können zwar verarbeitet werden, aber sie verlieren die exakten Endziffern. In folgender Zeile hängen wir einfach 67 an die kleinste darstellbare Zahl an.

```
document.write(-900719925474099267);
```

Der Interpreter rundet die drei Endziffern:

```
-900719925474099300
```

Bei einigen Anwendungen, vor allem den Bit-Operatoren, ist der mögliche Wertebereich kleiner. Er umfasst dann nur 32 Bit, geht also von -2^{31} (-2 147 483 648) bis $+2^{31}-1$ (2 147 483 647).

Die JavaScript-Funktion[1] `isFinite()` *testet, ob eine Zahl noch endlich, also kleiner bzw. größer als die größte bzw. kleinste* ***speicherbare*** *Zahl in JavaScript ist. Wenn dies der Fall ist, wird als Ergebnis der Wahrheitswert (Boolean)* `true` *zurückgeliefert. Beachten Sie, dass die größte bzw. kleinste speicherbare Zahl nichts mit der größten bzw. kleinsten exakt darstellbaren Zahl zu tun hat. Sie kann wesentlich größer bzw. kleiner sein, weist dann jedoch Ungenauigkeiten bei den letzten Stellen hinter dem Komma auf. Mehr zu* `inFinite()` *finden Sie in Kapitel 5.3 »Globale Funktionen«.*

Hexadezimale Schreibweise

Integer können auch anders geschrieben werden. Eine recht bekannte Möglichkeit ist die hexadezimale Schreibweise. Das hexadezimale System hat nicht die Basis 10 – wie unsere Zahlenschreibweise –, sondern die Basis 16.

1 Eine Funktion ist ein Block aus einer oder mehreren Anweisungen, die mit ihrem Namen aufgerufen werden können. `isFinite()` gehört zu den Funktionen, die schon in JavaScript vorhanden sind. Mehr zu Funktionen erfahren Sie in Kapitel 5 »Funktionen«.

Das heißt, es gibt 16 Ziffern und da unsere Ziffern von 0 bis 9 nicht ausreichen, werden die Zahlen 10 bis 15 von den Buchstaben A bis F dargestellt. In JavaScript stellen Sie den hexadezimalen Ziffern `0x` voran, um sie zu kennzeichnen, also beispielsweise:

```
0xAF
```

Die Umrechnung ist recht einfach: Die erste hexadezimale Ziffer `A` wird mit 16 multipliziert, dann wird die zweite `F` addiert. Für das angegebene Beispiel sieht die Rechnung wie folgt aus:

```
10 * 16 + 15 = 175
```

Diese Ausgabe können Sie auch im Browser überprüfen, da hexadezimale Werte automatisch umgerechnet werden:

```
document.write(0xAF);
```

Das hexadezimale System wird vor allem in HTML dazu verwendet, RGB-Farbwerte anzugeben. Für jeden der drei Farbkanäle bilden zwei hexadezimale Ziffern den jeweiligen Wert von 0 bis 255 ab. Die Reihenfolge ist dabei immer Rot, Grün und Blau. Folgender Farbwert entspricht dem RGB-Wert 255, 0, 0, der für reines Rot steht:

```
#FF0000
```

Oktale Schreibweise

Die oktale Schreibweise wird vom ECMA-262-Standard nicht unterstützt, aber von einigen JavaScript-Interpretern zugelassen. In Tabelle 1.1 finden Sie eine Übersicht, welcher aktuelle Browser diese Schreibweise unterstützt.

NS4.x	NS6	NS7	IE4	IE5	IE5.5	IE6	O7	M1	K3
	✓	✓		✓	✓	✓		✓	

Tabelle 3.1: Die oktale Schreibweise

In der oktalen Schreibweise beginnen Zahlen immer mit einer 0. Anschließend folgen Ziffern von 0 bis 7. Für die oktale Schreibweise werden also nur 8 Ziffern verwendet, da die Basis 8 ist (*oktal* ist der griechische Begriff für *Acht*). Die Umrechnung ist recht einfach:

```
0265
```

ergibt

```
2*64 + 6*8 + 5 = 181
```

!! STOP

Da einige Browser die oktale Schreibweise nicht unterstützen, sollten Sie sie normalerweise vermeiden. Die Schwierigkeit besteht für die Browser darin, dass sie nicht erkennen, ob die vorangestellte 0 für einen Dezimalwert oder die oktale Schreibweise steht.

Gleitkommazahlen

Gleitkommazahlen (Float) haben Nachkommastellen, die durch einen Dezimalpunkt abgetrennt werden. Der Punkt verdeutlicht, dass JavaScript aus dem anglophonen Sprachraum kommt. Dort ersetzt der Dezimalpunkt das bei uns bekannte Dezimalkomma.

```
4.65
```

Erscheint vor dem Punkt keine Zahl, ist die Zahl vor dem Komma eine 0:

```
.848
```

steht also für 0,848.

i INFO

In JavaScript wird im Allgemeinen nicht zwischen Integern und Fließkommazahlen differenziert. Beide werden auch als Datentyp `Zahl` *bezeichnet.*

Beachten Sie, wie schnell es geschehen kann, dass eine Gleitkommazahl auf einmal mit einem Komma statt einem Punkt im Skript landet. Der JavaScript-Interpreter ignoriert im Allgemeinen das Komma und gibt keinen Fehler aus, was die Fehlersuche sehr erschwert. Bei `document.write()` sind mehrere durch Kommata getrennte Parameter möglich. Die folgende Zeile

```
document.write(0,40 + 4);
```

entspricht also

```
document.write(0);
document.write(40 + 4);
```

In Abbildung 3.1 sehen Sie, dass der Browser statt 4,4 als Ergebnis 044 anzeigt.

