JavaScript

JavaScript wird direkt in HTML-Dokumente eingebunden.

Gib folgende Zeilen mit einem Texteditor (Notepad) ein:
(Falls der Editor nicht gefunden wird, öffne im Browser eine Datei mit der Endung html und gehe ins Menü Ansicht/Quelltext anzeigen.)

```
<html>
Dies ist normales HTML. <br>
< script language="JavaScript">
        document.write("Hallo Welt! Dies ist JavaScript.");
</script>
<br>
Wieder normales HTML. <br>
</html>
```

Speichere diese Datei unter dem Namen hallo.html und öffne das erzeugte Dokument in einem Browser. Im Browser wird folgende Ausgabe zu sehen sein:

Dies ist normales HTML.
Hallo Welt! Dies ist JavaScript.
Wieder normales HTML.

Alles, was innerhalb der <script>-Tags steht, wird als JavaScript interpretiert. Wir haben hier lediglich eine Anweisung, die der Browser ausführen soll, sie lautet

document.write("Hallo Welt! Dies ist JavaScript.");

Variablen

Um z.B. Zahlen zu verarbeiten, benötigen wir Speicherzellen, sogenannte Variablen, in die wir Werte speichern. Variablennamen beginnen mit einem Buchstaben, ansonsten sind sie frei wählbar. Damit der Speicherplatz einer Variablen reserviert wird und der Variablenname mit der internen Adresse dieses Platzes verknüpft wird, werden benötigte Variablen am Anfang des Programms angegeben, sie werden deklariert.
Man kann bei der Deklaration von Variablen einen Wert vorgeben (die Variable wird initialisiert). Wenn man z.B. eine Variable x mit dem Ausgangswert 17 haben möchte, schreibt man:

```
var x = 17;
```

Dies hat den gleichen Effekt wie die beiden Zeilen:

```
var x;
x = 17;
```

Das Semikolon am Ende der Anweisung kann entfallen, wenn in einer Zeile nur eine Anweisung steht.

In JavaScript müssen Variablen im allgemeinen nicht deklariert werden, jedoch werden wir dies wegen der besseren Lesbarkeit stets tun.

Besonders wichtig: JavaScript unterscheidet zwischen Groß- und Kleinbuchstaben.
Mit var eingabe und var Eingabe werden unterschiedliche Variablen deklariert.

In JavaScript werden keine Datentypen angegeben.

Intern bestimmt die erste Wertzuweisung den Typ:

integer, ganze Zahl, 0, 1, 2, 3, -1, -2, . . .

string, Zeichenkette, z.B. “JavaScript “

boolean, Wahrheitswert, eine boolesche Variable kann entweder den Wert true (= wahr) oder false (= falsch) annehmen.

Beispiele für die Ausgabe:

```
document.write(x);
document.write( “Das Ergebnis lautet: “+ y);
document.write( “Ergebnisse: “+ zahl + “<br> “);
```

Wertzuweisungen

```
x = y + 25;
x = x + 10;
```

Der Wert des Ausdrucks auf der rechten Seite wird der Variablen auf der linken Seite zugewiesen. Rechts kann ein beliebig komplizierter Rechenausdruck stehen, links steht nur der Name einer Variablen, die den Wert aufnimmt. Das Gleichheitszeichen hat also eine ganz andere Bedeutung als in der Mathematik. Das mathematische Gleichheitszeichen wird in JavaScript == geschrieben.

Um die Zahl 10 zu x zu addieren und den neuen Wert wieder in x zu speichern, gibt es die Kurzform:

```
x += 10;
```

Desweiteren bedeuten:

Abkürzung	Bedeutung
x += y	x = x + y
x −= y	x = x − y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Der Modulo-Operator % gibt den Rest einer Division zweier ganzer Zahlen zurück. Beispielsweise gibt der Ausdruck 14 % 5 den Wert 4 zurück und der Ausdruck 6 % 3 den Wert 0.

