# JavaScript-Grundlagen


JavaScript ist eine Programmiersprache zur dynamischen Webseitengestaltung. JavaScript wurde erfunden, um Webseiten dynamisch im Browser zu ändern und Interaktionen mit dem Benutzer zu realisieren. Das ist heute immer noch der Haupteinsatz.

JavaScript ist die einzige Sprache, die von fast allen vorhandenen Browsern nativ unterstützt wird.

Bei JavaScript handelt es sich um eine Programmiersprache, die speziell für Internetseiten entworfen wurde. Sie bietet viele Elemente, die auch aus anderen Programmiersprachen bekannt sind, wie zum Beispiel

- Schleifen zur schnellen Wiederholung von Programmteilen,
- Verzweigungen zur unterschiedlichen Behandlung verschiedener Situationen und
- Funktionen zur Zerlegung eines Programms in übersichtliche Bestandteile.

Außerdem hat man mit Hilfe von Objekten und dem **Document Object Model (DOM)** Zugriff auf alle Elemente der Internetseiten, so dass man sie dynamisch verändern kann.

JavaScript-Programme werden den Benutzern innerhalb von Internetseiten zusammen mit HTML zur Verfügung gestellt.

- Sie werden **auf dem Browser des Benutzers** ausgeführt und **können**
- **die Inhalte einer Internetseite dynamisch verändern**.

Dies geschieht entweder sofort nach dem Laden der Internetseite oder nach dem Eintreten eines Ereignisses, zum Beispiel der Betätigung eines Buttons durch den Benutzer.
JavaScript ermöglicht somit den Entwurf komplexer Anwendungen mit einer Benutzeroberfläche.
**JavaScript wurde entworfen, um dem Benutzer zusätzliche Möglichkeiten und Hilfen zu bieten, die er allein mit HTML nicht hat.**

Um HTML-Seiten mehr Interaktivität zu verleihen führte die Firma **Netscape 1995** in ihrem Browser Navigator 2.0 eine Skriptsprache "ActiveScript" ein. Die Programmiersprache „Java“ wurde im gleichen Jahr von der **Firma SUN** entwickelt, um ein plattformunabhängiges Entwicklungssystem zu haben. ActiveScript wurde dann von Netscape an Java angepasst und in JavaScript umbenannt.

Es gibt keine festen Vorschriften dafür, an **welcher Stelle** ein HTML-Datei ein JavaScript-Bereich definiert werden muss. **Es hat sich jedoch eingebürgert, es im Head hinter den Titel zu setzen**, dann ist es schon geladen, wenn es gebraucht wird.

Man kann mehrere Script-Bereiche auf der Seite verteilen, **sowohl im Head als auch im Body Bereich**. Die Skripte werden dann von oben nach unten abgearbeitet.

Einige Schlüsselwörter und Anweisungen werden vollständig klein geschrieben. Manche haben einen großen Anfangsbuchstaben, und einige haben große Anfangsbuchstaben für jedes Wort (wenn sie denn aus mehreren Wörtern bestehen). Wichtig ist, dass du Anweisungen so schreibst, wie sie vorgegeben sind. Das nennt man **case sensitive**, und es sieht beispielsweise so aus: alert(); oder charAt(); oder Math.sin()

Zum Schreiben der Programme genügt ein **Texteditor**, der die Markierungen von HTML und die Schlüsselwörter von JavaScript hervorheben kann. Ein Editor ist, wie ein Browser, auf jedem Rechner vorhanden. Nicht jedoch auf Smartphones und Tablets. Bessere Editoren sind z.B.

- ***http://notepad-plus-plus.org***
- ***https://code.visualstudio.com (von Microsoft)***

Bessere Editoren, wie die hier erwähnten haben alle guten Funktionen:

- Syntax-Highlighting (Syntaxhervorhebung): farbliche Gestaltung des Quellcodes. Befehle, Variablen und Sonderzeichen werden jeweils in einer anderen Farbe präsentiert. Dadurch ist der Code übersichtlicher und Schreibfehler werden schnell erkannt. Falsch geschriebene Befehle werden nämlich erst gar nicht farblich markiert.
- Code-Vervollständigung: Wenn man den Namen eines Objektes eingibt, werden mögliche Anweisungen oder Funktionen geliefert und man kann einfach auswählen.
- Man kann beliebig viele Dateien gleichzeitig geöffnet haben.
- Code Folding: zusammengehörige Programmteile werden erkannt. Und über kleine quadratische Kästchen am Rand kann man die Teile zusammenklappen und auch wieder aufklappen. Damit wird das Programm übersichtlicher.
- Eine Website kann mit einem Klick auf ein bestimmtes Icon direkt im Browser geöffnet werden.
- Fertige Plugins erweitern die Fähigkeiten, wenn gewünscht.

**KISS-Prinzip: Keep it simple, stupid!**
Eine einfache Lösung ist einer komplexen, umständlichen Lösung vorzuziehen. Denn dieser hat meist weniger versteckte Fehler, ist einfacher zu lesen und einfacher zu warten.

# 1)JavaScript einbinden

## 1a)Inline-JavaScript

Diese einfache Technik nutzen die für HTML-Elemente verfügbaren Attribute aus. Diese sind sogenannte „Event-Handler“, die den Code einfach ausführen.

Die gebräuchlichsten Event-Handler haben mit der Maus zu tun:

- onclick
- onMouseOver
- checked - z.B. in Formularen von HTML
- onload

Onclick reagiert auf das Klick-Ereignis. Es funktioniert bereits in einer einfache HTML-Umgebung, ohne einer <script>-Einbettung.

## 1b)eingebettetes JavaScript

Um mit umfangreicheren Codes und benutzerdefinierten Funktionen arbeiten zu können, muss man die Scripts zwischen <script>-Tags zusammenfassen. Es funktioniert genauso wie das <style>-Element für CSS, indem es den Code an seinem Platz hält und die restlichen Elemente im Dokument mittels Verweise anspricht.

