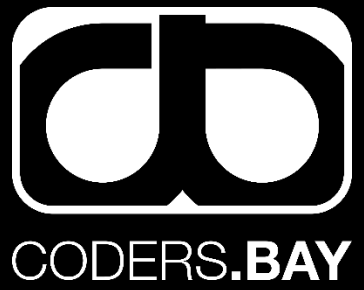


JavaScript Teil 1

JAVASCRIPT

- **Es ist eine richtige Programmiersprache.** Es wird gerne behauptet, dass es sich bei JavaScript um keine „echte“ Programmiersprache handelt, weil Funktionen fehlen. Dabei wurde bewusst auf diese Funktionen verzichtet (ganz besonders jene, bei denen es um die Fähigkeit geht, mit dem Dateisystem zu kommunizieren), um JavaScript sicher zu halten. JavaScript ist eine echte Sprache, und sie bildet eine sehr gute Ausgangsbasis für den Einstieg in die Programmierung
- **Es handelt sich nicht um Java.** Es gibt eine bekannte Programmiersprache, die nur Java heißt und die ebenfalls für die Webprogrammierung verwendet wird. Bei JavaScript und Java handelt es sich aber um zwei völlig unterschiedliche Sprachen, bei denen sich nur die Namen ähneln.
- **JavaScript ist eine Skriptsprache.** JavaScript ist relativ einfach zu erlernen. Sie verlangt lange nicht so viel wie andere sprachen und pflegt eine relativ entspannte Sicht auf viele Dinge. Da JavaScript eine Programm Sprache ist, gibt es aber natürlich auch hier Regeln, die befolgt werden müssen, aber Skriptsprachen sind doch nachsichtiger als die großen Programmiersprachen.

WICHTIG

Es ist hilfreich, beim Ausführen von Webseiten mit Skripten, immer einen Blick auf die Konsole zu werfen, da dort wichtige Fehlermeldungen erscheinen können! (Entwicklertool im Browser!!)

Das erste Skript

- Die erste Seite enthält ein einfaches JavaScript-Programm. Das den Satz „Hallo, Welt!“ in einem Element erscheinen lässt, das Dialogfeld heißt
- JavaScript-Code wird über das Tag `<script>` auf einer Webseite eingebudnen. Der Code selbst wird innerhalb des Tag-Paares `<script></script>` untergebracht. Im Tag `<script>` gibt es das Attribut, type, das normalerweise den Inhalt `text/javascript` hat
- In HTML5 kann die Angabe des MIME-*Type* `type="text/javascript"` weggelassen werden - in älteren HTML-Varianten ist die Angabe *notwendig*

`//<![CDATA[`

Die Umrahmung unseres Scripts mit `/*<![CDATA[*/` und `/*]]>*/` soll es nur sicherstellen, dass Sonderzeichen in dem Skript auch dann verwendet werden können, wenn das enthaltende Dokument als *XHTML*-Dokument ausgeliefert wird. Da viele Web-Autoren ihre Dokumente *nicht* als *XHTML*-Dokumente ausliefern, kann diese Umrahmung aber meist entfallen.

`//]]>`

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hallo Welt</title>
  </head>
  <body>
    <script type="text/javascript">
      //<![CDATA[
        // Hallo, Welt!
        alert("Hallo, Welt!");
      //]]>
    </script>
  </body>
</html>
```

KOMMENTARE

- Einzeilige Kommentare fangen mit `//` an
- Mehrzeilige Kommentare funktionieren wie in CSS mit `/* ich bin ein Kommentar */`

Die Methode alert();

- Es gibt mehrere Möglichkeiten der Datenausgabe
- Die Methode `alert()` (vom Englischen to alert = alarmieren, warnen, melden) erzeigt ein kleines Dialogfeld, in dem Text auf dem Bildschirm angezeigt wird. Dieses Meldungsfenster ist ein Beispiel für ein modales Dialogfeld. Sie unterbrechen den Ablauf eines Programms, bis der Benutzer sie beachtet und beantwortet. Im Programm geschieht so lange nichts und der Benutzer kann auf der Seite nichts anderes tun.

Das Semikolon

- Alle JavaScript-Anweisungen müssen mit einem Semikolon (;) abgeschlossen werden. Das Semikolon erfüllt in den meisten Computersprachen die gleiche Aufgabe wie ein Punkt ein einem Satz. Er schließt einen logischen Gedanken ab.
- In JavaScript funktioniert normalerweise auch dann fehlerfrei, wenn du auf das Semikolon verzichtest, du solltest es aber auf jeden Fall hinzufügen, weil deine Gedankengänge mit Semikolons besser verdeutlich werden können und in anderen Sprachen wie PHP sie zwingend erforderlich sind. Gewöhn dir deshalb besser derartige Schludrigkeiten gar nicht erst an.

VARIABLEN

- Computerprogramme erhalten ihre Leistungsfähigkeit durch das Arbeiten mit Daten.
- Dieses Programm ermöglicht Interaktionen mit dem Benutzer. Er kann einen Namen eingeben, der gespeichert wird und dann als Begrüßung angezeigt wird.
- Durch das Pluszeichen können beliebige Anzahl an Textstücken miteinander verknüpft werden. Die Verwendung des Pluszeichens zum Kombinieren von Text wird auch Verkettung genannt.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hallo Welt</title>
  </head>
  <body>
    <script>
      var person = "";
      person = prompt("Wie heißen Sie");
      // "Hallo, " ist ein literaler Wert, weil er zwischen
      // Anführungszeichen steht
      // Der Ausdruck person ist der Name einer Variable,
      // er wird nicht von Anführungszeichen umschlossen
      // "!" ist wieder ein literaler Wert
      alert("Hallo " + person + "!");
    </script>
  </body>
</html>
```