Um eine Variable x um den Wert 1 zu vergrößern oder zu verkleinern. schreiben wir abkürzend:

x++; bzw. x −−;

for-Schleife

Damit in einem Programm dieselbe Anweisung, mehrere sind auch möglich, eine bestimmte Anzahl, z.B. 1000 mal, ausgeführt wird, verwendet man eine for-Schleife.

```
var summe = 0 ;
for (var i = 0 ;  i <= 100 ;  i++)
{    summe = summe + i;
}
```

Die for-Schleife hat folgende Struktur (Syntax):

```
for (Zählvariablenanfang;  Zählvariablenbedingung;  Zählvariablenveränderung)
{     Anweisung;
      Anweisung;
      . . .
}
```

Im Kopf der for-Schleife kann die Zählvariable deklariert werden, ihr Anfangswert wird hier angegeben (die Zählvariable wird initialisiert).

Nachdem die Anweisungen in der Schleife ausgeführt wurden, wird die Zählvariable gemäß der Vorgabe geändert, im allgemeinen wird sie um eins erhöht. Der Vorgang wiederholt sich solange wie die *Zählvariablenbedingung* erfüllt ist.

Statt i++ könnte auch i = i + 1 oder kürzer i += 1 stehen.

```
for (var n = 0;  n <= 100;  n+=2) { ... }
```

Die Zählvariable n wird von 0 bis 100 in Zweierschritten hochgezählt.

```
for (var n = 1000;  n > 0;  n−−) { ... }
```

Die Zählvariable n wird von 1000 bis 1 heruntergezählt.

if else - Abfrage

Sollen Anweisungen nur dann ausgeführt werden, falls eine Bedingung erfüllt ist (sogenannte bedingte Anweisung), so ist eine if-Abfrage zu verwenden.

```
if (z == 4)    document.write("Z hat den Wert 4 ");
```

Weiter ist auch möglich:

```
if (z == 4)     document.write("Ergebnis"+z+"<br>");
       else     document.write("Es liegt kein Ergebnis vor.");
```

Sollen jeweils mehrere Anweisungen ausgeführt werden, so sind sie in einen Block mit geschweiften Klammern { ... } zusammenzufassen.

Die Anweisung if ... else wird gebraucht, um eine Gruppe von Anweisungen abhängig davon auszuführen, ob eine oder mehrere Bedingungen wahr oder falsch ergeben.

```
if ( Bedingung )
{    Anweisung;          1. Block
     Anweisung;
     . . .
}
  else
  {  Anweisung;          2. Block
     Anweisung;
     . . .
  }
```

Wenn die Bedingung wahr ergibt, werden die Anweisungen des 1. Blocks (einmal) ausgeführt und der 2. Block wird übersprungen. Wenn die Bedingung falsch ergibt, ist es umgekehrt. Dabei kann der else Teil weggelassen werden, wenn es keine Anweisungen gibt, die ausgeführt werden sollen, wenn die Bedingung nicht erfüllt ist.

Vergleiche

Operator	Vergleich
==	gleich
! =	ungleich
>	größer als
<	kleiner als
>=	größer oder gleich
<=	kleiner oder gleich

Bedingungen können mit *und* (&&), bzw. *oder* (||) verknüpft werden.

Funktionen

```
function Summe(a, b, c)
{    var sum;
     sum = a + b + c;
     return sum;
}
```

Jede Funktionsdefinition beginnt mit dem Schlüsselwort function,
gefolgt von dem Namen der Funktion - in diesem Fall Summe.

Nach dem Funktionsnamen kommen zwei Klammern, in denen die Parameter der Funktion stehen. Diese Parameter sind die Daten, die das aufrufende Programm an die Funktion übergibt. Bei Funktionen, die keine Parameter benötigen, folgen dem Funktionsnamen nur zwei leere Klammern ( ).

