Inhalt:

Kontrollstrukturen:

Kontrollstrukturen sind klassische Anwendungen von logischen oder vergleichenden Operatoren und spezielle Anweisungsschritte in einer Programmiersprache, mit denen ein Programmierer Entscheidungen über den weiteren Ablauf eines Programms oder Skripts vorgeben kann, wenn bestimmte Bedingungen eintreten. In JavaScript hat man die gleichen Kontrollstrukturen zur Verfügung, die es in fast allen Programmiersprachen in gleicher oder ähnlicher Form gibt. Es gibt drei Arten von Kontrollflussanweisungen:

1. Entscheidungsanweisungen. Diese suchen auf Grund einer Bedingung einen Programmfluss heraus.
2. Schleifen bzw. Iterationsanweisungen. Diese wiederholen eine bestimmte Anzahl an Anweisungen.
3. Sprunganweisungen. Diese verlassen eine Struktur.

1)Entscheidungsanweisungen

Ein Programm kann abhängig von bestimmten Bedingungen unterschiedliche Teile eines Programms durchlaufen. Man sagt auch: Es verzweigt sich. Verzweigungen gehören zu den wichtigen Kontrollstrukturen.

Verzweigungen mit »if ... else«

Verzweigungen werden mit Hilfe der Anweisung if ... else erzeugt. Nach dem if steht in runden Klammern eine Bedingung, die entweder erfüllt oder nicht erfüllt ist.

- Falls sie erfüllt ist, wird die folgende Anweisung oder der folgende Block von Anweisungen ausgeführt. Einen Block von Anweisungen erkennt man an den geschweiften Klammern { ... }.
- Falls die Bedingung nicht erfüllt ist, wird die Anweisung oder der Block von Anweisungen nach dem else ausgeführt, falls vorhanden.

```
if (Bedingung) {
      ...Anweisungen A
      } else {
      ...alternative Anweisungen
      }
```

Bedingungen werden mit Hilfe von Wahrheitswerten gebildet. Vergleichsoperatoren haben Wahrheitswerte als Ergebnis.

- Der Operator > steht für größer als;
- der Operator < steht für kleiner als;
- >= bedeutet größer als oder gleich;

- <= bedeutet kleiner als oder gleich.
- == prüft auf Gleichheit
- != auf Ungleichheit

Es gibt die if-Bedingung mit oder ohne nachgestellten else-Zweig. Wenn er da ist, kann darin wie oben eine Folge von alternativen Anweisungen stehen, die immer dann ausgeführt werden, wenn die Bedingung den Wert false liefert. Wenn bereits der if-Zweig ausgeführt wurde, wird der else-Zweig nicht ausgeführt.
Wenn der else-Zweig fehlt, bewirkt die gesamte Struktur nur etwas, wenn die Bedingung den Wert true liefert. Im Falle des Werts false passiert gar nichts.

Achtung:
Achte darauf, nach der Bedingung hinter einem if oder nach einem else **kein Semikolon** zu schreiben. Es sollte also

- nicht aussehen wie folgt: if(a > b);
- oder else;.

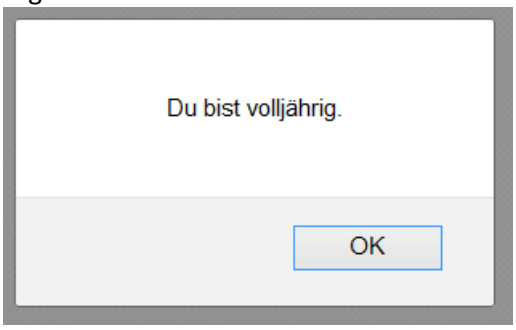
Dies würde dazu führen, dass die Verzweigung unmittelbar endet und die folgenden Anweisungen immer ausgeführt würden. Es könnte auch passieren, dass gar keine Ausgabe erfolgt, weil das else ohne sein zugehöriges if erzeugt wird. Diese typischen Einsteigerfehler sind schwer zu finden.