Gleitkommazahlen nehmen Werte von 10^{-324} bis 10^{308} an. Größere Werte werden unendlich (Rückgabe: Infinity).

Abbildung 3.1: Der Browser zeigt aufgrund des Kommas ein falsches Ergebnis an.

Exponential-Schreibweise

Neben der normalen Schreibweise existiert ebenfalls eine alternative Schreibweise mit Exponent. Zuerst folgt eine reelle Zahl, dann der Buchstabe E (oder e), ein Vorzeichen (Plus oder Minus) und eine ganze Zahl als Exponent. Wird das Vorzeichen weggelassen, nimmt der JavaScript-Interpreter automatisch einen positiven Wert an.

```
2.2E2
```

steht also für 2,2 * 10^2 = 220. Da das Ergebnis ein Integer ist, bildet die Exponential-Notation genau genommen sowohl eine Schreibweise für Gleitkommazahlen als auch für ganze Zahlen.

Wenn Sie eine Zahl in Exponential-Schreibweise ausgeben, wandelt der JavaScript-Interpreter sie nur bis zu einer bestimmten Größe in die normale Schreibweise um.


```
9e+20
```

wird als 900 000 000 000 000 000 000 ausgegeben,

```
9e+21
```

nur als 9e+21.

Im negativen Bereich werden nur Exponenten bis -6 als normale Zahlen ausgegeben; ab -7 erfolgt Exponential-Notation.

Zeichenketten

Zeichenketten oder auch Strings[2] enthalten Text, der allerdings durchaus auch aus Ziffern und anderen Zeichen bestehen kann. Sie erkennen einen String im Code immer an den Anführungszeichen.

2 Da der Begriff String aus Programmierersicht die richtige Bezeichnung für den Datentyp ist, verwenden wir im weiteren Verlauf des Buches hauptsächlich diesen Anglizismus. Taucht der Begriff Zeichenkette auf, ist dies als Synonym zu String zu verstehen.

```
"Text"
```

Ebenfalls ein String ist:

```
"12,30 Euro"
```

Die Anführungszeichen, die die Strings umgeben, können auch nur einfache Anführungszeichen sein:

```
'Text'
```

!! STOP

Der JavaScript-Interpreter erlaubt keine Mischung von einfachen und doppelten Anführungszeichen:

```
document.write("Text');
```

zur Ausgabe eines Strings ist also verboten. Der Browser gibt einen Fehler aus (siehe Abbildung 3.2).

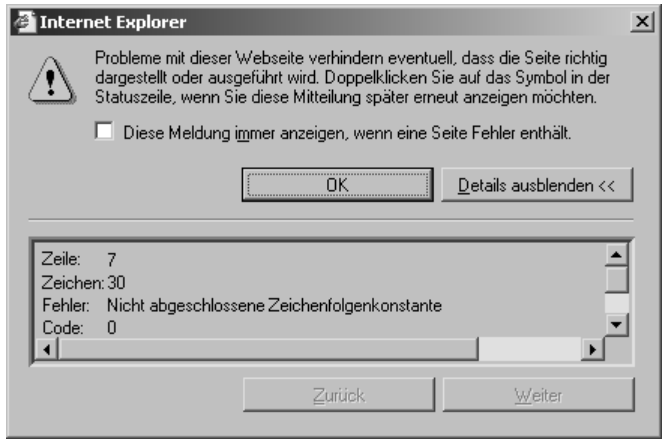


Abbildung 3.2: Die Fehlermeldung im Internet Explorer 6, wenn die Anführungszeichen nicht korrekt angegeben wurden.

String-Länge

Ein String darf in JavaScript eine unbegrenzte Länge haben. Allerdings sollten Sie den String nicht mit einem Zeilenumbruch im Texteditor zwischen den Anführungszeichen umbrechen, da dies zu einem Fehler führt.

:-) TIPP

Einige sehr alte Browser unterstützen nur Strings mit maximal 255 Zeichen. Dies ist wichtig, wenn Sie eine 100%-Abwärtskompatibilität anstreben.

Escape-Sequenzen

Wenn Sie in Strings Anführungszeichen oder Apostrophe (die geraden Anführungszeichen entsprechen) verwenden möchten, kann es Probleme geben. Folgender String ist problematisch:

```
'Ich denk's mir'
```

Der Interpreter nimmt an, das zweite gerade Anführungszeichen sei bereits das Ende des Strings. Um ihm zu signalisieren, dass das gerade Anführungszeichen keine JavaScript-Bedeutung hat, wird es mit einem Backslash (\)[3] entwertet.

```
'Ich denk\'s mir'
```

Der Backslash ist das allgemein gültige Zeichen für die Entwertung von Zeichen. Den Backslash in Verbindung mit dem Zeichen nennt man Escape-Sequenz (häufig auch *Steuerelemente*).

In JavaScript gibt es einige standardmäßig verfügbare Escape-Sequenzen, die in Tabelle 3.2 aufgeführt sind. Wenn Sie ein Zeichen, das nicht entwertet werden müsste, mit dem Backslash (\) entwerten, hat dies keine Auswirkungen, da der Browser den Backslash einfach ignoriert. Wollen Sie dagegen einen Backslash in den Text einfügen, sollten Sie ihn als Escape-Sequenz (\\) schreiben.

