JavaScript {

```
verfasser = "Thomas Maier";
inhalt[0] = "Unterrichtsplanung";
inhalt[1] = "Lernhandouts";
inhalt[2] = "Übungshandouts";
inhalt[3] = "Zielformulierungen";
```

}

2024

JavaScript
von Thomas Maier

Einstieg

JavaScript ist äußerst genügsam was die Leistung der Hard- und Software betrifft. Im Grunde reicht ein gängiger PC mit einem Editor und einem Browser. Das Betriebssystem spielt nur eine untergeordnete Rolle, da JavaScript plattformunabhängig ist. Ob nun Windows, OSX oder Linux – egal! Hier eine Darstellung der benötigten Software.

- ❑ Ein schneller Texteditor
- ❑ Eine umfangreiche IDE-Software (HTML Editor)
- ❑ sämtliche gängige Browser
- ❑ Eine Textverarbeitungssoftware
- ❑ weitere Tools: Color Picker, ZIP Software, Screenshot-Tools, PDF Viewer (Acrobat odgl.)
- ❑ kolaborative Anwendungen: z. B. Office 365, OneNote, Google, TeamViewer
- ❑ didaktische Tools: iTalc, Veyon, Moodle
- ❑ FTP Software (z. B. FileZilla)
- ❑ Virtualisierungssoftware
- ❑ CMS Editoren (z. B. Adobe Muse, Joomla, Wordpress)
- ❑ Entwicklungsumgebungen (z. B. Visual Studios)
- ❑ Bildbearbeitungssoftware
- ❑ **Audio und Videobearbeitungssoftware**

Eine Webanbindung des Schulservers (z. B. über IIS, Apache odgl.) wäre sicher ein Pluspunkt, damit die Studierenden ihre JavaScripte sofort online stellen können. Die Planung eines Webservers ist aber ein anderes Thema, das den Rahmen dieser Arbeit sprengen würde. Die Studierenden können aber ihre Lernprodukte auf gratis Webspace Anbietern, wie z. B. mur.at oder Lima-City.de publizieren (wenn nötig und wenn sie es wollen).

Beispiele für Basisinstallationen

		
Microsoft Windows	**Apple OSX**	**Linux**
• XAMPP (WAMPP) • Notepad ++ • Adobe Dreamweaver • ATOM • Edge, Chrome, Firefox • MS Office • MS Access • Color Picker • 7Zip • Snipping Tool • Adobe PDF Acrobat • FileZilla • OneNote • TeamViewer • Veyon (iTalc) • Gimp bzw. Adobe Photoshop • Corel Draw, Adobe Illustrator, Inkscape • Audacity • Oracle VirtualBox • MS SQL Server	• XAMPP • TextWrangler • Adobe Dreamweaver • ATOM • Safari, Chrome, Firefox • Apple iWork • MS Access • Color Picker • 7Zip • Skitch • Adobe PDF Acrobat • FileZilla • Evernote • TeamViewer • Veyon (iTalc) • Gimp bzw. Adobe Photoshop • Corel Draw, Adobe Illustrator, Inkscape • Audacity • Oracle VirtualBox • SQL Server	• XAMPP • Sublime • Bluefish • ATOM • Chrome, Firefox • LibreOffice • Kexi • KColorChooser • 7Zip • Shutter • Adobe Reader • FileZilla • Evernote • TeamViewer • Veyon (iTalc) • Gimp • Inkscape • Audacity • Oracle VirtualBox • SQL Server

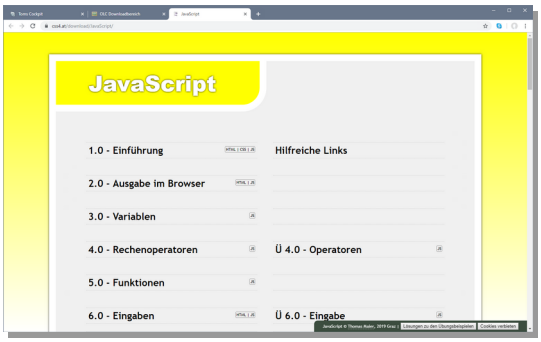
Die Hardware ist definitiv genügsamer:

- Bei Laptops muss ein externes Keyboard bereitgestellt werden. Ebenso eine Maus.
- Ein zweiter Monitor ist ein Nice-to-Have, aber nicht zwingend notwendig.
- Eine Breitband-Internet-Verbindung.
- Ein funktionierendes Netzwerk mit Serverseitigen Profilen und Netzlaufwerken.
- Audio-Ausgabe: Entweder über Lautsprecher oder besser über Kopfhörer.
- USB Anschlüsse für die Verwendung von USB-Speicher-Sticks.
- Ein Netzwerk-Drucker
- Mobile Devices (Smartphone und Tablets) um die Scripte zu testen

Abbildung: Die Bannernavigation von www.css4.at

Auf der Lernplattform www.css4.at befindet sich links oben ein Suchfeld. Über dieses kann man nach notwendigen Dateien, Suchbegriffen und Refcodes suchen.

Auf der Startseite befindet sich der Lernbereich als dunkelblauer Kasten. Dort findet man unter Web Development einen Link zur Lernplattform: JavaScript!



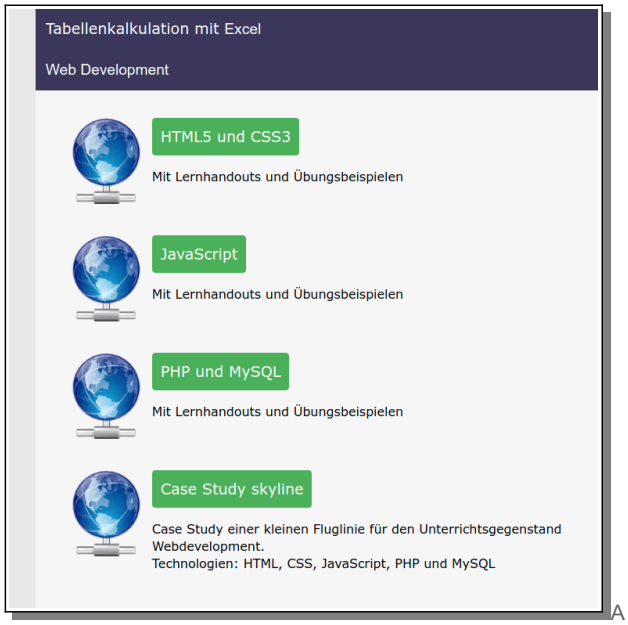


Abbildung: Im Lernbereich

Die Lernplattform für JavaScript erlaubt zu jeder Einheit die Downloads der Lern- und Übungshandouts und der benötigten Dateien für die Übungsbeispiele.

Zusätzlich findet man auf der Lernplattform noch hilfreiche Links zum Unterricht. In der linken Spalte findet man die Lernhandouts und korrespondierend rechts die dazu passenden Übungshandouts.

Handouts und Assignments

Die Handouts sind in Einheiten gegliedert. Jede Einheit beschreibt einen Themenkomplex aus dem jeweiligen Bezugssystem. Grundsätzlich unterliegen die Einheiten einer Progressionslogik, können aber auch voneinander unabhängig unterrichtet werden (je nach pädagogischen Bedarf).

Alle Handouts wurden nach dem Prinzip des "One-Page-Management" erstellt und können auch als Kopiervorlage genutzt werden. Die Handouts wurden mit Screenshots bebildert. Die Anleitungen sind auf die Übungsbeispiele hin konzipiert.

Die Symbole auf den Handouts sollen den Studierenden eine Ordnung im Lernprozess vermitteln.

 ⇦ ***Übungsbeispiel***

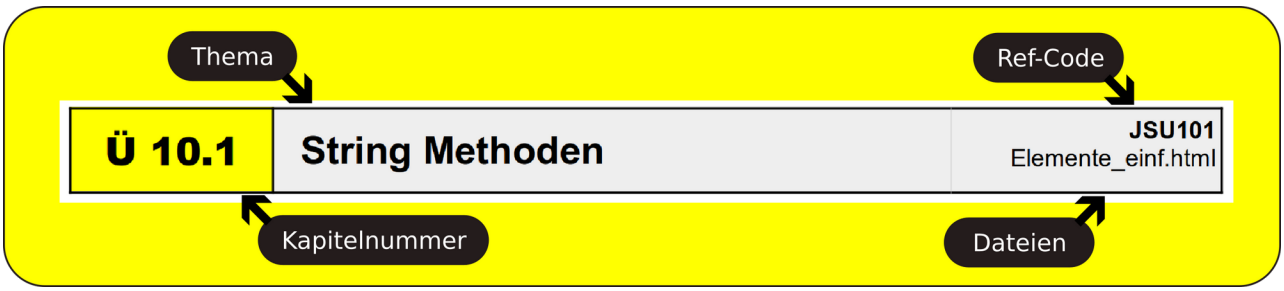
 ⇦ ***Information und Hervorherbung***

 ⇦ ***Nachschlagen und Bibliothek***

 ⇦ ***Tools und Werkzeuge***

 ⇦ ***Notizen, Codebeispiel und Anmerkungen***

Dieses One-Page-Management wurde der Wirtschaft entnommen. Es ist eine Methode um Führungspersönlichkeiten einen raschen Überblick über die aktuellen Fakten zu geben. Rasch und Einfach – was nun also für Manager_innen von großen Betrieben stimmig ist, sollte für den Schüler bzw. die Schülerin genauso gelten. Ein neues Thema – ein neues Blatt Papier.



Über die Kapitelnummer werden Themenbereiche gegliedert. Übungsblätter haben zusätzlich noch ein "Ü" bei der Kapitelnummer. Der Ref-Code identifiziert ein Handout eindeutig und kann im Suchfeld von www.css4.at abgerufen werden. Dateien, die für das Thema notwendig sind, findet man als rechts, unter dem Ref-Code. Die Dateien können von www.css4.at herunter geladen werden! In dieser Arbeit folgt immer zuerst die didaktische Konzeption (Feinzielen und Anmerkungen) und im Anschluss daran das Lernhandout bzw. Übungsblatt.

1.0 Einführung

Da die meisten Browser mit der Programmiersprache JAVA erstellt wurden, erlauben diese eine Schnittstelle um mit JavaScript zusätzliche Befehle einer Webseite abzuwickeln.Wie schon im Namen JavaScript steht, handelt es sich dabei nicht um eine Programmiersprache sondern eine Scriptsprache die clientseitig abgearbeitet wird. Im Vergleich zu PHP, welches serverseitig abgearbeitet wird, ist der Quellcode für alle lesbar.

Für die Arbeit mit JavaScript benötigen wir einen Texteditor (z. B. Notepad++, oder Dreamweaver, Atom) und einen Browser (z. B. Firefox, Chrome). JavaScript harmoniert wunderbar mit HTML und CSS, deshalb werden wir es in einem HTML-Konstrukt verwenden.

HTML Konstrukt mit **<script>** Tag im Body

```
<!doctype html>
  <html lang="de">
    <head>
      <meta charset="utf-8">
      <title>Titel</title>
      <meta name="author" content="Thomas Maier">
      <meta name="date" content="2022-11-10">
      <style> ... </style>
    </head>
   <body>
     <script> ... </script>
     <noscript> ... </noscript>
   </body>
  </html>
```

HTML

<script> ... </script>
<noscript> ... </noscript>

Innerhalb des **<script>** Tag wird der JavaScript Code geschrieben.
Das **<script>** Element kann auch im **<head>** stehen.

Innerhalb des **<noscript>** Tag befindet sich jener Content, der angezeigt wird wenn JavaScript in den Browsereinstellungen deaktiviert ist.

z. B. **<noscript> <h2>** JavaScript aktivieren! **</h2> </noscript>**

Eine Vorlage erstellen

- ❑ Starte einen Texteditor (z. B. Notepad++, Dreamweaver, Atom odgl).
- ❑ Schreibe das HTML Konstrukt wie oben dargestellt.
- ❑ Schreibe in den Meta-Tag **author** deinen Vor- und Nachnamen.
- ❑ **Speichere den Code als `vorlage.html` ab.**

Wir werden die **`vorlage.html`** öfters brauchen. Also speichere das File so ab, damit es jederzeit und schnell gefunden wird. (z. B: USB-Stick, Cloud, Netzlaufwerk). Bei jeder neuen Übung sollte das aktuelle Datum und ein Titel eingetragen werden!

2.0 Ausgabe im Browser

Die JavaScript Befehle werden innerhalb des **<script>** *Elements ausgeführt.*
Für die Ausgabe von Text gibt es vier Möglichkeiten.

1: **window.alert();** *2:* **document.write();**
3: **.innerHTML = "";** *4:* **console.log();**

Alle Anweisungen in JavaScript enden mit einem Semikolon ;
Der Text befindet sich innerhalb von zwei Anführungszeichen **"text"**

JS

`window.alert();`

Die Ausgabe erfolgt über ein Dialogfenster. Das Aussehen des Dialogfensters ist vom Browser abhängig, in den meisten Fällen gibt es eine OK Schaltfläche.
alert() ist eine Methode des Window Objekts.

```
<script>
   window.alert("Hallo Welt");
</script>
```

JS

`document.write();`

Schreibt an der Stelle im Textfluss wo sich das **<script>** Element befindet.
Man kann HTML Code in dieser Ausgabe mitgeben:
z. B: **document.write("<h1>Hallo Welt</h1>");**

```
<div> <script>
         document.write("Hallo Welt");
      </script> </div>
```

JS

document.getElementById().innerHTML = "";

Schreibt in ein bestimmtes HTML Element, das mit einer ID versehen ist.
Sollte im HTML Element ein Content sein, wird dieser überschrieben!
.getElementById("") ist ein Selektor. **.innerHTML** ist die Methode!

```
<p id="ausgabe">  </p>

<script>
  document.getElementById("ausgabe").innerHTML = "Hallo Welt";
</script>
```

JS

console.log();

Für die Fehlersuche (debugging) kann man eine Ausgabe in die Konsole des Browsers anweisen.
Die Konsole befindet sich in den Entwicklertools (z. B. bei Firefox).

```
<script>
   console.log("Hallo Welt");
</script>
```

3.0 Variablen

Mit Variablen erzeugt man eine Art von Behälter der einen Wert haben kann. Gleich wie im Mathematikunterricht, wo x und y einen Wert haben, sind Variablen in JavaScript mit Zahlenwerte, Text (Strings) oder Objekten zuzuweisen. Eine Variable wird mit **`var`** *vereinbart (erstellt, man spricht auch von deklarieren, dimensionieren).*

JS

```
var Variablenname;
```

Der *Variablenname* ist case-sensitve (Groß- und Kleinschreibung beachten) und sollte sich vom JavaScript-Code klar unterscheiden. Am besten verwendet man deutsche Worte.

Ohne Zuweisung, wird eine Variable als `undefined` typisiert und kann damit alle Werte übernehmen. z. B. `var flaeche;`

```
var ausgabe = "Das Ergebnis: ";   flaeche = breit * hoch;
```

Eine Wertzuweisung erfolgt immer von rechts nach links.

```
<script>
   var breit = 15;
   var hoch = 5;
   var flaeche;
   var ausgabe = "Das Ergebnis: ";
   flaeche = breit * hoch;    //← Der Stern * steht für eine Multiplikation
   document.write(ausgabe);
   document.write(flaeche);
</script>
```

Um nicht den Überblick zu verlieren, sollte man die Variablen immer ganz oben im Code deklarieren.

JS

```
const Konstantenname = Wert;
```

Mit **`const`** wird eine Konstante (ein Wert der sich nicht verändert) definiert.

```
const euler = 2.718;   //← Der Punkt steht für ein Dezimalzeichen.
const Pi = 3.142;
```

Mit Kommentaren kann man den Code übersichtlicher gestalten. Sie sind manchmal auch recht nützlich bei der Fehlersuche, weil man damit einen Codeabschnitt auskommentieren kann.

Einzeilige Kommentare werden mit **`//`** definiert

Mehrzeilige Kommentare werden mit **`/* ... */`** definiert.

```
// Berechnungen
/* Ausgabe im Dokument
   document.write(ausgabe); */
```

4.0 Rechenoperatoren

Bei einer mathematischen Operation in JavaScript gelten auch die Rangfolgen:
Zuerst die Berechnung einer Klammer und dann Punkt- vor Strichrechnung.

Operation	Zeichen Operator	Beispiel
Addition	**+**	`i = i + 1;`
Subtraktion	**-**	`var drei = 6 - 3;`
Multiplikation	*****	`x = y * 3;`
Division	**/**	`zinssatz = 8 / 100;`
Modulo (Rest einer Division)	**%**	`var rest = 5 % 3;` Lösung: `rest = 2`

Script zur Berechnung der Umsatzsteuer, wenn man den Bruttowert angibt.

```
var brutto = 240;
var USt = 20;
var steuer = brutto / (100 + USt) * USt;
document.write("Die Umsatzsteuer beträgt € ");
document.write(steuer);
```

Ein einfaches Pluszeichen + dient auch zur Verkettung eines Strings (Text).

```
var ausgabe = " bei einem Steuersatz von " + USt + " Prozent";
document.write(ausgabe);
```

JS

Math.

Das Standard Objekt **Math.** bietet weitere Methoden wie z. B:

Methode		Beispiel	
Potenz	`Math.pow( , );`	`quadrat = Math.pow(4,2);`	Lösung: 16
Wurzel	`Math.sqrt();`	`wurzel = Math.sqrt(9);`	Lösung: 3
Zufallszahl	`Math.random();`	`var zufall = Math.random();`	
Runden	`Math.round();`	`b = Math.round(12.6);`	Lösung: 13

Berechnung des Flächeninhalts eines gleichseitigen Dreiecks.

$$A = \frac{a^2}{4} \times \sqrt{3}$$

```
var a = 8;
var flaeche = Math.pow(a , 2) / 4 * Math.sqrt(3);
var ausgabe = "Der Flächeninhalt beträgt " + flaeche;
document.write(ausgabe);
```

Ü 4.0 Operatoren

Übung A: Variablentypen

- ❑ Ergänze die Tabelle um die korrekte Erstellung von Variablen.
- ❑ Finde passende Variablennamen zum Typ.
- ❑ Erfinde passende Werte zum Typ.
- ❑ **Solltest du einen Typ nicht kennen, dann recherchiere im Internet.**

Typ	Erstellung (bzw. Vereinbarung)
Fließkomma	`var fZahl = 4.334;`
Integer	
String	
Boolean	
Undefined	

Übung B: Fahrenheit und Celsius

- ❑ Übersetzte die mathematische Formel der Umrechnungen von Celsius in Fahrenheit in einen JavaScript Code.
- ❑ Deklariere die notwendigen Variablen.
- ❑ Speichere den Code in ein HTML Dokument.
- ❑ **Teste den Code: 45 °C sind 113 °F**

Übung C: Mitternachtsformel

- ❑ Übersetzte die a-b-c Formel zur Lösung von quadratischen Gleichungen in einen JavaScript Code.
- ❑ Deklariere die notwendigen Variablen.
- ❑ Speichere den Code in ein HTML Dokument.
- ❑ Runde die Ergebnisse von x_1 und x_2 auf zwei Kommastellen!
- ❑ **Teste den Code: a = 3, b = 5, c = 1 → x_1 = -0.23, x_2 = -1.43**

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

5.0 Funktionen

Eine Funktion kann man mit einem Unterprogramm im Quellcode vergleichen. Die Funktion wird nicht automatisch ausgeführt, sondern muss durch einen Funktionsaufruf gestartet werden. Sie hat keinen, einen oder mehrere Übernahmeparameter und meist einen oder keinen Rückgabewert. Funktionen können auch im `<head>` *stehen.*

JS

```
function Funktionname (Parameter1, Parameter2, ...)
    {Anweisungen ...; return;};
```

Eine Funktion beginnt immer mit dem Befehl **`function`**.

Der Funktionsname soll eindeutig sein und sich von anderen Funktionen unterscheiden. Der Funktionsname ist case-sensitive.

Die Parameter sind lokale Variablen, die in der Funktion abgearbeitet werden.

Der Anweisungsblock steht innerhalb von zwei geschwungenen Klammern { }. Jede Anweisung endet mit einem Semikolon ;

Mit **`return`** übergibt man einen Wert oder beendet die Funktion!
z. B. **`return ausgabe;`** oder **`return true;`**

```
<script>
  function geklickt(welcher) {
    window.alert("Button " + welcher + " wurde geklickt!");
    return;  }
</script>

<button onClick="geklickt('Absenden');" >Senden</button>
<button onClick="geklickt('Loeschen');" >Löschen</button>
```

Das Attribut `onClick=""` im HTML Element `<button>` führt JavaScript Code aus, wenn man auf das Element klickt. Hier wird die Funktion `geklickt()` mit einem String (`Absenden` bzw. `Loeschen`) aufgerufen.

Der Funktionsaufruf `geklickt('Absenden');` wurde mit einfachen Anführungszeichen für die Wertübergabe geschrieben, weil das `onClick=""` Attribut die doppelten Anführungszeichen benötigt.