Die Anweisungen der Funktion stehen in geschweiften Klammern.

Innerhalb der Funktion dient das Schlüsselwort return dazu, den Wert, der von der Funktion an das aufrufende Programm zurückgegeben werden soll, zu bestimmen. Einige Funktionen geben keinen Wert zurück, in diesem Fall lässt man das return einfach weg.

Im Programm wird die Funktion z. B. durch

```
n = Summe (1, 2, 3);
```

aufgerufen.

Das erste Argument, in diesem Fall 1, wird hierbei als erster Parameter a der Funktion verwendet. Das zweite und dritte Argument werden jeweils als zweiter und dritter Parameter verwendet.

while-Schleife

```
while (Bedingung)
{    Anweisung;
     Anweisung;
     . . .
}
```

So lange die *Bedingung* wahr ist, werden die Anweisungen in der while-Schleife ausgeführt.

Hier ist ein einfaches Beispiel, welches auf dem Browser von 4 zurück auf 1 zählt:

```
var z = 4;
while (z > 0)
{    document.write (z-- + "<br>");
}
```

Durch den HTML Tag
 wird jeder Wert von z in einer eigenen Zeile dargestellt.

Wenn die *Bedingung* bereits beim ersten Auswerten falsch ergibt, werden die Anweisungen innerhalb der geschweiften Klammern nicht ausgeführt.

Wenn die *Bedingung* am Anfang wahr ergibt und in der Schleife nichts geschieht, um sie auf falsch zu bringen, werden die Anweisungen in der Schleife immer wieder ausgeführt und die Schleife wird nie zu Ende sein (Endlosschleife).

Es gibt noch eine andere Variante der while-Schleife, die do-while-Schleife

```
do
{    Anweisung;
     Anweisung;
     . . .
}
  while (Bedingung);
```

Der Unterschied besteht darin, dass die do-while-Schleife die *Anweisungen* mindestens einmal ausführt, bevor die Bedingung geprüft wird.

switch-Anweisung

Beispiel:

```
switch (x/Math.abs(x))
{    case −1:      document.writeln("x ist negativ");
                   break;
     case 1:       document.writeln("x ist positiv ");
                   break;
          . . .
     default:      document.writeln("x ist Null ");
}
```

Die Syntax lautet:

```
switch (Ausdruck)
{    case Markierung:     Anweisungen;
                          break;
     case Markierung:     Anweisungen;
                          break;
     . . .
     default:             Anweisungen;
}
```

Mit der switch-Anweisung können verschachtelte if-else Anweisungen vermieden werden.
Die *Anweisungen* werden ausgeführt, wenn die *Markierung* mit *Ausdruck* übereinstimmt.
Fehlt die break-Anweisung, so werden nach einer gefundenen Markierung auch die weiteren ausgeführt.

break und continue

Wenn in einer while- oder for-Schleife auf ein break getroffen wird, wird die Schleife vollständig beendet und die Programmausführung setzt bei der ersten Anweisung unmittelbar nach der Schleife fort.

Wenn in einer while- oder for-Schleife ein continue erreicht wird, wird das Programm bei der ersten Anweisung in der Schleife fortgesetzt und alle Anweisungen zwischen dem continue und dem richtigen Ende der Schleife werden übersprungen.

Mit break könnte das Beispiel:

```
var z = 4;
while (z > 0)
{     document.write (z−− + “<br>“);
}
```

so aussehen:

```
var z = 4;
while (true)
{     document.write (z−− + “<br>“);
      if (z == 0) break;
}
```

Als Bedingung für das while wurde die logische Konstante true verwendet. Das bedeutet, dass die while-Schleife endlos laufen würde, überließen wir sie sich selbst.

Die folgenden Zeilen zeigen die Benutzung von continue. Was bewirken diese Zeilen?

```
var z = 10;
while (z > 0)
{     z−−;
      if (z == 6) continue;
      document.write(z + “< br> “);
}
```