

JavaScript

JavaScript ist eine Skriptsprache mit der man Client- und Server-Internetprogramme entwickeln kann. In einem Web-Browser wie Netscape oder Internet Explorer kann man die in eine HTML-Datei eingefügten Befehle direkt einlesen und ausführen; mit LiveWire zum Beispiel ist es weiters möglich server-basierte Programme zu schreiben, die den CGI-Applikationen (Common Gateway Interface) ähnlich sind.

JavaScript und Java

JavaScript ist mit der Programmiersprache Java in einem gewissen Grad verwandt, da es die gleiche Syntax besitzt und die meisten Befehle Javas unterstützt. Obwohl es viele Ähnlichkeiten zwischen diesen beiden Sprachen gibt, sind die Unterschiede leicht erkennbar. JavaScript besitzt zum Beispiel weniger und daher auch einfachere Datentypen. Der Datentyp einer Variable etwa muss nicht deklariert werden und außerdem besitzt JavaScript ein einfacheres Objekt-Modell. Im Gegensatz zum Compile-Time-System von Java, basiert JavaScript über ein Run-Time-System. JavaScript ist eine Art Ergänzung von Java, indem es die Eigenschaften und Leistung von Java-Applets oder Plug-Ins abfragen bzw. verändern kann.

Die folgende Tabelle zeigt ein paar Unterschiede und die Gemeinsamkeiten von Java und JavaScript:

JavaScript	Java
Vom Client direkt interpretiert – wird nicht kompiliert.	Auf dem Server kompiliert und dann auf dem Client ausgeführt.
Objekt-basierend. Keine Klassen oder Vererbung; „eingebaute“ Objekte.	Objekt-orientiert. Programme bestehen aus Objekten, Klassen mit Vererbung, etc.
Integriert in bzw. eingefügt in HTML.	Applets separiert von HTML (Aufruf durch HTML-Seiten).
Keine Datentyp-Deklaration von Variablen.	Datentypen von Variablen müssen deklariert werden.
Objekt-Referenzen werden in „Run-Time“ überprüft.	Objekt-Referenzen müssen schon in „Compile-Time“ existieren.
Sicher: kann nicht auf Festplatte schreiben.	Sicher: kann nicht auf Festplatte schreiben.

JavaScript in HTML

Einfügen von JavaScript in Dokumenten

Skripten werden in HTML mit Hilfe des Skript-Tags eingefügt:

```
<script>...</script>
```

Der Text eines Skripts wird dabei zwischen dem `<script>` und dem `</script>` eingefügt. Außerdem kann das Skript-Tag folgende Parameter haben:

```
<script language="JavaScript">...</script>
```

Der Parameter *language* ist erforderlich.

```
<script src="http://myscript.js">...</script>
```

Ein optionaler Parameter, der eine separate Datei angibt, welche JavaScript-Instruktionen enthält.

Hinweise

- Skripten innerhalb des Skript-Tags werden erst nach dem Seitenaufruf interpretiert. Funktionen werden dabei nur gespeichert, nicht ausgeführt. Diese werden nur dann ausgeführt, wenn sie von der HTML-Seite aufgerufen wird.
- Befehle, die mit dem *src*-Parameter eingelesen werden, werden wie normale Befehle innerhalb des Skript-Tags behandelt. Diese haben gegenüber Befehle innerhalb des Skript-Tags Vorrang.
- Die im *src*-Parameter angegebene Datei sollte die Erweiterung *.js* tragen.
- Skripten können von Kommentar-Anweisungen eingeschlossen werden, damit ältere Browser, die JavaScript nicht unterstützen, diese einfach ignorieren.
- Genau wie Java unterscheidet JavaScript zwischen Groß- und Kleinbuchstaben.
- Zeichenketten sollten von einfachen Anführungszeichen eingeschlossen werden, Parameterwerte hingegen von doppelten.

Werte, Namen und Datentypen

- Werte
- Variablennamen
- Datentypen

Werte (Values)

JavaScript erkennt folgende Typen von Werten:

- Zahlen wie 42 oder 3.14159
- Logische Werte (Boolsche Datentypen): wahr oder falsch (*true* oder *false*)
- Zeichenketten wie „Howdy!“
- null, ein spezielles Schlüsselwort, das einen Nullwert bezeichnet

In JavaScript gibt es keine Unterscheidung zwischen ganzen Zahlen (*integer*) und Fließkommazahlen (*real*). Zudem gibt es keinen eigenen Datentyp für Datum und Zeit. Das Objekt *date* und die damit verbundenen Funktionen ermöglichen die Arbeit mit Datum und Zeit.