Tabelle 3.2: Escape-Sequenzen von JavaScript

Escape-Sequenz	Beschreibung
\0	Nullzeichen; NUL. Alle Zeichen dahinter werden ignoriert. Das Nullzeichen terminiert den String.
\b	Backspace; entspricht der Entf-Taste, wird aber nicht von allen Browsern interpretiert.
\f	Seitenvorschub
\n	Zeilenumbruch; beachten Sie, dass dieser Zeilenumbruch nicht funktioniert, wenn Sie den String auf einer HTML-Seite ausgeben. Hier benötigen Sie das HTML-Tag .
\r	Wagenrücklauf; stammt noch aus der Schreibmaschinenzeit. Der Cursor geht bis an den Anfang auf das erste Zeichen in der nächsten Zeile zurück.
\t	Tabulator

3 Tastenkürzel AltGr + ß

Tabelle 3.2: Escape-Sequenzen von JavaScript (Forts.)

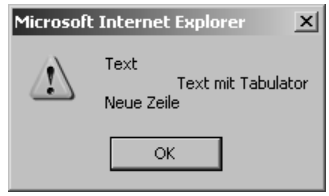
Escape-Sequenz	Beschreibung
`\v`	Vertikaler Tabulator; wird auch von neueren Browsern nicht unterstützt und ist daher zu meiden.
`\\`	Backslash; gibt **einen** Backslash aus.
`\'`	Einfaches Anführungszeichen
`\"`	Doppeltes Anführungszeichen
`\xAA`	Zeichen aus dem Latin-1-Zeichensatz, wobei die Zeichennummer durch den hexadezimalen Wert bei AA angegeben wird (Latin-1-Zeichen können auch in oktaler Schreibweise mit drei Ziffern angegeben werden. Dies wird von den neueren Browsern unterstützt, ist aber nicht Teil der ECMA-262 v3).
`\uAAAA`	Zeichen aus dem Unicode-Zeichensatz, wobei die Zeichennummer als hexadezimaler Wert mit vier Stellen (AAAA) angegeben wird (in Kapitel 3.3 unter »Zeichensatz« erfahren Sie Details zu den Zeichensätzen in JavaScript).

Die besonderen Escape-Sequenzen für den Zeilenumbruch, den Wagenrücklauf und den Tabulator sehen Sie in JavaScript am einfachsten in einem Warnfenster, das mit `alert()` erzeugt wird.

Listing 3.1: Escape-Sequenzen im Einsatz *(escape_sequenzen.html)*

```
<html>
  <head>
    <title>Escape-Sequenzen</title>
  </head>
  <body>
    <script language="JavaScript"><!--
      alert("Text\r\tText mit Tabulator\nNeue Zeile");
     //--></script>
  </body>
</html>
```

Abbildung 3.3: Die Escape-Sequenzen in einem Warnfenster



Bei einem Warnfenster oder ähnlichem entscheiden sowohl der Browser als auch das Betriebssystem, wie Escape-Sequenzen angezeigt werden. Entsprechend gibt es einige Unterschiede, vor allem beim Zeilenumbruch: Der Netscape Navigator unter Windows stellt beispielsweise den Wagenrücklauf nicht als Zeilenumbruch dar. Er benötigt ein `\n`*. Testen Sie also gut!*

Abbildung 3.4: Der Wagenrücklauf wird nicht angezeigt.

Boolean

Der Datentyp `Boolean`[4] besteht nur aus zwei Zuständen: *wahr* oder *falsch*. Entsprechend heißt er auch Wahrheitswert. Wann benötigt man diesen Datentyp? Recht häufig, da in der Programmierung oftmals eine Bedingung überprüft wird, die dann *wahr* oder *falsch* zurückliefert.

Ein Computer speichert Informationen in einem Bit. Dieses Bit kann ebenfalls nur zwei Zustände annehmen: 0 und 1. Entsprechend werden die beiden Wahrheitswerte manchmal auch als 0 (`false` = falsch) und 1 (`true` = wahr) geschrieben.

JavaScript verwendet also entweder 0 für *falsch* und 1 für *wahr* oder alternativ die englischen Begriffe: `true` und `false`.[5]

```
var a = true;
var b = false;
```

Im obigen Beispiel werden zwei Variablen definiert (siehe Kapitel 3.2 »Variablen deklarieren«). Variable `a` hat den Wert `true` und Variable `b` den Wert `false`.

4 Benannt ist der Datentyp nach dem englischen Mathematiker George Boole, geboren am 2. November 1815 in Lincoln, England und gestorben am 8. Dezember 1864 in Ballintempel, Irland. Er begründete die Boolesche Algebra, die Logik mit einfachen algebraischen Symbolen für die Mathematik greifbar machte.

5 In den Beschreibungen finden Sie im Folgenden sehr oft die Anglizismen *true* und *false*, da sie in JavaScript die entsprechenden Schlüsselwörter für Wahrheitswerte darstellen.

Objekte

Objekte – sowohl eingebaute als auch selbst definierte – sind für die Programmierung mit JavaScript sehr wichtig (siehe Kapitel 6 »Objekte«). Die komplexen Informationen, die in einem Objekt gespeichert werden, verlangen nach einem eigenen Datentyp: `Object`.[6]

Eigenschaften von Objekten, die Werte speichern, können einen beliebigen Datentyp, beispielsweise `String` *oder* `Boolean` *annehmen.*

Arrays

Arrays speichern Datenelemente und vergeben einen Index, mit dem auf die Werte zugegriffen werden kann (siehe Kapitel 7 »Arrays und Strings«). Intern entspricht ein Array ebenfalls dem Datentyp `Object`.

Funktionen

Funktionen stellen Anweisungen bereit, die von beliebiger Stelle aus dem Skript aufgerufen werden können (siehe Kapitel 5 »Funktionen«). Sie sind in JavaScript Daten des Typs `Object`.

