INP (05)

Prof. Dr.-Ing. S. Gössner

University of Applied Sciences Lippe & Höxter

Inhalt

Kontrollanweisungen

Normalerweise wird ein Programm *sequentiell* von der ersten bis zur letzten Anweisung ausgeführt. Eine solche sequentielle Ausführung ist der voreingestellte Programmfluss.

Auf der Grundlage dieses Verhaltens lassen sich nur sehr simple Programme schreiben.

Programmiersprachen stellen daher eine Menge von Anweisungen zur Verfügung, mittels derer *der Programmfluss kontrolliert* werden kann.

Javascript hat die Kontrollanweisungen der *C*-Sprachfamilie übernommen.

- Verzeigungsanweisungen oder Auswahlanweisungen
  - `if`
  - `if ... else`
  - `switch ... case`
  - Auswahloperator `?:`
- Wiederholungsanweisungen
  - `while`
  - `do ... while`
  - `for`
  - `for ... in`

Eine weitere Form der Kontrolle des Programmflusses ist die Verwendung von *Funktionen* und *Methoden*, bzw. die *Ausnahmebehandlung* mittels `throw`, `try`, `catch`.

if - Anweisung

Syntax

```
if (<Bedingung>) {
  <Anweisung>*
}
```

Nach dem reservierten Wort `if` folgt innerhalb eines runden Klammerpaars eine *Bedingung* als *Ausdruck*. Wenn und nur wenn diese Bedingung unmittelbar – oder nach der Umwandlung in einen Wert vom Typ *Boolean* – `true` liefert, wird der nachfolgende Block von Anweisungen ausgeführt.

Beachte:

- Wenn genau **eine** Anweisung nach `if (<Bedingung>)` folgt, dann kann das geschweifte Klammerpaar weggelassen, also `if (<Bedingung>) <Anweisung>;` geschrieben werden.
- Ein unmittelbar nach der `if`-Anweisung folgendes *Semikolon* ist i.A. ein Fehler! Ein einzelnes Semikolon steht hier für eine *leere Anweisung*, was i.d.R. nicht das ist, was man will.

if ... else - Anweisung

Syntax

```
if (<Bedingung>) {
   <Anweisung>*
}
else {
   <alternative Anweisung>*
}
```

Unmittelbar nach der `if`-Anweisung folgt das Schlüsselwort `else` und ein Block *alternativer Anweisungen*.

Beachte:

- `else` steht für ein *bedingungsloses **"andernfalls"***, und benötigt **keinen** in runden Klammern folgenden *Ausdruck*.
- Wenn genau **eine** Anweisung nach `else` folgt, dann kann das geschweifte Klammerpaar wegelassen, also "`else <Anweisung>;`" geschrieben werden.

if .. else - Verkettung

Syntax

```
if (<Bedingung>) {
  <Anweisung>*
}
else if (<Bedingung>) {
  <Anweisung>*
}
else {
  <Anweisung>*
}
```

Die Anweisung im `else`-Zweig einer `if ... else`-Anweisung kann wiederum eine `if`- oder `if .. else`-Anweisung sein. Hierdurch gelangen wir zu einer multikonditionalen Anweisungsfolge.

Beispiel

```
var color = "rot", action;

if (color == "grün")
   action = "gehen";
else if (color == "rot")
   action = "stehen";
else if (color == "grün+gelb")
   action = "fertigmachen";
else if (color == "gelb")
   action = "rennen";
else
   alert("ungültige Farbe/Farbkombination");
```

switch ... case - Anweisung

Syntax

```
switch(<Ausdruck>) {
  case <Wert-1>: <Anweisung>*;
                 break;
  case <Wert-2>: <Anweisung>*;
                 break;
  ...
  case <Wert-n>: <Anweisung>*;
                 break;
  default:       <Anweisung>*
                 break;
}
```

Die `switch...case`-Anweisung ist eine multikonditionale Anweisung, die den Wert eines *Ausdrucks* mit einer Reihe gegebener Konstanten vergleicht. Liefert der Vergleich `<Ausdruck> == <Wert-i>` den Wert `true`, so werden die zu diesem `case`-Zweig gehörigen Anweisungen ausgeführt. Liefern alle `case`-Vergleiche den Wert `true`, werden die Anweisungen eines vorhandenen `default`-Zweigs ausgeführt.