Konvertieren von Datentypen (Casting)

In JavaScript ist die Deklaration von Datentypen einer Variable nicht erforderlich. Die Datentypen werden während der Ausführung des Programms automatisch konvertiert. Da Ausdrücke von links nach rechts eingelesen werden, so muss JavaScript diesem Schema folgen und die Datentypen von links nach rechts einlesen. JavaScript wird daher versuchen den Datentyp eines Ausdrucks vom Datentyp des linken Wertes abzuleiten. Dazu ein Beispiel:

```
var zeichen = "7"  
var zahl = 42
```

Nun zwei Ausdrücke:

```
x = zeichen + zahl  
y = zahl + zeichen
```

Beim ersten Ausdruck wird die Zahl 42 in die Zeichenkette "42" umgewandelt und die Variable *x* erhält den Datentyp einer Zeichenkette (*string*), weil der linke Wert des Ausdrucks ("7") eine Zeichenkette ist. Durch Verbinden der beiden Zeichenketten "7" und "42" erhält man anschließend den Wert "742". Beim zweiten Ausdruck wird aber die Zeichenkette "7" in die Zahl 7 konvertiert und die Variable *y* wird zu einer Zahl. Durch Addieren der Zahl 42 und der Zahl 7 erhält man 49.

Es gibt jedoch einige Fälle, in denen JavaScript keine Konvertierung vornehmen kann. Der Ausdruck

```
var zeichen = "Wien"  
var zahl = 42  
  
y = zahl + zeichen
```

wird zu einem Fehler führen, da "Wien" nicht in eine Zahl konvertiert werden kann.

Variablenamen

Variablen werden verwendet, um gewisse Werte zu speichern. Diese Variablen müssen Namen tragen durch die man sie aufrufen kann. Bei der Vergebung von Namen muss man gewissen Regeln beachten:

Ein Identifikator oder Name muss mit einem Buchstaben oder dem Unterstrich („_“) anfangen. Die folgenden Zeichen können entweder Ziffern (0-9) oder Buchstaben (A-Z, a-z) sein.

Datentypen (Literals, Data types)

Ganze Zahlen (Integers)

Ganze Zahlen können als Dezimal- (Basis 10), Hexadezimal (Basis 16) oder als Oktalzahlen (Basis 8) dargestellt werden. Eine Dezimalzahl besteht aus einer mit einem Vorzeichen versehenen Sequenz von Ziffern ohne einer vorangestellten 0 (Null). Eine 0 (Null) am Anfang einer solchen Sequenz kennzeichnet eine Oktalzahl, ein 0x (oder 0X) eine Hexadezimalzahl. Dezimalzahlen können die Ziffern 0-9 enthalten, Hexadezimalzahlen die Ziffern 0-9 und die Buchstaben a-f bzw. A-F und Oktalzahlen lediglich die Ziffern 0-7.

Fließkommazahlen (Floating Point)

Eine Fließkommazahl kann aus folgendem bestehen: eine ganze Dezimalzahl, einem Punkt („.“), einer Nachkommazahl, einem Exponenten und einem Vorzeichen. Der Exponent wird mit einem „e“ oder „E“, einem Vorzeichen und einer Zahl gekennzeichnet. Eine Fließkommazahl muss aus mindestens einer Ziffer und einem Punkt oder einem Exponent bestehen.

Boolsche Datentypen (Boolean)

Boolsche Datentypen können folgende Werte annehmen: *true* oder *false*.

Zeichenketten (Strings)

Eine Zeichenkette besteht aus keinem oder mehreren Zeichen, die von einfachen (‘) oder doppelten (“) Anführungszeichen eingeschlossen werden, wobei nur gleiche Anführungszeichen verwendet werden dürfen.

Sonderzeichen

Man kann folgende spezielle Zeichen in Zeichenketten verwenden:

- `\b` kennzeichnet einen Backspace.
- `\f` kennzeichnet ein Form Feed.
- `\n` kennzeichnet ein Return.
- `\r` kennzeichnet einen Zeilenumbruch.
- `\t` kennzeichnet das Tabulatorzeichen.