Spezielle Werte

Neben den einfachen (primitiven) Datentypen `Zahl`, `String` und `Boolean` und dem komplexen Datentyp `Object`, gibt es noch einige spezielle Werte. Das sind eigentlich Datentypen, die nur einen Wert haben.

null

Der Wert `null` steht für einen Datensatz *ohne Wert*. Er ist in JavaScript nicht gleichbedeutend mit 0, auch wenn er manchmal so verwendet wird.

Sie erhalten beispielsweise 0, wenn Sie alle Elemente eines Arrays mit einer Schleife durchgehen (siehe Kapitel 7 »Arrays und Strings«) und das Ende des Arrays erreicht haben. Der nächste – nicht mehr vorhandene – Datensatz hat dann den Wert `null`.

undefined

Wenn eine Variable zwar deklariert ist, aber keinen Wert zugewiesen bekommen hat, hat sie den Wert `undefined`. Ebenso ist eine Objekteigenschaft ohne Wert `undefined`. Mit diesem Wert können Sie also prüfen, ob eine Variable oder Eigenschaft überhaupt einen Wert hat.

6 Der Datentyp `Object` wird auch als komplexer Datentyp bezeichnet. Einfache (oder primitive) Datentypen sind Zahl (Integer und Fließkomma), String und Boolean.

Der Vergleichsoperator[7] für Gleichheit in JavaScript (==) setzt `null` *und* `undefined` *gleich. Intern handelt es sich allerdings um unterschiedliche Werte. Dies wird mit dem Operator Genau gleich (===) sichtbar. Er berücksichtigt auch den Datentyp der zu vergleichenden Elemente und liefert bei einem Vergleich zwischen* `null` *und* `undefined` *das Ergebnis* `false`.

Infinity

`Infinity` steht für *unendlich*. Dieser Wert wird ausgegeben, wenn eine Rechnung oder Zahl in JavaScript nicht mehr dargestellt werden kann. Sie ist dann unendlich.

NaN

`NaN` besagt, dass ein Wert keine Zahl ist. Dies wird beispielsweise ausgegeben, wenn bei einer Berechnung nicht nur der Datentyp `Zahl`, sondern beispielsweise ein String verwendet wird.

3.2 Variablen deklarieren

Um Variablen als Datenspeicher zu nutzen, müssen sie erst einmal erzeugt werden. Dieser Vorgang heißt Deklarieren. Der JavaScript-Interpreter erkennt an dem Schlüsselwort `var`, dass eine Variable deklariert werden soll:

```
var x;
```

Diese Zeile deklariert die Variable `x`.

Was geschieht beim Deklarieren? Der JavaScript-Interpreter erkennt, dass eine neue Variable angelegt werden soll. Er reserviert dann für die Variable Raum im Hauptspeicher und verknüpft diesen Platz mit dem Variablennamen, sodass er jederzeit darauf zugreifen kann, wenn die Variable wieder zum Einsatz kommt.

In JavaScript müssen Sie bei der Deklaration der Variablen ***keinen*** *der soeben besprochenen Datentypen zuweisen. JavaScript ist* ***nicht typisiert****, das heißt, eine JavaScript-Variable kann jeden Datentyp speichern, im Laufe ihrer Lebensdauer sogar unterschiedliche. Mehr dazu erfahren Sie in Kapitel 3.3 unter »Typkonvertierung«.*

Ist eine Variable erst einmal deklariert, kann Sie jederzeit mit ihrem Namen, in unserem Beispiel also `x`, aufgerufen werden. Ein einfacher Aufruf kann in der Ausgabeanweisung `document.write()` erfolgen:

```
document.write(x);
```

7 Siehe Kapitel 4.1 »Operatoren«

Da die Variable `x` bisher noch keinen Wert hat, wird `undefined` ausgegeben.

Abbildung 3.5: Bisher ist die Variable noch `undefined`.

Sie können auch mehrere Variablen in einer Zeile deklarieren. Trennen Sie die Variablen mit Kommata:

```
var x, y;
```

definiert also `x` *und* `y`.

Wenn Sie `var` *weglassen, reserviert JavaScript für die Variable dennoch Platz im Hauptspeicher. Allerdings ist die Variable dann grundsätzlich global, das heißt im kompletten Skript verfügbar. Lokale Variablen sind auf eine Funktion beschränkt (siehe Kapitel 5.1 Abschnitt »Lokale und globale Variablen«). Wenn Sie eine lokale Variable in einer Funktion deklarieren wollen und* `var` *vergessen, kann es unter Umständen zu Problemen kommen, vor allem wenn das Skript noch gleichnamige globale Variablen besitzt.*

Werte zuweisen

`undefined` ist natürlich auf Dauer kein schöner Wert[8] für eine Variable. Daher muss sie einen Wert erhalten. Dies erledigt in JavaScript das Ist-Gleich-Zeichen (`=`).

```
var x = 3;
```

Diese Zeile weist der Variablen `x` den Wert 3 zu. Das Ist-Gleich heißt wegen seiner Funktion auch Zuweisungsoperator. Andere Operatoren finden Sie in Kapitel 4.1 »Operatoren«.

Wird einer Variable, wie im aktuellen Beispiel, bereits beim Deklarieren ein Wert zugewiesen, spricht man auch von einer Initialisierung der Variablen. Abgesehen vom Sprachgebrauch kann eine Variable natürlich auch erst später einen Wert erhalten:

8 Der Wert einer Variable wird auch als Literal bezeichnet (siehe oben). Er tritt einmal auf. Ändert sich der Wert, handelt es sich um ein neues Literal.

```
var x;
x = 3;
```

Diese Zeilen bewirken dasselbe wie die vorhergehende Initialisierung: Die Variable `x` erhält den Wert 3.