VARIABLEN

- Variablen sind nicht mehr, als Stellen im Arbeitsspeicher, in denen Daten vorübergehend gespeichert werden können. Variablen besitzen normalerweise folgende Merkmale:
  - **Die Anweisung `var`** zeigt an, dass eine Variabel erstellt werden soll
  - **Einen Namen.** Wenn du eine Variable anlegst, musst du ihr einen Namen geben
  - **Einen Anfangswert.** Es ist möglich, einer Variable einen Anfangswert zuzuweisen.
  - **Einen Datentyp.** JavaScript kann den Datentyp einer Variablen zwar automatisch erkennen, aber man sollte zumindest wissen, welche Art von Daten (welcher Datentyp) in einer Variable erwartet wird.
- Variablen sind interessanter, wenn sie Daten enthalten. JavaScript kennt mit Eingabeaufforderungen ein einfaches Werkzeug, das im Computerbereich oft auch prompt genannt wird. Es erlaubt schlicht, Fragen zu stellen und die Antwort in Variablen abzulegen.
- Die Anweisung **`prompt();`** macht mehrere interessante Dinge:
  - **Sie zeigt ein Dialogfeld an.** Die Methode erzeugt ein modales Dialogfeld, das dem durch alert erzeugten ähnelt.
  - **Sie stellt eine Frage**. Die Anweisung prompt erwartet, dass dem Benutzer eine Frage gestellt wird
  - **Sie sorgt dafür, dass eine Antwort eingegeben werden kann.** Im Dialogfeld gibt es eine Möglichkeit, eine beliebige Antwort einzugeben, und Schaltflächen, die darauf hinweisen, dass die Eingabe angeschlossen ist oder Benutzer die Operation abbrechen wollen
  - **Sie leiten die Daten an eine Variable weiter**. Die Anweisung soll die vom Benutzer eingegebenen Daten aufbewahren. Deshalb geht diese Anweisung auch fast immer mit einer Variablen einer. Ach der Ausführung des Codes enthält die Variable dein eingegebenen Wert.

Das String-Objekt

- Der Begriff String, der dem deutschen Begriff Kette entspricht, hat seinen Ursprung in der Art, wie Text im Arbeitsspeicher eines Computers gespeichert wird. Jedes Zeichen wird im Arbeitsspeicher in einer eigenen Zelle abgelegt, und die Zeichen eines Wortes oder einer Phrase erinnerten die frühen Programmierer an die Kugeln einer Kette.

OOP Einführung

- JavaScript stützt sich (wie auch viele andere Programmiersprachen) auf das Modell der objektorientierten Programmierung (OOP). Diese Art der Programmierung erlaubt den Zugriff auf einige von Haus aus sehr leistungsfähige Objekte.
- *Objekte* werden verwendet, um komplizierte Dinge mit vielen Merkmalen zu beschreiben. Beispiel Kuh:
  Es funktioniert nun noch nicht einmal ansatzweise, eine auch nur halbwegs vernünftige Beschreibung einer Kuh in eine ganzzahlige Variable (einer sogenannten Integervariable) zu packen.

OOP Einführung

- Entsprechend können Objekte, wie bspw. Das kuh-Objekt, in objektorientierter Umgebung normalerweise auch durch zahlreiche Merkmale beschreiben werden. Objekte besitzen:
  - **Eigenschaften**: Merkmale des Objektes wie Rasse und Alter
  - **Methoden**: Dinge, die die Objekte machen können , wie zum Beispiel `Muhh()` und `gebeMilch()`
  - **Ereignisse**: Bei diesen handelt es sich um Auslöser (Trigger), auf die das Objekt reagiert wie `InsektAufNase()`
- Derartige Begriffe werden dann beschreiben, wenn sie gebraucht werden. Naturgemäß müssen und werden aber nicht alle Objekte die genannten Merkmale unterstützen.
- Die meisten JavaScript-Variablentypen sind eigentlich Objekte, und die meisten JavaScript-Objekte verfügen über Eigenschaften und Methoden. Viele dieser Typen besitzen zudem auch noch eigene Ereignisbehandlungsroutinen, die eigens für den geeigneten Umgang mit den Ereignissen existieren, mit denen es ein Objekttyp zu tun bekommen kann.

.. Es wird darüber gestritten, ob JavaScript wirklich eine objektorientierte Sprache ist oder nicht. Viele reden hier von einer objektbasierenden Sprache..

Länge eines Strings ermitteln

- Wenn einer Variable ein Text zugewiesen wird, behandelt JavaScript die Variable automatisch als String-Objekt. Sofort werden die einem String-Objekt entsprechenden Merkmale übernommen. String-Objekte besitzen eine Reihe von Eigenschaften und Methoden, unter anderem die Eigenschaft **`length`** (Länge). Diese Eigenschaft gibt die Anzahl der Zeichen im String zurück.
- Dieser Code verwendet die Eigenschaft **`length`** so, als ob es sich um eine Art Untervariable handelt. **`person`** ist eine variabel – und **`person.length`** ist die Eigenschaft **`length`** der Variable **`person`**. In JavaScript werden Objekte und Eigenschaften durch einen Punkt (ohne Leerzeichen dazwischen) miteinander verbunden.

```
<script>
  var person = prompt("Wie heißen Sie");
  var laenge = person.length;
  alert("Der Name " + person + " ist " + laenge + " Zeichen lang.");
</script>
```