Ausdrücke und Operatoren in JavaScript

Ausdrücke

Ein Ausdruck ist eine Reihe von Variablen, Datentypen, Operatoren und anderen Befehlen, die zu einem einzigen Wert führen. Der neue Wert kann entweder eine Zahl, eine Zeichenkette oder ein logischer Wert sein. Es gibt zwei Arten von Ausdrücken: Die, die einer Variable einen Wert zuweisen, und die, die einfach einen Wert besitzen.

JavaScript besitzt folgende Arten von Ausdrücken:

- Arithmetische: für Zahlen, etc.
- String: für Zeichenketten
- Logische: wahr oder falsch

Bedingungs-Ausdrücke

Diese Ausdrücke können eine aus zwei gegebenen Werten haben, wobei der Wert von einer Bedingung abhängig ist. Die Syntax lautet:

$$(\text{Bedingung}) ? W1 : W2$$

Wenn die Bedingung erfüllt wird, erhält der Ausdruck den Wert von $W1$, ansonsten den von $W2$.

Zuweisungs-Operatoren

Zuweisungs-Operatoren weisen der Variable links vom Operator einen Wert, der von der Variable rechts vom Operator abhängig ist, zu. Der einfachste Zuweisungs-Operator ist das Ist-Gleich-Zeichen (`„=“`).

Hier ein paar Vereinfachungen von oft verwendeten mathematischen Berechnungen:

- `x += y` bedeutet `x = x + y`
- `x -= y` bedeutet `x = x - y`
- `x *= y` bedeutet `x = x * y`
- `x /= y` bedeutet `x = x / y`
- `x %= y` bedeutet `x = x % y`

Und hier noch ein paar Vereinfachungen für binäre Berechnungen:

- $x \ll= y$ bedeutet $x = x \ll y$
- $x \gg= y$ bedeutet $x = x \gg y$
- $x \gg\gg= y$ bedeutet $x = x \gg\gg y$
- $x \&= y$ bedeutet $x = x \& y$
- $x \^= y$ bedeutet $x = x \^ y$
- $x |= y$ bedeutet $x = x | y$

Operatoren

In JavaScript gibt es logische, arithmetische und String-Operatoren. Weiters gibt es noch unäre und binäre Operatoren: Binäre Operatoren erfordern zwei Werte, einen vor und einen nach dem Operator:

wert1 operator wert2

Ein unärer Operator verlangt nur einen Wert:

operator wert

oder

wert operator

Integer-Operatoren

Unäre Integer-Operatoren

Operator	Bedeutung
-	unäre Negation
~	bitweises Komplement
++	Inkrement
--	Dekrement

Die unäre Negation ändert das Vorzeichen des Integers. Das bitweise Komplement ändert jedes einzelne Bit, d.h. eine 0 wird zu 1, eine 1 wird zu 0. Inkrement erhöht den Integer um 1, Dekrement erniedrigt ihn um 1. Der Inkrement- und der Dekrement-Operator können jeweils als Postfix oder Präfix verwendet werden. Wird er wie bei

$y = x++$

als Postfix verwendet, dann erhält y den Wert von x und anschließend wird x erhöht. Der Ausdruck

$y = ++x$

bedeutet, dass y den Wert von x erhält, nachdem x um 1 erhöht wurde.

Binäre Integer-Operatoren

Binäre Operatoren verlangen zwei Werte. Diese beiden Werte werden berechnet und einer Variable zugewiesen.

&&											
?:											
=	+=	-=	*=	/=	%=	<<=	>>=	>>>=	&=	^=	=
,											

JavaScript Objekt-Modell

JavaScript besitzt ein relativ einfaches Objekt-Modell. Ein Objekt ist ein Gebilde mit Eigenschaften, wobei die Eigenschaften durch Variablen definiert werden. Eigenschaften können selber Objekte sein. Funktionen, die einen Bezug zu einem Objekt haben, bezeichnet man als die Methoden des Objekts. Zusätzlich zu bereits vorhandenen Objekten wie zum Beispiel in Web-Browsern, kann man seine eigenen Objekte definieren.

Objekte und Eigenschaften

Ein JavaScript-Objekt besitzt Eigenschaften. Auf solche Eigenschaften kann man wie folgt zugreifen:

```
objektName.objektEigenschaft
```

Eigenschaften werden definiert, indem man ihnen einen Wert zuweist:

```
objektName.objektEigenschaft = "Wert"
objektName.objektEigenschaft = 567
```

Funktionen und Methoden

Eine Funktion ist eine JavaScript-Prozedur – eine Reihe von Ausdrücken – die man aus jeder Stelle im Programm aufrufen kann. Funktionen werden mit dem function-Befehl definiert. Es ist ratsam, Funktionen im HEAD-Abschnitt einer HTML-Datei zu definieren, da diese beim Seitenaufbau zuerst geladen werden.