Beispiel:
Bei Volljährigkeit soll eine entsprechende Meldung ausgegeben werden.

```
 7 <body>
 8     <script>
 9     var alter = 20;
10
11     if (alter >= 18) {
12         alert("Du bist volljährig.");
13     } else {
14         alert("Du bist zu jung und darfst nicht hinein.");
15     }
16     </script>
17 </body>
```

Speicher unter „3_js_alter.html“

Ergebnis:

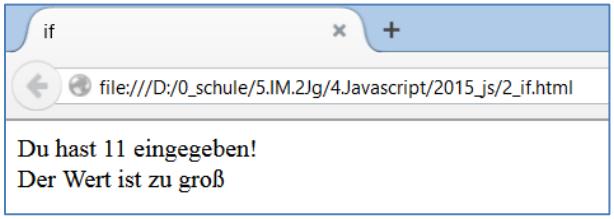


Beispiel:
In dem Beispiel wird mit der prompt()-Methode vom Anwender eine Zahl zwischen 0 und 9 entgegengenommen und dieser Wert der Variablen „erg“ zugewiesen.

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <script>
        var erg = prompt("Gib eine Zahl zw. 0 und 9 ein");
        document.write("Du hast " + erg + " eingegeben! <br>");
        if (erg > 9){
            document.write("Der Wert ist zu groß");
        } else {
            document.write("Alles klar");
        }
    </script>
```

Speichern unter „3_js_erg.html“.

Ergebnis:



Hausübung Body-Mass-Index:

Errechne den Body-Mass-Index mit folgender Berechnung:

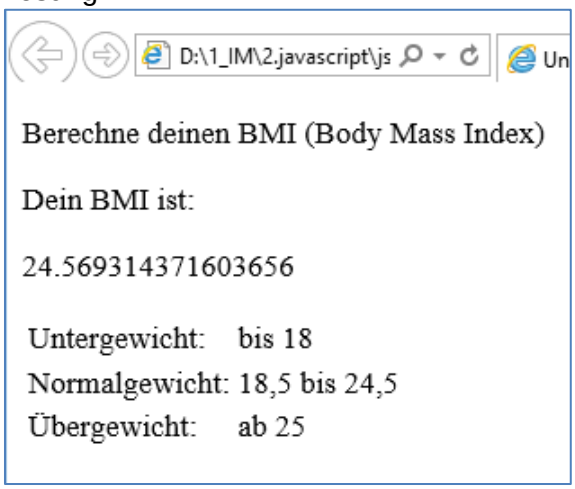
bmi = gewicht / (groesse * groesse) * 10000

- Die Größe und das Gewicht sollen jeweils vom Kunden mit Hilfe einer „prompt()“ – Anweisung eingegeben werden können.
- Darunter soll mit einer <table> eine kurze Erklärung erfolgen. Mit <tr> und <td>.

Es soll eine „if“-Anweisung regeln, ob der Kunde eine Größe eingibt:

```
if(groesse == 0) {
    alert("Körpergröße angeben!")
  } else {
    document.write(bmi);
  }
```

Lösung:



Übung: Was passiert hier?

```
// Zeichenketten
var land = "Spanien";
if(land == "Spanien"){
document.write("<p>Land ist Spanien<\/p>");}
else {
document.write("<p>Land ist nicht Spanien<\/p>");}
```

2)Bestätigung anfordern mit confirm()

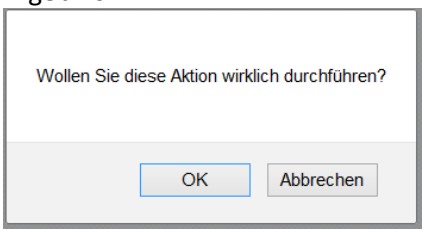
Die Funktion confirm() stellt im Zusammenhang mit einer Verzweigung eine weitere Möglichkeit zur Kommunikation mit dem Benutzer dar. Nach dem Aufruf der Methode erscheint ein Dialogfeld mit einer Frage sowie zwei Buttons mit der Aufschrift **OK bzw. Abbrechen**. Damit kann man eine einfache Ja – oder Nein Frage stellen.

Beispiel:

```
<script>
        var antwort = confirm("Wollen Sie diese Aktion wirklich durchführen?");
        if(antwort) {
        alert("Diese Aktion wird durchgeführt");
} else {
        alert("Diese Aktion wird nicht durchgeführt");}
</script>
```

Falls der Benutzer den Button OK betätigt, wird true zurückgeliefert. Dies ist für die Verzweigung bereits ausreichend, da ein Wahrheitswert geliefert wird. Man benötigt keinen Vergleichsoperator. Die Abfrage if(antwort == true) würde zum selben Ergebnis führen, ist aber unnötig lang.
Nach Betätigung des Buttons Abbrechen wird false geliefert. Dann wird der else-Zweig durchlaufen.

Ergebnis:



Message-Box mit OK/Abbrechen-Button:

werden mit dem Befehl confirm(„Text“) aufgerufen. Klickt man auf OK, wird True zurückgegeben, bei Abbrechen False.

```
<script>
a = confirm(„Das Wetter ist schön.“);
document.write(a);
</script>
```

3)Programmteile wiederholen - Schleifen

Neben den Verzweigungen gehören die Schleifen zu den wichtigen Kontrollstrukturen. Viele Vorgänge, die sich auf dieselbe oder recht ähnliche Weise wiederholen, können mit Hilfe von Schleifen effizient programmiert werden. Sie nutzen die Fähigkeit eines Rechners, sehr viele Schritte in sehr kurzer Zeit durchzuführen.

Schleifen mit „for“

Eine Schleife mit „for“ nutzt man, wenn man als Entwickler weiß, **wie oft ein Programmteil** wiederholt werden soll. Sie ist auch dann sinnvoll, wenn ein Programmteil für eine regelmäßige Abfolge von Zahlen, die von einem Startwert bis zu einem Endwert laufen, wiederholt werden soll. Bei for-Schleifen wird zur Steuerung meist eine Variable eingesetzt. Sie wird Schleifenvariable genannt.

Der Aufbau einer for-Schleife gliedert sich in drei Teile (Parameter):

- eine Anweisung für den Startwert der Schleifenvariablen
- eine Bedingung, die während des gesamten Ablaufs der Schleife für die Schleifenvariable gelten muss
- eine Anweisung für die Änderung

bzw. man kann auch sagen:
for mit zwei runden Klammern und dann die geschweiften Klammern für den Code. Die runde Klammer besteht aus drei Bestandteilen, zwischen denen jeweils ein Semikolon steht
for (;;)

- am Beginn die Initialisierung
- an zweiter Stelle steht eine Bedingung
- es folgt eine Aktion

```
<script>
for (initialisierung; bedingung; aktion) {
    //Code
}
</script>
```

Was passiert nun:
Ganz am Anfang wird die Initialisierung ausgeführt. Dann Schritt 2: Wird bei der Überprüfung der Bedingung ein „wahr“ also ein „true“ ausgegeben, wird der Code in der geschweiften Klammer ausgeführt. Nachdem dieser ausgeführt wurde, wird die Aktion ausgeführt. Dann wiederholen sich die Bedingung, bei „true“ der Code und die Aktion usw.

So könnte die Reihenfolge aussehen: von 1 bis 7