Eine eigene Rundenfunktion mit der Übergabe einer Fließkommazahl und der Anzahl der Stellen.

```
<script>
  function runden(zahl, stellen) {
      zahl = zahl * Math.pow(10, stellen);
      zahl = Math.round(zahl) / Math.pow(10, stellen);
      return zahl;}
  var ausgabe = runden(34.3862345, 3); // <-- Start der Funktion
  window.alert(ausgabe);
</script>
```

Der Funktionsaufruf wurde mit einer Variablen-Vereinbarung aufgerufen.
Die lokale Variable `zahl` wurde abgearbeitet und mit `return` an die Variable `ausgabe` übergeben.

6.0 Eingabe

Damit der User etwas auf einer Webseite eingeben kann, gibt es zwei praktikable Möglichkeiten. Erstens, eine Eingabe mit der **`window.prompt();`** *Methode oder eine Eingabe über* **`<input>`** *Elemente, die mit* **`document.getElementById().value;`** *abgerufen werden.*

In der Programmierung gilt das EVA-Prinzip. EVA steht für Eingabe – Verarbeitung - Ausgabe. Das gilt sowohl für Programmier- als auch für Scriptsprachen.

JS

`window.prompt("text", "default text");`

Der Browser stellt ein Eingabe-Dialog-Fenster zur Verfügung. **`prompt();`** ist eine Methode des Window-Objekts. Der User kann etwas eingeben, mit OK bestätigen und diese Eingabe wird dann einer Variable zugewiesen.

text	Ein Text der den Abfrage Dialog begleitet.
default text	Ein Text der im Eingabe Feld steht (Placeholder).

```
<script>
   var ort = window.prompt("Ihr Wohnort?", "Wien");
   document.write("Wir senden das Paket nach " + ort);
</script>
```

Mit **`\n`** erzwingt man eine Zeilenschaltung.
Im HTML Code sollte man jedoch **`<br>`** verwenden.

```
window.prompt("Willkommen \n Bitte gib deinen Vornamen ein!");
```

JS

`document.getElementById().value;`

Ermittelt den Wert eines HTML Elements (z. B. **`<input>`**, **`<textarea>`** usw.) über seine eindeutige ID. Es ist Sinnvoll, die Abfrage in eine Funktion zu schreiben und über einen Button zu starten.

```
<script>
  function deinName()
  {
    var ausgabe = document.getElementById("derVorname").value;
        ausgabe = "Hallo " + ausgabe;
    document.getElementById("dieAusgabe").innerHTML = ausgabe;
    return;
  }
</script>

<input id="derVorname" placeholder="Dein Vorname?">
<button onClick="deinName();">Start</button>
<p id="dieAusgabe"></p>
```

Ü 6.0 Eingabe

Übung A: Pythagoras

Im rechtwinkligen Dreieck gilt für die drei Seiten: **$a^2 + b^2 = c^2$**
Der Benutzer gibt die Werte von a und b ein und erhält die Länge von c.
Erstelle auch einen Button zum Starten der Funktion.

- ❑ **HTML** Zwei Eingabefelder, ein Button, ein Ausgabefeld
- ❑ **JS** Verarbeitung in einer Funktion
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ Teste deine Seite [a = 4, b = 3, c = 5]

Übung B: Zinseszins

Schreibe ein Skript, welches durch Eingabe von Anfangskapital, Zinssatz und einer Laufzeit in Jahren das **Endkapital** einer Spareinlage errechnet und ausgibt.

Die Formel für eine Zinses-Zins-Rechnung lautet:

- ❑ **HTML** Drei Eingabefelder, ein Button und ein Ausgabefeld
- ❑ **JS** Verarbeitung in einer Funktion
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ Teste deine Seite
  $K_0 = 1200$, $p = 4$, $n = 5$, $K_n = 1459{,}98$

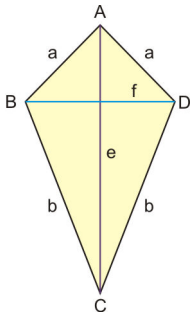
$$K_n = K_0 \times \left(1 + \frac{p}{100}\right)^n$$

```
Kn _____ Endkapital
K0 _____ Anfangskapital
p  _____ Zinssatz (z. B. 4 %)
n  _____ Laufzeit in Jahren
```

Übung C: Deltoid

Auf einer Webseite sollen die Längen von a, b, und f eines Deltoids eingegeben werden. Ausgegeben werden soll der Flächeninhalt.

- ❑ **HTML** Drei Eingabefelder, ein Button, ein Ausgabefeld
- ❑ **JS** Verarbeitung in einer Funktion
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ Teste deine Seite [a = 5, b = 8, f = 6, A = 34,25]



$$A = \frac{e \cdot f}{2}$$

$$e = \sqrt{a^2 - \left(\frac{f}{2}\right)^2} + \sqrt{b^2 - \left(\frac{f}{2}\right)^2}$$

6.1 Ereignis Attribute

Ereignis Attribute (event attributes) werden für den Aufruf von JavaScript (Funktionen oder direkt ein Code) verwendet. Achte dabei auf die Anführungszeichen.

```
<body onLoad="window.alert('Servus');">
<button onClick="MeineFunktion();">Absenden</button>
```

Window Events	**onBeforePrint**	← *Bevor ein Dokument gedruckt wird.*
	onError	← *Wenn ein Fehler auftritt.*
	onLoad	← *Nachdem ein Dokument/Objekt geladen wurde.*
	onResize	← *Wenn sich die Größe des Browserfenster verändert.*
	onUnload	← *Nachdem ein Browserfenster geschlossen wurde.*

```
<body onResize="WieHoch();">
<p>Die Viewport-Höhe: <span id="brz"></span> Pixel</p>
<script>
   function WieHoch() {
       var hoch = window.innerHeight;
       document.getElementById("brz").innerHTML = hoch;
   }
</script>
```

window.innerHeight *ermittelt die Höhe des Viewports in Pixel.*

Form Events	**onChange**	← *Wenn sich der Wert verändert.*
	onFocus	← *Wenn der Fokus auf dem Element ist.*
	onSelect	← *Wenn ein Text in der Eingabe markiert wird.*
	onBlur	← *Wenn man das Eingabefeld verlässt.*
	onInput	← *Bei jeder Eingabe im Eingabefeld (Tastendruck).*
Keyboard und Mouse Events	**onKeyDown**	← *Bei einem Tastendruck.*
	onKeyUp	← *Nach einem Tastendruck.*
	onClick	← *Bei einem Mausklick (linke Maustaste).*
	onDBLclick	← *Bei einem Doppelklick (linke Maustaste).*
	onMouseOut	← *Mauszeiger verlässt ein Element.*
	onMouseOver	← *Mauszeiger bewegt sich über dem Element.*
	onWheel	← *Wenn das Mausrad gedreht wird.*

```
<input type="password" id="Passwort" onKeyUp="zeige();">
<p id="Anzeige">Hier ist das Passwort lesbar!</p>
<script>
   function zeige(){
      var ausgabe = document.getElementById("Passwort").value;
      document.getElementById("Anzeige").innerHTML = ausgabe;
   }
</script>
```

7.0 Verzweigung

Eine Verzweigung besteht aus Bedingungen und Anweisungen. Wird eine Bedingung erfüllt (true), dann werden auch die Anweisungen ausgeführt. Wird die Bedingung (bzw. die Bedingungen) nicht erfüllt (false), dann werde die Anweisungen des Else Block ausgeführt, wenn ein solcher vorhanden ist.

JS

`if(Bedingung) {Anweisung; Anweisung; Anweisung; usw.}`
`else {Anweisung; Anweisung; Anweisung; usw.}`

Für die Definition einer Bedingung gibt es Vergleichsoperatoren.
Die Operatoren **== istgleich** und **!= ungleich** können für alle Typen (Text, Boolean, Nummer usw.) verwendet werden.

```
var eingabe = window.prompt("Wie viel ist 7 x 8?");
if (eingabe == 56) {window.alert("Richtig"); }
else {window.alert("Falsch"); }
```

Operator	Bedeutung	Beispiel
==	istgleich	`if(x == 2) {window.alert("Zwei gewählt");}`
!=	ungleich	`if(y != 10) {x = x / 100;}`
>	größer	`if(eingabe > 50) {document.write(eingabe);}`
>=	größergleich	`if(breite >= 1024) {window.alert("Zu groß");}`
<	kleiner	`if(wert < 0) {ausg = "Keine negativen Zahlen";}`
<=	kleinergleich	`if(breite <= 1024) {schalter = true;}`

Beispiel: Die größere von zwei Zahlen ermitteln.
`Number()` *wandelt die Eingabe in eine Zahl!*

```
var benutzer = window.prompt("Welcher Benutzer?", "User");
var zahl_1 = Number(window.prompt("Erste Zahl eingeben!"));
var zahl_2 = Number(window.prompt("Zweite Zahl eingeben!"));

if(zahl_1 > zahl_2) {var ausgabe = "Erste Zahl ist größer";}
else {var ausgabe = "Zweite Zahl ist größer";}

if(zahl_1 == zahl_2) {var ausgabe = "Beide Zahlen sind gleich";}
window.alert(ausgabe);
```

Es können auch mehrere Bedingungen auf einmal definiert werden. Dazu verbindet man die Bedingungen mit den logischen Operatoren && oder ||
&& bedeutet UND also alle Bedingungen müssen erfüllt werden
|| bedeutet ODER also eine einzige Bedingung reicht aus.

```
if (benutzer == "User" && zahl_1 == zahl_2)
   {window.alert("User hat zwei gleiche Zahlen eingegeben!");}
if (benutzer == "Admin" || benutzer == "Root")
   {window.alert("Egal, Root bzw. Admin darf sowieso alles!");}
```

Ü 7.0 Verzweigung

Übung A: Frau oder Mann

In einem Formular wird das Geschlecht (männlich oder weiblich) über Radio-Buttons abgefragt. Zusätzlich noch der Nachname. Bei weiblich wird "Sehr geehrte Frau" und bei männlich "Sehr geehrter Herr" mit dem Nachnamen ausgegeben.

- ❑ **HTML** zwei Radiobuttons, ein Eingabefeld, einen Button, ein Ausgabefeld
- ❑ **JS** eine Funktion mit Verzweigung
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)

HTML

```
<input type="radio" id="mann" name="geschlecht" >
<input type="radio" id="frau" name="geschlecht" checked >
```

JS

```
var Mann = document.getElementById("mann").checked;
```

Auswertung eines Radiobuttons. `Mann` *ist eine boolesche Variable.*
Ist ein Radiobutton aktiviert, wird `true` *an die Variable übergeben.*

Übung B: Rabattberechnung

Ein Olivenöl-Hersteller vermarktet sein Öl über das Internet. Die 1-Liter Flasche kostet € 12,40. Bestellt man 8 oder mehr Flaschen, bekommt man 5 % Rabatt. Bestellt man 16 oder mehr Flaschen, bekommt man 12 % Rabatt auf die Gesamtsumme.

- ❑ **HTML** ein Eingabefeld, einen Button, ein Ausgabefeld
- ❑ **JS** eine Funktion mit Verzweigung, Berechnung der Gesamtsumme und des Rabatt.
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)

Übung C: Systemvoraussetzungen

Ein Softwarehersteller hat drei Versionen der 3D Anwendung "Threedim" auf den Markt gebracht. Jede Version hat unterschiedliche Systemvoraussetzungen.

- Threedim Lite (RAM: 2 GiB, Prozessor: 1.8 GHz, Festplatte: 30 GiB)
- Threedim Full (RAM: 4 GiB, Prozessor: 2 GHz, Festplatte: 40 GiB)
- Threedim Prof (RAM: 16 GiB, Prozessor: 3 GHz, Festplatte: 120 GiB)

❑ Erstelle eine Webseite mit der Eingabe für die Größe von RAM, Prozessor und Festplatte. Ein Script soll dann die möglichen Software-Versionen ermitteln und ausgeben.

- ❑ **HTML** Drei Eingabefelder, ein Button
- ❑ **JS** eine Funktion mit Verzweigungen
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)

7.1 switch case

Alternativ zu einer IF-Else Verzweigung gibt es die Switch Case Verzweigung. Dabei wird mit **`switch`** *eine Variable ausgewählt und über* **`case`** *die Bedingung definiert. Die* **`case`** *Anweisungen werden mit* **`break;`** *beendet.*

JS

```
switch(var) {
   case 1: Anweisung;
           Anweisung;
           break;
   case 2: Anweisung;
           Anweisung;
           break;
 default:  Anweisung;
           break;}
```

Nach `case` wird der Wert zur Bedingung angeführt und mit einem Doppelpunkt abgeschlossen. Mit `break;` beendet man den Anweisungsblock. `default:` wird dann ausgeführt, wenn keine andere `case`-Bedingung übereinstimmt.

Beispiel: Produktionsstätten von deutschen Automarken

```
<script>
  var prodOrt = "";
  function meineFunktion() {
    var dasAuto = document.getElementById("deAuto").value;

    switch(dasAuto) {
       case "BMW":
         prodOrt = "München"; break;
       case "VW":
         prodOrt = "Wolfburg"; break;
       case "Opel":
         prodOrt = "Rüsselsheim am Main"; break;
       case "Porsche":
         prodOrt = "Stuttgart"; break;
       default: prodOrt = "Gültige Automarke eingeben!"; break;
    }
    document.getElementById("ausgabe").innerHTML = prodOrt;
  }
</script>

<h3>Geben Sie bitte eine deutsche Automarken ein!</h3>
<input type="text" id="deAuto" onKeyUp="meineFunktion();" >
<p id="ausgabe"></p>
```

Variablen die in einer Funktion vereinbart werden, sind auch nur innerhalb der Funktion gültig. Variablen außerhalb, sind globale Variablen und stehen jeder Funktion zur Verfügung.

Im Beispiel oben ist: **`var prodOrt = "";`** *eine globale Variable.*

Ü 7.1 switch case

JSU071

Übung A: Postleitzahl

Noch wärend der Eingabe einer Postleitzahl soll das dazupassende Bundesland angezeigt werden. Die erste Ziffer einer Postleitzahl steht für das Bundesland bzw. die Leitzone. **Quelle:**
https://de.wikipedia.org/wiki/Postleitzahl_(Österreich)

- ❑ **HTML** ein Eingabefeld (passendes Ereignisattribut), ein Ausgabefeld
- ❑ **JS** **eine Funktion mit switch case**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

1xxx	Wien (auch teilweise für benachbarte Orte in Niederösterreich)
2xxx	östliches und südliches Niederösterreich, auch teilweise für Nordburgenland
3xxx	westliches Niederösterreich und kleine Teile südöstliches Oberösterreich
4xxx	Oberösterreich und kleine Teile westliches Niederösterreich
5xxx	Salzburg und westliches Oberösterreich
6xxx	Nordtirol und Vorarlberg
7xxx	Burgenland, soweit es nicht unter 2xxx oder 8xxx fällt
8xxx	Steiermark, Teile des Bezirks Bezirk Jennersdorf im Südburgenland
9xxx	9xxx – Kärnten und Osttirol

Übung B: Gegenüberstellung if vs. switch

Erstelle eine Webseite mit einer Gegenüberstellung (z. B. Vor- und Nachteile, Syntax, Schreibarbeit, Übersichtlichkeit usw) von
`IF ELSE` vs. **`SWITCH CASE`**

Recheriere im Internet und baue eigene Überlegungen mit ein.

- ❑ **HTML** **HTML Dokument**
- ❑ **JS** **Kein Code notwendig**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

IF ELSE	VS.	SWITCH CASE

8.0 while Schleifen

Mit einer Schleife wiederholt man einen Anweisungsblock. **`while`** *Schleifen eignen sich für Durchläufe, deren Anzahl noch nicht bekannt ist.*

Kopfgesteuerte Schleife

JS

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

Für die Bedingungen gelten die selben Regeln wie bei einer IF-ELSE Verzweigung. Siehe 7.0 Verzweigung (Vergleichsoperatoren). Die Anweisungen werden solange wiederholt, solange auch die Bedingung erfüllt ist (true).

```
<p>Du musstest <span id="ausgabe"></span> mal raten!</p>
<script>
  var eingabe = "";
  var versuch = 0;
  while (eingabe != "Innsbruck") {
         eingabe = window.prompt("Hauptstadt von Tirol?");
         versuch++;
         document.getElementById("ausgabe").innerHTML = versuch;}
</script>
```

***`versuch++;`** ist ein Zähler. Dabei wird pro Durchgang eines hinzuaddiert. Es entspricht einem* **`versuch = versuch + 1;`**

Fussgesteuerte Schleife

JS

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

Der Anweisungsblock wird einmal sicher durchlaufen. Am Ende werden die Bedingungen mit `while ();` definiert, die ein weiteres durchlaufen bestimmen sollen. Wird die Bedingung erfüllt (true), dann wird der Anweisungsblock nochmals ausgeführt.

```
var eingabe = "";
var zaehler = 0;
do {eingabe = window.prompt("Mathequiz: 4 x 3 = ?");
    zaehler ++;
    if (eingabe == 12) {
          window.alert("Bravo! " + zaehler + " Versuche");
          break;}     }
while (eingabe != 12);
```

Mit **`break;`** *verlässt man sofort eine Schleife!*

***!!! ACHTUNG !!!** Wenn die Bedingungen falsch gesetzt sind, kann eine Endlosschleife entstehen. Hier ein Beispiel für eine Endlosschleife:*

```
var zaehler = 5;
do {zaehler++; window.alert(zaehler);}
while (zaehler > 5);
```

8.1 for Schleifen

Wenn man abschätzen kann, wie viele Durchgänge (Wiederholungen bzw. Iterationen) ein Anweisungsblock zu durchlaufen hat, dann verwendet man am Besten for Schleifen. Diese haben eine Zählvariable, eine Bedingung für die Zählvariable und eine Zählvariante die entweder höher oder niedriger wird.

JS

```
for (Zählvariable; Bedingung; Zählvariante)
    {Anweisungen;}
```

Zählvariable eine Zählvariable wird mit **var** vereinbart und definiert den Startwert z. B. **var i = 5;**
Die Zählvariable kann man für die Anweisungen verwenden.

Bedingung für die Zählvariable wird eine Bedingung mit Vergleichsoperatoren definiert. z. B. **i <= 50;**

Zählvariante um welchen Betrag sich die Zählvariable verändert. Der Betrag kann größer oder kleiner werden – abhängig von der Zählvariante. z. B. **i++** oder **i = i - 1**

```
var zahl = window.prompt("Bis zu welcher Zahl?");
for (var i = 1; i <= zahl; i++) {
     document.write(i + "<br>");
}
```

Verändert die Schriftgröße um plus 2 Pixel pro Durchlauf. Dazu brauchen wir ein HTML Element welches unser Vorhaben schön darstellt:
`<p style="font-size: 10px;">Schrift mit 10 Pixel Größe</p>`

```
const anfang  = '<p style="font-size:';
const mitte   = 'px">Schrift mit ';
const schluss = ' Pixel Größe</p>';
for (var i = 10; i <= 46; i = i + 2) {
     document.write(anfang + i + mitte + i + schluss);
}
```

Beispiel: Buchstabe für Buchstabe

```
<script>
   var meinText = window.prompt("Überschrift?");
   for (var i in meinText) {
      document.write('<p>' + meinText[i] + '</p>');
   }
</script>
```

Durch die Bedingung **(var i in meinText)** *wandert die Zählschleife Buchstabe für Buchstabe den String der Variable* **meinText** *ab (von links nach rechts).*
Ausgegeben wird der Buchstabe an der Position **i → meinText[i]**
Mehr dazu in der Einheit über Arrays.

Ü 8.1 Schleifen

Übung A: Geschlecht eingeben

Eine Benutzerin oder ein Benutzer wird nach ihrem bzw. seinem Geschlecht gefragt. Die Frage soll solange wiederholt werden, bis sie oder er ein »w« oder ein »m« eingibt.

- ❑ **JS** **Schleife, Eingabeprompt**

Wenn man mehrere Bedingungen angeben will, dann verknüpft man diese mit &&.

Übung B: Die drei Schleifen

Schreibe ein Script, das alle ganzen Zahlen von 1 bis 10 ausgibt. Verwende dafür alle drei Schleifen (Kopf- und Fussgesteuerte, Zählschleifen).

- ❑ **HTML** **drei Ausgabefelder**
- ❑ **JS** **drei Schleifen für die Ausgabe der Zahlen 1 bis 10**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

Übung C: Primzahlen

Es soll ein Zahlenbereich (von – bis) definiert werden. Per Mausklick auf einen Button sollen alle Primzahlen innerhalb des Zahlenbereichs ausgegeben werden.