Text mit String-Methoden ändern

- Objekte weisen nicht nur Eigenschaften auf, sondern verfügen auch über *Methoden*. Bei Methoden handelt es sich um Dinge, die Objekte machen können, so gesehen also um Mittel zum Zweck. String-Objekte verfügen unter anderem über folgende Methoden:
  - `toUpperCase()`: Erstellt eine Kopie des Strings in Großbuchstaben (Upper Case)
  - `toLowerCase()`: Erstellt eine Kopie des Strings in Kleinbuchstaben (Lower Case)
    - → `toUpperCase()` *und* `toLowerCase()`*:* Nimmt den Wert der Variablen und wandelt in auf die vorgegebene Weise um. Wird häufig verwendet, wen Wert darauf gelegt wird, den Inhalt einer Variablen in Gro- oder Kleinbuchstaben geliefert zu bekommen.
  - `substring()`: Gibt einen bestimmten Teil des Strings zurück. Achtung: das erste Zeichen in einem String ist an der Position 0 (null). `substring(1,3)` gibt die Positionen 2, 3, 4 eines Strings aus!
    - → *Substring(Anfang, Ende)*: Liefert die Teilzeichenkette der Variablen vom beginnenden Zeichen bis zum Zeichen an der Endposition.
  - `indexOf()`: Ermittelt, ob eine Zeichenfolge (ein String) Bestandteil einer anderen Zeichenfolge ist
    - → `indexOf(Teilzeichenkette)`: Gibt die Position der Teilzeichenkette (engl. Substring) in der Variablen an. Wenn die Variable die Teilzeichenkette nicht enthält, liefert die Variable den Wert -1. Achtung: die Zählung beginnt hier mit 0!

Text mit String-Methoden ändern

Methoden werden wie Eigenschaften über einen Punkt an das Objekt gehängt. Methoden haben als besonderes Merkmal ein Paar Klammern, das manchmal Informationen enthält, die Parameter genannt werden. Bei Parametern handelt es sich um Informationen, die an die Methode übergeben werden, damit diese ihren Job erledigen kann. Einige Methoden benötigen zwingend Parameter, andere nicht.

```
<script>
  var text = prompt("Geben Sie bitte einen Text ein.");

  alert("Dies ist Ihr Text in Großbuchstaben: " + text.toUpperCase());
  alert("Un nun in Kleinnbuchstaben ... " + text.toLowerCase());
  alert("Das erste 'a' befindet sich an Position... " + text.indexOf("a"));
  alert("Die ersten drei Buchstaben sind... " + text.substring(0, 3));
</script>
```

Zahlen addieren

- JavaScript nimmt es nicht sonderlich genau mit der Frage, ob eine Variable Text oder eine Zahl enthält.
- Dieses Programm enthält drei Variablen. Die Variable `x` und `y` haben jeweils einen Zahlenwert der in der dritten Variablen addiert wird. Die letzte Zeile zeigt das Ergebnis an.
- Diese Dinge sind hier wichtig:
  - **Man kann einer Variablen einen Wert zuweisen.** Das Gleichheitszeichen bedeutet daher so viel wie „der Variablen x wird der Wert 5 zugewiesen“.
  - **Numerische Werte werden nicht in Anführungszeichen geschrieben.** Wenn es um einen literalen Textwert geht, steht dieser immer in Anführungszeichen. Numerische Werte (wie der Wert 5) werden nicht in Anführungszeichen gesetzt
  - **Du kannst numerische Werte addieren.** Da sowohl `x` als auch `y` einen numerischen Wert enthält, kannst du beide addieren
  - **Das Ergebnis einer Operation kann in einer Variablen abgelegt werden.** Das Ergebnis der Rechenoperation `x + y` wird in der Variablen summe abgelegt.
  - **Alles funktioniert wie erwartet.** Das Programm verhält sich wie gewünscht. Das ist aber nicht immer der Fall.

```
<script>
  var x = 5;
  var y = 3;
  var summe = x + y;
  alert(x + " plus " + y + " gleich " + summe);
</script>
```

Zahlen aus Benutzereingaben addieren

- Alle in einem Computer gespeicherten Daten, werden als Haufen von nullen und Einsen und damit als Binärdaten abgelegt. Der Wert 01000001 kann dann zum Beispiel die Zahl 65 oder der Buchstabe A sein. Derselbe binäre Wert kann also etwas ganz Unterschiedliches bedeuten, wenn er als Zahl, als Farbe oder als Teil eines Textes interpretiert wird.
- In JavaScript kann in eine Variable jegliche Art von Daten hineingepackt werden, und der Datentyp verändert werden. Eine Variable kann hier eine Ganzzahl und zu einem anderen Zeitpunkt Text enthalten. JavaScript bestimmt aus dem Kontext heraus, wie die Daten in einer bestimmten Variablen interpretiert werden.
  Folgende Kategorien kennt JavaScript:
  - **Ganzzahl:** Bei Ganzzahlen (oder auf Computerdeutsch Integerwert) handelt es sich um ganze Zahlen (ohne Dezimaltrennzeichen). Diese können positiv oder negativ sein.
  - **Fließkommazahlen:** Eine Fließkommazahl, die eigentlich Fließpunktzahl heißen müsste, hat ein Dezimaltrennzeichen wie in 3.14 und können auch negativ sein.
    Als Dezimaltrennzeichen wir ein Punkt verwendet!
  - **Boolesch:** Die Boolesche Werte sind Wahrheitswerte, die `true` (wahr) und `false` (falsch) sein können
  - **String:** In Programmiersprachen wird Text als String oder Zeichenkette bezeichnet. String-werte werden in JavaScript normalerweise in Anführungszeichen umschlossen
    Sollen Anführungszeichen in Strings verwendet werden, kann man diese im Quelltext über die sogenannte Escape-Sequenz `\"` von JavaScript erzeugen.
  - **Arrays und Objekte:** Komplexe Datentypen (später mehr)

Mit dem ECMAScript 5-Standard zieht Disziplin in JavaScript ein. Nun kennt JavaScript den „strengen Modus“ (strict mode), mit dem eine standardisierte und um Fehlerquellen bereinigte Untermenge der Sprache aktiviert werden kann. Den restriktiven Modus kann man mit `"use strict"` im Skript selbst aktivieren. Ältere Browser alten die Aktivierung des strengen Modus für nichts weiter als einen String, mit dem sie nichts anzufangen wissen.