```
<script>
for (initialisierung (1); bedingung (2)(5); aktion (4)(7)) {
    //Code (3)(6)
}
</script>
```

Tipp: Eigentlich sollte man die Variable mit “let“ definieren und nicht mit „var“. Der Vorteil dabei ist nämlich, dass „let“ nur in der Schleife gültig ist und nicht außerhalb.

Beispiel:

Eigentlich hat es sich eingebürgert, die Variable für die Initialisierung „i“ zu nennen, daher:

1. Die Zahl wird mit „1“ festgelegt: **let i = 1;**
2. Damit nun das Hinaufzählen um eins nicht endlos läuft, kommt die Bedingung nun ins Spiel: die „Bedingung“ steht an zweiter Stelle, damit sie vorher überprüft wird, bevor der Teil 3, nämlich die „Aktion“ ausgeführt wird: **i <= 10;**
   Hier wird sozusagen die Gültigkeit angegeben.
3. Die Aktion soll sein, dass die Zahl „i“ um eins erhöht wird: **i++**

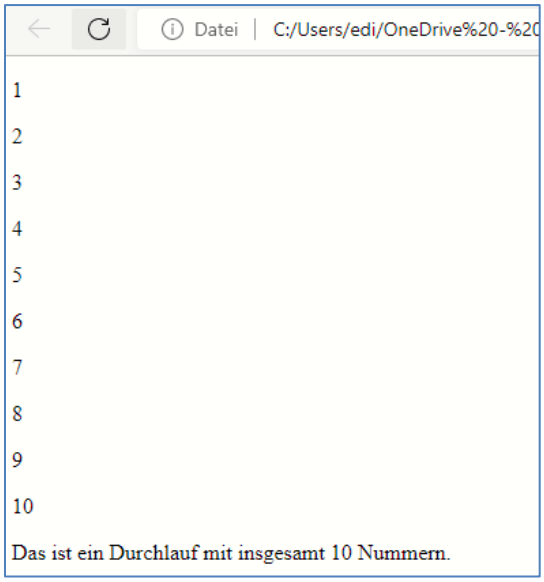
Dann müssen die Daten ausgegeben werden: mit „alert“ nicht sinnvoll, da müsste man 10mal das Fenster schließen. Daher wähle „document.write“.

Damit die Zahlen ausgegeben werden, muss man **IN DER FOR-Schleife die Ausgabe** schreiben.

Die Ausgabe des Textes erfolgt außerhalb der Schleife!

```
 8 <body>
 9   <script>
10     for (var i = 1; i <= 10; i++) {
11         document.write("<p>" + i + "</p>");    //Ausgabe aller Zahlen
12
13         text = "Das ist ein Durchlauf mit insgesamt " + i  + " Nummern.";
14     }
15     document.write(text);
16   </script>
17 </body>
18 </html>
```

Ergebnis:



Weiteres Beispiel:

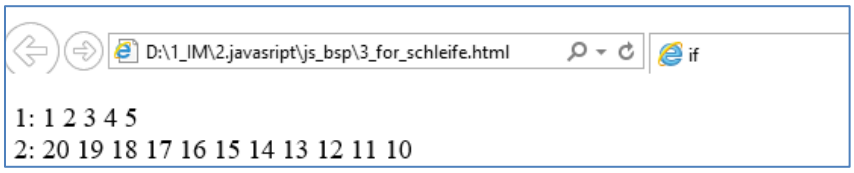
```
<body>
 <script>
        let i;
        // 1: Aufwärts
        document.write("<p>1: ");
        for(i=1; i<=5; i++)
        document.write(i + " ");