Sollten Sie eine Variable häufiger mit dem Schlüsselwort `var` *deklarieren, bleibt JavaScript gnädig. Wird bei der zweiten Deklaration ein Wert zugewiesen, übernimmt JavaScript den Wert als normale Zuweisung und ignoriert* `var`.

Erfolgt bei der ersten Deklaration eine Zuweisung, bei der zweiten dagegen nicht, behält die Variable den Wert aus der ersten Zuweisung.

```
var x = 1;
var x;
document.write(x);
```

gibt also 1 aus.

Werte ändern

Die Werte von Variablen sind beliebig änderbar. Um einer Variablen einen neuen Wert zuzuweisen, verwenden Sie einfach wieder den Zuweisungsoperator Ist-Gleich (=):

```
var x = 3;
x = 2;
```

Die Variable `x` erhält hier den Wert 3, der anschließend auf 2 geändert wird.

Variablen sind aber noch mächtiger: Mit Berechnungen und Operatoren können Sie Variablen Rechnungsergebnisse zuweisen und sie miteinander verknüpfen.

```
var x = 4 * 3;
var y = x / 2;
```

Das Multiplikationszeichen (*) und das Divisionszeichen (/) sind so genannte arithmetische Operatoren. Sie und einige andere nützliche Operatoren lernen Sie in Kapitel 4.1 »Operatoren« kennen.

Strichpunkt (;)

In den bisherigen Beispielen folgte nach jeder Variablendeklaration oder -zuweisung, aber auch nach jeder Ausgabe mit `document.write()` immer ein Strichpunkt (;)[9].

Der Strichpunkt beendet eine Anweisung in JavaScript. Muss er aber nicht. In anderen Sprachen, beispielsweise Java oder PHP, ist die Angabe des Strichpunkts Pflicht, der JavaScript-Interpreter dagegen meckert nicht, wenn Sie ihn weglassen.

Die Beispiele in diesem Buch sind dennoch alle mit Strichpunkt ausgestattet.[10] Dafür gibt es einen Grund: Anweisungen sehen wesentlich übersichtlicher aus, wenn sie beendet werden. Der Programmierstil wird sauberer.

Wollen Sie mehrere Anweisungen in eine Zeile schreiben, ***müssen*** *Sie diese durch Strichpunkte trennen:*

```
var x, y;
x = 2; y = 4;
```

Fügen Sie keine Strichpunkte an, übernimmt das der JavaScript-Interpreter automatisch. Dies ist bei Anweisungen über mehrere Zeilen problematisch. Im Folgenden sehen Sie die Rückgabe eines Wertes in einer Funktion, über zwei Zeilen verteilt:

```
return
x;
```

Der Interpreter nimmt aber an, dass die obere Zeile eine eigenständige Anweisung ist und behandelt dies wie folgt:

```
return;
x;
```

Mit diesen Zeilen wird aber nicht `x` zurückgegeben, sondern die Funktion verlassen, ohne dass eine Rückgabe erfolgt. Die zweite Zeile `x;` wird ignoriert.

Mehrere Variablen

Wenn Sie mehrere Variablen gleichzeitig, das heißt mit nur einem `var`-Schlüsselwort, deklarieren wollen, trennen Sie sie durch Kommata:

9 Der Strichpunkt heißt auch *Semikolon*.

10 Wenn er hinter einer Anweisung fehlt, handelt es sich um ein Versehen des Autors. Eine kurze E-Mail an `th@hauser-wenz.de` genügt, und der Fehler wird in der nächsten Auflage behoben.

```
var x, y, z;
```

definiert `x`, `y` und `z` als Variablen.

Das funktioniert auch mit Wertzuweisung:

```
var x = 2, y = 4, z = 5;
```

3.3 Bezeichner und Typen

Bevor Sie mit Variablen zu arbeiten beginnen und sich ans Programmieren machen, sollten Sie noch einige Grundlagen über Variablen und JavaScript mit auf den Weg nehmen. Dieses Kapitel behandelt hauptsächlich die Variablennamen (auch Bezeichner genannt) und die Typkonvertierung in JavaScript.

Groß- und Kleinschreibung

JavaScript unterscheidet zwischen Groß- und Kleinschreibung[11]. Dies gilt nicht nur bei Variablennamen, sondern auch bei Funktionsnamen, JavaScript-Programmierelementen und Objekten.

```
var Test = 3;
```

ist also eine andere Variable als

```
test = 4;
```

Bei den Objekten gab es zeitweise Einschränkungen, was die Groß- und Kleinschreibung betrifft. Der Internet Explorer 3 hat in beiden Varianten bei Browserobjekten und -eigenschaften nicht auf Groß- und Kleinschreibung geachtet. Da dies in den späteren Versionen allerdings geändert wurde, hat das heute keine praktischen Auswirkungen mehr.

Variablennamen

Ein Variablenname darf in JavaScript aus Buchstaben, Zahlen und dem Unterstrich (_) bestehen. Beginnen darf er allerdings nur mit einem Buchstaben oder dem Unterstrich, nicht aber mit einer Zahl. Darüber hinaus sind Sonderzeichen im Variablennamen nicht erlaubt.