Beispiel

```
var color = "rot", action;

switch (color) {
  case "grün";
     action = "gehen";
     break;
  case "rot":
     action = "stehen";
     break;
  case "grün+gelb":
     action = "fertigmachen";
     break;
  case "gelb":
     action = "rennen";
     break;
  default:
     alert("ungültige Farbe/Farbkombination");
}
```

Beachte:

- Der `default`-Zweig ist optional.
- Die `break`-Anweisung verhindert, dass nach einem zutreffenden `case`-Zweig die nachfolgenden `case`-Zweige ebenfalls ausgeführt werden. Das `break` fehlt also nur in Ausnahmefällen

Konditionaloperator

Syntax

```
<Bedingung> ? <Ausdruck> : <Alternativausdruck>
```

Dieser *ternäre* Operator liefert den Wert eines *Ausdrucs* und stellt keine *Anweisung* dar. Wegen der Ähnlichkeit zur `if...else`-Anweisung bietet sich eine Diskussion hier jedoch an.

Liefert `<Bedingung>` den Wert `true`, erhält der Gesamtausdruck den Wert von `<Ausdruck>`, sonst von `<Alternativausdruck>`.

Beispiel:

```
var alter = 8,
    eintritt = alter < 13 ? "frei" : "kostenpflichtig";
```

Schleifen

Um eine Folge von Anweisungen mehrfach hintereinander ausführen zu können, bietet Javascript die üblichen *Wiederholungsanweisungen* der C-Sprachfamilie.

Wiederholungsanweisungen werden auch als *Schleifen* oder *Iterationen* bezeichnet. Allen ist gemeinsam, dass sie in jedem Durchlauf eine Ausdruck auswerten und im Falle von `true` die Iteration fortsetzen, andernfalls abbrechen.

Javascript bietet:

- while
- do … while
- for
- for … in

while - Schleife

Syntax

```
while (<Laufbedingung>) {
  <Anweisung>*
}
```

Dem Schlüsselwort `while` folgt in einem *runden Klammerpaar* die `<Laufbedingung>`. Besitzt diese den Wert `true`, wird der nachfolgende Block von Anweisungen ausgeführt, sonst nicht.

Beispiel

```
var eingabe = prompt("Eingabe", "");

while (eingabe != "") {
  document.write(eingabe + "\n");
  eingabe = prompt("Eingabe", "");
}
```

Beachte:

- Wenn genau **eine** Anweisung nach `while (<Laufbedingung>)` folgt, dann kann das geschweifte Klammerpaar weggelassen, also `while (<Laufbedingung>) <Anweisung>;` geschrieben werden.
- Ein unmittelbar nach der `while`-Anweisung folgendes *Semikolon* ist i.A. ein Fehler! Ein einzelnes Semikolon steht hier für eine *leere Anweisung*, was i.d.R. nicht das ist, was man will.

do ... while - Schleife

Syntax

```
do {
  <Anweisung>*
} while (<Laufbedingung>)
```

Ein Anweisungsblock wird von den Schlüsselwörtern `do` und `while` eingeschlossen, gefolgt von einem *runden Klammerpaar*, das eine `<Laufbedingung>` beinhaltet. Besitzt diese den Wert true, wird der eingeschlossene Block von Anweisungen wiederholt ausgeführt, sonst nicht.

Da die Laufbedingung am Ende der `do...while`-Schleife geprüft wird, bietet diese sich für solche Fälle an, in denen mindestens eine Iteration durchgeführt werden soll.