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

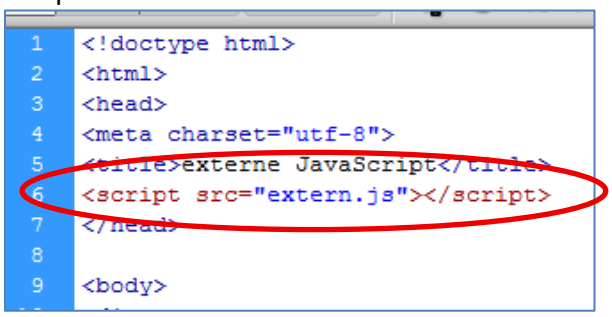
- Es wird dieser script-Container benötigt.
- Innerhalb des Containers befinden sich die JavaScript-Anweisungen, die der Reihe nach ausgeführt werden.
- Jede Anweisung sollte mit einem Semikolon abgeschlossen werden. Dadurch ist klar, wo die Anweisung zu Ende ist und wo die nächste beginnt.
- Man muss auf die richtige Schreibweise achten: die Anweisung ist so zuschreiben, wie sie vorgesehen ist. Das nennt man „case sensitive“.
  Beispiele: alert(); charAt(); Math.sin().

Was sich innerhalb dieses Tags befindet, wird für die Darstellung der Webseite erst einmal gar nicht berücksichtigt. Der Browser übergibt den Inhalt dieses Tags an den JavaScript-Interpreter. Der JavaScript-Interpreter ist ein besonderes Programm innerhalb des Browsers. Er analysiert und übersetzt die Anweisungen im „script“-Tag so, dass der Computer sie versteht.

## 1c)Externe JavaScript-Datei

Es ist sinnvoll JavaScript-Code in einer oder mehreren externen Dateien zu speichern und mit dem Attribut „src“ aufzurufen. Das verkürzt den Download, erhöht die Produktivität und eröffnet die Möglichkeit, den Code in jedem Dokument wiederzuverwenden.

Beispiel:




**Welche Themen werden wichtig sein?**

Variablen und Datentypen
Conditionals (IF)
Loops – for-Schleife
Funktionen
Arrays
Objekte
DOM-Manipulation

**START:**

1. **Zuerst wird mit der Konsole im Browser gearbeitet**
2. **Dann im Browser**

## Datentypen:

Damit kann man unterschiedliche Daten in verschiedene Kategorien organisieren.

| | |
|---|---|
| string: | „Hallo, wie geht es dir?“ -WICHTIG: Anführungszeichen |
| number: | 5, 20 |
| boolean: | true or false |
| undefined: | undefined |
| null: | null |

Beispiel für boolean: ist man eingeloggt, dann gilt „true“ und man kann den Inhalt sehen; ist man nicht eingeloggt, das gilt der Wert „false“ und man kommt zur Seite „Registrieren“.

Beispiel „string“: das ist nicht limitiert auf das Alphabet, man kann auch Zahlen oder Sonderzeichen benutzen oder Emojis oder auch Leerzeichen.

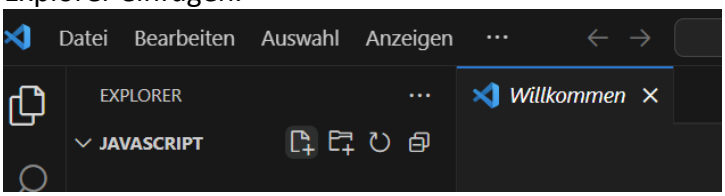
Beispiel für „undefined“ oder „null“: wenn man ein „VIP-Kunde“ ist, dann wird was angezeigt, wenn nicht, dann wird diese Anzeige auf Null oder Nichts gesetzt. Dann wird auch nichts angezeigt, kein Wert.

- In einer Variablen können beliebige Werte gespeichert werden.
- Texte werden in der Programmierung **Strings** genannt.

```
var playerName = 'Max'; // string
var highscore = 100; // number
var gameOver = false; // boolean
var test; // undefined
var nüscht = null // object
```

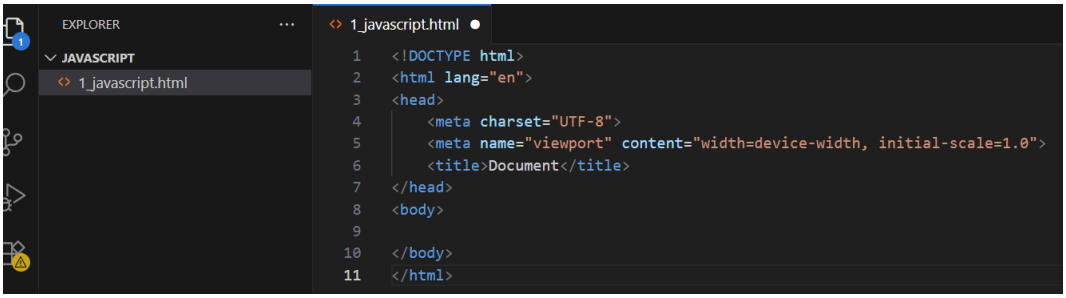
**Übung 1:**
Erstelle einen neuen Ordner mit dem Namen „javascript“.


Öffne den Editor „VisualStudio-Code“.
Ziehe mit der linken Maustaste den obigen Ordner einfach in VSC hinein und er wird sich im Explorer einfügen.




Erstelle eine neue HTML-Datei namens „1_javascript.html“ und erstelle einen Boilerplate-Code. **Boilerplate-Code - das** bezeichnet **Standardcode**, der in vielen Programmen oder Dateien gleich oder sehr ähnlich ist. Er dient oft als **Grundgerüst**.