- ❑ **HTML** **Zwei Eingabefelder, ein Button, ein Ausgabefeld**
- ❑ **JS** **Funktion mit Schleife, IF-Verzweigung für die Primzahlenermittlung**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

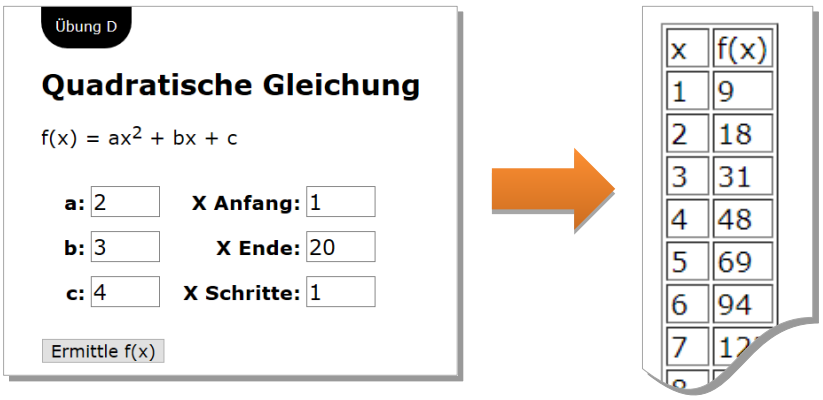
Übung D: Quadratische Funktion

Eine quadratische Funktion hat die Form **$f(x) = ax^2 + bx + c$.**

Schreibe eine Webseite mit der Eingabe für a, b, c. Damit ist die Funktion berechenbar. In Folge kann der User einen Zahlenbereich für x eingeben und eine Schrittweite für x – die dann f(x) bzw. y ausgibt.

- ❑ **HTML** **sechs Eingabefelder, eine Ausgabetabelle**
- ❑ **JS** **Funktion mit Schleife**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

Mit `parseFloat();` *kann man einen String in eine Dezimalzahl umwandeln.*
z. B. `var fx = parseFloat(y);`



9.0 Arrays

Ein Array (Feld) ist eine Variable, das mehrere Werte speichern kann. Die Positionen der Werte werden von links nach rechts ins Array vergeben. Die erste Position hat die Nummer 0, die zweite die Nummer 1 usw.
Ein leeres Array wird mit **`var arrayname = [];`** *vereinbart. Je nach Typ, muss man die Schreibweise beachten. So wird ein String mit Anführungszeichen und ein Integer-wert nur durch einen Beistrich (ohne Anführungszeichen) getrennt.*
z. B. **`var alter = [32, 28, 12, 54];`**

JS

`var arrayname = [Wert0, Wert1, Wert2];`

Die Werte können von jedem Typ sein (z. B: String, Integer usw). Den Wert ruft man mit **`arrayname[0]`** ab. Im Beispiel unten, wird Dienstag ausgegeben.

```
var tage = ["Sonntag", "Montag", "Dienstag",
            "Mittwoch", "Donnerstag", "Freitag", "Samstag"];
window.alert(tage[2]);
```

Einen Wert verändert man gleich, wie eine bekannte Variablenzuweisung.
Im Beispiel wird "Montag" durch "Wochenbeginn" ersetzt.

```
tage[1] = "Wochenbeginn";
window.alert(tage[1]);
```

Ohne Nummer, wird der gesamte Inhalt des Array angezeigt.

```
console.log(tage);
```

Natürlich kann die Nummer auch durch eine Variable gesetzt werden. Im Beispiel wird der Wochentag "Donnerstag" angezeigt, weil die Variable xy gleich 4 ist.

```
var xy = 4;
window.alert(tage[xy]);
```

Die `.length` Eigenschaft

JS

`.length;`

Die **`.length`** Eigenschaft gibt die Anzahl der Elemente eines Arrays zurück.
Mit der **`.length`** Eigenschaft kann man auf das letzte Element zugreifen.

```
window.alert(tage.length);
window.alert(tage[tage.length - 1]);
```

Die **`.length`** Eigenschaft kann auch auf eine String Variable angewandt werden.
Dabei wird die Anzahl der Zeichen im String zurückgegeben.

```
var meinText = "Wochentage";
window.alert(meinText.length);
```

Übung A: Österreichische Offiziere

Erstelle eine Webseite, in welcher ein Benutzer die Zahl einer Rangfolge eingibt und dann die Bezeichnung des Offiziersrang angezeigt wird.

Quelle: **http://www.bundesheer.at/abzeichen/dienstgrade.shtml**

- ❑ **HTML** Eingabefeld, Ausgabefeld, Button
- ❑ **JS** Funktion mit einem Array
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)

0. Fähnrich 	**6.** Oberst 
1. Leutnant 	**7.** Brigadier 
2. Oberleutnant 	**8.** Generalmajor 
3. Hauptmann 	**9.** Generalleutnant 
4. Major 	**10.** General 
5. Oberstleutnant 	

Übung B: Österreichische Offiziere II

Erweitere das Beispiel A um die Anzeige der Abzeichen. Verwende für die Eingabe einen Range Slider `<input type = "range" … >`

- ❑ **HTML** Range Slider, Ausgabefeld, Bilder
- ❑ **JS** Funktion mit Array
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)

9.1 Array Methoden

.unshift() – Vorne an Position [0] ein neues Element hinzufügen

JS `arrayname.unshift()`

```
var sport = ["Golf", "Tennis", "Fussball"];
sport.unshift("Skispringen");
window.alert("Die Sportarten: " + sport);
```

.shift() – Das erste Element löschen

JS `arrayname.shift()`

```
var wegdamit = sport.shift();
window.alert("Gelöscht wurde: " + wegdamit);
window.alert("Übrig bleibt: " + sport);
```

.push() – Ein Element am Schluss hinzufügen

JS `arrayname.push()`

```
var neuerSport = window.prompt("Neue Sportart hinzufügen");
sport.push(neuerSport);
sport.push("Tanzen");
window.alert(sport);
```

.pop() – Löscht das letzte Element aus einem Array

JS `arrayname.pop()`

```
sport.pop();
window.alert(sport);
```

.splice() – Ein Element einfügen oder löschen

JS `arrayname.splice(Position, Löschen, neue Elemente)`

Position	An welcher Stelle das neue Element eingefügt werden soll.
Löschen	Wie viele Elemente gelöscht werden sollen. Wenn man kein Element löschen will, dann gibt man eine 0 an.
neue Elemente	Die Werte, die hinzugefügt werden sollen.

```
sport.splice(2, 1, "Reiten", "Laufen", "Ice Hockey");
window.alert(sport);
```

.join() – Umwandlung in einen String mit Trennzeichen

JS `arrayname.join(" - ")`

```
var meineSportarten = sport.join(" und ");
window.alert(meineSportarten);
console.log(typeof(meineSportarten));
```

Mit **`typeof(var)`** *kann man den Typ einer Variable ermitteln.*

9.2 Arrays sortieren

Arrays lassen sich recht einfach sortieren. Dafür gibt es die **`.sort()`** *Methode bzw. die* **`.reverse()`** *Methode um die Inhalten umgekehrt aufzulisten.*

JS

arrayname**`.sort();`**

Die Arrayelemente werden alphabetisch aufwärts sortiert. Bei dieser Methode werden die Elemente wie ein String behandelt.

```
var studium = ["Philosophie", "Chemie", "Medizin", "Jus"];
studium.sort();
window.alert(studium);
```

JS

arrayname**`.reverse();`**

Dreht die Reihenfolge um, sodaß das letzte Element zum Ersten wird und das Erste zum letzten.

```
var studium = ["Philosophie", "Chemie", "Medizin", "Jus"];
studium.reverse();
window.alert(studium);
```

JS

arrayname**`.sort(function(a, b) {return a - b});`**

Da die .sort() Methode nur Strings in eine alphabetische Reihenfolge sortiert und keine Zahlen aufsteigend (z. B. 1, 2, 3, 33, 4, 456, 5 …) gibt es einen Trick um das Problem zu umgehen. Die Funktion(a, b) übernimmt zwei Werte und subtrahiert sie. Je nachdem wie das Ergebnis ausfällt (Minus, Null, Plus) wird die Zahl auch sortiert.

```
var punkte = [50, 101, 1, 5, 25, 12];
punkte.sort(function(a, b){return a - b});
window.alert(punkte);
```

JS

arrayname**`.forEach(`**eineFunktion**`);`**

Startet eine Funktion für jeweils jedes Element im Array. Übergeben wird der Wert und der Index → deshalb sollte die Funktion beide Variablen vereinbaren.

```
var zahlen = [65, 44, 12, 4];
zahlen.forEach(meineFunktion);

function meineFunktion(item, index) {
window.alert("Position: " + index + " ist der Wert " + item);}
```

Selbiges erzielt man auch mit einer **`for…in`** Schleife. Diese durchwandert jedes Element im Array. **`for…in`** kann auch auf einen String angewandt werden, dann wandert die Schleife jeden Buchstaben ab.

```
for (var index in zahlen) {
window.alert("Bei: " + index + " steht " + zahlen[index]);}
```

Ü 9.2 Arrays sortieren

Übung A: Mittelwert

Fülle ein Array mit 100 Zufallszahlen zwischen 1 und 10. Berechne den Mittelwert und ermittle die Anzahl der Zahlen, die über dem Mittelwert liegen. Sortiere das Array aufsteigend und gib alle Zahlen zusätzlich als String aus.

- ❑ **HTML** Ausgabefelder und Startbutton
- ❑ **JS** Funktion mit Array
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)

Eine Zufallszahl zwischen 1 und 10 ermittelt man mit:

```
var zufallszahl = Math.round(Math.random() * 9) + 1;
```

Übung B: Wettkampf

Für einen Wettkampf, sollen die Zeiten in Sekunden und der Name des Wettkämpfers protokolliert werden. Der Schnellste soll automatisch ermittelt werden.

- ❑ **HTML** Eingabefeld für den Namen,
  Start-Stop Button,
  Ausgabe in einer Tabelle.
- ❑ **JS** Funktionen mit Arrays, Berechnungen
- ❑ **CSS** Ansprechende Gestaltung
  (Freies Design)

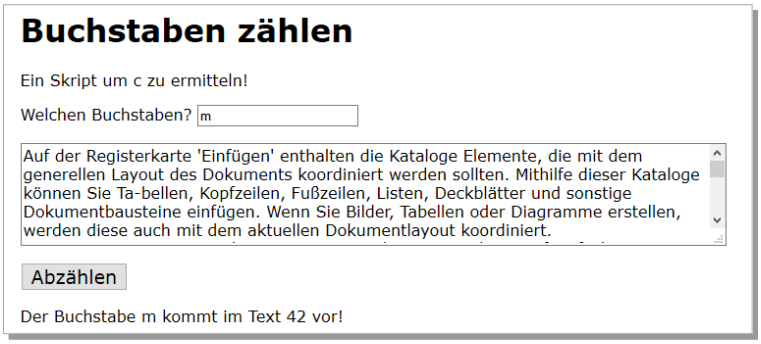
Die Zeit in Millisekunden ermittelt man mit:

```
var datum = new Date();
var zeit = datum.getTime();
```

Übung C: Buchstaben zählen

In ein Textarea-Feld soll ein beliebig langer Text eingegeben bzw. hineinkopiert werden. Im Anschluss wird ein Buchstabe im Text gesucht und gezählt.

- ❑ **HTML** Textarea-Feld, Inputfeld, Button
- ❑ **JS** Funktion mit einem Text-Array
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)



10.0 Text Operationen

Bisher haben wir Strings mit dem + Operator verbunden. Wir haben auch einen String als Array betrachtet und mit der **`.length`** *Methode die Anzahl der Zeichen ermittelt.*

Positionen von Text in einem String finden

JS

`.indexOf("Textteil", start);`

start … gibt die Position an, ab welcher die Suche starten soll.
z. B. 4 → also nach dem vierten Zeichen.

Ist der Textteil nicht im String, wird – 1 ausgegeben.
Mit **`.lastIndexOf`** sucht man nach der letzten Übereinstimmung.

```
var eingabe = "O Genie, der Herr ehre dein Ego!";
var pos = eingabe.indexOf("ehre", 4);
window.alert(pos);
```

Text aus einem String ausschneiden

JS

`.slice(start, ende);`

start … Startposition im Index z. B. 5 → also das fünfte Zeichen.

ende … Ende im Index z. B. 12 → bis zum 12 Zeichen. Wird kein Wert für das Ende angegeben, dann gibt die **`.slice`** Methode den Rest des Strings zurück.

Gibt man eine negative Zahl als start Wert an, dann wird vom Ende des Strings die Startposition ermittelt. z. B. **`eingabe.slice(-5);`** schneidet die letzten fünf Zeichen ab.

```
var eingabe = "O Genie, der Herr ehre dein Ego!";
var ausgabe = eingabe.slice(23, 31);
window.alert(ausgabe);
```

Die **`.substr()`** *Methode schneidet ebenfalls einen Textteil heraus. Nur wird hier der Startwert und die Anzahl der Zeichen angegeben.*

```
var ausgabe = eingabe.substr(23, 8);
```

UTF-16 Code ermitteln

JS

`.charCodeAt(index);`

index … *Die Position des Zeichens. Das erste Zeichen hat die Position 0.*

Zurück gegeben wird der UTF-16 Code des Zeichen. z. B. e = 101, D = 68

```
var satz = "Dreh mal am Herd!";
var derUTFcode = satz.charCodeAt(2);
window.alert(derUTFcode);
```

Mit **`String.fromCharCode()`** *dreht man die Sache um.*
Man gibt den UTF-16 Code ein und bekommt das Zeichen zurück.

```
var ausgabe = String.fromCharCode(83, 116, 114, 105, 110, 103);
window.alert(ausgabe);
```

Ü 10.0 Text Operationen

Übung A: E-Mail-Adresse

Eine E-Mail-Adresse besteht aus einem lokalen Teil (local-part, vor dem @-Zeichen) und einem Domänenteil (domain-part, nach dem @-Zeichen). Schreibe ein Script, das den Domänenteil einer eMail-Adresse ermittelt und ausgibt.

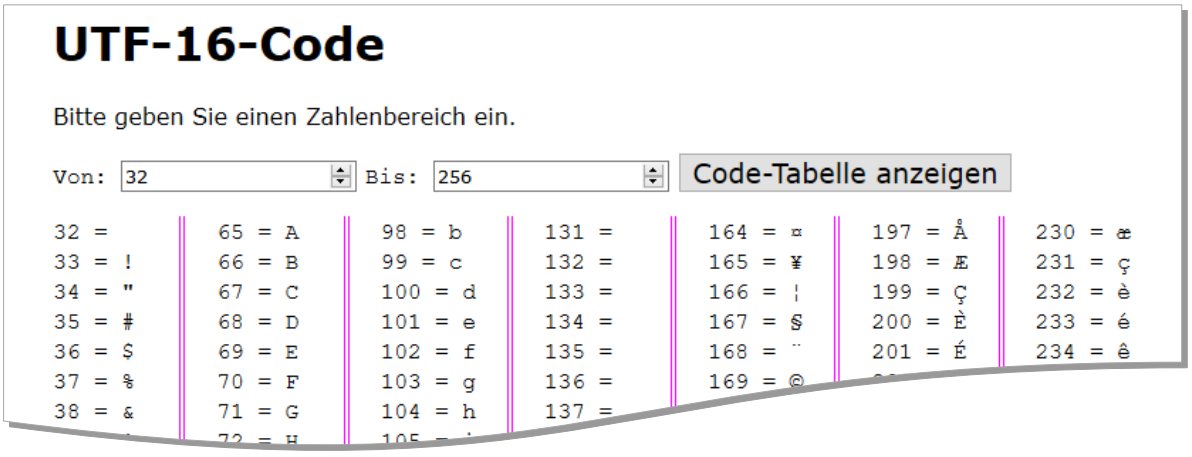
z. B. webmail@css4.at → **local-part:** webmail → **domain-part:** css4.at

- ❑ **HTML** **E-Mail-Eingabefeld, Ausgabefeld**
- ❑ **JS** **Funktion mit Textoperationen**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

Übung B: UTF-16 Code

Schreibe ein Script, dass die ersten 256 Zeichen des UTF-16-Codes ausgibt (Code und Zeichen). Erweitere die Eingabe um einen Zahlenbereich!

- ❑ **HTML** **zwei Eingabefelder, ein Ausgabefeld**
- ❑ **JS** **Funktion mit Zählschleife und CharCode Ausgabe**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**



Übung C: Kryptographie

Schreibe ein JavaScript, welches eine Eingabe verschlüsselt (also für Menschen unlesbar macht). Der verschlüsselte String soll wieder lesbar gemacht werden. Zusätzlich soll ein Schlüssel Feld eingebaut werden. Der Schlüssel ist notwendig für die Ent- bzw. Verschlüsselung.

Achtung: Wenn man mit dem Webdokument online geht, sollte man bedenken, dass der JavaScript-Code jederzeit über den Quelltext gelesen werden kann und damit auch der kryptographische Algorithmus.

10.1 String Methoden

Textteile ersetzten

JS

```
.replace("old_string", "new_string");
```

Die Methode liefert einen neuen String mit der gewünschten Ersetzung. Es wird ausschließlich die erste Übereinstimmung ersetzt und `.replace` ist Case-Sensitive.

```
var meinText = "Das Gegenteil von umfahren ist umfahren";
var neuerText = meinText.replace("umfahren","durchfahren");
window.alert(neuerText);
```

Will man alle Übereinstimmungen im String ersetzten, dann muss man den Textteil zwischen `/Text/g` setzten. Ein `/g` Flag ersetzt dann die Anführungszeichen.

```
var meinText = "Das Gegenteil von umfahren ist umfahren";
var neuerText = meinText.replace(/umfahren/g,"anhalten");
window.alert(neuerText);
```

Ein `/i` Flag hebt Case-Sensitive auf
z. B: `meinText.replace(/UMFAHREN/i,"anhalten")`

In Groß- und Kleinbuchstaben konvertieren

JS `.toUpperCase();` ← In Großbuchstaben

JS `.toLowerCase();` ← In Kleinbuchstaben

```
<p id="gross"></p>
<p id="klein"></p>
<script>
    var derName = window.prompt("Bitte den Namen eingeben");
    var grosserName = derName.toUpperCase();
    var kleinerName = derName.toLowerCase();
    document.getElementById("gross").innerHTML = grosserName;
    document.getElementById("klein").innerHTML = kleinerName;
</script>
```

Leerzeichen (Whitespace) vorne und hinten entfernen

JS `.trim();`

```
var farben = "       Rot und Blau     ";
alert(farben.trim());
```

String in ein Array splitten

JS

`.split();`

Der `.split` Methode wird das Trennzeichen mitgegeben.
z. B. `.split("|")` oder `.split(",")`

```
var tageszeit = "Morgen, Mittag, Abend, Nacht";
var ausgabe = tageszeit.split(",");
window.alert(ausgabe[2]);
```

Übung A: HTML Zeichenreferenz

In eine **<textarea>** kann man HTML Code schreiben. Erstelle ein Script, daß die HTML-eigenen Zeichen in HTML Code umwandelt.

z. B. aus `<p id="inhalt">` wird `<p id=&quot;inhalt&quot;>`

- ❑ **HTML** Zwei **<textarea>** **für Ausgabe und Eingabe**, Button
- ❑ **JS** **Funktion mit Zeichenersetzung**

Zeichen	Bezeichnung	Code
"	Anführungszeichen	&quot;
&	kaufmännisches Und	&amp;
<	öffnende spitze Klammer	&lt;
>	schließende spitze Klammer	&gt;
'	einfaches Anführungszeichen	&apos;

Übung B: Suche mit JS

Scripte eine Suche für die Webseite Elemente_einf.html. Das eingegeben Wort (bzw. der String) soll hervorgehoben werden. Sollte das Wort (bzw. der String) öfters vorkommen, dann soll jede Übereinstimmung hervorgehoben werden.

- ❑ Öffne die Datei **Elemente_einf.html**

Es gibt mehrere Lösungen. Für eine Lösung mit **`.replace`** *benötigt man* **`RegExp(suchVariable, 'g')`** *um die /g flags zu setzen.*

Übung C: IBAN Validierung

Schreibe ein JavaScript zur Validierung einer IBAN Kontonummer.
Infos: https://de.wikipedia.org/wiki/Internationale_Bankkontonummer

- ❑ Teste dein Script mit mind. 3 IBANs aus 3 unterschiedlichen Ländern.

1.	Die ersten zwei Zeichen einer IBAN-Nummer geben an, aus welchem Land sie kommt.	**DE68** 2105 0170 0012 3456 78
2.	Die ersten vier Zeichen werden an das Ende gesetzt.	2105 0170 0012 3456 78 **DE68**
3.	Alle Buchstaben werden durch ihre Position im Alphabet + 9 ersetzt. (A = 10, B = 11, C = 12, D = 13, E = 14 … Z = 35).	210501700012345678**1314**68
4.	Nun wird der Rest berechnet, der sich beim ganzzahligen Teilen der Zahl durch 97 ergibt (Modulo 97). **Das Ergebnis muss 1 sein, ansonsten ist die IBAN falsch.**	210501700012345678**1314**68 **mod 97 = 1**

***ACHTUNG**: Eine Ganzzahldivision (Modulo) wird mit % durchgeführt. Nun ist aber JavaScript nicht fähig so große Integer-Zahlen zu berechnen (wegen dem Rundungsfehler, Integer-Zahlen sind nur bis zu 9 Stellen als sicher einzustufen). Überlege dir also eine Funktion, die eine Ganzzahldivision in Teilschritte aufteilt.*

10.2 Datum

JS

`var objektname = new Date();`

Bei der Instanziierung wird dem neuen Objekt das aktuelle Datum des Browsers übergeben.

```
<p>Das aktuelle Datum ist <time id="ausgabe"></time></p>
<script>
   var heute = new Date();
   document.getElementById("ausgabe").innerHTML = heute;
</script>
```

Das `<time>` Element ist nicht unbedingt notwendig für die Ausgabe. Es dient zur besseren Strukturierung.