Beispiel

```
var eingabe;
do {
  eingabe = prompt("Eingabe", "");
  document.write(eingabe + "\n");
} while (eingabe != "")
```

for - Schleife

Syntax

```
for (<Initialisierung>; <Laufbedingung>; <Iteration>) {
  <Anweisung>*
}
```

Hinter dem Schlüsselwort `for` steht innerhalb eines *runden Klammerpaars* eine Folge von drei durch Semikoli getrennte Sektionen mit der Bedeutung:

<Initialisierung>
: Vorbelegung von Variablen, die den Schleifenablauf steuern *(Laufvariablen)*.

<Laufbedingung>
: Besitzt diese den Wert `true`, wird der nachfolgende Block von Anweisungen ausgeführt, sonst nicht.

<Iteration>
: Hier werden die Laufvariablen aktualisiert.

Die Arbeitsweise:

1. Zum Schleifeneintritt werden die Anweisungen in der <Initialisierung>-Sektion ausgeführt.
2. Es wird die <Laufbedingung> geprüft.
3. Liefert die <Laufbedingung> den Wert `true`, werden die Anweisungen im Schleifenblock ausgeführt, sonst wird die Schleife abgebrochen.
4. Es werden die Ausdrücke in der <Iteration> - Sektion ausgewertet.
5. Weiter mit Schritt 2.

Die `for`-Schleife ist sehr leistungsfähig und die weitaus häufigst benutzte Wiederholungsanweisung.

Beispiel

```
var sum = 0;
for (var i=1, max=100; i<=max; i++)
  sum += i*i;
```

for ... in - Schleife

Syntax

```
for (<Eigenschaft> in <Objekt>) {
  <Anweisung>*
}
```

Hinter dem Schlüsselwort `for` steht innerhalb eines runden Klammerpaars eine Variable(ndefinition) gefolgt vom reservierten Wort `in` und einer Objektvariablen.

Während der Schleifenausführung nimmt die Variable *<Eigenschaft>* nacheinander alle Eigenschaftsbezeichnungen des Objekts *<Objekt>* an.

Beispiel

```
var adresse = { ort="lemgo", plz="32657",
                str="Liebigstr.", nr=87 };

for (var e in adresse)
  document.write(e + "=" + adresse[e] + "\n");
```

break und continue - Anweisung

break

Die `break` - Anweisung kann neben ihrem Einsatz in der `switch...case` - Anweisung in allen Schleifen verwendet werden. Sie veranlasst den unmittelbaren Abbruch der Schleife.

Beispiel

```
var eingabe;

for (;;) { // for ever
  eingabe = prompt("Eingabe", "");
  if (eingabe == "")
    break;
  document.write(eingabe + "\n");
}
```

continue

Die `continue` kommt ausschliesslich in Schleifen zum Einsatz. Sie veranlasst unmittelbar eine neue Überprüfung der Laufbedingung.

Beispiel

```
var zahl;

while (zahl != 0) {
  zahl = parseFloatprompt("Zahl bitte!", "0"));
  if (isNaN(zahl)) {
    alert("Bitte gültige Zahl eingeben!")
    continue;
  }
  document.write(zahl + "\n");
}
```

Verschachtelte Schleifen

Verschachtelte Schleifen haben eine grosse Bedeutung bei der Behandlung von zwei- und mehrdimensionalen Strukturen.

```
for (var i=0; i<n; i++) {        // Zeilensteuerung
   for (var j=0; j<m; j++) {     // Spaltensteuerung
      // ...                     // Zelleninhalt
   }
}
```

Beispiel: Ausgabe eines ASCII-Musters

```
-------
      /
     /
    /
   /
  /
-------
```

```
var n = 7, out = document.write;

for (var i=0; i<n; i++) {
   if (i==0 || i==n-1)
      for (var j=0; j<n; j++)
         out("-");
   else {
      for (var j=0; j<n-i-1; j++)
         out(" ");
      out("/");
   }
   out("\n");
}
```