Tastenkombination: **Rufzeichen und Enter-Taste**

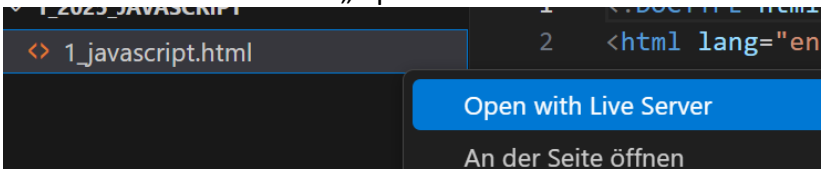


Im <body> soll ein Text ausgegeben werden.

Um JavaScript in HTML zu verwenden, muss es in einem Bereich stehen, der lautet:
<script>  </script>

```
8   <body>
9       <script>
10          document.write("<h2>JavaScript in HTML</h2>");
11      </script>
12
13  </body>
14  </html>
```

Speichere und öffne die Datei im Browser, z.B. mit dem „Live Server“ – rechter Mausklick in VSCode auf die Datei und „Open with Live server“:



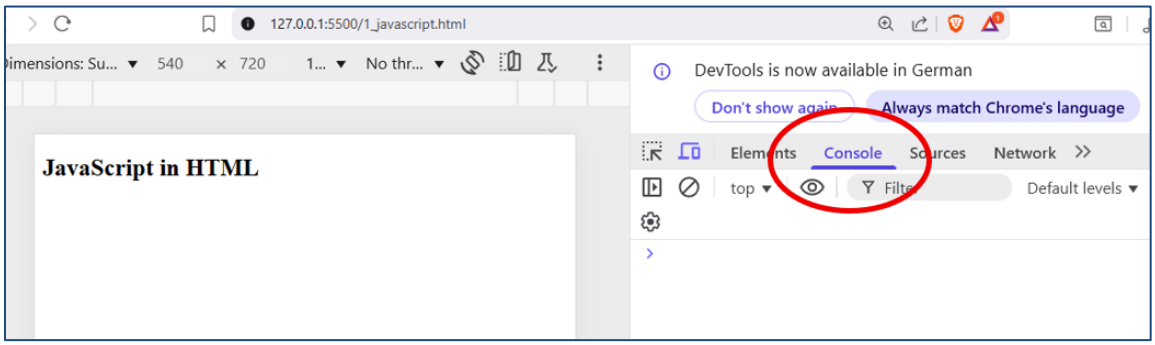

Öffne es im Browser:


## Konsole nutzen

Um sofort die Auswirkungen von JavaScript-Code zu sehen.
Nutze die Seite im Browser von oben und öffne die Konsole mit

- Rechte Maus in der Seite im Browser und wähle dann „Untersuchen“
  oder
- Funktionstaste F12

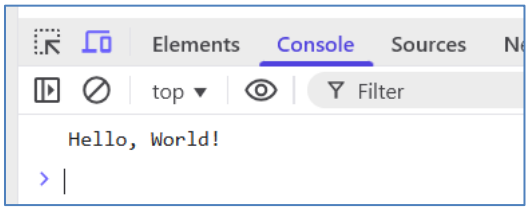
Klicke auf „Console“.



Ändere in der HTML-Site den Code auf

```
<body>
    <script>
        console.log("Hello, World!");
    </script>
</body>
</html>
```

Als Ergebnis sieht man die Ausgabe in der Konsole:



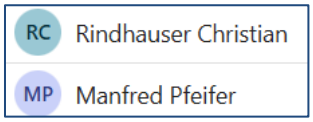

Nun soll eine Fähigkeit von JavaScript hervorgehoben werden – sofortige Aktivität

- Anfangsbuchstabe des Strings ermitteln
- Länge des Strings ermitteln und anzeigen

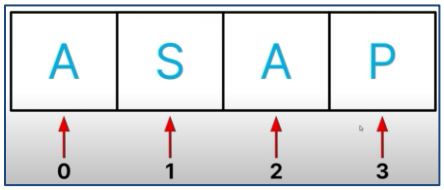
**Einzelne Stelle eines Strings auswählen und anzeigen lassen:**

Es soll der Inhalt der ersten Stelle des Strings gesucht und in der Konsole ausgegeben werden. (Das kann auch mit jeder anderen Stelle geschehen.)

Das wird sehr häufig benutzt, z.B. in Outlook werden die ersten Buchstaben des Vor- und Nachnamens gewählt und in einem Kreis angezeigt, als „Profilbild“.




In einem String hat jeder Inhalt eine Nummer. Das nennt man Index und beginnt mit 0.

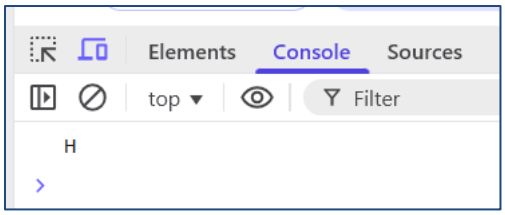



Dazu werden direkt nach dem String diese Klammern aufgenommen, die die gewünschte Stelle anzeigen.

- erste Stelle: 0

```
console.log("Hello, World!"[0]);
```

Ergebnis in der Konsole:

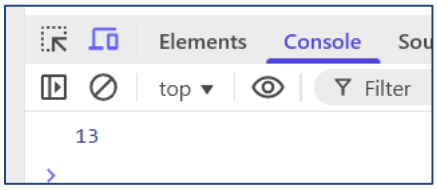



**Länge des Strings anzeigen lassen:**

Lösche die neuen Klammern wieder.
Direkt nach dem String wird die Methode „.length“ angehängt inkl. Punkt davor

```
console.log("Hello, World!".length);
```

Ergebnis:




## Mit Variablen arbeiten:

Variablen werden erzeugt mit

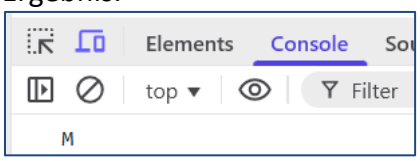
- var
- const

Erstelle eine Variable „vorname“ und darunter die Frage nach der ersten Stelle:

```
    <script>
        let vorname = "Max";

        console.log(vorname[0]);
    </script>
</body>
```

Ergebnis:




## Variablen:

Eine Variable ist eine Art „Container“, in dem man Informationen ablegen kann. Also ein Speicher für Daten, z.B. Geburtsdatum und andere Benutzerdaten.
Innerhalb des JavaScripts kann sich der Wert der Variable beliebig oft verändern und kann jederzeit ausgelesen und angezeigt werden.