Zahlen aus Benutzereingaben addieren

- In diesem Code sind **x** und **y** Textvariablen. Die Zeile **ergebnis = x + y** wird als „verknüpfe x mit y“ interpretiert, und das Ergebnis ist „Hallo, Sie da!“

```
<!-- BSP 6 -->
<script>
  // 1.
  var x = "Hallo ";
  var y = "Sie da!";
  var summe = x + y;
  alert(summe);
  // 2.
  var x = 5;
  var y = 3;
  var summe = x + y;
  alert(summe);
</script>
```

- In dem zweiten Beispiel ist der Code fast identisch, jedoch sind **x** und **y** Zahlen. Der Operator Plus übernimmt zwei unterschiedliche Aufgaben. Wenn er von Zahlen umgeben ist, addiert er. Geht es um Text, verknüpft er automatisch!
- Wenn der Benutzer Daten in die Dialogfelder eingibt, behandelt JavaScript diese Daten als Text. Es findet somit keine Berechnung statt.

```
<!-- BSP 7 -->
<script>
  var x = prompt("Erste Zahl:");
  var y = prompt("Zweite Zahl:");
  var summe = x + y;
  alert(x + " plus " + y + " gleich " + summe);
</script>
```

VARIABLENTYPEN ÄNDERN

Die Umwandlungsfunktionen sind leistungsstark, werden aber nur benötigt, wenn die automatische Umwandlung für Probleme sorgt.

- **`parseInt()`**: Wird verwendet, um Text in eine Ganzzahl (Integer) umzuwandeln. Wenn du einen Textwert in Anführungszeichen setzt, gibt die Funktion einen ganzzahligen Wert zurück. Wenn die Zeichenkette (der String) eine Fließkommawert enthält, wird ein ganzzahliger Wert (ein Integerwert) zurückgegeben
- **`parseFloat()`**: Wandelt Text in einen Fließkommawert um
- **`toString()`**: Nimmt jeden beliebigen Variablentyp und wandelt ihn in einen String um. Diese Funktion wird normalerweise nicht benötigt, weil sie bei Bedarf automatisch aufgerufen wird
- **`eval()`**: Hierbei handelt es sich um eine besondere Methode, die als Eingabe einen String akzeptiert. Sie versucht dann, den String als JavaScript-Code auszuwerten und die Ausgabe zurückzugeben. Damit können variablen umgewandelt werden oder einfache Rechenoperationen durchgeführt werden.
- **`Math.ceil()`**: Eine von mehreren Methoden, um eine Fließkommazahl in eine Ganzzahl umzuwandeln. Diese Technik rundet immer auf.
- **`Math.floor()`**: Rundet eine Fließkommazahl immer ab.
- **`Math.round()`**: Arbeitet wie die Standard-Rundungstechniken. Jeder Teilwert, der kleiner als .5 ist, wird abgerundet, und jeder Teilwert, der größer als oder gleich .5 ist, wird aufgerundet.

```
<script>
  var x = prompt("Erste Zahl:");
  var y = prompt("Zweite Zahl:");
  var summe = parseFloat(x) + parseFloat(y);
  alert(x + " plus " + y + " gleich " + summe);
</script>
```

Variablentypen ändern

Funktion	Von	In	Beispiel	Ergebnis
`parseInt()`	String	Integer	`paseInt("23")`	`23`
`parseFloat()`	String	Fließkomma	`parseFloat("21.45")`	`21.45`
`toString()`	Beliebiger Datentyp	String	`meineVar.toString()`	`Verschieden`
`eval()`	Ausdruck	Ergebnis	`eval(" + 3")`	`8`

`Math.ceil()`
`Math.floor()`
`Math.round()`
	Fließkomma	Integer	`Math.ceil(5.2)`
			`Math.floor(5.2)`
			`Math.round(5.2)`
				`6`
				`5`
				`5`

Mit Zufallszahlen arbeiten

- Die meisten Computersprachen kennen die eine oder andere Funktion für das Erzeugen von Zufallszahlen. Die Funktion `Math.random()` gibt eine "zufällige" Fließkommazahl zwischen `0` und `1` zurück.
- Streng formal und technisch gesehen, sind die vom Computer erzeugten Werte keine wirklichen Zufallszahlen. Stattdessen werden mehr oder weniger komplexe Formeln genutzt, die mit einem Anfangswert starten und daraus weitere, schwer vorhersehbare Werte berechnen. Bei JavaScript und vielen anderen Sprachen wird für den ersten Wert der Stand der Systemuhr in Millisekunden ausgelesen, wodurch die weiteren Werte tatsächlich eine Zufallskomponente beinhalten.

Der Würfel

1. **Lass einen zufälligen Fließkommawert erzeugen**
   Verwende die Funktion `Math.random()`, um einen Fließkommawert zwischen 0 und 1 zu erhalten
2. **Multipliziere den erhaltenen Fließkommawert mit 6**
   Damit erhältst du einen Fließkommawert zwischen 0 und 5.9999 (aber niemals 6)
3. **Verwende `Math.ceil()`, um aufzurunden**
   An dieser Stelle wandelst du die Zahl in eine Ganzzahl um. `Math.ceil()` rundet immer auf, wodurch immer ein Integerwert zwischen 1 und 6 zustande kommt.

```
<script>
  var zahl = Math.random();
  alert(zahl);

  var groessereZahl = zahl * 6;
  alert(groessereZahl);

  var wuerfel = Math.ceil(groessereZahl);
  alert(wuerfel);
</script>
```

Vermeide in Programmcode und innerhalb von Dateinamen möglichst die Verwendung von Sonderzeichen Du weißt nie, unter welchen Ländereinstellungen ein Betriebssystem läuft und ob es überhaupt mit Sonderzeichen umgehen kann.