11 Dies wird auch als *case-sensitive* bezeichnet.

Folgende Namen sind korrekt:

```
var _Hallo;
var T_H_2;
var x9302;
```

Nicht erlaubt sind dagegen:

```
var Jo&Jo;
var 90210;
var 0acht;
```

Des Weiteren sind in Variablennamen die reservierten Schlüsselwörter von JavaScript verboten (siehe nächster Abschnitt)

Was erlaubt ist, ist aber nicht unbedingt sinnvoll. Sie sollten sich angewöhnen, mit sprechenden Variablennamen oder einer schlüssigen Namenskonvention zu arbeiten. Besteht ein Variablenname aus mehreren Wörtern, trennen Sie ihn am besten mit Unterstrichen oder, indem Sie neue Wörter mit einem Großbuchstaben beginnen.

```
var test_account;
var testAccount;
```

Die zweite Variante gleicht auch der Konvention, die JavaScript bei seinen Methoden und Eigenschaften verwendet: Der Anfang – also das erste Wort – mit Kleinbuchstaben, neue Worte beginnen mit Großbuchstaben.

Reservierte Schlüsselwörter

JavaScript verwendet – wie jede andere Programmiersprache auch – bestimmte Worte für Programmierkonstrukte. Beispielsweise gibt das Wort `var` dem JavaScript-Interpreter an, dass danach eine Variablendeklaration folgt. Diese Wörter heißen Schlüsselwörter.

Sie sind reserviert, das heißt, Variablenbezeichner und jede andere Art von Bezeichner[12] dürfen nicht so benannt werden.

Die Vorgaben zu reservierten Schlüsselwörtern beginnen mit Wörtern, die bereits in der Sprache verwendet werden. Sie dürfen auf jeden Fall nicht als Bezeichner eingesetzt werden. Zusätzlich gibt es häufig für die Zukunft vorgesehene Schlüsselwörter, die oftmals noch als Bezeichner funktionieren würden. Wenn es dann aber neue Browserversionen gibt, versagen die alten Skripten mit diesen Bezeichnern. Daher gilt grundsätzlich die Regel, keine Schlüsselwörter als Bezeichner in Skripten zu verwenden.

12 beispielsweise Funktions- und Objektnamen

Die grundlegenden Vorgaben für JavaScript kommen offiziell von der ECMA. In der Spezifikation ECMAScript-262 v3 werden die in Tabelle 3.3 und Tabelle 3.4 aufgeführten Schlüsselwörter reserviert.

break	else	new	var
case	finally	return	void
catch	for	switch	while
continue	function	this	with
default	if	throw	
delete	in	try	
do	instanceof	typeof	

Tabelle 3.3: Reservierte Schlüsselwörter von ECMAScript-262 v3

abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchronized
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	

Tabelle 3.4: Für die Zukunft reservierte Schlüsselwörter von ECMAScript-262 v3

Netscape folgt in der JavaScript-Version 1.5 weitgehend den Vorgaben von ECMAScript-262 v3 (siehe Tabelle 3.5).

abstract	delete	function	null	throw
boolean	do	goto	package	throws
break	double	if	private	transient
byte	else	implements	protected	true
case	enum	import	public	try
catch	export	in	return	typeof
char	extends	instanceof	short	var

Tabelle 3.5: Reservierte Schlüsselwörter (Netscape JavaScript 1.5)[13]

13 Quelle: `http://developer.netscape.com/docs/manuals/js/core/jsref15/keywords.html#1004016`

Tabelle 3.5: Reservierte Schlüsselwörter (Netscape JavaScript 1.5) (Forts.)

class	false	int	static	void
const	final	interface	super	volatile
continue	finally	long	switch	while
debugger	float	native	synchronized	with
default	for	new	this	

Microsoft hat die JScript-Referenz auch an JScript.NET angepasst (läuft auch unter dem Namen JScript 7). Daher gibt es noch einige neue reservierte Schlüsselwörter. Diese sind zwar eigentlich nicht alle für den clientseitigen Einsatz vorgesehen. Aus Gründen der Kompatibilität zu zukünftigen Browsern und JavaScript-Implementationen sollten Sie sie allerdings dennoch nicht als Bezeichner verwenden.

Tabelle 3.6: Die reservierten Schlüsselwörter in JScript (mit JScript.NET)[14]

break	case	catch	class	const
continue	debugger	default	delete	do
else	export	extends	false	finally
for	function	if	import	in
instanceof	new	null	protected	return
super	switch	this	throw	true
try	typeof	var	while	with

Tabelle 3.7: Neue reservierte Wörter[15]

abstract	boolean	byte	char	decimal
double	enum	final	float	get
implements	int	interface	internal	long
package	private	protected	public	sbyte
set	short	static	uint	ulong
ushort	void			

14 Quelle: `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/jscript7/html/jsreserved.asp`

15 Wörter dürfen aufgrund der Abwärtskompatibilität als Bezeichner eingesetzt werden. Empfehlenswert ist dies allerdings nicht, da spätere Browserversionen dies eventuell untersagen.

assert	ensure	event	goto	invariant
namespace	native	require	synchronized	throws
transient	use	volatile		

Tabelle 3.8: Für die Zukunft reservierte Schlüsselwörter in JScript

Ein genauer Blick auf die Listen verrät, dass `undefined` *bei Netscape und Microsofts JScript kein reserviertes Schlüsselwort ist,* `null` *dagegen schon*[16]*. Der Grund dafür ist, dass* `undefinied` *als globale Variable gesehen wird, deren Wert für alle Variablen und Eigenschaften ohne Wert gilt.*

!! STOP

Formatierung

JavaScript ignoriert Leerzeichen und Tabulatoren zwischen Ausdrücken (auch Token). Ein Ausdruck kann eine Variable, ein Programmierkonstrukt oder ein Bezeichner für ein Objekt sein.