**Variablen besitzen normalerweise folgende Merkmale:**

- Die Anweisung „let“ zeigt an, dass eine Variable erstellt werden soll.
- einen Namen
  Wenn man eine Variable anlegt, muss man ihr einen Namen geben
- einen Anfangswert
  Es ist üblich, einer Variablen sofort einen Wert zuzuweisen.
- einen Datentyp
  JavaScript kann den Datentyp zwar automatisch erkennen, aber man sollte zumindest wissen, welche Art von Daten in einer Variablen erwartet wird. Anders als in typisierten Programmiersprachen muss man nicht den Typ einer Variablen festlegen, dies passiert automatisch bei er Wertezuweisung. Sie sind also nicht typisiert. Sie können jeden beliebigen Wert aufnehmen.

Sprechende **Namen**, die auch ein halbes Jahr, nachdem das Skript geschrieben wurde, noch signalisieren, welche Bedeutung die Variable oder Funktion haben, erleichtern das Lesen und die Wartung des Codes erheblich! Längere Namen für Variable sind besonders selbsterklärend.

Dabei hat sich die sogenannte **camelCase-Schreibweise** eingebürgert: Der Name beginnt mit einem Kleinbuchstaben, jedes neue Wort beginnt mit einem Großbuchstaben,

buttonStart und buttonStop
dasHierIstEinfacherZuLesen ….. besser als dashieristeinfacherzulesen

**Info**: Eine Variable zu erstellen nennt man auch eine Variable „**deklarieren**“.

Beachte: **Kommazahlen in JavaScript werden durch einen Punkt** dargestellt (z.B.: 1.2).

Detail:

- Deklaration nennt man es, wenn man einen Namen gibt: let nachname
- Initialisierung nennt man es, wenn man einen Wert zuweist:

```
let nachname=“Huber“;

oder:

let nachname;
nachname = “Huber“;
```

Zu Beginn des Programms wird eine Variable mit dem Namen „nachname“ deklariert, mit Hilfe des JavaScript-Schlüsselworts „let“. Das heißt also, der Vorgang, bei dem eine Variable zum ersten Mal verwendet und mit „let“ eingeführt wird, nennt man „Deklaration“.

Nach der Deklaration folgt eine Zuweisung: Der Variablen „nachname“ wird der Wert "Huber" zugewiesen.

Ein Zeichenkettenwert muss zwischen doppelten oder zwischen einfachen Anführungsstrichen notiert werden.

## REGELN:

- Muss ein Wort sein, ohne Leerzeichen darin. Daher ist eine Schreibweise mit CamelCase sinnvoll, um es besser lesen zu können.
  Beispiel: „userEmail“ – falsch wäre „user Email“ (wegen Leerzeichen)
- Variablen müssen mit einem Buchstaben oder dem Grundstrich beginnen (z.B. nachname, _nachname, Nachname); NICHT mit einer Zahl.
- Sie dürfen nur aus Buchstaben und Ziffern bestehen und sogar aus dem Dollar-Zeichen $
- Daher keine:
  Leerzeichen, Umlaute, Sonderzeichen und keine reservierten JavaScript-Befehle, wie z.B. date, form, function usw.

Variablen können

- **Zahlen,**
- **Boolesche Werte** (true, false),
- **strings** (Texte in Anführungszeichen z.B. "ein Text")
- **null** (Schlüsselwort für einen Nullwert) enthalten. Eine Variable hat dann den Wert **null** (nicht zu verwechseln mit der Zahl 0) wenn sie nicht definiert ist, ihr also noch kein Wert zugewiesen wurde. Wird der Inhalt einer undefinierten Variable ausgegeben erscheint der Wert **undefined**.
- **Und sogar das Ergebnis einer komplexen Berechnung.**

**Beispiel:**

```
let userEmail = false;
let adminEmail = „geheim007“;
let alter = 16;
```

Erst wenn das Programm beendet wird, gehen die Variablen und die darin gespeicherten Werte wieder verloren.

Das Gleich-Zeichen „=“ ist hier nicht mathematisch zu verstehen, sondern ist eine Zuweisung.

Beachte:

- JavaScript ist per Definition case-sensitiv, d.h. **zahl** und **Zahl** sind zwei verschiedene Variablen.
- Hexadezimale Zahlenwerte müssen mit 0x beginnen, z.B.: 0x3f.

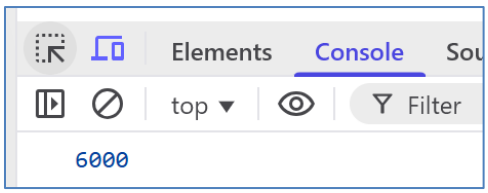
**Übung:**

Umrechnung eine Unze Gold = 3.000 Euro
Ziel: Ausgabe vom Wert von 2 Unzen

```
9     <script>
10        // 1 unze = 3000
11        let unze = 1;
12        let betrag = unze * 3000;
13
14        console.log(2 * betrag);
15    </script>
16  </body>
```

oder man ändert den Wert von der Variable „unze“

```
9     <script>
10        // 1 unze = 3000
11        let unze = 1;
12
13        unze = 2;
14        let betrag = unze * 3000;
15        console.log(betrag);
16    </script>
```



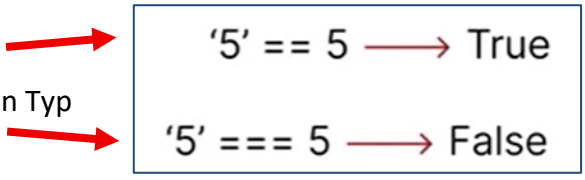

## Variabel vergleichen

Operatoren wie < oder >.