"IF" VERZWEIGUNG

- ✓ **Eine Bedingung einrichten.** Eine Bedingung ist eine mit wahr oder falsch zu beantwortende Frage. `if`-Anweisungen enthalten immer eine Art in Klammern angegebene Bedingung.
- ✓ **Einen Block mit Code beginnen.** if-Anweisungen verlangen eine Menge Code, der nicht immer ausgeführt wird. Das Ende der `if`-Zeile wird nicht durch ein Semikolon, sondern durch eine geschweifte Klammer festgelegt.
- ✓ **Es folgt normalerweise ein eingerückter Code.** Die Zeile oder die Zeilen, die der `if`-Anweisung unmittelbar folgen, sind Teil des Codeblocks, weshalb sie eingerückt werden
- ✓ **Die Anweisung endet einige Zeilen später.** Das Ende der `if`-Anweisung legt die geschlossene geschweifte Klammer fest, die im Regelfall einige Zeilen unterhalb des Anweisungsbeginns stet.

- ✓ **Der Code wird eingerückt.** Es ist üblich, den gesamten Code zwischen Anweisungsbeginn und Anweisungsende einzurücken

- › Den zentralen Teil der `if`-Anweisung bildet, zusätzlich zu anderen wichtigen Strukturen, eine Bedingung. Bei einer Bedingung handelt es sich um einen Ausdruck, der ausgewertet werden und wahr oder falsch sein kann.

```
<script>
  /*
    Holen einer Zufallszahl
    Wenn es eine 2 ist, hast du gewonnen
  */
  var wuerfeln = Math.ceil(Math.random() * 6);
  alert(wuerfel);
  if (wuerfel == 2) {
    alert("Du hast mit einer Zwei gewonnen");
  } // ende if
</script>
```

"IF" VERZWEIGUNG

- Die drei Hauptarten von Bedingungen:
  - **Vergleich:** Dies ist die am häufigsten eingesetzte Bedingungsart. Es werden in der Regel eine Variable mit einem Wert oder zwei Variablen miteinander verglichen.
  - **Boolesche Variable.** Sie sind Variablen, die als Wert nur `true` oder `false` enthalten können. Es ist überflüssig, einen booleschen Wert mit irgendetwas zu vergleichen, weil er bereits wahr oder falsch ist.
  - **Boolesche Funktion:** Es gibt Funktionen, die als Wert `true` oder `false` zurück geben. Auch diese Art von Funktionen kann als Bedingung verwendet werden.

- ✓ **Achte darauf, dass die variablentypen wirklich miteinander vergleichbar sind.** Man erhält unvorhersehbare Ergebnisse, wenn bspw. Ein Fließkommawert mit einem String verglichen wird.
- ✓ **String-Variablen können miteinander verglichen werden.** In JavaScript kann der Operator *Ungleich* verwendet werden, um die alphabetische Reihenfolge zweier Werte zu vergleichen, und man kann dasselbe Gleichheitssymbol für Zeichenketten verwenden.
- ✓ **Gleichheit benötigt ein doppeltes Gleichheitszeichen.** Das einfache Gleichheitszeichen (`=`) gibt eine *Zuordnung* an. Wenn Variablen verglichen werden, werden die doppelten Gleichheitszeichen (`==`) benötigt.

"IF" VERZWEIGUNG

Name	Operator	Beispiel	Anmerkung
`Gleich`	`==`	`(x==3)`	Funktioniert bei allen Variablentypen einschließlich String
`Ungleich`	`!=`	`(x != 3)`	Wahr, wenn Werte nicht gleich sind
`Kleiner als`	`<`	`(x < 3)`	Numerischer oder alphabetischer Vergleich
`Größer als`	`>`	`(x > 3)`	Numerischer oder alphabetischer Vergleich
`Kleiner als oder gleich`	`<=`	`(x <= 3)`	Numerischer oder alphabetischer Vergleich
`Größer als oder gleich`	`>=`	`(x >= 3)`	Numerischer oder alphabetischer Vergleich

"ELSE"-KLAUSEL

- ✓ **die `if`-Anweisung legt die Bedingung fest:** Das `if` zeigt den Anfang eines Bedinungszweiges an und bereitet den Weg für die Bedingung vor.
- ✓ **Die Bedingung führt eine Prüfung durch:** Bedingungen sind wahre oder falsche Ausdrücke, deshalb kann das Ergebnis der Bedingung wahr oder falsch sein.
- ✓ **Wenn (`if`) die Bedingung wahr (`true`) ist:** Der Code zwischen der Bedingung und der `else`-Klausel wird ausgeführt. (`else` kann mit dann oder ansonst übersetzt werden).
- ✓ **Wenn (`if`) die Bedingung falsch (`false`) ist:** Der Code zwischen `else` und dem Ende von `if` wird ausgeführt
- ✓ `else` **Einsatz:** `else` kann nicht für sich alleine eingesetzt werden sondern nur in Kombination mit einer `if`-Anweisung

```
<script>
  /*
    Holen einer Zufallszahl
    Wenn es eine 2 ist, hast du gewonnen
  */
  var wuerfel = Math.ceil(Math.random() * 6);
  alert(wuerfel);
  if (wuerfel == 2) {
    alert("Du hast eine 2 gewürfelt");
  } else {
    alert("Das war nur eine " + wuerfel + ".");
  } // ende If
</script>
```

"SWITCH-CASE"

- Dem Schlüsselwort `switch` (umschalten) folgt sofort ein Ausdruck in Klammern. Dabei handelt es sich normalerweise um eine Variable mit mehreren möglichen Werten. Die `switch`-Struktur stellt dann eine Reihe zu prüfender Werte und den Code bereit, der jeweils ausgeführt werden soll (`case`).

- So wird eine `switch`-Anweisung erstellt:

1. **Beginne mit dem Schlüsselwort switch**
   Dies richtet die Struktur ein. Alles in der geschweiften Klammer wird eines eingerückt.