Leerzeichen und Tabulatoren in Ausdrücken werden natürlich nicht ignoriert, sondern teilen den Ausdruck in Mehrere.

!! STOP

Für die Formatierung Ihrer Skripten sollten Sie also durchaus beispielsweise Leerzeichen verwenden. In diesem Buch werden die Zeilen für jede logische Skriptebene um jeweils zwei Leerzeichen eingerückt.

Zeichensatz

Bisher haben Sie bereits die Beschränkungen bei Bezeichnern für Variablen, Funktionen und Objekte und die Verwendung von Escape-Sequenzen bei Strings kennen gelernt. Abgesehen von diesen Einschränkungen unterstützt JavaScript in neueren Browsern allerdings den kompletten Unicode-Zeichensatzes[17].

	NS4.x	NS6	NS7	IE4	IE5	IE5.5	IE6	O7	M1	K3
	✓[18]	✓	✓			✓	✓	✓	✓	✓

Tabelle 3.9: Unicode

16 außer in ECMAScript!

17 Unicode steht für *Universal Code* und vereinheitlicht und erweitert bestehende Zeichensätze.

18 ab Netscape Navigator 4.0.6

INFO

Unterstützt der Browser kein Unicode, wird normalerweise automatisch nur der ASCII-Zeichensatz unterstützt. In ECMAScript ist Unicode erst in v3 allgemein gültiger Standard (außer in Bezeichnern). Vorher waren reine Unicode-Zeichen nur in Strings und Kommentaren (siehe Kapitel 3.4 unter »Kommentare«) zugelassen.

Der Unicode-Zeichensatz hat eine Tiefe von 16 Bit und enthält die Zeichen für die meisten Sprachen der Welt. Ein Unicode-Zeichen wird wie folgt notiert:

uAAAA

AAAA steht dabei für die hexadezimale Zahl des Unicode-Zeichens.

Unter Windows 2000/XP ist die Zeichentabelle (START/PROGRAMME/ZUBEHÖR/SYSTEMPROGRAMME/ZEICHENTABELLE) sehr komfortabel geworden und zeigt von den Zeichen eines Zeichensatzes auch den Unicode an.

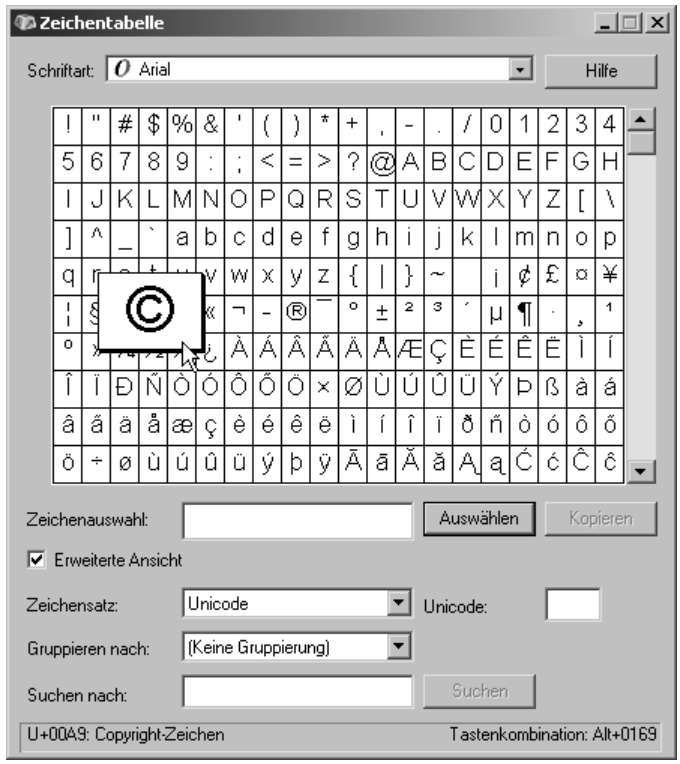


Abbildung 3.6: Die Zeichentabelle unter Windows XP

Typkonvertierung

JavaScript konvertiert Datentypen von Variablen automatisch:

```
var x = "drei";
x = 3;
```

Zuerst wird Variable `x` als Datentyp `String` initialisiert. Anschließend wird ihr ein Integer zugewiesen. Die Typkonvertierung von `String` in `Zahl` erfolgt automatisch.

Die automatische Datentypkonvertierung unterscheidet JavaScript von anderen Programmiersprachen wie Java und C#, in denen der Datentyp für eine Variable explizit angegeben werden muss.

Funktionen zur Typkonvertierung

Benötigen Sie eine Zahl aus einem String für eine Berechnung oder möchten Sie einen anderen Datentyp in einen `String` umwandeln, müssen Sie Hilfsfunktionen verwenden. Diese Funktionen sind globale Funktionen, das heißt, sie stehen überall in JavaScript zur Verfügung:

- `Element.toString()`wandelt ein Element mit einem anderen Datentyp in einen String um.
- `Number(String)` ändert einen String in eine Zahl und entscheidet automatisch, ob das Ergebnis ein Integer oder eine Fließkommazahl ist.
- `parseFloat(String)` extrahiert eine Fließkommazahl aus einem String.
- `parseInt(String)` wandelt einen String in einen Integer um.

Wie die Funktionen zur Typkonvertierung arbeiten, erfahren Sie in Kapitel 5.3 im Abschnitt »Datentypen ändern«. Dort finden Sie auch jeweils ein Beispiel.