Vergleich auch mit == oder ===

Unterschied zweimal == und dreimal ===

- Zweimal == überprüft die Werte
- Dreimal === überprüft die Werte UND den Typ

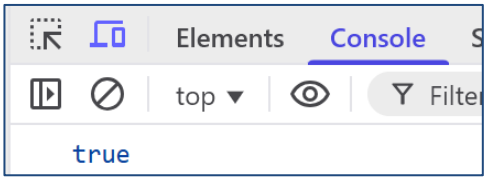



Es wird ein String mit dem Wert 5 mit einer Nummer mit dem Wert 5 verglichen.

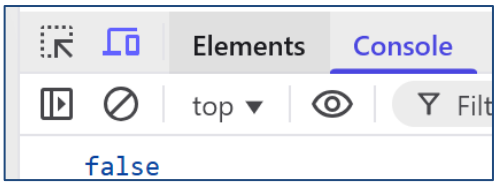
DAHER sollte man eigentlich immer dreimal das === benutzen.

**Übung:**

```
9     <script>
10        let vergleich = "5" == 5;
11
12        console.log(vergleich);
13    </script>
```



```
9       <script>
10          let vergleich = "5" === 5;
11
12          console.log(vergleich);
13      </script>
```




## Entscheidungen (IF-Anweisung)

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 so aussehen

- if(a > b){…};  Semikolon ist falsch hier!!
- 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.

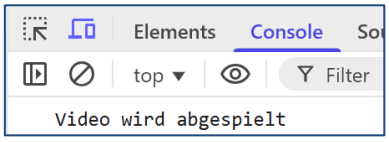
**Übung:**

Wenn die Mitgliedschaft „true“ ist, dann soll das Video angezeigt werden

```
<script>
    let mitgliedschaft = true;

    if (mitgliedschaft === true) {
        console.log("Video wird abgespielt");
    } else {
        console.log("Du bist kein Mitglied");
    }

</script>
```




**Übung mit Alter-Abfrage**

- if inkl. „else if“
- mit Ausgabe der Variablen

Es soll gefragt werden, ob eine Person in den Club reindarf. Die Grenze ist 18 Jahre. Ist das Alter über 18 oder unter 18? Ist der Kunde genau 18, dann soll eine besondere Meldung kommen.

```
<body>
    <script>
        let alter = 17;

        if (alter > 18) {
            console.log("Du darfst eintreten");
        } else if (alter === 18) {
            console.log("Gratulation, du darfst eintreten");
        }
       else {
            console.log("Du darfst nicht eintreten");
        }
    </script>
</body>
```

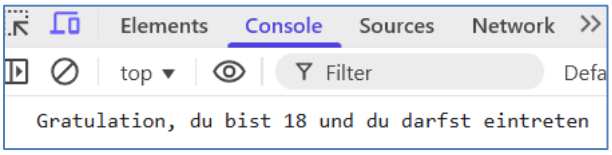
Nun soll die Variable „alter“ ebenfalls eingebaut werden und ausgegeben werden.

Dazu wird die Variable eingebunden. Das funktioniert mit Hilfe des

- + Symbol vor und nach der Variable
- und den String vorher und nachher mit den Anführungszeichen beenden und wieder starten

```
let alter = 18;
if (alter > 18) {
    console.log("Du bist " + alter + "  und du darfst eintreten");
  } else if (alter === 18) {
      console.log("Gratulation, du bist " + alter + " und du darfst eintreten");
  }
  else {
      console.log("Du darfst nicht eintreten");
  }
```

Ergebnis in der Console:




## Ausgabe mit „verketten“:

Steht ein + zwischen zwei Strings werden diese dadurch automatisch zu einem Gesamtstring verbunden.
**Der Operator + führt hier dann also keine Addition durch**. Der Fachbegriff lautet hier: **concatenation**.
Man kann aber auch Strings mit Variablen verketten, wie in dem Beispiel oben.

**Beispiel:**

```
console.log("Du bist " + alter + "  und du darfst eintreten");
```

Dabei muss man auf die Lesbarkeit achten und bewusst Leerzeichen einbauen. So ist hier z.B.  ein Leerzeichen am Ende und am nachfolgenden Beginn des Strings .

Hier gilt die Regel: Der + Operator addiert nur dann, wenn keiner der Operanden ein String ist. Im nächsten Beispiel handelt es sich um Zahlen (keine Strings):

**Ab nun schreiben wir alles nicht mehr in die Konsole sondern in das Browserfenster mit „document.write()“**

## Punktnotation: document.write()

```
document.write('Guten Tag');
```

Die Syntax der Sprache JavaScript nennt man Punktnotation:

- vorne steht das Objekt und danach folgt, getrennt durch einen
- Punkt, die Methode, die auf dem Objekt ausgeführt werden soll. In unserem Fall ist das Objekt das „document“, also die im Browser angezeigte HTML-Seite. Die Methode „.write()“ gibt die Worte „Hello World“ aus, die in Klammern direkt nach dem Methodennamen definiert werden.

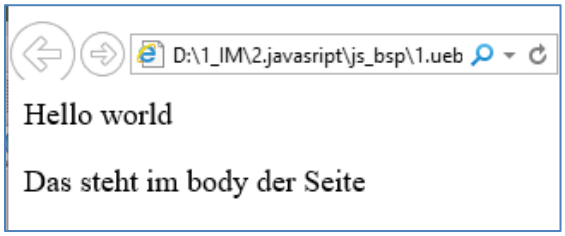
Da es sich um einen

- String, also eine Zeichenkette, handelt, muss dieser durch Anführungszeichen gekennzeichnet werden. Anders verhält es sich bei
- Zahlen, diese müssten ohne Anführungszeichen übergeben werden.