JS

`.toLocaleString('de-DE')`

Die Methode wandelt das Datum in eine deutsche Schreibweise um.

`.toLocaleTimeString('de-DE');` ← gibt nur die Uhrzeit zurück

`.toLocaleDateString('de-DE');` ← gibt nur das Datum zurück

```
var heute = new Date();
heute = heute.toLocaleString('de-DE');
window.alert(heute);
```

JS

`var objekt = new Date(`jahr, monat, tag, stunde, minute, sekunde, ms**`);`**

Ein bestimmtes Datum kann auch bei der Instanziierung definiert werden. Das Jahr, Monat und der Tag sind notwendig, die anderen Angaben sind optional.

```
<p>Friedrich Dürrenmatt starb am <span id="ttag"></span>
   um <span id="tzeit"></span> Uhr an Herzversagen.</p>
<script>
  var gestorben = new Date(1990, 12, 14, 22, 34);
  var todestag = gestorben.toLocaleDateString('de-DE');
  var todeszeit = gestorben.toLocaleTimeString('de-DE');
  document.getElementById("ttag").innerHTML = todestag;
  document.getElementById("tzeit").innerHTML = todeszeit;
</script>
```

JS

`.getTime()`

Gibt die Anzahl der Millisekunden vom 01. Jänner 1970, 0:00 UTC bis zum gesetzten Datum zurück.

```
var heute = new Date();
var mondfin = new Date(2022, 4, 16, 4, 12, 42, 0);
mondfin = mondfin.getTime();
heute = heute.getTime();
var zeitbis = (mondfin - heute) / (1000 * 60 * 60 * 24);
window.alert(zeitbis + " Tage bis zur nächsten Mondfinsternis");
```

11.0 .style

Mit JavaScript lassen sich alle CSS Styles verändern. Dafür wird ein Element ausgewählt, --→ um **`.style`** *erweitert → die CSS Eigenschaft hinzugefügt und schließlich der neue Wert zugewiesen.*

JS

```
Element.style.Eigenschaft = 'wert';
```

Das Element wird ausgewählt z. B. mit `document.getElementByID("id")`

für Eigenschaft wird die Bezeichnung der CSS Eigenschaft angegeben. z. B. `color` oder `margin`

Die Wertzuweisung erfolgt immer mit einfachen oder doppelten Anführungszeichen. z. B. `= "red";` oder `= "3em";`

```
<h1>Lizenzvereinbarung</h1>
<p id="info">... unsere EDV Dienstleistungen unterliegen der
                GNU/GPL und der CommonCreatives.</p>

<button onClick="meineFunktion();"> OK </button>

<script>
   function meineFunktion() {
       document.getElementById("info").style.color = 'gray';}
</script>
```

Für CSS Eigenschaften mit einem Bindestrich wird die ***CamelCase*** *Schreibweise angewendet. Aus* `font-size` *wird* `fontSize`*. Aus* `border-radius` *wird* `borderRadius`*. Aus* `background-color` *wird* `backgroundColor` *usw.*

```
<input type="text" id="pwd" onKeyUp="hervor();"
       placeholder="Mind. 8 Zeichen">

<script>
 function hervor() {
  if (document.getElementById("pwd").value.length >= 8) {
      document.getElementById("pwd").style.backgroundColor = 'lime';
    }
  }
</script>
```

Best Practice

Um eine gut strukturierte Webseite zu scripten sollte man:

- ❑ Den HTML Code gut überlegt aufbauen.
- ❑ Das Aussehen gänzlich mit CSS gestalten.
- ❑ `<noscript>` Elemente verwenden. Mit HTML5 kann das `<noscript>` Element auch im `<head>` eingesetzt werden. Ein `<style>` Element ist dann im `<noscript>` möglich.
- ❑ `Element.style` nicht für den Aufbau und das Design, sondern nur für dynamische Veränderungen verwenden!

11.1 .style Objekt

Wenn man einen Element-Selektor z. B. **`document.getElementById()`** *öfters benötigt, kann man ihn als Objekt vereinbaren. Damit erspart man sich viel Schreibarbeit und der Code wird übersichtlicher.*

JS **`var Objektname = document.getElementById();`**

```
<h1>Österreich-Quiz</h1>
<p>Wie heißt die Hauptstadt von Österreich?
   <input type="text" id="Hauptstadt" ></p>
<p><button onClick="aufloesen();">Auflösen</button></p>

<script>
 function aufloesen() {
     var antwort = document.getElementById("Hauptstadt");
     if (antwort.value == "Wien") {
         antwort.style.backgroundColor = 'lime';
         antwort.style.border = '3px solid green';}
   else {antwort.style.backgroundColor = 'red';
         antwort.style.color = 'white';
         antwort.style.textDecoration = 'line-through';}
 }
</script>
```

Das vereinbarte Objekt funktioniert nicht nur mit **`.style`** *sondern auch mit allen anderen JS Befehlen, die auf das Element zugreifen.*
Im Beispiel oben: **`antwort.value`**

Ein im HTML Element gesetztes Eventattribut mit dem Wert **`"(this)"`**
(z. B. **`<input onClick="this" id="dasHier" >`***) übergibt das Element als Objekt an eine JavaScript-Funktion (z. B.* **`function anzahl(eingabe);`***)*

Im Beispiel wird eine Funktion ausgeführt, sobald der Fokus (`onFocus`*) auf das* **`<input>`** *Element gesetzt wird. Wenn man das Input-Feld verlässt (*`onBlur`*) wird eine andere Funktion ausgeführt. Es ändert sich jedes Mal der Rahmen. Die JS Funktionen* **`hervor(); raus();`** *übernehmen den Selektor vom HTML Element und greifen auch wieder auf dieses zu.*

```
<input type="text" id="ant1" onFocus="hervor(this);"
       onBlur="raus(this);" placeholder="Vorname" ><br>
<input type="text" id="ant2" onFocus="hervor(this);"
      onBlur="raus(this);" placeholder="Nachname" ><br>
<input type="text" id="ant4" onFocus="hervor(this);"
      onBlur="raus(this);" placeholder="eMail" ><br>

<script>
   function hervor(eingabefeld) {
      eingabefeld.style.border = '3px solid blue';}
   function raus(dasFeld) {
      dasFeld.style.border = '1px solid gray';}
</script>
```

Ü 11.1 .style Objekt

Übung A: Heute

Scripte eine Webseite die den heutigen Wochentag ausgibt. Bei Montag, Mittwoch und Freitag färbt sich der Hintergrund dunkelblau. Am Dienstag und Donnerstag wird der Hintergrund gelb. Am Wochenende ist der Hintergrund grün.

- ❑ **HTML** Ausgabe des Wochentags
- ❑ **JS** Funktion zur Ermittlung des Wochentages. Style Zuweisungen.

Die `.getDay()` *Methode ermittelt den Wochentag zu einem Datum. Ausgegeben wird eine Zahl → Sonntag = 0, Montag = 1, usw.*

Übung B: Lesegeschwindigkeit

Erstelle eine Webseite um die Lesegeschwindigkeit zu testen. Suche dafür einen Text mit mind. 570 Wörtern. Verteile über die Absätze `<p>` Tags.

Vor dem Text, ist ein Start-Button. Nach dem Text ein Stop-Button. Bevor der User beginnt zu lesen, drückt er oder sie auf den Start-Button. Wenn der Text zu ende gelesen wurde drückt er oder sie auf den Stop-Button. Ein Script soll dann ermitteln, wie viele Worte pro Minute der_die Leser_in geschafft hat.

Zusätzlich soll der_die Benutzer_in die Schriftgröße des Textes verändern können. Klickt er_sie auf einen Absatz, dann wird dieser Absatz durchgestrichen und ausgegraut.

- ❑ Text mit mind. 570 Wörtern.
- ❑ **HTML** `<p>` Tags mit Event-Attributen,
  Buttons (Start, Stop, Schriftgröße)
  Ausgabefeld
- ❑ **JS** Start-Stop-Funktion (Ermittlung der Worte pro Minute),
  Schriftgröße-Funktion
  Funktion zum ausgrauen und durchstreichen von Absätzen.
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Teste deine Seite**

Bewertung der Lesegeschwindigkeit

(von Hunziker, H.-W.: Im Auge des Lesers, www.learning-systems.ch)

- ❑ **weniger als 150 Wörter pro Minute:**
  Langsamer Leser.
- ❑ **150 bis 200 Wörter pro Minute:**
  Durchschnittliche Lesegeschwindigkeit.
- ❑ **200 bis 240 Wörter pro Minute:**
  Guter Leser
- ❑ ***240 bis 300 Wörter pro Minute:***
  Schneller Leser

11.2 Selektoren

Bisher haben wir ein Element nur mit **`document.getElementById()`** *angesprochen. Mit der* **`querySelector()`** *Methode kann man auch Klassen oder Elemente per Tag-Name ansprechen.*

JS

`document.querySelector()`

Die Methode spricht immer das erste Element im HTML Code bzw. DOM an.
`.querySelector("h1")` → das erste **`<h1>`** Element
`.querySelector(".hervor")` → erstes Element mit `class = "hervor"`.
`.querySelector("#antwort")` → erstes Element mit `id = "antwort"`.
`.querySelector("[title]")` → erstes Element mit dem Attribut `title`

```
<h1>Knifflig?</h1>
<div class="frage">Was wird beim Trocknen nass?</div>
<p id="antwort"></p>
<button onClick="antworten();" title="Auslösen">Auflösen</button>
<script>
  function antworten() {
    document.querySelector("h1").style.color = "green";
    document.querySelector(".frage").style.fontStyle = "italic";
    document.querySelector("#antwort").innerHTML = "Das Handtuch";
    document.querySelector("[title]").style.display = "none";}
</script>
```

JS

`document.querySelectorAll();`

`.querySelectorAll();` spricht alle Elemente im Dokument an und speichert diese in ein Array. Es können, gleich wie bei **`.querySelector()`**, Tags, Klassen, ID's, Attribute usw. ausgewählt werden.

```
<p>Was ist bei der Diät erlaubt, und was nicht!</p>
<p class="raus">Konditoreien und Fast-Food-Buden</p>
<p>Meiden Sie <span class="raus">Zucker</span> und
             <span class="raus">Fett</span></p>
<script>
    var r = document.querySelectorAll(".raus");
    r[1].style.color = "red";
</script>
```

Weil die Elemente in ein Array gespeichert wurden, kann man auf alle über eine Zählschleife zugreifen.

```
<script>
   var RausKlasse = document.querySelectorAll(".raus");
   for (var i = 0; i < RausKlasse.length; i++) {
        RausKlasse[i].style.textDecoration = "line-through";
        RausKlasse[i].style.fontVariant = "small-caps"
   }
</script>
```

11.3 .setAttribute

Mit der **`.setAttribute()`** *Methode kann man einem HTML Element ein zusätzliches Attribut verleihen. Es benötigt einen Attributnamen und einen Wert!* **`.getAttribute()`** *ließt einen Attributwert aus.* **`.removeAttribute()`** *entfernt ein Attribut inkl. Wert und mit* **`.hasAttribute()`** *kann man prüfen, ob ein Attribut vorhanden ist.*

JS

```
Element.setAttribute(Attribut, Wert);
```

Das Element wird selektiert mit **`getElementById()`** oder **`querySelector()`**. Das Attribut wird dann mit Anführungszeichen angegeben z. B. "**`href`**". Nach einem Beistrich gibt man den Wert des Attributes an. z. B. "https://www.css4.at".

```
<style>
  .meinLink {text-decoration:none;}
  .meinLink:hover {border-bottom:1px dashed gray;}
</style>
<a id="lern">Lernplattform</a>
<button onClick="nLink();">Link herstellen</button>
<script>
   function nLink() {
     var neuerLink = document.querySelector("#lern");
     neuerLink.setAttribute("href", "https://www.css4.at");
     neuerLink.setAttribute("class", "meinLink");
   }
</script>
```

JS

```
Element.getAttribute(Attribut);
Element.removeAttribute(Attribut);
```

`.getAttribute` gibt den Wert eines selektierten Elements zurück.

`.removeAttribute` löscht ein Attribut aus dem HTML Element.

Das Beispiel übernimmt den Wert des **`Placeholder`** *Attributes, entfernt das Attribut und fügt den Wert als* **`title`** *wieder ein!*

```
<input type="email" id="Mail" onFocus="pruefe(this);"
       placeholder="eMail Adresse eingeben">

<script>
    function pruefe(obj) {
       var titletext = obj.getAttribute("placeholder");
       obj.removeAttribute("placeholder");
       obj.setAttribute("title", titletext);}
</script>
```

JS

```
Element.hasAttribute(Attribut);
```

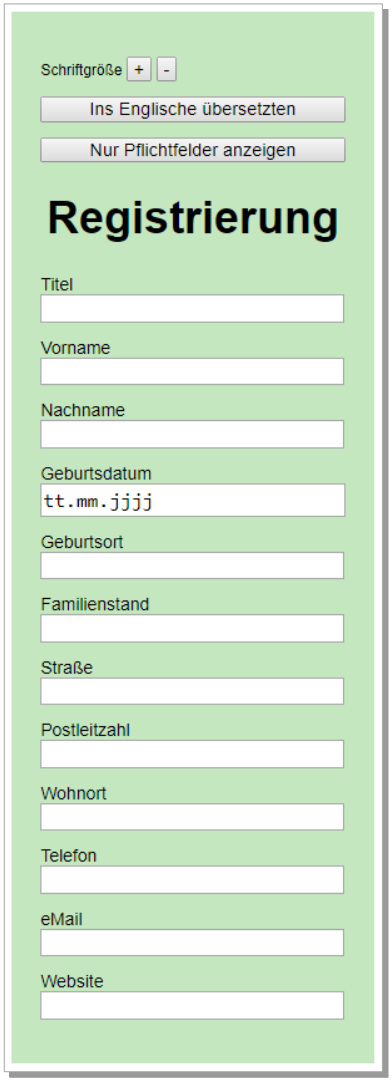
Ermittelt, ob es ein bestimmtes Attribut im Element gibt.
Zurückgegeben wird **`true`** oder **`false`** (also Boolean).

```
var meinElement = document.getElementById("Mail");
console.log(meinElement.hasAttribute("placeholder"));
```

Übung A: Schriftgröße ändern

Öffne das Webdokument ***Registrierung.html***. Mit den + und – Buttons soll sich die Schriftgröße von den Eingabefeldern und der Beschriftung ändern.

- ❑ Bei Klick auf Plus: Schriftgröße
  +1pt für Beschriftungen und
  +2pt für die Eingabefelder.
- ❑ **Bei Klick auf Minus: Schriftgröße**
  -1pt für Beschriftungen und
  -2pt für die Eingabefelder.



Übung B: Englisch

Öffne das Webdokument ***Registrierung.html***. Bei einem Klick auf den Button "Ins Englische übersetzten" soll die englische Beschriftung angezeigt werden.

Übung C: Englisch II

Öffne deine Lösung von Übung B (***Registrierung.html***). Zu jedem Eingabefeld soll nach einem Klick auf den Button "Ins Englische übersetzen" ein `title` Attribut mit dem Wert "`Only english allowed`" hinzugefügt werden.

TIPP: `.setAttribute`

Übung D: Pflichtfelder anzeigen

Öffne das Webdokument ***Registrierung.html***.
Bei einem Klick auf den Button "Nur Pflichtfelder anzeigen" werden alle Eingabefelder ausgeblendet, die nicht als Pflichtfeld (required) gekennzeichnet sind.

Übung E: Erweiterungen

Öffne deine Lösung von Übung D (***Registrierung.html***) und ergänze es um die Lösungen von Übung A, B und C.

- ❑ **HTML** Füge neue Eingabefelder für das Geburtsdatum, Geburtsort und Familienstand hinzu!
- ❑ **JS** Per Klick soll zwischen Englisch und Deutsch gewechselt werden.
  Title Attribute sind auf Englisch und Deutsch verfügbar
  Alle Felder sollen per Mausklick wieder einblendbar sein!

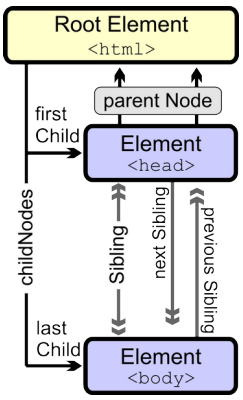
12.0 Nodes im DOM

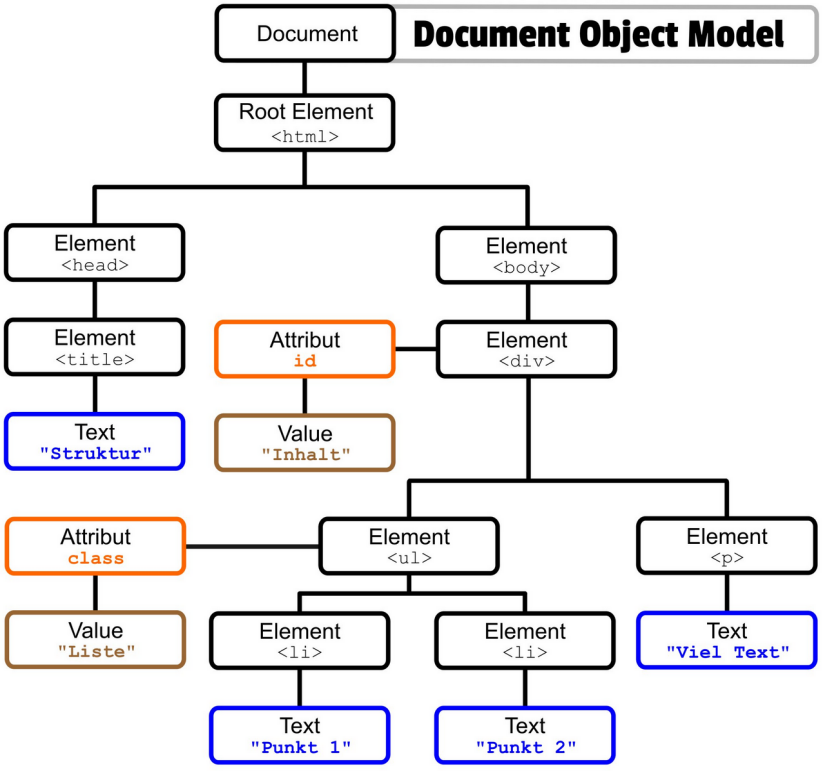
Nachdem ein Webdokument geladen wurde, erstellt der Browser das DOM – Document Object Model. Es ist die Schnittstelle zwischen HTML und JavaScript.
Der DOM Standard wird vom W3C (WWW Consortium) herausgegeben – die Browserhersteller (Mozilla, Microsoft usw.) halten sich in der Regel an diese Standards.

Das DOM wird oft als Baumstruktur dargestellt! Die einzelnen Bestandteile in dieser Baumstruktur werden auch als Knoten (Nodes) bezeichnet. Zu den drei wichtigsten Knotentypen gehören: **Element**knoten, **Attribut**knoten und **Text**knoten.

Wir unterscheiden:

Elternelemente
parentNode
Kindelemente
childNodes
firstChild
lastChild
Geschwister
sibling
nextSibling
previousSibling





"Verwandtschaftsverhältnisse" wenn der **<body>** *Tag als Parent betrachtet wird!*

```
<html>
  <head>
   <title>Struktur</title>
  </head>
  <body>                    ← Parent (Elternelement)
   <div id="Inhalt">        ← First Child (Kindelement) von <body>
    <ul class="Liste">      ← Child von <div>, Parent von <li>, Sibling von <p>
     <li>Punkt 1</li>       ← Child von <ul>
     <li>Punkt 2</li>       ← Child von <ul>,  next Sibling von  <li>Punkt 1</li>
    </ul>
    <p>Viel Text</p>        ← Second Child von <div>, Sibling (Geschwister) von <ul>
   </div>                     und last Child von  <div>
  </body>
</html>
```

Ü 12.0 Nodes im DOM

Übung A: Verwandtschaft

Erstelle eine Graphik die **alle** Verwandtschaftsverhältnisse der Elemente vom HTML Code unten beschreibt.

- ❑ **Nutze dafür eine DP Anwendung deiner Wahl**
  (z. B. LibreOffice, Inkscape, CorelDraw, Illustrator odgl.)

```
<div id="hauptinhalt">
  <h1 class="ueber1">Was bedeutet DP?</h1>
  <p>DP ist die Abkürzung für Desktop Publishing</p>
</div>
```

Übung B: Baumstruktur

Erstelle eine Baumstruktur vom HTML Code unten. Es sollen alle drei wichtigen Knotenpunkte (Elemente, Attribute und Texte) ersichtlich sein!