Eine Funktions-Definition besteht aus dem function-Befehl, gefolgt vom

- Namen der Funktion,
- einer Liste von Parametern, eingeklammert und durch Beistriche voneinander getrennt, und
- den Ausdrücken, die die Funktion ausführen soll, wenn sie aufgerufen wird. Diese sind durch geschwungene Klammern eingeschlossen.

Eine Definition einer Funktion:

```
function schreiben(string) {
    document.write("<p>" + string)
}
```

Bei der Definition einer Funktion wird diese noch nicht ausgeführt, sie muss erst aufgerufen werden>

```
<script>
schreiben("Dieser Text wird angezeigt.")
</script>
```

Parameter müssen nicht immer Zahlen oder Zeichenketten sein. Man kann auch ganze Objekte als Parameter übergeben.

Methoden

Methoden sind Funktionen, die mit einem Objekt verknüpft sind. Methoden werden wie Funktionen definiert. Danach müssen sie nur noch mit einem Objekt verknüpft werden:

```
objekt.MethodenName = FunktionsName
```

Das Aufrufen der Methode geschieht auf folgende Weise:

```
objekt.MethodenName(Parameter);
```

Neue Objekte erstellen

Um ein Objekt zu erstellen sind zwei Schritte notwendig:

- Den Typ des Objekts definieren, indem man eine Funktion definiert
- Eine Instanz des Objekts mit *new* kreieren

Beispiel:

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

Diese zwei Argumente erstellen zwei Objekte vom Typ person:

```
Lucero = new person("Lucero Joe", 18, "M")  
Gates = new person("Gates Bill", 40, "M")
```

Verwendung von built-in (= eingebauten) Objekten und Funktionen

JavaScript stellt folgende built-in Objekte und Funktionen zur Verfügung:

- *String*-Objekt
- *Math*-Objekt
- *Date*-Objekt
- *eval*-Funktion

String-Objekte sind einfach Zeichenketten, die als Objekte behandelt werden. Die Methoden dienen der Formatierung der Zeichenkette. Das *Math*-Objekt enthält Eigenschaften oder Methoden, wobei Eigenschaften mathematische Konstanten verkörpern und Methoden mathematische Funktionen. Das *pi* ist zum Beispiel eine Eigenschaft vom Objekt *Math* und wird mit *Math.PI* aufgerufen; die *Sinus*-Funktion ist eine Methode auf die mit *Math.sin(Zahl)* zugegriffen werden kann. Das *Date*-Objekt ermöglicht die Arbeit mit Datum und Zeit. Es besitzt keine Eigenschaften, dafür viele Methoden für das Einstellen, das Abrufen und andere Veränderungen der Zeit oder des Datums. Die *eval*-Funktion nimmt einen Ausdruck als ein Argument, berechnet es und gibt den Wert zurück.

Array

Ein Array ist eine Reihe von Werten, die mit einem Index versehen werden und in einer Variable gespeichert werden. Der Aufruf der einzelnen Werte erfolgt durch die Angabe des Index des jeweiligen Wertes. JavaScript besitzt keinen eigenen Datentyp für Arrays, stattdessen verwendet man Objekte und ihre Eigenschaften als Arrays.

Objekt-Hierarchie in Web-Browsern

Web-Browser definieren automatisch einige Objekte; ihre Anzahl ist dabei abhängig vom Inhalt der HTML-Seite. Das Fenster, in dem die HTML-Seite angezeigt wird ist beispielsweise ein Objekt (*window*-Objekt). Der Inhalt der Seite ist eine Eigenschaft vom *window*-Objekt (*window.document*), gleichzeitig ist er aber auch ein selbstständiges Objekt. Die Überschrift dieser HTML-Seite ist dann schließlich auch eine Eigenschaft der HTML-Seite (*window.document.title*). Diese HTML-Elemente bilden eine Hierarchie von Objekten und Eigenschaften