Garbage Collection

Variablen reservieren in JavaScript automatisch einen Platz im Hauptspeicher. Wenn sich dann der Wert im Hauptspeicher ändert, müssten C- und C++-Programmierer den alten Wert von Hand löschen. JavaScript erledigt das automatisch. Dieses Prinzip heißt Garbage Collection.[19]

19 Für Informatik-Interessierte: Die meisten Browser verwenden Garbage-Collection-Varianten des *Mark-and-Sweep-Algorithmus*. Er kennzeichnet und entfernt Objekte, auf die von keinem anderen Element verwiesen wird.

3.4 Hilfsmittel

Das nächste Kapitel bietet einen tiefer gehenden Einstieg in die Programmierung. Bevor wir damit beginnen, finden Sie hier noch einige nützliche Hilfsmittel, beispielsweise Kommentare, um Skripten mit eigenen Anmerkungen versehen zu können.

Kommentare

Ein Kommentar ist ein Text, der vom JavaScript-Interpreter ignoriert wird. In diesen »ignorierten« Text können Sie also schreiben, was Sie möchten. In der Praxis werden Kommentare verwendet, um den Code zu erklären. Insbesondere bei längeren Skripten ist es sehr wichtig, ausführlich zu kommentieren, denn in einem Jahr wissen Sie sicherlich häufig nicht mehr, wie der alte Code genau aufgebaut war. Außerdem helfen Kommentare beim Austausch von Code zwischen verschiedenen Programmierern.

JavaScript unterstützt einzeilige und mehrzeilige Kommentare. Einzeilige Kommentare werden mit zwei Schrägstrichen (//) begonnen.

```
var user = "Meier";  //Variable für den Nutzernamen
```

Ab der Stelle, wo die Kommentarzeichen eingefügt sind, wird der restliche Text oder Code entwertet. So können Sie natürlich auch ganze Code-Zeilen deaktivieren. Dies ist insbesondere beim Testen und bei der Fehlersuche sinnvoll:

```
// document.write(user);
```

Mehrzeilige Kommentare werden in /* und */ eingeschlossen. Der gesamte Code dazwischen ist inaktiv.

```
var user = "Meier";  /*Variable für
der Nutzername wird deklariert*/
var pass = "test";
```

Ein mehrzeiliger Kommentar kann auch einen eigens formatierten Block mit einer Beschreibung, beispielsweise einer Funktion[20], enthalten:

Listing 3.2: Ein Kommentarblock *(kommentar.html)*

```
<html>
  <head>
    <title>Kommentarblock</title>
  </head>
  <body>
    <script language="JavaScript"><!--
```

20 Siehe hierzu Kapitel 5 »Funktionen«

```
      function quadrat(x) {
        return x*x;
      }
      /*------------------------
       |quadrat() gibt das Qua- |
       |drat eines Parameters   |
       |zurück.                 |
       ------------------------*/
      document.write(quadrat(4));
    --></script>
  </body>
</html>
```

Diese relativ aufwändige Formatierung im Kommentar sorgt zwar für optische Übersichtlichkeit, sollte allerdings erst vorgenommen werden, wenn die Definition der Funktion abgeschlossen und vollständig beschreibbar ist, denn eine nachträgliche Änderung ist je nach Umfang der Dokumentation ein beträchtlicher Mehraufwand.

Konstanten

Konstanten sind einmal definierte Werte, die sich im Gegensatz zu Variablen nicht ändern und auch nicht ändern sollen. Konstanten werden in JavaScript mit dem Schlüsselwort `const` eingeleitet:

```
const Name = Wert;
```

Leider werden Konstanten zurzeit nur vom Netscape Navigator ab Version 6 und von Mozilla unterstützt.

NS4.x	NS6	NS7	IE4	IE5	IE5.5	IE6	O7	M1	K3
	✓	✓						✓	

Tabelle 3.10: `const`

Für die Namen (Bezeichner) von Konstanten gelten dieselben Regeln wie für Variablennamen.

Im folgenden Beispiel wird der Umrechnungskurs Euro/DM in einer Konstanten hinterlegt und dann dazu verwendet, einen DM-Betrag in Euro umzuwandeln.

Listing 3.3: Arbeiten mit einer Konstante *(konstante.html)*

```
<html>
  <head>
    <title>Konstanten</title>
  </head>
  <body>
    <script language="JavaScript"><!--
```

```
      const kurs = 1.95583;
      var dm = 200;
      var euro = dm / kurs;
      document.write(euro);
    --></script>
  </body>
</html>
```

Abbildung 3.7: Der Navigator 7 zeigt die korrekte Umrechnung an.

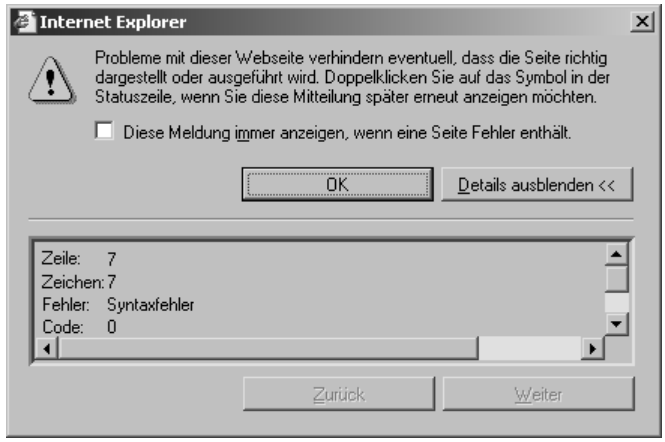


Abbildung 3.8: Der Internet Explorer 6 meldet einen Syntaxfehler, da er `const` nicht unterstützt.