- ❑ Trage bei den Elementen ihre Verwandtschaft ein!
- ❑ **Nutze dafür eine DP Anwendung deiner Wahl**
  (z. B. LibreOffice, Inkscape, CorelDraw, Illustrator odgl.)

```
<html lang="de">                    ← _____
<head>                              ← _____
<title>Printmedien</title>          ← _____
</head>
<body>                              ← _____
<div id="Inhalt">                   ← _____
<h1>Zeitungen</h1>                  ← _____
<ol id="Liste">                     ← _____
<li>Standard</li>                   ← _____
<li>Die Presse</li>                 ← _____
</ol>
</div>
</body>
</html>
```

12.1 Nodes einfügen

Mit den Nodes (Knoten) können HTML Elemente hinzugefügt, ersetzt und gelöscht werden. Dafür wird das neue Element erzeugt und das Elternelement selektiert. In das Elternelement wird dann das neue HTML Element hinzugefügt.

JS 1

`document.createElement();`

Ein neues HTML Element wird erzeugt!

```
var neuesElement = document.createElement("li");
```

2

`document.createTextNode();`

Ein neuer Text wird definiert.

```
var neuerText = document.createTextNode("Das hier ist neu");
```

3

`.appendChild();`

Der Text wird an das neue Element angehängt (hinzugefügt).

```
neuesElement.appendChild(neuerText);
```

4

`.insertBefore();`

1. Das Elternelement wird ausgewählt
2. Jenes Kindelement, vor dem das neue Element eingefügt werden soll, wird ausgewählt.
3. Mit Elternelement.**`insertBefore(`**`neuesElement`, Kindelement) wird das neue Element hinzugefügt.

Beispiel: Ein neuer Eintrag **`<li>`** *in einer Nummerierung* **`<ol>`**

```
<ol id="meineListe">
   <li id="Eintrag1">Redbull Salzburg</li>
   <li id="Eintrag2">Rapid Wien</li>
   <li id="Eintrag3">LASK</li>
</ol>

<script>
   var neuesElement = document.createElement("li");
   var neuerText = document.createTextNode("Sturm Graz");
   neuesElement.appendChild(neuerText);

   var elternElement = document.getElementById("meineListe");
   var kindElement = document.getElementById("Eintrag1");
   elternElement.insertBefore(neuesElement, kindElement);
</script>
```

Ein Attribut hinzufügen (z. B. title, id, class usw.)

JS 1:	`document.createAttribute();`	← Attribut wird erstellt
JS 2:	`.value = "";`	← Wert des Attribut wird definiert
JS 3:	`.setAttributeNode();`	← Attribut wird zugewiesen

```
var neuesAttribut = document.createAttribute("title");
neuesAttribut.value = "Die beste Fussballmannschaft";
neuesElement.setAttributeNode(neuesAttribut);
```

12.2 Nodes ersetzen

*Um ein HTML Element zu ersetzen, wird zuerst das neue Element erstellt (.**createElement**). Dem neuen Element wird ein Text zugewiesen (.**createTextNode**). Zusätzlich kann man dem neuen Element auch ein Attribut zuweisen (.**createAttribute**).*

*Das Elternelement vom HTML Tag, welcher ersetzt werden soll, wird selektiert und dimensioniert. Ebenso das HTML Element selbst. Im Anschluss wird mit .**replaceChild()** das HTML Element ersetzt.*

JS

.replaceChild()

Wird als Methode an das Elternelement angehängt. Übergeben wird dann, das neue Element und das alte Element.

```
elternElement.replaceChild(neuesElement, altesElement);
```

Beispiel: Ersetzt die Überschrift mit der `id="ue1"` durch einen `<p>` Tag mit der Klasse `class="gelesen"`.

```
<style>
  .gelesen {color: red;}
</style>


<div id="inhalte">
   <h1 id="ue1">Datenschutz</h1>
   <h1 id="ue2">Allgemeine Bestimmungen</h1>
   <button onClick="meineFunktion();">Gelesen</button>
</div>

<script>
  function meineFunktion() {
    var neuesElement = document.createElement("p");
    var neuerText = document.createTextNode("Gelesen");
    neuesElement.appendChild(neuerText);

    var neuesAtt = document.createAttribute("class");
    neuesAtt.value = "gelesen";
    neuesElement.setAttributeNode(neuesAtt);

    var elternElement = document.getElementById("inhalte");
    var altesElement = document.getElementById("ue1");
    elternElement.replaceChild(neuesElement, altesElement);
   }
</script>
```

1
2
3
4
5

Erklärung zum Beispiel!

1. *CSS Eigenschaften im **<head>***
2. *HTML mit der **<h1 id="ue1">** Überschrift*
3. *Neues Element **<p>** mit dem Text "Gelesen".*
4. *Neues Attribut für das Element (die CSS Klasse .gelesen wird zugewiesen).*
5. *Eltern- und Kindelement werden selektiert und ersetzt.*

12.3 Nodes entfernen

Um einen Knoten (bzw. ein HTML Element) zu entfernen, muss man das Elternelement und das zu löschende Element kennen und selektieren.

JS

`.removeChild()` ← Löscht ein HTML Element.

Im Gegensatz zu der CSS Anweisung **`display: none;`** wird das HTML Dokument gänzlich gelöscht und nicht nur ausgeblendet.

```
<ol id="einkaufsListe">
  <li id="e1">Milch</li>
  <li id="e2">Schokolade</li>
  <li id="e3">Äpfel</li>
</ol>
<button onClick="Entfernen();">Löschen</button>
<script>
 function Entfernen () {
   var elternElement = document.getElementById("einkaufsListe");
   var kindElement = document.getElementById("e2");
   elternElement.removeChild(kindElement);
 }
</script>
```

Warum braucht man das Elternelement? Diese Frage ist berechtigt, da das zu löschende Element sowieso direkt selektiert wird.
Die Antwort ist: Der DOM benötigt es!

*Die Lösung jedoch: mit **`.parentNode`** wird automatisch das Elternelement selektiert. Das Script zum Beispiel oben, sieht dann wie folgt aus:*

```
<script>
 function Entfernen() {
   var elternElement = document.getElementById("e2").parentNode;
   var kindElement = document.getElementById("e2");
   elternElement.removeChild(kindElement);
 }
</script>
```

JS

`.parentNode` ← Selektiert ein Elternelement!

Mit **`.parentNode`** wird das Elternelement selektiert. Im Beispiel ist das **`<div>`** das Elternelement von **`<button>`** - ausgeblendet wird damit das **`<div>`** und damit auch das **`<h1>`** und das **`<p>`** Element.

```
<div>
   <h1>Informationen</h1>
   <p>Vieles zu lesen und noch mehr</p>
   <button onClick="ausblenden(this);">Ausblenden</button>
</div>
<script>
  function ausblenden(kindObjekt) {
      kindObjekt.parentNode.style.display = 'none';
  }
</script>
```

12.4 Nodelist

Auf die einzelnen Kindknoten kann man wie in einer Liste mit **`.childNodes[ ]`** *bzw. mit* **`.childNodes.item()`** *zugreifen.* **`.childNodes.length`** *gibt die Anzahl der Knoten eines Elternelements zurück. Diese Befehle geben aber auch den Whitespace (Leerzeichen, Zeilenschaltungen usw.) und Kommentare als Knotenpunkt zurück. Alternativ kann ein Array mit den Kindelementen als Nodelist vereinbart werden.*

JS

`.appendChild()`

Mit **`.appendChild()`** kann man auch einen neuen Knoten (neues Element) am Ende innerhalb eines Elternelements hinzufügen bzw. anhängen!

```
<ul id="meineListe">
  <li>Physik</li>
  <li>Chemie</li>
  <li>Mathe</li>
</ul>
<input id="neuesFach" type="text" placeholder="Neues Fach">
<button onClick="einesMehr();">Fach hinzufügen</button>
<script>
  function einesMehr() {
    var neuerEintrag = document.getElementById("neuesFach").value;
    var neuesElement = document.createElement("li");
    var neuerText = document.createTextNode(neuerEintrag);
    neuesElement.appendChild(neuerText);
    var elternElement = document.getElementById("meineListe");
    elternElement.appendChild(neuesElement);  // ← Fügt neues Element hinzu!
  }
</script>
```

***Erweiterung des Beispiel**: Eine Nodelist in Form eines Arrays wird erstellt. Der* **`.querySelectorAll()`** *durchsucht nur das Elternelement.*

```
<button onClick="listeAnzeigen();">Nr. 2 anzeigen</button>
```

```
function listeAnzeigen() {
   var dListe = document.getElementById("meineListe");
   var mListe = dListe.querySelectorAll("li");
   alert("Anzahl " + mListe.length);
   alert("Nr. 2 ist " + mListe[1].innerHTML);  }
```

Weitere Knoteneigenschaften

JS		
	`.firstChild`	← Selektiert den ersten Kindknoten.
	`.lastChild`	← Selektiert den letzten Kindknoten.
	`.previousSibling`	← Selektiert den unmittelbar vorherigen Knoten.
	`.nextSibling`	← Selektiert den unmittelbar nächsten Knoten.

Diese Eigenschaften durchsuchen nur die Struktur des Elternelements. Es werden keine Kindes-Kinder gesucht! Beachte dass der Whitespace auch selektiert wird! Ausnahme ***`.nextElementSibling`***

	`.nextElementSibling`	← Selektiert das unmittelbar nächste Element.

Ü 12.4 Nodelist

Übung A: Einkaufsliste

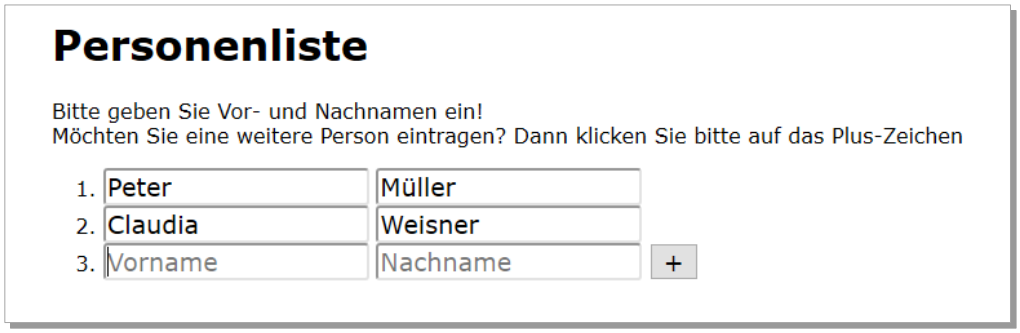
Erstelle eine online Einkaufliste. Die Produkte werden in ein Eingabefeld eingegeben und in einer Liste aufgeschrieben.

- ❑ **HTML** Überschrift und kurzer erklärender Text, Eingabefeld, 2 Buttons, Ungeordnete Liste
- ❑ **JS** Funktion mit Nodes
- ❑ **CSS** Ansprechende Gestaltung

Übung B: Personenliste Plus

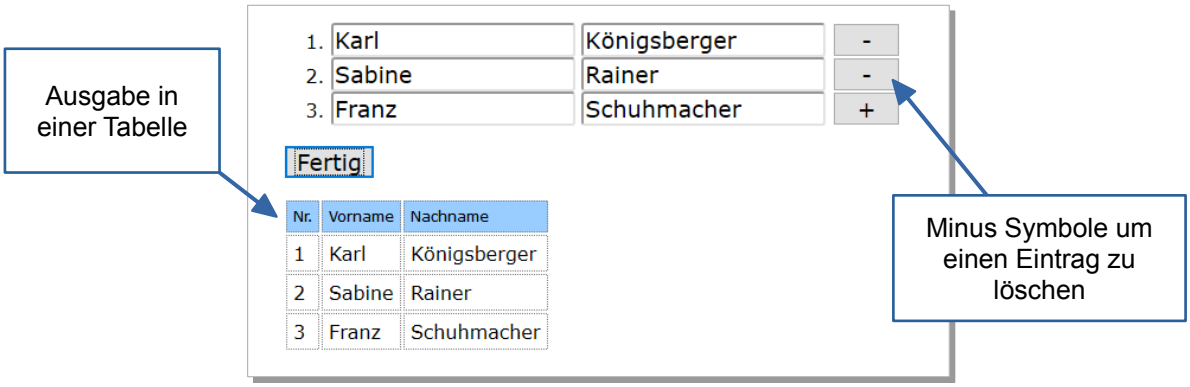
Gestalte eine Webseite mit Eingabefelder für eine Personenliste. Die Einträge sollen durchnummeriert sein. Es sollen zwei Eingabefelder für Vor- und Nachnamen verfügbar sein. Am Ende von jedem neuen Eintrag soll ein + Symbol sein, das einen neuen Eintrag ermöglicht. Es darf nur ein + Symbol geben und dieses muss auf der gleichen Höhe sein, wie der letzte Eintrag.

- ❑ **HTML** **Überschrift, Text, Eingabefelder, Button, Nummerierung**
- ❑ **JS** Mind. eine Knotenoperation
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)



Übung C: Personenliste Minus

Öffne deine Lösung von Übung B. Bei jedem Eintrag soll ein Minus Symbol sein. Klickt man darauf, wird der Eintrag aus der Personenliste gelöscht. Zusätzlich soll noch ein Fertig Button hinzugefügt werden. Klickt man darauf, dann werden alle Einträge in einer HTML Tabelle zusammengefasst!



13.1 document

Bisher haben wir das **document** *Objekt zum Selektieren von Elementen im DOM verwendet. Dafür verwendeten wir* **.getElement**… *und* **.querySelector**… *Wir nutzen das* **document** *Objekt auch für Nodes. Es kann aber mehr. Hier eine kleine Auswahl von Eigenschaften und Methoden des* **document** *Objekts …*

JS		
	document.URL;	← Gibt die Internetadresse URL des Dokuments zurück.
	document.domain;	← Gibt die Domain des Dokuments zurück.
	document.characterSet;	← Gibt den Zeichensatz des Dokuments zurück.
	document.title;	← Gibt den Titel des Dokuments zurück.

Ein neuer Titel wird mit **document.title = "Neuer Titel";** *gesetzt!*

```
<!doctype html>
<html lang="de">
 <head>
    <meta charset="utf-8">
    <title>Mein Dokument</title>
 </head>

 <body>
 <div class="inhalt">
     <h1>Dokumenteingenschaften</h1>
     <p id="ausgabe"></p>
 </div>
 <script>
   var ausgabe = "Hier die Eigenschaften: <br>";
   ausgabe = ausgabe + "<br>URL: " + document.URL;
   ausgabe = ausgabe + "<br>Domain: " + document.domain;
   ausgabe = ausgabe + "<br>Zeichensatz: " + document.characterSet;
   ausgabe = ausgabe + "<br>Titel: " + document.title;
   document.getElementById("ausgabe").innerHTML = ausgabe;
   document.title = "Dokument geladen!";
 </script>

  </body>
</html>
```

JS		
	document.hasFocus();	← Ermittelt ob das Dokument fokussiert ist (true|false).
	document.readyState;	← Ermittelt ob ein Dokument vollständig geladen wurde!

```
    <script>
      alert(document.readyState);
      function meineFunktion() {alert(document.readyState);}
    </script>
    <button onClick="meineFunktion();">Fertig</button>
```

JS		
	**document.designMode = "on";**	← Macht das Dokument editierbar.
	Mit **document.designMode = "on";** kann man das gesamte Webdokument im Browser editieren (zum Beispiel einen Text hinzufügen oder löschen)! Mit **= "off"** wird das Editieren deaktiviert.	

13.2 Timer

Die **`setInterval()`** *Methode ruft eine Funktion in bestimmten Intervallen auf. Jedes Intervall wird in Millisekunden angegeben. Lässt man den Timer alle 1000 ms ticken, dann wird zu jeder Sekunde die Funktion aufgerufen. 1000 ms = 1 Sekunde!*

JS

`setInterval(``Funktion, Millisekunden`**`)`**

Der Timer kann als Objekt dimensioniert werden. Mit der Vereinbarung wird der Timer auch gestartet. Will man der Funktion Werte übergeben, dann werden die Parameter nach den Millisekunden hinzugefügt.
z. B.: **`setInterval(`*`Funktion, Millisekunden, param1, param2, …`*`);`**

```
<script>
  var meinTimer = setInterval(Uhrzeit, 1000);
  function Uhrzeit() {
     var JetztZeit = new Date();
     JetztZeit = JetztZeit.toLocaleTimeString('de-DE');
     document.getElementById("ausgabe").innerHTML = JetztZeit;}
</script>
<p id="ausgabe"></p>
```

JS

`clearInterval();`

Stoppt den Timer. Der Methode wird das Timer Objekt übergeben.
z. B. `clearInterval(`*`meinTimer`*`);`

```
<script>
   var lauf = 10;
   var meinTimer = setInterval(meinCountdown, 500);

   function meinCountdown() {
        lauf = lauf - 1;
        document.getElementById("ausgabe").innerHTML = lauf;
        if (lauf == 0) {clearInterval(meinTimer);}
   }
</script>
<p id="ausgabe"></p>
```

Im Beispiel wird von 10 bis 0 hinunter gezählt. Bei 0 wird der Timer gestoppt. Die Variable **`lauf`** *und das Timer-Objekt* **`meinTimer`** *sind global und stehen damit jeder Funktion zur Verfügung.*

JS

`setTimeout(`*`Funktion, Millisekunden`*`)`

Die Methode ruft eine Funktion nach einer bestimmten Anzahl von Millisekunden auf. Die Funktion wird nur einmal aufgerufen.
Mit `clearTimeout()` kann man das Aufrufen verhindern.

```
<script>
    var meinTimer = setTimeout(meineFunktion, 5000);
    function meineFunktion() {
       alert("Ich habe 5 Sekunden gewartet!");    }
</script>
```

Ü 13.2 Timer

Übung A: Ladezeit

In einer Webseite soll ein wirklich großes Bild (mind. > 40 MiB) aus dem Internet dargestellt werden. Ermittle die Ladezeit des gesamten Dokuments.

- ❑ Suche im Internet nach einem richtig großen Bild.
- ❑ **HTML** Binde das Bild mit einem **<img>** Tag ins Dokument ein.
- ❑ **JS** **Timer und readyState Abrage, getTime()**
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ Teste deine Seite und erweitere die Seite mit Content um die Ladezeit zu erhöhen!
- ❑ **Setze ein Timeout, welches nach 20 Sekunden die Meldung "Zeit abgelaufen" ausgibt.**

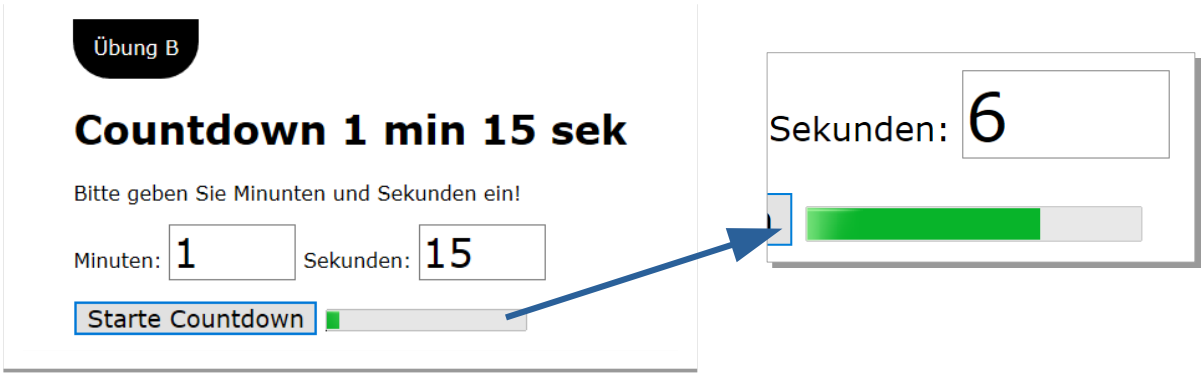
Ladezeit: 6.864 Sekunden

Übung B: Countdown

Schreibe eine Webseite mit einer Countdown-Funktion. Über Eingabefelder kann man Minuten und Sekunden eingeben. Das Skript soll dann die Gesamtanzahl der Sekunden herunterzählen.

- ❑ **HTML** **2 Eingabefelder, Button, Ausgabefeld**
- ❑ **JS** **Timer Funktionen**
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Erweitere das Beispiel um eine optische Rückmeldung.** (z. B. eine Progress-Bar)

Das **<progress>** *Element kann für die optische Rückmeldung verwendet werden. Es kennt die Attribute* **value** *und* **max**.



Übung C: Schreibmaschinen-Effekt

Ein Text soll Buchstabe für Buchstabe (so als würde man auf einer Schreibmaschine schreiben) auf dem Bildschirm ausgegeben werden!