2. **Gebe den Ausdruck an.**
   Hierbei handelt es sich normalerweise um eine Variable, die man mit mehreren Werten vergleichen will. Die Variable steht in den Klammern; ihr folgt eine geschweifte Klammer

3. **Lege die Alternativen fest**
   Lege den ersten Wert fest, mit dem die Variable verglichen werden soll. Achte darauf, dass die Alternative (`case`) vom selben Datentyp ist wie die Variable.

4. **Beende die `case`-Beschreibung mit einem Doppelpunkt (`:`).**
   arbeite sorgfältig!!

5. **Schreibe den Code für die Alternative.**
   Du kannst so viele Zeilen Code schreiben, wie du willst. Dieser Code wir nur dann ausgeführt, wenn der Ausdruck mit der gegebenen Alternative übereinstimmt.

6. **Kennzeine das Ende der jeweiligen Alternative mit der Anweisung `break`.**
   Die Anweisung break weist den Computer an, aus der `switch`-Struktur herauszuspringen, sobald die `case`-Alternative ausgewertet worden ist.

7. **Wiederhole den Vorgang mit weiteren Alternativen.**
   Schreib für die übrigen Alternativen, die du abprüfen willst, einen ähnlichen Code.

8. **Richte dich mit der Klausel `default` auf Überraschungen ein.**
   Die Alternative `default` funktioniert wie `else` in einer `else-if`-Struktur. Sie kümmert sich um jene Alternativen, die nicht berücksichtigt worden sind.
   In der `default` Klausel kann auf eine `break`-Anweisung verzichtet werden, weil sie immer am Ende einer `switch`-Struktur steht.

IF-ELSE VS. SWITCH-CASE

```
<script>
  var wuerfel = Math.ceil(Math.random() * 6);
  if(wuerfel == 1) {
    alert("1 gewürfelt!");
  } else if (wuerfel == 2) {
    alert("2 gewürfelt!");
  } else if (wuerfel == 3) {
    alert("3 gewürfelt!");
  } else if (wuerfel == 4) {
    alert("4 gewürfelt!");
  } else if (wuerfel == 5) {
    alert("5 gewürfelt!");
  } else if (wuerfel == 6) {
    alert("6 gewürfelt!");
  } else {
    alert("Huston, wir haben ein Problem!");
  } // ende IF
</script>
```

```
<script>
  var wuerfel = Math.ceil(Math.random() * 6);
  var ausgabe = "";
  switch(wuerfel) {
    case 1:
      ausgabe = "I";
      break;
    case 2:
      ausgabe = "II";
      break;
    case 3:
      ausgabe = "III";
      break;
    case 4:
      ausgabe = "IV";
      break;
    case 5:
      ausgabe = "V";
      break;
    case 6:
      ausgabe = "VI";
      break;
    default:
      ausgabe = "PROBLEM!!!";
  } // Ende Switch
  alert(ausgabe);
</script>
```

Merke bei "Switchen"

- ✓ **Du kannst jede Art von Ausdrücken miteinander vergleichen.** Unter JavaScript kann switch für beliebige Datentypen verwendet werden. Ob das Resultat sinnvoll ist, sei dahingestellt.
- ✓ **Es liegt an dir, den Datentyp richtig vorzugeben.** Wenn du mit einer numerischen Variablen arbeitest und diesen mit einer String-Variablen vergleichst, wirst du kaum die Ergebnisse erhalten, die du erwartest.
- ✓ **Vergiss die Doppelpunkte nicht.** Am Ende der meisten Zeilen verwendet die Anweisung switch Semikolons wie die meisten JavaScript-Befehle. Die Zeilen, die die Alternativen (mit `case`) beschreiben, enden aber mit einem Doppelpunkt!

Verschachtelte "if"-Anweisung

- Bedingungen können miteinander kombiniert werden. Eine Entscheidung kann andere Entscheidungen enthalten, in denen es wiederum andere Entscheidungen gibt. Man kann `if`-Anweisungen ineinander packen, um komplexe Logik zu verwalten.
    - **Achte auf die Einrückungen.** Einrückungen bieten die Möglichkeit auszudrücken, in welcher Code-Ebene man sich befindet. Achte aber auf das Einrückungsschema, für das du dich entschieden hast.
    - **Verwende Kommentare.** Füge regelmäßig Kommentare hinzu, um sich selbst daran zu erinnern, wo du dich jeweils in der logischen Struktur befindest.
    - **Teste den code.** Nur weil du der Meinung bist, dass alles funktioniert, muss das noch lange nicht der Fall sein. Teste sorgfältig, um sicher zu gehen, dass der Code auch wirklich das macht, was er machen soll.

Eine Zählerschleife mit "for"

- Die `for`-Schleife zählt zu den Standardschleifen.
- `for`-Schleifen basieren auf einer ganzzahligen *Zählervariable*, die hier runde genannt wurde. Diese Variable wird dazu verwendet, um in der Schleife die Anzahl der Wiederholungen festzulegen.
  Die `for`-Anweisung besteht aus drei eigenständigen Teilen

  1. **Initialisierung**: Dieses Segment (`runde = 1`) legt den Startwert des Wächters fest
  2. **Bedingung**: Die Bedingung (`runde <= 10`) ist eine ganz normale Bedingung
  3. **Änderung**: Der letzte Teil der `for`-Struktur (`runde++`) gibt an, dass die Zählvariable nach jedem Durchlauf der Schleife geändert wird. Im Beispiel wird die Variable runde bei jedem Schleifendurchlauf um `1` erhöht.

- Die `for`-Struktur enthält ein Klammernpaar, in das der zu wiederholende code gesetzt wird. In einer Schleife kann es beliebig viel Code geben.