        // 2: Abwärts
        document.write("<br>2: ");
        for(i=20; i>=10; i--)
        document.write(i + " ");
 </script>
</body>
```

Erklärung:
Die Variable i soll zunächst als Schleifenvariable dienen.

\\1:
In der ersten Schleife erhält i den Startwert 1. Als Bedingung für die gesamte Schleife gilt: Sie läuft, solange i kleiner als 5 oder gleich 5 ist. Nach jedem Durchlauf wird der Wert von i mit Hilfe des Zuweisungsoperators ++ um 1 erhöht. Damit ergibt sich eine Zahlenfolge von 1 bis 5, in Schritten von 1. Startwert, Bedingung und Änderung werden jeweils durch Semikolon voneinander getrennt.
Innerhalb der ersten Schleife steht nur eine Anweisung: Die Ausgabe der Schleifenvariablen.

\\2:
Die zweite Schleife läuft abwärts, im Gegensatz zur ersten Schleife. Sie startet bei 20 und läuft,

solange der Wert von i größer als 10 oder gleich 10 ist. Nach jedem Durchlauf wird der Wert von i um 1 vermindert. Damit ergibt sich die Zahlenfolge 20, 19, 18 ... 10.
Die verschiedenen Teile einer Schleife müssen aufeinander abgestimmt sein. Eine Schleife mit for(i=10; i>=5; i++) läuft endlos. Eine Schleife mit for(i=10; i<=5; i--) läuft nie. Solche Schleifen sollten Sie natürlich vermeiden.



Übung mit einem „counter“

Solange die Variable kleiner ist als der Wert (hier 15), wird der Code innerhalb der Schleife ausgeführt. Es soll der untenstehende Satz solange ausgegeben werden, bis der Wert vom „counter“ erreicht wird.
Erstelle das File „3.for_counter.html“:

```
<!doctype html>
<head>
<meta charset="UTF-8">
<title>counter</title>
</head>
<body>
  <script>
    for(var counter = 0; counter <=15; counter++) {
        document.write("Der Counter liegt bei " + counter + "<br>");
    }
  </script>
</body>
</html>
```

Ergebnis:

4)Die switch-case-Fallunterscheidung

In JavaScript gibt es auch die Fallunterscheidung über switch-case. Ein Vorteil gegenüber if-else ist, dass man auf einfache und übersichtliche Weise mehrere Falle unterscheiden kann. Das ist zwar auch mit if-else moglich, indem man mehrere if-Abfragen hintereinander schreibt und ggf. mit vorangestelltem else verknüpft. Jedoch erscheint die Fallunterscheidung mit switch-case vielen Programmierern als eleganter. Die Syntax einer switch-case-Anweisung sieht formal folgendermaßen aus:

```
switch(auswah]){
        case 1: ausgabe = “one“;
        break;
        case 2: ausgabe = “two“;
        break;
        ...
        default: ...
}
```

1. Mit dem Schlüsselwort „switch“ leitet man die Fallunterscheidung ein.
2. Als Argument wird in runden Klammern eingeschlossen eine Testvariable oder ein Ausdruck angegeben, für dessen aktuellen Wert man die Fallunterscheidung durchführt.
3. Die einzelnen Fälle, zwischen denen unterschieden werden soll, werden untereinander aufgelistet und innerhalb geschweifter Klammern als Blöcke notiert. Jeden einzelnen Fall leitet man mit dem Schlüsselwort case ein, gefolgt von der konkreten Angabe des Werts, auf den man prüfen will. Beendet wird die Beschreibung mit einem Doppelpunkt, um anzuzeigen, dass die Alternative beginnt.
4. Der Code für die Alternative folgt dem Doppelpunkt. Dieser Code wird nur ausgeführt, wenn der Ausdruck übereinstimmt.
5. Die Anweisung break, die am Ende jedes Blocks notiert ist, ist eine optionale Sprung-anweisung. Es wird aus der switch-Struktur herausgesprungen, sobald die case-Alternative ausgewertet worden ist. **Beim letzten Block in der switch-case-Anweisung ist das break immer überflüssig**.
6. Der Fall default ist dafür da, eine Standardreaktion für den Fall zu definieren, dass keiner der von uns definierten Falle zutrifft. Dieser Zweig ist optional und kann entfallen, wenn es keinen Vorgabefall geben soll.

Der gravierende **Nachteil** der Programmflusssteuerung über switch und case ist, dass nur diskrete Fälle unterschieden werden können. Also ist nur der Vergleich auf exakte Gleichheit mit einem Wert möglich. Es gibt keinen Vergleich auf größer oder kleiner, wie es bei einer if-else-Konstruktion möglich ist.