Schreibe nun auch in den <head> (nach dem <title>) der Datei folgenden Inhalt:

```
<script>
document.write("Hello World!");
</script>
```

Ergebnis:




**Wichtig:**

- In Anweisungen (hier: document.write()) dürfen keine Zeilenumbrüche vorkommen.
- Am Ende eines Befehles muss ein STRICHPUNKT stehen!
- JavaScript unterscheidet zwischen Groß- und Kleinschreibung.
  Beispiel: Ein großes „W“ in „document.Write“ ist erfolglos, das es diesen Befehl nicht gibt.
- Man kann auch mehrere Anweisungen in eine Zeile schreiben. Hauptsache, es steht ein Semikolon am Ende jeder Anweisung.

## Übung: new Date()

Der Konstruktor „Date()“ teilt dem JavaScript-Interpreter mit, dass die Variable ein Datum ist. Dadurch wird es möglich, die Methode des Date-Objektes zu verwenden, um Datum und Uhrzeit abzurufen.

**1.)Schreiben eines angepassten Aktualisierungsdatums**

```
document.write('Zuletzt aktualisiert am: ' + new Date() );
```

Ergebnis:

Zuletzt aktualisiert am: Thu May 15 2025 10:11:29 GMT+0200 (Mitteleuropäische Sommerzeit)

BESSER:

```
document.write('Zuletzt aktualisiert am: ' + new Date().toLocaleString() );
```

Zuletzt aktualisiert am: 15.5.2025, 10:13:03

## Keine Zeilenumbrüche – im Code (wegen besserer Lesbarkeit)

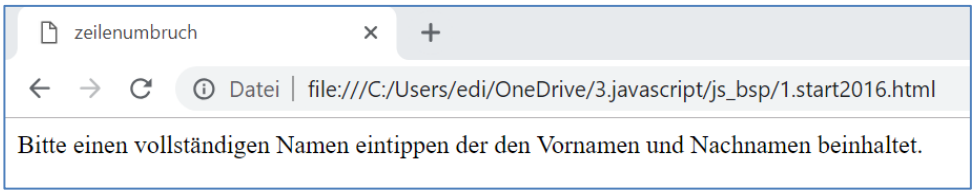
Jeder Text in JavaScript

- muss für sich in einer Zeile stehen,
- ganz im Gegenteil zu Texten in HTML.
- Ein echter Zeilenumbruch in einem Text ist nicht erlaubt und führt zu einem Fehler.

Beispiel:

document.write("Bitte einen vollständigen Namen eintippen, der den Vornamen und Nachnamen beinhaltet.");

Ergebnis: passt




Falsch: wenn man nach dem „eintippen“ einen Zeilenumbruch mit der Taste „enter“ setzt:

document.write(" Bitte einen vollständigen Namen eintippen
der den Vornamen und Nachnamen beinhaltet. ");

```
10 ▼ <script>
11   document.write("<p> Bitte einen vollständigen Namen eintippen
12                  der den Vornamen und Nachnamen beinhaltet.</p>");
```

**RICHTIG:**

Verbindung mit einem Pluszeichen, aber vorher und nachher das Zeilen-Ende bzw. der Zeilen-Start mit Anführungszeichen.

**Verketten:** Steht ein + zwischen zwei Strings werden diese dadurch automatisch zu einem Gesamtstring verbunden.

```
document.write("<p> Bitte einen vollständigen Namen eintippen "
        + "der den Vornamen und Nachnamen beinhaltet.</p>");
```

```
<script>
document.write("<p> Bitte einen vollständigen Namen eintippen "
               + "der den Vornamen und Nachnamen beinhaltet.</p>");
```

Das ist nur relevant für den Code, nicht für das Ergebnis im Browser. Da müsste man auch den Paragrafen  extra 2mal starten und beenden:

```
document.write('<p>Hallo world </p>' +
               '<p>noch einmal</>');
```

Hallo world

noch einmal

## Kommentare

Kommentare dienen zur Beschreibung der einzelnen Teile des Programms. Sie erleichtern die Lesbarkeit des Programms. Der Inhalt der Kommentare wird nicht im Browser dargestellt.

Es gibt drei verschiedene Arten von Kommentaren:

- Ein Kommentar im **HTML-Bereich** kann sich über eine oder mehrere Zeilen erstrecken. Er steht zwischen den Zeichenfolgen <!-- und -->.

- **Im JavaScript-Bereich** wird ein Kommentar, der über eine oder mehrere Zeilen geht, zwischen den Zeichenfolgen /* und */ notiert.
- Falls man nur einen kurzen Kommentar im JavaScript-Bereich notieren möchte, zum Beispiel hinter einer Anweisung, so eignet sich die Zeichenfolge //. Dieser einzeilige Kommentar geht nur bis zum Ende der jeweiligen Zeile.

Unterschiede zwischen HTML und JavaScript:

```
<body>
        <!--Das ist ein Kommentar im HTML-Bereich -->
<script>
                /* Das ist ein Kommentar über mehrere
                   Zeilen im JavaScript-Bereich */
                // Ein Kommentar, der nur bis zum Zeilenende geht
</script>
</body>
```

HTML

JavaScript

**Tipp:** im Editor VisualStudioCode wird automatisch der richtige Kommentar erzeugt wenn man folgende Tastenkombination drückt:

**STRG + #**

## Ein- und Ausgabe von Zeichenketten

Zwei nützliche Methoden:

- Die Methode **prompt()** erwartet, dass der Benutzer eine Eingabe macht, also einen Text eingibt. Die Eingabe wird in der Variable gespeichert.
- Mit Hilfe der Methode **alert()** wird ein Dialogfeld mit einer Meldung für den Benutzer ausgegeben. In den Klammern benötigt man die Anführungszeichen, außer es handelt sich um eine Zahl z.B. alert(34);
  „alert“ ist Englisch und bedeutet „Warnung“.