- Bei dem Operator `++` handelt es sich um eine besondere Abkürzung, ausgeschrieben heißt es: `runde = runde + 1`
  Andere Möglichkeiten:
  Rückwärtslaufende Schleifen:

```
for (runde = 10; runde >= 1; runde--)
```

  Um mehr als 1 erhöhen

```
for (i = 5; i <= 25; i += 5)
```

```
<script>
  for (runde = 1; runde <= 10; runde++) {
    alert("dies ist Schleifenrunde: " + runde + ".");
  } // ende for
</script>
```

- `for`-Schleifen sind nützlich, wenn du vorher weißt, wie oft etwas wiederholt werden soll

WHILE-SCHLEIFE

- Das Skript stellt dem Benutzer eine einfache mathematische Frage – und fragt so lange nach, bis der Benutzer richtig antwortet.

- Arbeitsweise von `while`-Schleifen:

1. **Lege eine Variable fest**
   Diese Variable fungiert als Zählvariable der Schleif.
2. **Gebe der Variable einen Startwert**
   Der Startwert der Variable wird auf `-99` gesetzt, was eigentlich nicht richtig sein kann. Dadurch wird aber garantiert, dass die Schleife mindestens einmal läuft.
3. **Lege fest, was sich in der Variable befinden soll**
   Im Beispiel lautet die richtige Antwort `5`. Wenn die Antwort anders aussieht, läuft die Schleife weiter. D der Anfangswert auf `-99` gesetzt ist, gibt es mindestens einen Schleifendurchlauf.
4. **Stelle dem Benutzer eine Frage**
   Es ist in dem Beispiel wichtig, den Wert der Antwort ändern zu können, damit man den Wert `5` erhalten kann und die Schleife irgendwann verlassen wird.
5. **Gib dem Benutzer eine Rückmeldung**

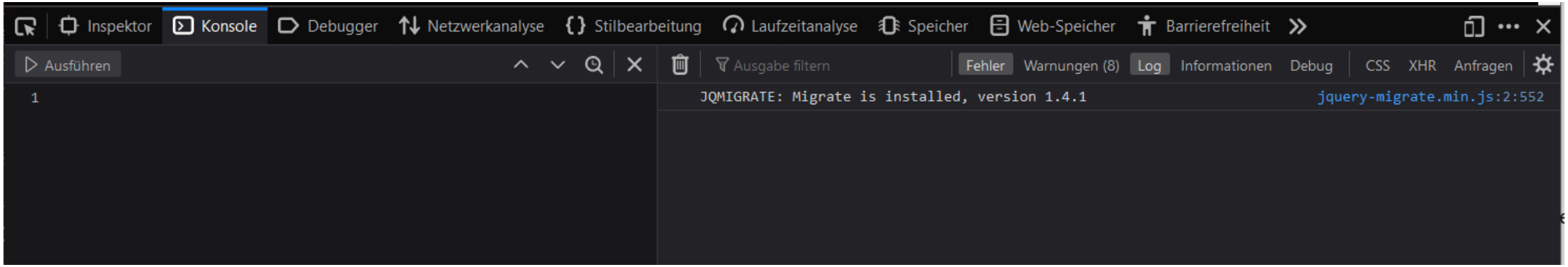
```
<script>
  antwort = "-99";
  while (antwort != "5") {
    antwort = prompt("was ergibt 3+2");
    if (antwort == "5") {
      alert("Klasse");
    } else {
      alert("Versuch es nochmal....");
    } // ende IF
  } // ende WHILE
</script>
```

Vermeide Schleifenfehler

- `while`-Schleifen sind scheinbar einfacher als for-Schleifen, stellen aber im Gunde die gleichen Anforderungen:
  - **Es gibt normalerweise eine entscheidende Zählvariable.** `while`-Schleifen werden normalerweise (aber nicht immer) von einer zentralen Variablen kontrolliert.
  - **Die Zählvariable muss mit einem Startwert versehen werden.** Wenn sich die Schleife wie gewünscht verhalten soll, muss der Zählvariablen ein geeigneter Startwert zugewiesen werden. Meistens soll die Schleife auf jeden Fall einmal durchlaufen werden
  - **Du musst eine Bedingung haben.** Die `while`-Schleife basiert wie die `for`-Schleife auf Bedingungen. Die Schleife wird so lange durchlaufen, wie die Bedingung wahr ist.
  - **Es muss einen Mechanismus geben, über den die Zählvariable geändert werden kann.** Irgendwo in der Schleife benötigst du eine Zeile, die einen Wert so ändert, dass eine Abbruchbedingung erfüllt wird, ansonsten kommt es zu einer Endlosschleife.

Fehler im Firefox finden

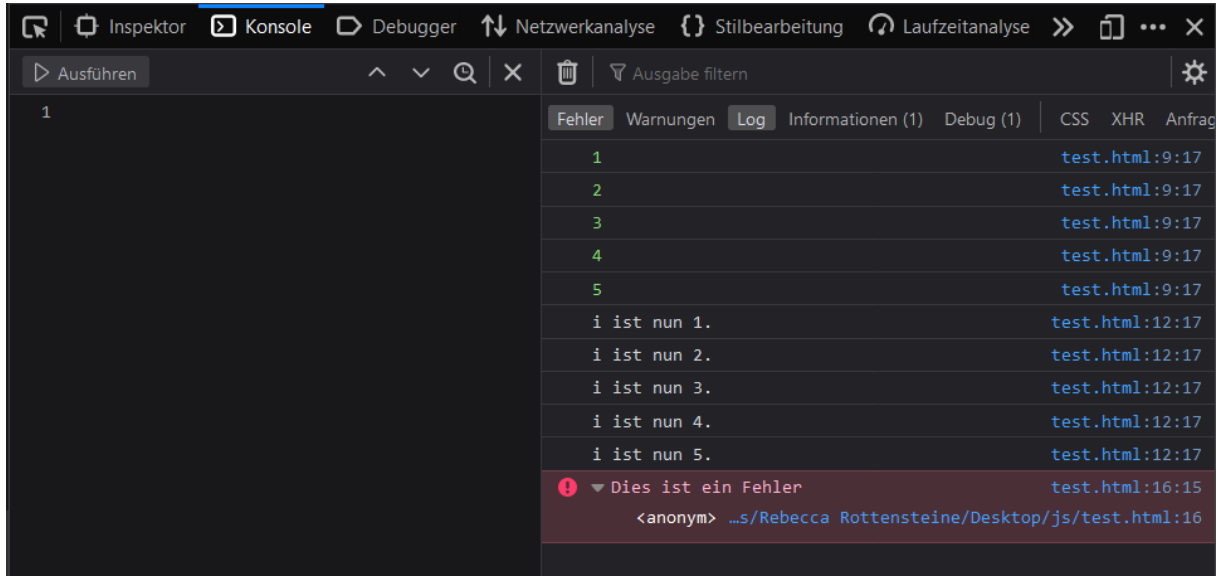
- Traditionell hat Firefox in der Web-Konsole eine deutlich bessere Fehlerbehandlung und aussagekräftigere Fehlermeldungen als die älteren Internet-Explorer.
- Aufzurufen mit der Taste **F12** oder rechte Maustaste -> Element untersuchen