- ❑ **HTML** **Ausgabe**
- ❑ **JS** **Timer-Funktion**

14.0 Window Objekt

Das Window Objekt ist das globale Objekt, welches ein offenes Browser-Fenster definiert. Untergeordnete Objekte sind z. B. das **document**, **history**, **location** *oder* **navigator** *Objekt. Sie werden auch als Eigenschaften des* **window** *Objekts betrachtet. z. B.* **window.document.querySelectorAll("p");**

Fenstergrößen

JS		
	.innerHeight	← Die Höhe des Viewports inkl. Scrollbars in Pixel
	.innerWidth	← Die Breite des Viewports inkl. Scrollbars in Pixel

```
<body onResize="fensterGR();">
    <h1>Der Viewport inkl. Scroolbars</h1>
    <p>Breite: <b id="Breit"></b> Pixel</p>
    <p>Höhe: <b id="Hoch"></b> Pixel</p>
 <script>
  function fensterGR() {
   document.getElementById("Breit").innerHTML = window.innerWidth;
   document.getElementById("Hoch").innerHTML = window.innerHeight;}
 </script>
</body>
```

Window Methoden

JS		
	confirm()	← Öffnet ein Dialog-Fenster mit OK oder Abbrechen. Rückgabewert ist Boolean: true oder false

```
function meineFunktion() {
    var ausgabe = window.confirm("Sind Sie sicher?");
    alert(ausgabe);
}
```

JS		
	prompt()	← Öffnet ein Dialog-Fenster mit Eingabemöglichkeit

```
function meineFunktion() {
    var antwort = window.prompt("Ihr Name bitte");
    alert(antwort);
}
```

JS		
	open()	← Öffnet ein Fenster window.open("https://www.css4.at");
	print()	← Öffnet den Druck-Dialog window.print();

Base-64-Kodierung

JS		
	.btoa()	← Dekodiert einen String mit Base-64
	.atob()	← Enkodiert einen Base-64-Code

```
function kodierFunktion() {
   var str = "Streng Geheim, naja!";
   var enc = window.btoa(str);
   var dec = window.atob(enc);
   var ausgabe = "Dekodiert: " + enc + "\nEnkodiert: " + dec;
   window.confirm(ausgabe);
 }
```

Ü 14.0 Window Objekt

Übung A: Seitenlayout

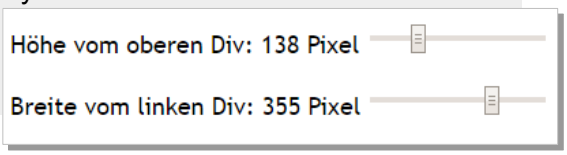
Erstelle rein mit HTML und JavaScript (ohne CSS) ein Seitenlayout wie unten dargestellt. Das Layout soll den Viewport voll ausfüllen und es dürfen keine Scrollbalken erscheinen. Wenn man die Größe des Browserfensters (onResize) verändert, wird das Layout angepasst.

- ❑ **CSS** **Ohne CSS!**

Übung B: dynamisches Seitenlayout

Öffne deine Lösung von Übung A: Seitenlayout und erweitere sie um zwei Schieberegler (**`input type="range"`**). Einer verändert die Höhe des div oben und der andere die Breite von div links – natürlich dynamisch!

- ❑ **JS** Dynamische Veränderung
- ❑ **CSS** **Für die Schieberegler!**



Übung C: En- und Decodieren

Scripte eine Webseite zum En- und Dekodieren mit Base-64. Die Eingabe soll über ein `prompt()` Fenster passieren – die Ausgabe in einem `confirm()` Fenster.

Übung D: Duden.de

Auf duden.de findet man zu beinahe allen deutschen Wörtern einen Eintrag. Die Duden-Site ist so aufgebaut, dass über die URL auch nach einem Begriff gesucht werden kann. z. B. Eine Suche nach dem Wort **`Mahlzeit`** ist über `https://www.duden.de/suchen/dudenonline/Mahlzeit` möglich!

Scripte eine Webseite die nach einer Eingabe eines Begriffs die dazu passende Seite auf duden.de automatisch öffnet!

14.1 Pop-Up Window

Mit der **`open()`** *Methode des* **`window`** *Objekts kann man auch Pop-Up Fenster erstellen. Diese Pop-Up Fenster sind für Benutzer meist eine "nervige" Sache (besonders Werbe-Pop-Up) und sollten deshalb eher vermieden werden. Viele Browser bieten deshalb Pop-Up Blocker an, also sollte man Pop-Up's gut überlegt einsetzen.*

JS

`window.open(`URL, Name, Feature**`);`**

Die Werte werden mit Hochkommas geschrieben. Die Features werden durch Beistriche getrennt.
z. B.: `window.open("http://www.css4.at", "css4", "menubar=no");`

URL

Hier wird eine URL zu einer Webseite festgelegt.
Gibt es keine URL, so wird ein leeres Fenster mit about:blank geöffnet!

Name

Über den Namen wird das target Attribut definiert, bzw. dem Fenster ein Name zugewiesen (vgl. <iframe>). Mögliche Werte sind:

`_blank`	Wird in ein neues Fenster geladen!
`_parent`	Wird in ein übergeordnetes Fenster geladen.
`_self`	Ersetzt die aktuelle Seite.
`_top`	Ersetzt ein Frameset.

Feature

`height=pixels`	Die Höhe eines Fenster in Pixel. Mindestens 100
`width=pixels`	Die Breite eines Fenster in Pixel. Mindestens 100
`menubar=yes|no`	Definiert, ob das Menü des Browserfensters angezeigt werden soll.
`scrollbars=yes|no`	Anzeige des scrollbar (Bildlaufleiste).
`status=yes|no`	Darstellung der Statusanzeige.
`titlebar=yes|no`	Titelbar Anzeige.
`toolbar=yes|no`	Anzeige der Browser-Toolbar.
`left=pixels`	Position des neuen Fensters, gemessen links vom Bildschirmrand in Pixel.
`top=pixels`	Position des neuen Fensters, gemessen vom oberen Anfang des Bildschirms in Pixel.
`location=yes|no`	Zeigt das Adressfeld (nicht in allen Browser).
`resizable=yes|no`	Erlaubt die Veränderung der Fenstergröße (nicht in allen Browser)

Statt `yes` oder `no` kann auch `1` oder `0` eingetragen werden!

```
var radioFenster = window.open("", "Radiostream", "width=250,
                   height=400, menubar=no,
                   toolbar=no, left=200, top=300");
    radioFenster.focus();
    radioFenster.document.write("<h1>Jazzradio.com</h1>");
```

JS **`.close()`** ← *Schließt das Fenster wieder!*

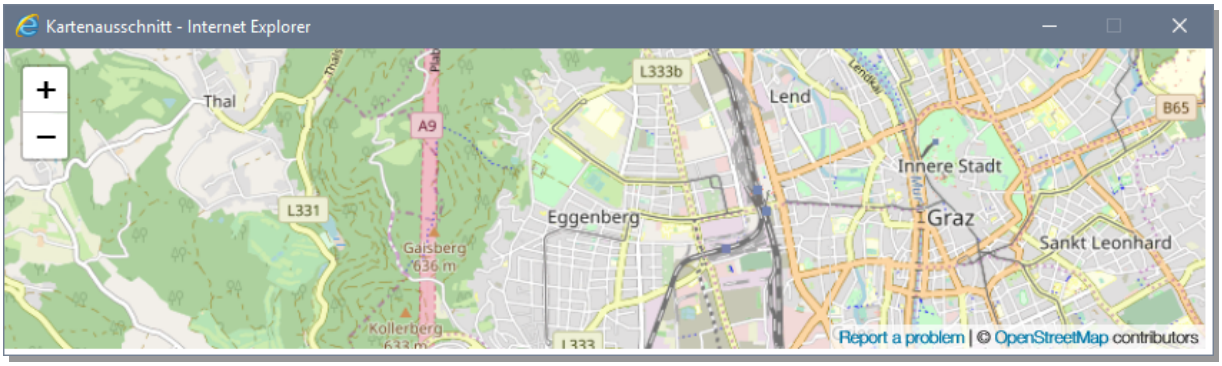
```
<button onClick="radioFenster.close();">Schließe Fenster</button>
```

Mit **`if(!radioFenster){…} else {…}`** kann man überprüfen, ob das neue Fenster instanziiert wurde oder ob ein Pop-Up Blocker es verhindert hat.

Übung A: Open Street Map

Auf openstreetmap.org findet man freie Straßenkarten. Suche dir einen Punkt auf der Landkarte und teile diese in Form von einem iframe. Auf der Website von openstreetmap.org gibt es einen Teilen-Button der den HTML Code ausgibt. Schreibe eine Webseite, die den Kartenausschnitt als Pop-Up öffnet.

- ❑ Suche auf openstreetmap.org nach dem Teilen-Button mit dem HTML Code.
- ❑ **HTML** Ein Button zum Öffnen des Pop-Ups.
- ❑ **JS** Funktion zum Öffnen des Pop-Ups, blende so viel aus, wie möglich.
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Verpacke alles in einem einzigen HTML Dokument!**



Übung B: Linkliste

Öffne die Webseite `aboutcss.html`. Über einen Button soll sich ein PopUp Fenster mit allen Hyperlinks `<a href …>` aus dem Dokument öffnen.

- ❑ **HTML** Button
- ❑ **JS** Alle Links ermitteln, Pop Up Fenster öffnen
- ❑ **CSS** Freie Gestaltung

Lösungsansatz: Mit `.outerHTML` wird der HTML Code (Tag, Attribute, Text usw.) eines Elements ausgegeben.

Übung C: Linkliste erweitert

Öffne deine Lösung von **Übung B: Linkliste**.

Erweitere das JavaScript um eine Prüfung, ob das Fenster geöffnet wurde mit einer passenden Rückmeldung (z. B. Button verschwindet, odgl) für den User.

Zusätzlich soll das PopUp Fenster am rechten Bildschirmrand erscheinen.

`var breite = screen.width;` *ermittelt die Bildschirmbreite in Pixel.*
`.moveTo(x,y)` *bewegt ein Fenster zu den Koordinaten am Bildschirm.*

14.2 Window Methoden

Ein mit **open()** *geöffnetes Fenster lässt sich noch weiter verändern, bzw. modifizieren.*

JS

.moveTo(x, y) ← bewegt das Fenster zu einer bestimmten Position.

.moveBy(x, y)

Bewegt das Fenster um eine bestimmte Anzahl von Pixel, relativ zu seiner Position. x-Werte bewegen das Fenster rauf und runter, y-Werte links und rechts. Negative Werte sind möglich!

```
<button onclick="fensterAuf()">Neues Fenster öffnen</button>
<button onclick="fensterMove()">Bewege das Fenster</button>
<script>
  var fenster;
  function fensterAuf() {
     fenster = window.open("", "Info",
                           "width=200, height=100, top=200");
     fenster.document.write("<p>Wichtige Infos</p>");
  }
  function fensterMove() {
     fenster.moveBy(10, -10);
     fenster.focus();      }
</script>
```

JS

.resizeBy(Breite, Höhe) ← relativ zu seiner eignen Größe in Pixel.

.resizeTo(Breite, Höhe) ← Vergrößert oder verkleinert ein Fenster in Pixel.

```
<button onclick="fensterAuf()">Create window</button>
<button onclick="fensterSize()">Resize window</button>
<script>
  var meinFenster;
  function fensterAuf() {
     meinFenster = window.open("", "", "width=100, height=100");
  }
  function fensterSize() {
     meinFenster.resizeTo(450, 650);
     meinFenster.focus();       }
</script>
```

JS

window.getComputedStyle(Element**).getPropertyValue(**CSS-Eigenschaft**);**

Ermittelt das tatsächliche Aussehen eines Elements und gibt den Wert einer CSS-Eigenschaft zurück! z. B. bei einem mit einer Breite von 100 % wird die Breite in Pixel ermittelt und zurückgegeben.

```
<div id="rotesDiv" style="height: 50px; background: red;"></div>
<script>
  var element = document.getElementById("rotesDiv");
  var dieCSSeigenschaft = window.getComputedStyle(element);
  var ausgabe = dieCSSeigenschaft.getPropertyValue("width");
  alert("Breite in Pixel: " + ausgabe);
</script>
```

14.3 pageOffset

Mit der **`window.pageYOffset`** *Eigenschaft ermittelt man, wie weit ein Benutzer nach unten gescrollt hat (z. B: durch drehen am Mausrad). Gemessen wird der Wert in Pixel vom Dokumentanfang.* **`window.pageXOffset`** *gibt die gescrollte Entfernung in Pixel über die Horizontale (von links nach rechts) zurück.*

JS		
	`window.pageYOffset;`	← Gescrollt über die Y-Achse (Vertikal)
	`window.pageXOffset;`	← Gescrollt über die X-Achse (Horizontal)

Im Beispiel werden 200 durchnummerierte Zeilen eingefügt. Im `<body>` *Tag ist das Attribut* `onscroll`*. Sobald das Mausrad gedreht wird, feuert die Funktion die Y-Position in das fixierte* `<div id="pos">`*.*

```
<body onscroll="runterScrollen();">
  <div style="position:fixed; top:10px; right:10px;"></div>

  <script>
     for (var i = 0; i <= 200; i++) {
        document.write("<p>Zeile:" + i +  "</p>");
     }

     function runterScrollen() {
        document.querySelector('div').innerHTML = window.pageYOffset;
     }
  </script>
</body>
```

JS		
	`window.scrollBy(x, y);`	← Scrollt das Dokument um einen xy-Wert
	`window.scrollTo(x, y);`	← Scrollt das Dokument zu einem xy-Wert

Sobald um 500 Pixel nach unten gescrollt wird, springt (scrollt) das Dokument zurück zum Anfang `window.scrollTo(0, 10);`

```
if(window.pageYOffset >= 500) {
   window.scrollTo(0, 10);
}
```

JS		
	`.offsetTop;`	← Ermittelt die Y-Position eines Elements

Per Click auf ein **<p>** *wird seine Position in Pixel, gemessen vom Dokumentanfang ermittelt und in einem alert Dialog ausgegeben.*

```
<p onClick="posY(this);" style="height:300px; background:red;"></p>
<p onClick="posY(this);" style="height:300px; background:blue;"></p>
<p onClick="posY(this);" style="height:300px; background:green;"></p>

    <script>
        function posY(meinEl){
           alert(meinEl.offsetTop);
        }
    </script>
```

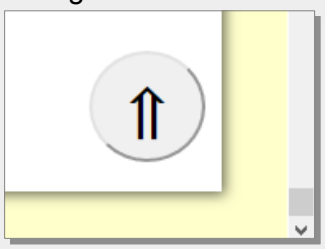
Übung A: Scrollen

Öffne die Webseite **`aboutcss.html`** bzw. deine Lösung von **14.1 C: Linkliste**.

Sobald um 300 Pixel nach unten gescrollt wurde, soll ein Button erscheinen, der per Klick das Dokument wieder nach oben zum Anfang scrollt.

Arbeite hier **nicht** mit Anker sondern finde eine pageOffset Lösung.

- ❑ **HTML** Button
- ❑ **JS** Funktion zur Ermittlung des pageOffset, Funktion für das Scrollen
- ❑ **CSS** **Button ansprechend gestalten, unten rechts am Viewport fixieren!**



Mit der CSS Eigenschaft `html {scroll-behavior: smooth;}` *scrollt das Dokument geschmeidig und springt nicht nur zu seiner neuen Position!*

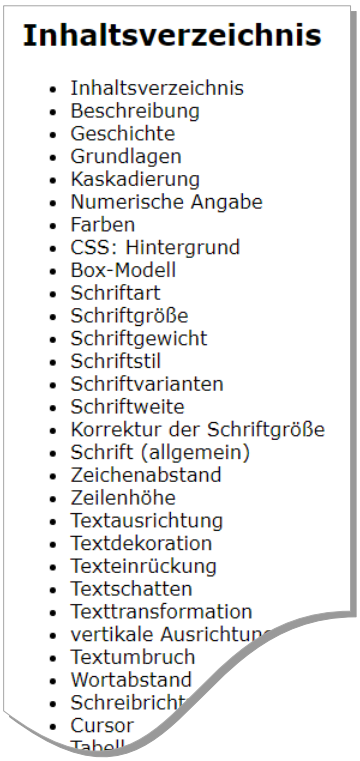
Übung B: Inhaltsverzeichnis

Öffne das Webdokument **`aboutcss.html`** bzw. deine Lösung von **Übung A: Scrollen**.

Füge unter der Hauptüberschrift **`<h1>`** ein automatisch generiertes Inhaltsverzeichnis mit allen **`<h2>`** Elementen hinzu.

Per Klick auf ein Element im Inhaltsverzeichnis soll automatisch zu diesem Punkt im Dokument gescrollt werden. Arbeite hier **nicht** mit Anker sondern finde eine pageOffset Lösung.

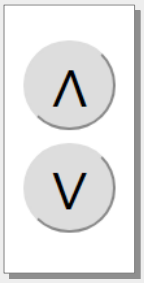
- ❑ **HTML** Ausgabeelement (z. B. **`<ul>`**)
- ❑ **JS** Funktion zur Erstellung des Inhaltsverzeichnis, Verlinkung der Verzeichnispunkte mit den **`<h2>`** im Dokument
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)



Übung C: Rauf und runter

Öffne das Webdokument **`aboutcss.html`** bzw. deine Lösung von **Übung B: Inhaltsverzeichnis**. Erweitere die Webseite um zwei Pfeilbuttons für Rauf und Runter. Per Klick soll zur nächsten **`<h2>`** Überschrift gescrollt werden (bzw. zur vorherigen zurück).

- ❑ **HTML** Zwei Buttons
- ❑ **JS** Entsprechende Funktionen
- ❑ **CSS** **Ansprechende Gestaltung**



15.1 Navigator

Das Navigator Objekt gibt Informationen über den Web-Browser des Users aus. Es ist ein unmittelbares Unterobjekt von window → **`var navObj = window.navigator;`**

Eigenschaften von window.navigator

JS		
	.appCodeName	← Spitzname des Browsers
	.appName	← offizieller Name des Browsers
	.appVersion	← Browser-Version
	.cookieEnabled	← Cookies erlaubt (true|false)
	.onLine	← besteht eine Internetverbindung (true|false)
	.platform	← Plattform, bzw. Betriebssystem

```
<p id="info"></p>
<script>
   var navObj = window.navigator;
   var ausgabe =  "Browser:    " + navObj.appName + "<br>";
       ausgabe += "Spitzname:  " + navObj.appCodeName + "<br>";
       ausgabe += "Version:    " + navObj.appVersion + "<br>";
       ausgabe += "Cookies:    " + navObj.cookieEnabled + "<br>";
       ausgabe += "Online:     " + navObj.onLine + "<br>";
       ausgabe += "Platform:   " + navObj.platform + "<br>";
   document.getElementById("info").innerHTML = ausgabe;
</script>
```

JS		
	**.language**	← Browser-Sprache

`.language` liefert den Sprachcode (en = Englisch, de = Deutsch usw.) und Browserabhängig das Land (AT = Österreich, DE = Deutschland usw), dann wird z. B. de-DE oder de-AT ausgegeben. Mit `.indexOf()` durchsucht man den String ob ein bestimmter Sprachcode vorkommt. Ist der Teilstring vorhanden, dann gibt die Methode die Position im String zurück, also: größer als -1.

```
if (navigator.language.indexOf("de") > -1) {alert("Guten Tag");}
if (navigator.language.indexOf("fr") > -1) {alert("Bonjour");}
if (navigator.language.indexOf("AT") > -1) {alert("Servus");}
```

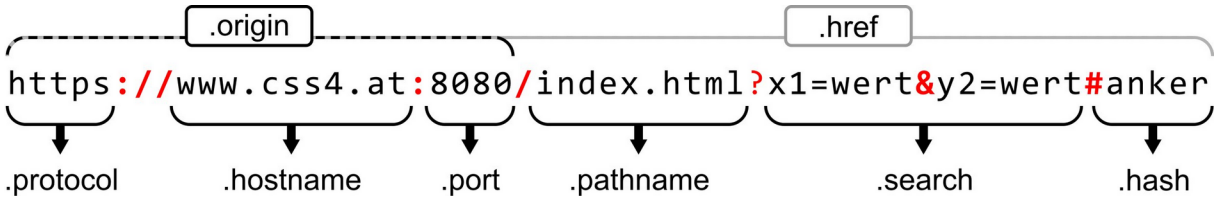
Um die exakte geographische Position eines Device (z. B. Smartphone mit GPS) zu ermitteln, bietet **`.navigator`** *das Objekt* **`.geolocation`** *an.*

JS		
	.plugins	← Ermittelt installierte Plug-Ins im Browser

```
function instPlugIns(){
  var ausgabe = "";
  for (var i = 0; i < navigator.plugins.length; i++) {
   ausgabe += " NAME: " + navigator.plugins[i].name;
   ausgabe += " ART:  " + navigator.plugins[i].description;
   ausgabe += " FILE: " + navigator.plugins[i].filename + "<br>";}
  return ausgabe;    }
```

15.2 Location

Mit dem Location Objekt (Unterobjekt von **`window`***) hat man Zugriff auf die vollständige URL der aktuellen Webseite. Alle Eigenschaften des* **`location`** *Objekts können sowohl ausgegeben als auch gesetzt werden! URL steht für eng. Uniform Resource Locator.*



JS	**`location.href`** ← Die gesamte URL Definiert die gesamte URL (vom Protokoll bis zum Anker). Mit **`location.origin`** bekommt man das Protokoll, den Hostnamen und den Port geliefert.
JS	**`location.protocol`** ← Protokoll bzw. Schema Liefert bzw. setzt das Protokoll (auch Schema genannt), einer URL. Das kann HTTP, HTTPS, FTP aber auch mailto, file, news usw. sein.
JS	**`location.hostname`** ← Definiert den Hostnamen (z. B. www.css4.at)
	`location.port` ← Port Die Angabe des Ports erlaubt die Ansteuerung eines TCP-Ports. Wird kein Port angegeben, so wird der Standard-Port des jeweiligen Protokolls verwendet – zum Beispiel bei HTTP 80, bei HTTPS 443 und bei FTP 21. Mit **`location.host`** wird Hostname und Port definiert.
JS	**`location.pathname`** ← Definiert den Pfad mit Datei (z. B. /html/index.html)
	`location.search` ← Abfragestring z. B. Übergabewerte Liefert bzw. setzt Übergabewerte in Form eines "Query-String". Mit einem Fragezeichen (?) beginnt der Query-String. Das Fragezeichen wird ebenfalls ausgegeben. Mehrere Übergaben sollten durch ein kaufmännisches Und (&) getrennt werden.
JS	**`location.hash`** ← Definiert den Anker (z. B. #anker)

Location Methoden

JS	**`location.reload()`** ← Ladet eine Seite neu
	`location.replace()` ← Ersetzt eine Webseite durch eine andere.
	`location.assign()` ← Gleich wie **`.replace()`** nur ohne 'Zurück-Button'.