**Unterschied von „document.write“ und „alert“:**
Mit „document.write“ gibt man das Ergebnis direkt in der Webseite aus. Dafür gibt es einen kleinen Nachteil, nämlich gibt der Browser den Inhalt der Seite und somit alles von unserem „document.write“ erst aus, wenn das JavaScript beendet ist. Im Gegensatz zu „alert“, das jederzeit aktiv wird, auch wenn das Programm noch läuft.

Beide Methoden arbeiten mit Zeichenketten. Außerdem führen beide Methoden dazu, dass das Programm in seinem **Verlauf anhält**. Erst nach einer Eingabe bzw. nach einer Bestätigung der Meldung läuft das Programm weiter. Auf diese Weise hat man die Möglichkeit, den Benutzer zu einer Eingabe oder zum Lesen einer Ausgabe zu zwingen.

Die Methode alert() kommt vom Englischen „to alert“ und heißt alarmieren bzw. warnen. Sie erzeugt ein kleines Dialogfeld, in dem ein Text auf dem Bildschirm angezeigt wird.

## Übung mit prompt und alert

**Tipp: Sonderzeichen „\n“ bei „alert“ (nicht bei „document.write“)**

Damit der Text schöner ausgegeben wird, verwende das Steuerzeichen **\n** (Backslash und n). Es erzeugt einen Zeilenumbruch innerhalb des Dialogfeldes. Der Buchstabe n steht für „new line“.

Speichern unter „1.ausgabe.html“:

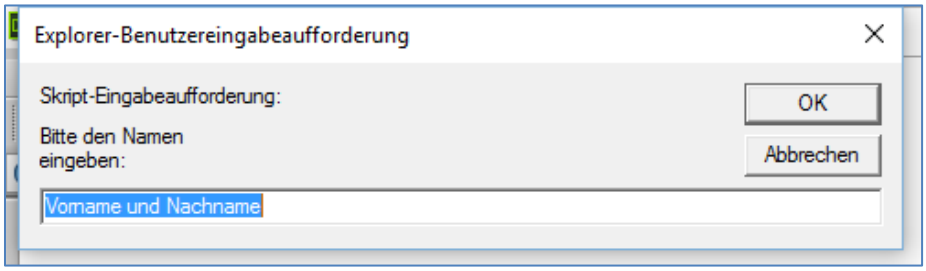
```
 8  <body>
 9  <script>
10  var deinName = prompt("Bitte den Namen \neingeben:", "Vorname und Nachname");
11  alert("Du hast eingegeben:\n" + deinName + "\nVielen Dank");
12  </script>
13  </body>
```

Das Steuerzeichen überlässt einem somit selbst und nicht dem Browser, wie der Text aussehen soll. Denn irgendwann, und meist an einer unpassenden Stelle, wird langer Text vom Browser automatisch in eine neue Zeile umgebrochen.

Weiteres Beispiel:
(man muss kein Leerzeichen machen)

alert("So eine Zeile kann ziemlich lang werden.\nAber so sieht es viel besser aus.");

Sobald Sie das Programm starten, erscheint eine Eingabeaufforderung wie hier in der Ausgabe:

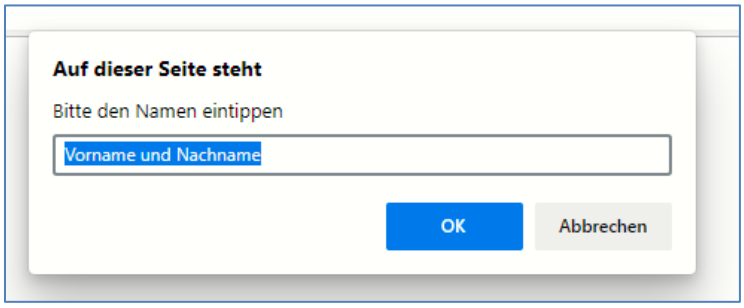



## PROMPT

Die Methode prompt() erzeugt die Eingabeaufforderung mit dem Text, der aus der ersten Zeichenkette stammt (=Bitte den Namen eintippen).
Man kann bei prompt() eine **zweite Zeichenkette angeben**, durch Komma getrennt. Diese enthält eine mögliche Vorgabe für das Eingabefeld, hier ist dies der Text „Vorname und Nachname“.
Falls man die zweite Zeichenkette weglässt, bleibt das Eingabefeld leer. Also: Bei prompt kommt in die Klammer vor das Komma der Text, welcher in der Prompt Box angezeigt wird. Hinter dem Komma kommen in der Regel nur 2 Anführungszeichen, es sei denn man will im Eingabefeld einen Text vorgeben.
Kurz gesagt: prompt(“fenstertext”, “eingabe”)

```
prompt("Bitte den Namen eintippen","Vorname und Nachname");
```

Ergenis:




Nach Betätigung des Buttons OK wird der aktuelle Inhalt des Eingabefeldes zurückgeliefert und in der Variablen deinName gespeichert.
Falls man den Button Abbrechen betätigt, ohne dass man was eingegeben hat, wird der Wert null gespeichert.

## Übung: Willkommensgruß auf der Startseite:

Speichern unter: 1.ausgabe.willkommen.html

```
<script>
     var nameEingabe;
     nameEingabe= prompt("Bitte den Namen eintippen");
     document.write("Hallo, "  + nameEingabe + " willkommen auf
     der Site. ");
</script>
```

Hier wird zuerst die Variable nameEingabe deklariert.
Anschließend folgt die Begrüßung auf der Seite, welche zwischen die Klammern von document.write () eingefügt wird. Dabei wird die Variable nameEingabe mit 2 Verkettungs-operatoren + mit dem anderen Text verknüpft. „textstring“ + variable + „textstring“

## Übung: Prompt Box mit String und „Number“:

Speichern unter: 1.ausgabe.number.html

```
<script>
     var x, y;
     x=prompt("Gib eine Zahl ein","");
     y=prompt("Gib noch eine Zahl ein","");
     alert (Number(x) +  Number(y));

</script>
```

Der Wert des Eingabefeldes wird an die Variable zurückgegeben.
Die Prompt Box gibt immer den Datentyp string zurück. Im oberen Beispiel wird dieser mittels Number() in eine numerische, rechnerische Zahl umgewandelt.