KONSOLENMELDUNGEN

- Mit dem **console**-Objekt zu arbeiten, macht vieles sehr viel komfortabler. Es kann damit an den strategisch wichtigen Stellen im Code ein paar Ausgaben eingebaut werden, um auf diese Weise zu überprüfen, ob der Code wie gewünscht funktioniert.

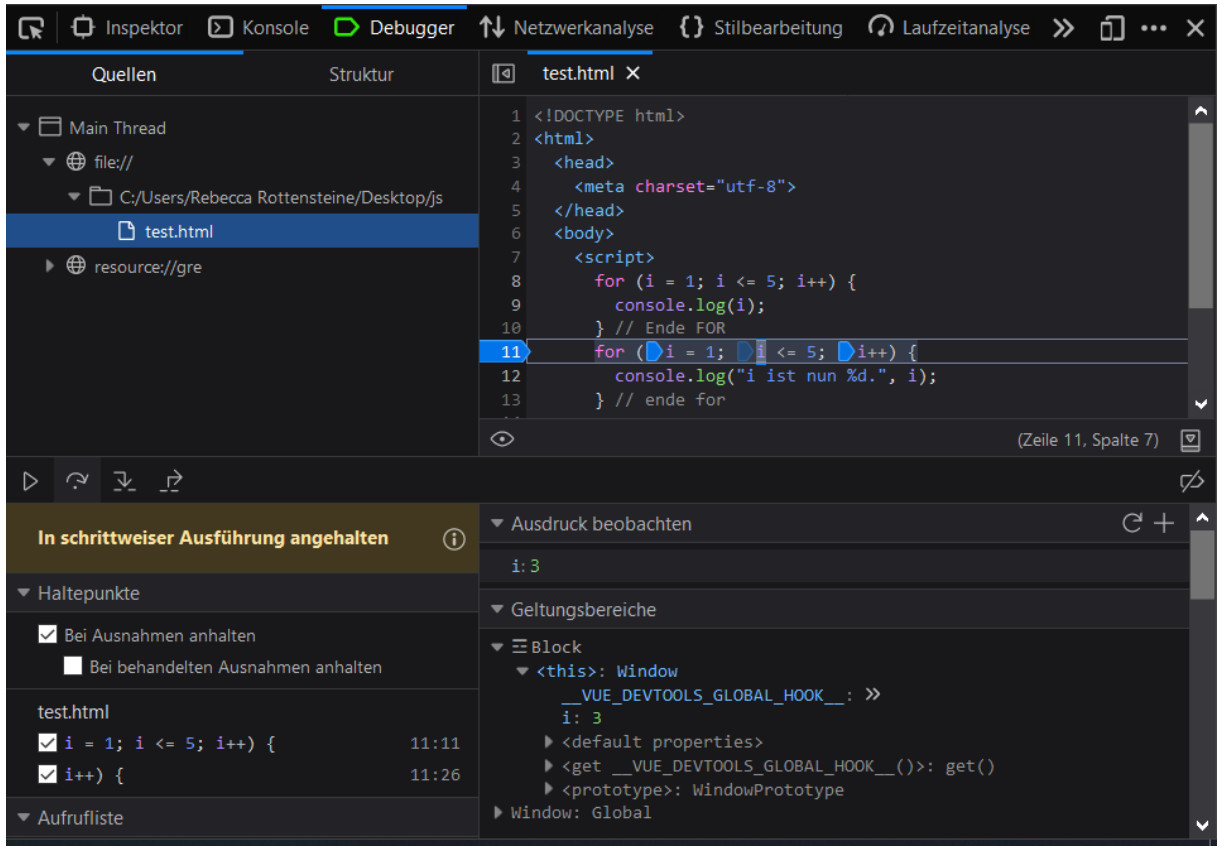
```
<script>
  for (i = 1; i <= 5; i++) {
    console.log(i);
  } // Ende FOR

  for (i = 1; i <= 5; i++) {
    console.log("i ist nun %d.", i);
  } // ende for

  console.error("Dies ist ein Fehler");
</script>
```



Konsolenbefehle: https://developer.mozilla.org/de/docs/Web/API/Console

HALTEPUNKT SETZEN

- Wenn Programme umfangreicher werden, ist es bei der Fehlersuche einfacher, Haltepunkte (**`Breakpoints`**) zu setzen.
- Wenn du das Programm jetzt neu startest, wird es bis zum Haltepunkt ausgeführt. Man sollte nicht nur einen Haltepunkte setzen, sondern auch noch einen zu beobachtenden Ausdruck hinzufügen.



Quelle: JavaScript für Dummies

ISBN: 978-3-527-71444-5
(Keine Kaufempfehlung!)