History Methoden

JS	**`history.back()`** ← Öffnet die vorherige Seite nochmals (Zurück-Button).
	`history.forward()` ← Ladet die nächste Seite aus der **`history`** Liste.
	`history.go()` ← Springt innerhalb der **`history`** Liste. z. B. **`history.go(-2);`** springt zwei Seiten zurück.

Ü 15.2 Location

Übung A: Browser Informationen

Schreibe eine Webseite die alle Informationen über den Browser ermittelt und in einer Tabelle ausgibt. Zusätzlich soll die URL ausgewertet werden.

- ❑ **HTML** **Tabelle**
- ❑ **JS** **Funktion, .navigator Objekt, .location Objekt,**
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Teste deine Seite mit unterschiedlichen Browsern, wenn möglich online!**

Im Internet gibt es zahlreiche gratis Anbieter von Webspace.

window.navigator

Browser	Netscape	.appName
Spitzname	Mozilla	.appCodeName
Version	5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36	.appVersion
Cookies	true	.cookieEnabled
Online	true	.onLine
Platform	Win32	.platform
Sprache|Land	de-AT	.language

Installierte Plug-Ins

Plug-In	Chrome PDF Plugin
Plug-In	Chrome PDF Viewer

Übung B: Error Docs

Die meisten Webhoster (Domain, Webspace usw.) stellen den Webdevoloper_innen ein Verzeichnis für Error Docs zur Verfügung. Error Docs werden angezeigt, wenn ein Server-Fehler auftritt z. B. 404 – Seite nicht gefunden. So können "Server-Fehlermeldungen" dem Design der Domain angepasst und technische Informationen mit JavaScript und PHP gesammelt werden.

Scripte ein Error Doc deiner Wahl!

- ❑ Mindestanforderungen: ein Zurück Button und eine Fehlerbeschreibung in englischer und deutscher Sprache (abhängig von den Spracheinstellungen des Browsers).

400 - Bad Request (Ungültige Anforderung)
401 - Authorization Required (Erlaubnis benötigt)
403 - Forbidden (Verboten)
404 - Page Not Found (Seite nicht gefunden)
405 - Method Not Allowed (Methode nicht erlaubt)
406 - Not Acceptable (Inakzeptabel)
407 - Proxy Authentication Required (Proxy-Authentifizierung erforderlich)
412 - Precondition Failed (Vorbedingung fehlgeschlagen)
414 - Request-URI Too Long (Anforderungs-URI zu lang)
415 - Unsupported Media Type (Nicht unterstützter Medientyp)
500 - Internal Server Error (Interner Serverfehler)
501 - Not Implemented (Nicht implementiert)
502 - Bad Gateway (Gateway Fehler)
503 - Service Temporarily Unavailable (Dienst vorübergehend nicht verfügbar)

15.3 Wertübergabe GET

*Bei einer Wertübergabe eines Formulars mit der Methode GET (**method="GET"**), werden die Namen und die Werte in die URL geschrieben. Die Übergabe (Querystring) erfolgt mit einem Fragezeichen zu Beginn, Name und Wert werden durch ein Ist-Gleich-Zeichen getrennt, und zwischen den Übergabeparametern steht ein Kaufmännisches Und.* **?name1=wert1&name2=wert2&name3=wert3&...**
Mittels **window.location.search** *wird der Querystring aus der URL ermittelt.*

Formular mit einer GET Übergabe.
name *und* **value** *werden mit der URL übergeben!*

```
<form method="get" action="auslesen.html">
   <p>Benutzer: <input type="text" name="Benutzer" ></p>
   <p>eMail:    <input type="text" name="eMail" ></p>
   <p>Passwort: <input type="password" name="Passwort" ></p>
   <p><button type="submit" >Absenden</button></p>
</form>
```

Auswertung der URL
Hier der Code im Webdokument **auslesen.html**
Werte und Namen der Variablen werden in einer Tabelle ausgegeben.

```
<table>
  <thead><tr><th>Variablenname</th><th>Wert</th></tr></thead>
  <tbody id="ausgabeTab"></tbody>
</table>
<script>
  var abfrageURL = window.location.search;     // Einlesen aus der URL
      abfrageURL = abfrageURL.slice(1);        // Fragezeichen löschen
  var werteGesamt = abfrageURL.split("&");          // Array erzeugen
  var paar, ausgabe = "";
  for (var i = 0; i < werteGesamt.length; i++) {
       paar = werteGesamt[i].split("=");     // Name und Wert trennen
       paar[0] = unescape(paar[0]);          // Sonderzeichen umwandeln
       paar[1] = unescape(paar[1]);
       ausgabe += "<tr><td>" + paar[0] + "</td>"
       ausgabe += "<td>" + paar[1] + "</td></tr>";      }
  document.getElementById("ausgabeTab").innerHTML = ausgabe;
</script>
```

unescape(); *verwandelt Sonderzeichen wieder in einen lesbaren String.*

Auswertung mit URLSearchParams *Achtung:* **URLSearchParams** *ist zurzeit noch nicht in allen Browsern verfügbar! Mit* **.get** *wird der Wert einer Variable ausgegeben. Weitere Methoden wären .getAll(), .has(), .forEach(), .sort() und viele mehr.*

```
var abfrageURL = window.location.search;
var sucheURL = new URLSearchParams(abfrageURL);
alert(sucheURL.get("Passwort"));
```

Ü 15.3 Wertübergabe GET

Übung A: Google Suche

Öffne `www.google.at` in einem Browser und Suche nach einem beliebigen Begriff. Analysiere danach die URL im Adressfeld `https://www.google.com/…`

Betrachte die übergebenen Schlüsselpaare `?q=…` und überlege was sie bedeuten können. Schreibe danach eine Webseite, mit einem eigenen Google Suchfeld und einer passenden Wertübergabe.

- ❑ **HTML** Form mit der Methode GET und der Action GoogleURL.
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)

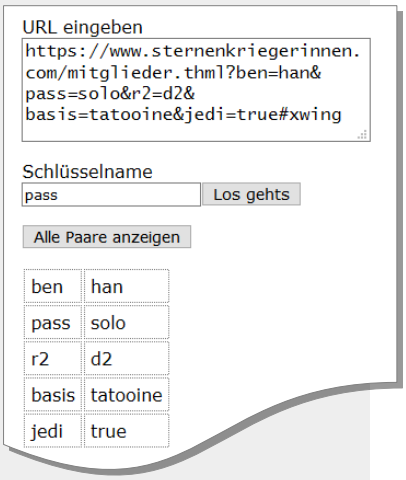
Übung B: URL Analyse

Scripte eine Get Funktion, die aus einer URL einen bestimmten Schüsselwert ausgibt. Die URL soll über ein Textfeld ermittelt werden!

Beispiel:
URL: …**`com/?ben=han&pass=solo&r2=d2#xwing`**

Funktion: **`zeigeWert("pass");`** Ausgabe: **`solo`**

- ❑ **JS** Funktion mit Return Value, Auswertung einer URL
- ❑ **HTML** Eingabefeld für die URL
- ❑ **Verwende keine `URLSearchParams`**



Übung C: Bücher Tauschbörse

Erstelle eine einfache Bücher Tauschbörse mit zwei Webdokumenten. Im ersten wird das Buch durch Felder beschrieben (Titel, Autor, Erscheinungsjahr, ISBN, Auflage, Verlag) – die Eingaben werden in einer URL zusammengefasst. Im zweiten Webdokument wird die URL ausgelesen und optisch ansprechend ausgewertet. Die Idee: Man braucht einfach nur einen Link versenden bzw. in sozialen Netzwerken posten!

- ❑ **HTML** **Zwei HTML Dokumente,**
  HTML 1: Eingabeformular, HTML 2: Darstellung
- ❑ **JS** Funktion zur Generierung einer URL, Auswertung der URL
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

15.4 localStorage

Mit dem Web Storage API kann man Daten lokal auf dem Rechner des Users speichern. Es werden nur der Schlüsselname und der Schlüsselwert in Form eines String gespeichert. Die lokal gespeicherten Daten können dann von allen Skripten der Domain abgerufen werden. Das Web Storage API beinhaltet zwei Objekte: `window.localStorage` *(speichert unbegrenzt und ohne Ablaufdatum) und* `window.sessionStorage` *(nur für eine Sitzung, nachdem das Browser-Tab geschlossen wurde, werden auch die Daten gelöscht).*

Für **sessionStorage** *gelten dieselben Methoden wie für* **localStorage**:
`.setItem(); .getItem(); removeItem(); und .clear();`

window.localStorage Methoden

JS `window.localStorage.setItem(Name, Wert)` ← Speichern

[Name] Der Name des Eintrags
[Wert] Der String zum gespeicherten Eintrag
.setItem() speichert den Eintrag lokal im Cache des Browsers. Wenn ein Eintrag schon vorhanden ist, wird dieser ohne Rückmeldung überschrieben.

```
var eMailAdresse = prompt("Ihre eMail Adresse");
localStorage.setItem("benutzerMail", eMailAdresse);
localStorage.setItem("Vorname", "Mario");
```

JS `localStorage.getItem(Name)` ← *Abrufen der Speicherung*

```
var neueMail = localStorage.getItem("benutzerMail");
alert(neueMail);
```

JS `localStorage.removeItem(Name)` ← *Löscht den Eintrag*

```
localStorage.removeItem("benutzerMail");
```

JS `localStorage.clear()` ← *Löscht alle Einträge zur Domain*

```
localStorage.clear();
```

Speicherung eines Array
`JSON.stringify();` *wandelt ein Array (bzw. ein Objekt) in eine JSON-Notation. Die* ***J****ava****S****cript* ***O****bjekt* ***N****otation (JSON) ist ein Dateiformat für den Datenaustausch zwischen Anwendungen und erlaubt das Speichern eines Arrays in einen String.*

```
var meinFeld = ["Blumen", "Pflanzen", "Bäume", "Gräser"];
var meinJSON = JSON.stringify(meinFeld);
localStorage.setItem("nimmDas", meinJSON);
```

Das Array abrufen
`JSON.parse();` *wandelt die JSON-Notation wieder in ein Array um.*

```
var dasFeld = localStorage.getItem("nimmDas");
var meinFeld = JSON.parse(dasFeld);
for(var i =0; i < meinFeld.length; i++) {alert(meinFeld[i]);}
```

Ü 15.4 localStorage

Übung A: Besuchzähler

Jedes mal wenn das Webdokument geöffnet wird, soll ein Besuchzähler mitprotokollieren und die Anzahl der Besuche auf der Webseite zurückgeben.

- ❑ **HTML** **Ausgabe in einem HTML Element**
- ❑ **JS** **localStorage des Zählers**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

Übung B: Datum

Öffne deine Lösung von **Übung A: Besuchzähler** und erweitere sie um eine Datumsausgabe des letzten Besuch. (Datum und Uhrzeit).

Füge zusätzlich noch einen Button hinzu, der den localStorage wieder löscht.

- ❑ **HTML** **Ausgabeelement, Button**
- ❑ **JS** **localStorage des Datums, Datumsverarbeitung, Löschfunktion von localStorage**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

Übung C: Einkaufsplan

Erstelle eine Art "Einkaufsliste" in einem Webdokument. In ein Input Feld soll die Bezeichnung der Ware eingeben werden. In einem weiteren Input Feld der dazu passende Preis.

Alle Preise sollen addiert und als Summe ausgegeben werden. Per Mausklick kann man ein Element (Bezeichnung und Preis) von der Liste löschen.

Der Einkaufsplan soll lokal gespeichert werden und wird bei einem nochmaligen Aufruf automatisch aus dem lokalen Speicher geladen.

- ❑ Arbeite mit Arrays
- ❑ **HTML** **Zwei Eingabefelder, Ausgabe (z. B. Tabelle), Buttons**
- ❑ **JS** **Arrays verarbeiten, localStorage**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

15.5 Cookies

Cookies sind kleine Textfiles die auf dem Device (Smartphone, Computer usw.) des Users lokal gespeichert werden. Aus Datenschutzgründen muss die Speicherung von Cookies explizit erlaubt werden. Ein Cookie besteht mind. aus dem Schlüsselnamen, dem Schlüsselwert, und einer Lebenszeit. Die Cookies sind nur innerhalb einer Domain gültig – also, nur jener der das Cookie setzt kann es auch wieder auslesen!
`navigator.cookieEnabled;` *ermittelt ob die Browsereinstellungen überhaupt Cookies erlauben – die Rückgabe ist ein Boolean Wert, also true oder false!*

Cookie erstellen

JS

```
document.cookie = "name=wert; max-age=sekunden";
```

Achtung: Wenn ein Cookie bereits besteht, und ein neues mit dem gleichen Schlüsselname gespeichert wird, dann wird das alte Cookie überschrieben.

`name`	Schlüsselname bzw. Bezeichner
`wert`	Schlüsselwert z. B. **`farbe=red;`**
`max-age`	Lebenszeit des Cookie in Sekunden, ohne Angabe verfällt das Cookie nachdem der Browser geschlossen wurde (Session-Cookie)
`sekunden`	86 400 Sekunden sind ein Tag

```
document.cookie = "benutzer=herbert; max-age=86400";
```

Statt `max-age` kann das Ablaufdatum mit `expire` gesetzt werden, also ein bestimmtes Datum. Damit die Datumsübergabe gut funktioniert, sollte man ein Datumsobjekt vereinbaren `new Date()` und das Objekt dann mit `.toGMTString();` übergeben.
Im Beispiel werden zum heutigen Datum 86 400 000 Millisekunden hinzuaddiert – also das Ablaufdatum wird auf Morgen gesetzt.

```
var jetzt = new Date();
jetzt.setTime(jetzt.getTime() + 86400000);
document.cookie = "ablauf=morgen; expires=" + jetzt.toGMTString();
```

Cookies lesen

JS

```
var xyz = document.cookie;
```

Alle Cookies werden wie in einem String zurückgegeben und können dann durch Textoperationen ausgewertet werden.
Die Rückgabe hat das Schema: `name1=wert1; name2=wert2; name3=wert3`

```
var meineKekse = document.cookie;
alert(meineKekse);
```

Cookies löschen

Da Cookies überschrieben werden, braucht man nur das zu löschende Cookie nochmals mit einem Ablaufdatum von 0 Sekunden setzten.

```
document.cookie = "benutzer=; max-age=0";
```

Ü 15.5 Cookies

Übung A: Datenschutzverordnung

Recherchiere im Internet die aktuellen Bestimmungen zur Datenschutzverordnung, dem Telekommunikationsgesetz bzw. dem E-DSVO oder der E-Privacy-Verordung der EU hinsichtlich der lokalen Speicherung von Daten (personenbezogene, notwendige, sensible usw.).

Mit anderen Worten: "Was ist rechtlich zur Speicherung von Cookies (bzw. localStorage) zu beachten!".

- ❑ **Erstelle aus deinem Rechercheergebnis eine Webseite.**

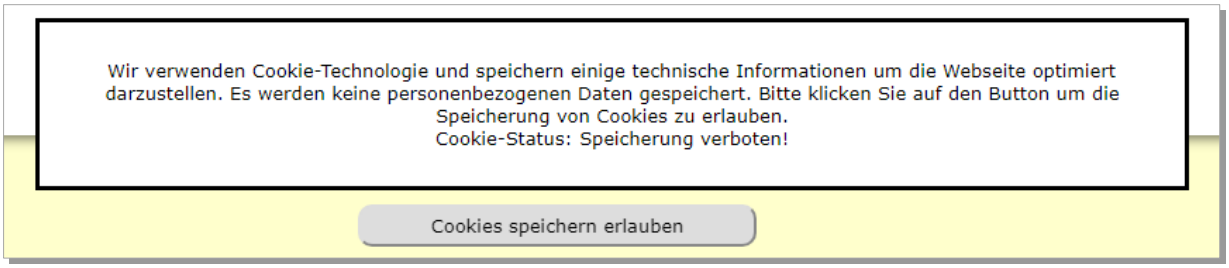
Übung B: Cookie OK

Öffne dein Webdokument aus der **Übung A: Datenschutzverordnung** und erweitere es um eine Schaltfläche zur ausdrücklichen Erlaubnis zum Speichern von Cookies.

Klickt der User auf die Schaltfläche, dann wird die Erlaubnis als Cookie gespeichert. Erlaubt der User die Verwendung von Cookies, dann soll bei einem weiteren Öffnen der Webseite, die Erlaubnis nicht wieder erscheinen.

- ❑ **JS** **Abfragen und setzen von Cookies**
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Teste deine Seite**

Cookies sind nur innerhalb einer Domain gültig. Localhost bzw. 127.0.0.1 werden von den Browsern als Domain interpretiert.



Übung C: Gescrollt

Öffne deine Lösung von **Übung B: Cookie OK** und ändere das Aussehen mit CSS. Man soll im Dokument vertikal scrollen können. Um den Bildlauf zu ändern können z. B. die Schriftgröße geändert, das Padding erhöht oder der Zeilenabstand vergrößert werden. Das Dokument kann auch um zusätzlichen Text erweitert werden.

Die vertikale Scrollposition soll lokal gespeichert werden. Bei einem erneuten Öffnen des Webdokuments scrollt ein JavaScript automatisch zur letzten Position – aber nur, wenn die Speicherung von Cookies (Übung B) erlaubt wurde!

- ❑ **JS** `.pageYOffset` und `.scrollTo();` Lokal speichern wahlweise mit **localStorage** oder als **Cookie**.
- ❑ **CSS** Text so verändern, dass vertikal gescrollt werden kann.

16.1 EventListener

Ereignisse bzw. Events können mit **`addEventListener()`** *an jedes HTML Element angehängt werden. Dabei können beliebig viele Events auf ein HTML Element gelegt werden (sogar vom gleichen Typ). Die Events können dynamisch vergeben werden und erleichtern damit auch die Lesbarkeit vom HTML-Code.*

```
HTML   <button id="Schalter" onClick="starte();">Drück mich</button>
JS     document.getElementById("Schalter").addEventListener("click", starte);
```

JS

`.addEventListener(`*Event*`, `** *Funktion*`);`**

Der Event wird mit Anführungszeichen definiert.

```
<button id="Sender">Absenden</button>

<script>
 document.getElementById("Sender").addEventListener("click", Sende);
 function Sende() {alert("Absenden"); }
</script>
```

JS

`.removeEventListener(`*Event*`, `** *Funktion*`);`**

Ein dynamisch hinzugefügter Event kann mit **`.removeEventListener`** wieder entfernt werden!

```
document.getElementById("Sender").removeEventListener("click", Sende);
```

Am besten schreibt man den JavaScript-Code am Schluss des **`<body>`** *Elements. Dann ist der DOM eingelesen und die Eventlistener können definiert werden!*

Druck und Zwischenablagen Events

Die Events für Druck und Zwischenablage dürfen in allen Elementen vorkommen außer im `<head>`*. Die Events können direkt auf das globale window Objekt gelegt werden!*
z. B. `window.addEventListener("afterprint", meldung);`

Druck	**`afterprint`**	Nach dem Druckauftrag, bzw. nach geschlossener Druckvorschau
	`beforeprint`	Vor dem Druck-Dialog
Zwischenablage	**`copy`**	Wenn eine Auswahl in die Zwischenablage kopiert wird
	`cut`	Wenn eine Auswahl in die Zwischenablage ausgeschnitten wird.
	`paste`	Beim Einfügen aus der Zwischenablage.