Objekte

anchor	Eine Textstelle, die als Ziel eines (Hypertext-) Links dient.
button	Ein Druckknopf in einem HTML-Formular.
checkbox	Ein Kästchen, das vom User aktiviert und deaktiviert werden kann.
Date	Ermöglicht das Arbeiten mit Datum und Zeit.
document	Enthält Informationen über das dargestellte Dokument.
form	Ein Formular.
frame	Ein Frame ist ein Teil des Anzeigefensters eines Web-Browsers.
history	Enthält Informationen über die vom User besuchten Seiten.
link	Eine Textstelle, die als eine Art "Tür" zu einer anderen HTML-Seite dient.
location	Enthält Informationen über die URL ("Adresse") der angezeigten Seite.
Math	Enthält u.a. wichtige Konstanten wie π (pi) oder trigonometrische Funktionen
navigator	Enthält Informationen über den vom User benutzten Web-Browsers.
password	Ein Textfeld, in dem der User sein Passwort eingeben kann.
radio	Eine Liste, von der der User ein Element aussuchen kann
reset	Ein Druckknopf, der die Eingaben des Benutzers löscht.
select	Ein Pull-Down-Menü oder eine Liste.
string	Eine Zeichenkette.
submit	Sendet die vom Benutzer in ein Formular eingetragenen Informationen.
text	Ein einzeliges Textfeld.
textarea	Ein mehrzeiliges Textfeld.
window	Das Fenster, in dem die HTML-Seite angezeigt wird.

Wichtige Befehle

break-Befehl

Unterbricht eine *while*- oder *for*-Schleife und übergibt die Kontrolle an die nächste Anweisung nach der Schleife.

Syntax:

```
break
```

Kommentare

Kommentare sind Notizen des Programmierers und dienen zur Erklärung des Codes. Sie werden vom Interpreter ignoriert. JavaScript unterstützt einzeilige und mehrzeilige Kommentare.

Syntax:

```
// einzeiliges Kommentar
/* mehrzeiliges Kommentar */
```

continue-Befehl

Unterbricht die Ausführung einer *while*- oder *for*-Schleife, verlässt im Gegensatz zum *break*-Befehl die Schleife nicht. Stattdessen springt die Kontrolle bei einer *while*-Schleife zur Ausgangsbedingung, bei einer *for*-Schleife zum Schrittwert.

Syntax:

```
continue
```

for-Befehl

Eine *for*-Schleife besteht aus drei optionalen Argumenten, die eingeklammert sind und voneinander durch Beistriche getrennt werden. Diese sind

- Der Anfangswert (der Wert bei dem die *for*-Schleife beginnen soll)
- Die Bedingung (solange diese Bedingung erfüllt wird, werden die Ausdrücke einer *for*-Schleife ausgeführt)
- Der Schrittwert (erhöht den Anfangswert um den gegebenen Wert)

Syntax:

```
for (Anfangswert, Bedingung; Schrittwert) {
    Ausdrücke
}
```

function-Befehl

Dieser Befehl dient zur Deklaration von Funktionen und ihren Parametern. Soll die Funktion einen Wert übergeben, dann verwendet man den Befehl *return*. Funktionen können nicht verschachtelt werden.

Syntax:

```
function name([parameter1] [..., parameter]) {
    Ausdrücke
}
```

if...else-Befehl

Der *if...else*-Befehl führt, abhängig von einer Bedingung, bestimmte Befehle aus. Wenn die Bedingung wahr ist, werden die Befehle unter *if* ausgeführt; sind sie falsch, werden die *else*-Befehle ausgeführt. Der *else*-Befehl ist optional.

Syntax:

```
if (Bedingung) {
    Befehle
}
else {
    Befehle
}
```

return-Befehl

Veranlasst eine Funktion einen Wert an die Kontrolle zu übergeben.

Syntax:

```
return Ausdruck;
```

var-Befehl

Dient der Deklaration von Variablen. Optional können gleich Werte zugewiesen werden.

Syntax:

```
var variablenName [= Wert] [..., variablenName [= Wert] ]
```

while-Befehl

Der *while*-Befehl kennzeichnet eine Schleife, die solange durchlaufen wird, solange die Bedingung wahr ist. Wenn die Bedingung falsch ist, wird die Schleife verlassen.

Syntax:

```
while (Bedingung) {
    Anweisungen
}
```

with-Anweisung

Die *with*-Anweisung verwendet ein gegebenes Objekt als Standard-Referenzobjekt für alle Zugriffe auf Eigenschaften ohne Objektreferenz.

Syntax:

```
with (objekt) {
    Anweisungen
}
```