### Methode „Number“:  (das N ist groß geschrieben)

Diese Methode ist fix in JavaScript vorhanden (ist eine globale Methode) und konvertiert Variablen zu Nummern:

Beispiele:

```
Number("10");       // returns 10
Number("John");     // returns NaN
```

# Mathematischer Operatoren:

Mit den arithmetischen Operatoren kann man wie aus der Mathematik gewohnt Berechnungen durchführen. Wenn die Berechnung mehrere Operatoren verwendet, gilt die Punktvor-Strich-Regel. Beispiel:

```
var ergebnis = 1 + 2 * 3;
```

**Übung**: Neue Datei mit dem Namen „1.datentypen.html“

Lege eine Zahlenvariable „i“ an („i“ für „integer“) und ein „j“

```
 8 <body>
 9 <script>
10     var i = 12345;
11     var j = 333;
12 </script>
```

Summe bilden: var summe = i + j

Diese soll auch ausgegeben werden: alert(summe)

```
 8 <body>
 9 <script>
10     var i = 12345;
11     var j = 333;
12     var summe = i + j;
13     alert(summe);
14 </script>
```

**Übung Kreisumfang:**

Berechne den Umfang eines Kreises, der einen Radius von 4 hat.

```
 3 <title>Kreisumfang</title>
 4 <script>
 5     var pi = 3.1415;
 6     var radius = 4;
 7     var kreisumfang;
 8     kreisumfang = 2 * pi * radius;
 9     document.write(kreisumfang);
10 </script>
11 </head>
```

**Mathematische Funktion verwenden:**
**Beachte die andere Variablen-Deklaration**

```
//Var Deklaration
var pi, radius, kreisumfang;
pi = Math.PI;
radius = 5;
kreisumfang = 2 * pi * radius;
document.write(kreisumfang);
```

**Übung**: Wahrheitswerte

```
15      var t = true;
16      var f = false;
17      alert(t);
```

**Speichern von Wahrheitswerten**

In einer Variablen können neben Zeichenketten und Zahlen auch Wahrheitswerte (boolesche Werte) gespeichert werden. Es gibt zwei Wahrheitswerte:

- true (deutsch: wahr)
- false (deutsch: falsch, unwahr).

Auf diese Weise kann man zum Beispiel den Zustand eines Kontrollkästchens (englisch checkbox) in einem Formular speichern. Falls die CheckBox markiert ist, dann hat eine bestimmte Eigenschaft der CheckBox den Wert true, ansonsten den Wert false.

## Übung: Berechnung des Bruttopreises:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Bruttopreis</title>
</head>

<body>
<h2>Berechnung des Bruttopreises</h2>
<script>
var Preis;
var Netto;
var Steuer = 0.2;
var Währung = "Euro";

Netto=100;
Preis=Netto*(1+Steuer);
document.write("Der Bruttopreis lautet " + Preis + " " + Währung);
</script>
</body>
</html>
```

Verbesserungsvorschlag: Variablen bitte klein schreiben, keine Umlaute z.B. Währung

Speichern als „1.bruttopreis.html“

**Aufgabe:**
Die Werte „Steuer“ und „Netto“ sollen mit prompt-Abfragen eingegeben werden können. Benutze auch die Funktion „parseFloat()“.

## Methode „parseFloat()“: (das F ist groß geschrieben)

Diese Methode ist fix in JavaScript vorhanden (ist eine globale Methode) und konvertiert Strings zu Nummern. Abstand ist erlaubt, aber nur die erste Nummer wird ausgegeben.

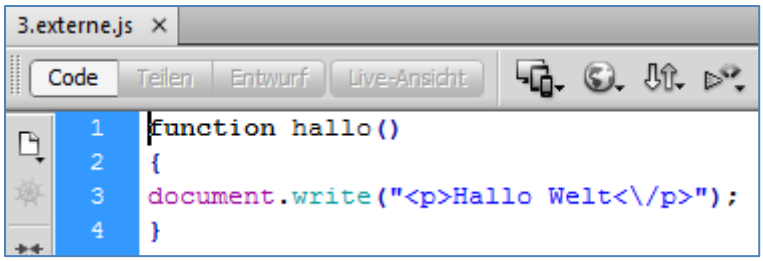
Beispiele:

```
parseFloat("10");         // returns 10
parseFloat("10 years");   // returns 10
parseFloat("years 10");   // returns NaN
```


## Tipp: auslagern in eine externe js-Datei

Die Definition von Funktionen, die man häufig und in verschiedenen Programmen benötigt, kann man in externe Dateien auslagern. So kann man eigene Funktionsbibliotheken erzeugen. Große Bibliotheken mit vielen nützlichen Funktionen, wie zum Beispiel die Bibliothek **jQuery**, wurden auf diese Weise erschaffen.

**Übung:** Erstelle eine neue Datei mit folgendem Inhalt:




Speicher diese als JavaScript-Datei als „2_externe.js“

In der „normalen“ HTML-Datei muss nun die Verbindung aufgenommen werden:
Der script-Container zur Einbindung der externen Datei steht im head des Dokuments. Das Attribut src verweist auf die externe Datei. Die Ausgabe sieht genauso aus wie im vorherigen Abschnitt.

```
<head>
<meta charset="UTF-8">
<title>Funktion Einstieg - externe JS</title>
<script src="2.externe.js">
</script>
</head>
<body>
<script type="text/javascript">
    hallo();
    hallo();
</script>
</body>
```

Im Head und im Body ist <script> und </script> nötig.
Speichern unter „2.funktion_extern.html“