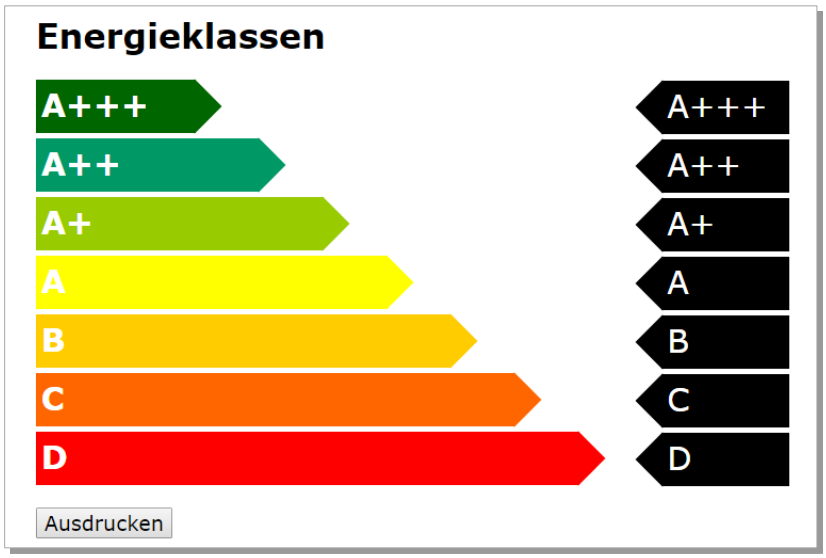
```
window.addEventListener("copy", meldung);
function meldung() {alert("Urheberrechte beachten"); }
```

Ü 16.1 EventListener

Übung A: Energieklassen

Erstelle eine Webseite mit einer Aufstellung von Energieeffizienzklassen (siehe unten). Füge noch einen Button für den Druck hinzu.
Sobald der User einen Druckbefehl erteilt, soll das aktuelle Datum mit Uhrzeit und einem Copyright Vermerk hinzugefügt werden.

- ❑ **HTML** **Ein Webdokument mit Energieklassen Darstellung (siehe unten)**
- ❑ **JS** **Eventlistener (`beforeprint` und `afterprint`)**
- ❑ **CSS** **Medienabfrage für den Druck**



Übung B: Energieklassen II

Öffne dein Lösung von **Übung A: Energieklassen**. Alle schwarzen Pfeile (rechts) sollen ausgeblendet werden. Per Klick auf eine farbige Energieklasse soll der dazupassende schwarze Pfeil wieder erscheinen.

Am Ausdruck darf auch nur **ein** schwarzer Pfeil sein, natürlich jener der angeklickt wurde. Die farbigen Pfeile bleiben wie sie sind.

- ❑ Löse dieses Beispiel mit Eventlistener
- ❑ **JS** **Eventlistener und .style Anweisungen**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

Code-Tipp: Hier ein Eventlistener, der eine Funktion mit dem this Argument startet:

```
.addEventListener("click", function(){einblenden(this);} )
```

16.2 Keyboard Events

Mit einem Keyboard Event kann ein Tastendruck ermittelt werden. Die Events können als Attribut in einem HTML Element definiert werden bzw. dynamisch mit einem Eventlistener. Ein Objekt wäre z. B. `var objekt = document.querySelector("input");`

onkeydown	feuert sobald eine Taste heruntergedrückt wird
JS	`objekt.addEventListener("keydown", meineFunktion);`
HTML	`<div onkeydown="meineFunktion();">`

onkeyup	feuert sobald eine Taste losgelassen wird
JS	`objekt.addEventListener("keyup", meineFunktion);`
HTML	`<div onkeyup="meineFunktion();">`

onkeypress	wenn Taste heruntergedrückt und festgehalten wird
JS	`objekt.addEventListener("keypress", meineFunktion);`
HTML	`<div onkeypress="meineFunktion();">`

Objektübergabe mit `event`

Startet man eine Funktion mit der Objektübergabe **`event`**, *dann wird der Tastendruck an die Funktion übergeben. Mit* **`event.key`** *wird der Tastendruck ausgewertet.*

JS

```
var meineTaste = event.key;
```

```
<input type="text" onKeyDown="welcheTaste(event);" >
<script>
  function welcheTaste(event) {
       var meineTaste = event.key;
       alert(meineTaste);      }
</script>
```

`event.keyCode` Eigenschaft

JS

`event.keyCode` ← Gibt den Code der Taste zurück.
z. B.: a = 97, b = 98, Enter = 13

```
function welcheTaste(event) {
   if(event.keyCode == 13) {alert("Enter wurde gedrückt");}   }
```

Weitere Eigenschaften die einen Unicode zurückgeben sind:
`event.charCode` *oder* `event.which`.

Beachte: `event.which` *wird nicht vom Internet Explorer unterstützt und* `event.keyCode` *funktioniert nicht als* `onkeypress` *Event im Firefox.*

Deshalb gibt es eine Cross-Browser-Solution in der einfach beide Eigenschaften vereinbart werden (abhängig vom Browser):

```
var meineTaste = event.which || event.keyCode;
```

Ü 16.2 Keyboard Events

Übung A: Wordpress Suche

Wordpress.com ist einer der größten Blog Anbieter im Netz. Demgemäß gibt es auch eine Vielzahl an Blogs mit Blogbeiträgen (auch zum Thema JavaScript).

Schreibe eine Webseite mit einem Suchfeld ohne Buttons. Sobald ein Suchbegriff eingeben und die Enter (bzw. Return) Taste gedrückt wurde, soll sich wordpress.com öffnen und die Suche automatisch ausführen.

- ❑ **HTML** **Ein Suchfeld**
- ❑ **JS** **Keyboard Event (Enter- bzw. Returntaste)**
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Alternativ kann auch eine Suche auf www.post.at gescriptet werden!**

Gib auf wordpress.com ins Suchfeld einen Begriff ein und analysiere dann die URL.

Übung B: Schriftgröße

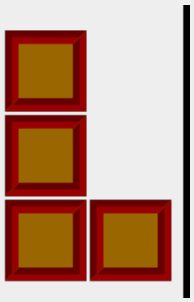
Erstelle eine Webseite mit einem beliebigen Fülltext.
Mit einem Tastendruck auf die Minustaste (-) wird die Schriftgröße um 1pt kleiner.
Mit einem Tastendruck auf die Plustaste (+) wird die Schriftgröße um 1pt größer.

- ❑ **HTML** **Ein <p> mit einem coolen Text. Thema: frei wählbar.**
- ❑ **JS** **Keyboard Event mit .style Anweisung**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

Übung C: Tetris

Lasse ein L-förmiges Tetriselement langsam vom Bildschrimrand oben herunterfallen. Mit den Pfeiltasten kann das Element nach links oder rechts navigiert werden. Mit der Pfeiltaste hinauf und hinunter wird das Element gedreht.

- ❑ **HTML** Ein L-Förmiges Tetriselement
- ❑ **JS** Alle notwendigen Funktionen
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ Automatisches Herunterfallen
- ❑ ← Pfeil Taste: Bewegung nach Links
- ❑ → Pfeil Taste: Bewegung nach Rechts
- ❑ ↑ Pfeil Taste: Drehung gegen den Uhrzeigersinn
- ❑ ↓ Pfeil Taste: Drehung im Uhrzeigersinn



16.3 Mouse Event Typen

Die Mouse Events starten Funktionen sobald der Mauszeiger bewegt, bzw. wenn ein Mousebutton gedrückt wird.

click	Wird ausgeführt, sobald auf das Element mit der linken Maustaste gedrückt wird
JS	`objekt.addEventListener("click", meineFunktion);`
HTML	`<div onclick="meineFunktion();">`

```
<h2 onClick="textAn()">Großer Flohmarkt</h2>
<h2 id="h2u">Kleiner Megastore</h2>
<script>
 document.getElementById("h2u").addEventListener("click", textAn);
      function textAn() {
      alert("Auch Sie sind herzlich eingeladen!");    }
</script>
```

dblclick	Feuert nach einem Doppelklick mit der linken Maustaste
JS	`objekt.addEventListener("dblclick", meineFunktion);`
HTML	`<div ondblclick="meineFunktion();">`

contextmenu	Startet eine Funktion, wenn die rechte Maustaste gedrückt wird
JS	`objekt.addEventListener("contextmenu", meineFunktion);`
HTML	`<div oncontextmenu="meineFunktion();">`

mouseenter	sobald der Mauszeiger auf ein Element bewegt wird.
JS	`objekt.addEventListener("mouseenter", meineFunktion);`
HTML	`<div onmouseenter="meineFunktion();">`

mouseleave	sobald der Mauszeiger das Element verlässt.
JS	`objekt.addEventListener("mouseleave", meineFunktion);`
HTML	`<div onmouseleave="meineFunktion();">`

```
<div id="inhalt" style="height:200px; width:200px;
                        background:blue; margin:auto;"></div>
<script>
   var mausEle = document.getElementById("inhalt");
   mausEle.addEventListener("mouseenter", rein);
   mausEle.addEventListener("mouseleave", raus);
   function rein() {mausEle.style.backgroundColor = "green";}
   function raus() {mausEle.style.backgroundColor = "red";}
</script>
```

mousemove	feuert jedes mal, wenn die Mouse in einem Element bewegt wird.
JS	`objekt.addEventListener("mousemove", meineFunktion);`
HTML	`<div onmousemove="meineFunktion();">`

16.4 Mouse Eigenschaften

Das Mouse Event Objekt ermittelt die Maustasten bzw. die Position des Mauszeigers. z. B.: **`onClick="meineFunktion(event)"`** *Zusätzlich kann abgefragt werden, ob eine Funktionstaste (Shift, Alt, Ctrl) gedrückt wurde.*

JS

`.buttons`

Gibt eine Zahl zurück die einer Maustaste entspricht. Wenn zwei oder mehr Tasten gedrückt wurden, wird die Zahl addiert: Linke Maustaste (1) plus rechte Maustaste (2) ergibt (3). Alternative Eigenschaften sind: `.button` und `.which`

1.....Linke Maustaste	2.....rechte Maustaste
4.....Mausrad-Button	8.....Vierte Maustaste
16.....Fünfte Maustaste	

```
<div style="background: green; height: 5em;"
     onMouseDown="mausTaste(event);"> </div>
<script>
   function mausTaste(event) {
     var taste = event.buttons;
     switch(taste) {
          case 1: console.log("Linke Taste"); break;
          case 2: console.log("Rechte Taste"); break;
          case 3: console.log("Linke und Rechte Taste"); break;
          case 4: console.log("Mittlere Taste"); break;}}
</script>
```

Das `onMouseDown` Attribut startet eine Funktion sobald eine Maustaste gedrückt wird – egal ob linke oder rechte Maustaste.

JS

`.offsetX` ← Position des Mauszeiger über die X-Achse eines Elements.

`.offsetX` ← Position des Mauszeiger über die X-Achse eines Elements.

Die Eventeigenschaften .offsetX und .offsetY geben die Position des Mauszeigers innerhalb eines HTML Elements zurück.

```
<textarea onMouseMove="mausPosition(event);"> </textarea>
<p>Koordinaten X: <span id="xAchse"></span></p>
<p>Koordinaten Y: <span id="yAchse"></span></p>
<script>
  function mausPosition(event) {
    document.getElementById("xAchse").innerHTML = event.offsetX;
    document.getElementById("yAchse").innerHTML = event.offsetY;}
</script>
```

`.pageX` und `.pageY` ermitteln die Position relativ zum Dokument.
`.screenX` und `.screenY` ermitteln die Position relativ zum Bildschirm.
`.clientX` und `.clientY` ermitteln die Position relativ zum Viewport.

Funktionstasten in Verbindung mit einem `onMouseDown` Event.

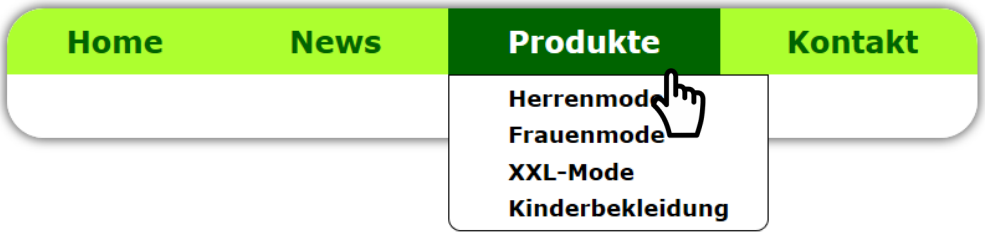
`.altKey`	Bei gleichzeitig gedrückter Alt Taste und Maustaste.
`.ctrlKey`	Bei gleichzeitig gedrückter CTRL Taste und Maustaste.
`.shiftKey`	Bei gleichzeitig gedrückter Shift Taste und Maustaste.

Ü 16.4 Mouse Eigenschaften

Übung A: Navigation

Erstelle eine horizontale Navigation mit den Buttons für Home, News, Produkte und Kontakt. Wenn die Maus über den Navigationsbutton Produkte fährt, erscheint ein Drop-Down Menü mit den Unterpunkten: Herrenmode, Frauenmode, XXL-Mode und Kinderbekleidung.

- ❑ **HTML** **Navigation mit HTML Elementen**
- ❑ **JS** **Hover-Effekt mit JavaScript**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**



Übung B: Fadenkreuz

Schreibe ein Script, dass über den gesamten Viewport ein Fadenkreuz legt. Also eine durchgezogene Linie vom Bildschirmrand links nach rechts und von oben nach unten. Das Fadenkreuz verändert sich mit der Position der Maus – der Schnittpunkt des Fadenkreuz ist also die Mausposition.

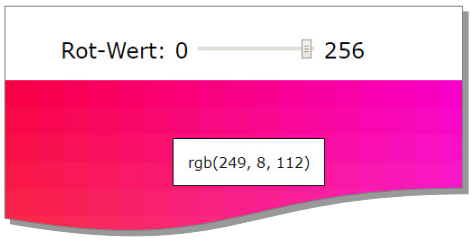
- ❑ **JS** **Entsprechende Mouseevents und Eigenschaften**

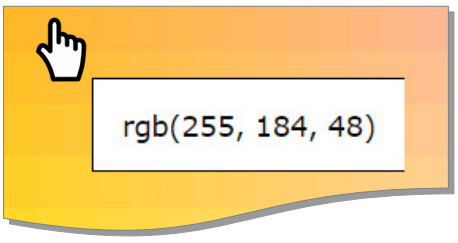
Die CSS Eigenschaft `cursor:crosshair;` verwandelt den Mauszeiger in ein Fadenkreuz. `cursor:none;` blendet den Mauszeiger komplett aus.

Übung C: Color Picker

Füge eine Farbpalette für einen bestimmten Rot-Wert ein. Sobald sich der Rot-Wert verändert, wird die Farbpalette (Grün und Blau-Werte) aktualisiert. Wenn man mit dem Mauszeiger über eine Farbe fährt, soll der RGB-Wert in einem Tool-Tip neben dem Mauszeiger angezeigt werden.

- ❑ **HTML** **Schieberegler für den Rot-Wert, Ausgabeelement für die Farbpalette.**
- ❑ **JS** **Farbpalette erzeugt durch Zählschleifen, Mouseevents und Eigenschaften für den Tool-Tip**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**





16.5 Touch Events

Natürlich sollen unsere Webseiten auch auf mobilen Geräten gut funktionieren. Dafür gibt es Touch Events für berührungssensitive Touchscreens. Sie ähneln den Mouse-Events.

touchstart	Beginn der Berührung ⇔ mousedown
JS	`objekt.addEventListener("touchstart", meineFunktion);`
HTML	`<div ontouchstart="meineFunktion();">`
touchmove	Wischen ⇔ mousemove, pointermove
JS	`objekt.addEventListener("touchmove", meineFunktion);`
HTML	`<div ontouchmove="meineFunktion();">`
touchend	Ende der Berührung ⇔ mouseup, pointerup
JS	`objekt.addEventListener("touchend", meineFunktion);`
HTML	`<div ontouchend="meineFunktion();">`
touchcancel	Abbruch der Berührung ⇔ mouseup, pointerup
JS	`objekt.addEventListener("touchcancel", meineFunktion);`
HTML	`<div ontouchcancel="meineFunktion();">`

JS

.targetTouches

Die `.targetTouches` Eigenschaft liefert eine TouchList mit Touch Objekten für jeden Finger der das aktuelle Element berührt.
z. B. `.targetTouches[0]` ← für den ersten Finger
Das Beispiel ermittelt, wieviel Finger ein `<div>` berühren.

```
<div ontouchstart="meinTouch(event);">
<p id="ausgabe" style="height: 500px; background: green;">
</p></div>
<script>
   function meinTouch(event) {
     var wieViele = "Anzahl der Finger: ";
     wieViele += event.targetTouches.length;
     document.getElementById("ausgabe").innerHTML = wieViele;}
</script>
```

JS

.touches ← Liefert eine TouchList mit aktuellen Berührungen am Screen

Das Beispiel ermittelt die Koordinaten des Touch Events.

```
<body ontouchstart="zeigeXY(event);" ontouchmove="zeigeXY(event);">
    <div id="anzeige" style="height:400px; background:red"></div>
    <script>
     function zeigeXY(event) {
       var x = event.touches[0].clientX;
       var y = event.touches[0].clientY;
       document.getElementById("anzeige").innerHTML = x + ", " + y;}
     </script></body>
```

17.1 Audio

JSL171
Europahymne.mp3

Mit HTML5 ist es möglich, ganz einfach ein Audioelement mit `<audio>` *hinzuzufügen.*
`<audio src="Europahymne.mp3" controls>.`
Hier wird gezeigt, wie ein Audiofile mit JavaScript geladen und gespielt wird.

***Audio Element** als Knoten hinzufügen.*
Das Audioelement wird zum **<body>** *hinzugefügt und mit der* **.src** *Eigenschaft wird die Quelle ausgewählt (diese kann absolut oder relativ sein).*

```
var audioSpieler = document.createElement("audio");
document.body.appendChild(audioSpieler);
audioSpieler.src = "Europahymne.mp3";
```

audioSpieler *wurde als globales Objekt vereinbart. Damit kann es über eine Funktion abgespielt werden.*
Die Funktion kann z. B. von einem Button gestartet werden:

```
<button onClick="abspielen();">Abspielen</button>
```

oder als onLoad Attribut im Body Tag `<body onLoad="abspielen();">` *dann wird das Soundfile sofort gestartet.*

```
function abspielen(){audioSpieler.play();}
```

Mit einer weiteren Funktion lässt sich das Audiofile pausieren.

```
<button onClick="pause();">Pause</button>
```

```
function pause(){audioSpieler.pause();}
```

Audio Eigenschaften

Alle Eigenschaften sind sowohl als **get** *wie auch* **set** *verfügbar. Sie können also ausgelesen bzw. gesetzt werden. (*`.duration` *ist natürlich nur read-only).*

JS	**.currentSrc**	← Die URL des aktuellen Soundfile.

```
console.log(audioSpieler.currentSrc);
```

JS	**.loop**	← Boolesch (true|false), Bei true als Endlosschleife.
	.muted	← Boolesch (true|false), Bei true wird es auf Lautlos gestellt.

```
audioSpieler.loop = true;
audioSpieler.muted = false;
```

JS	**.volume**	← Liest oder setzt die Lautstärke. 0.3 = 30 % Lautstärke.
	.currentTime	← Liest oder setzt die Position in Sekunden.

```
audioSpieler.volume = 0.3;
audioSpieler.currentTime = 3.43;
```

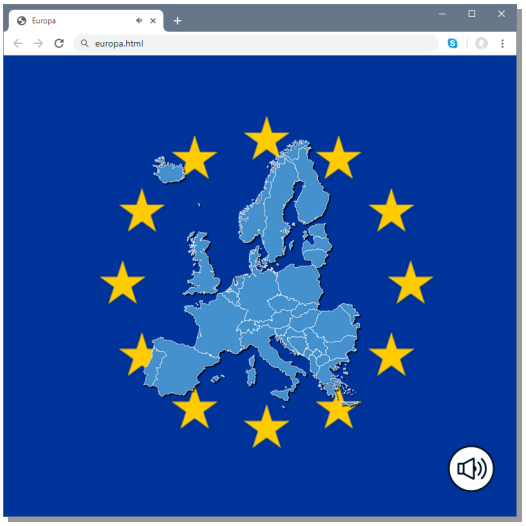
JS	**.controls**	← (true|false) Anzeige mit oder ohne Kontrollpaneel
	.duration	← Read-only, gibt die Länge des Soundfile zurück.

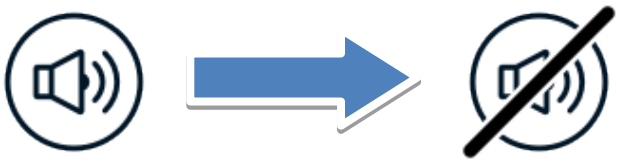
Übung A: Europa

Erstelle eine Webseite mit der EU-Flagge im Hintergrund und einem transparenten PNG von Europa (EU-Mitgliedstaaten) im Vordergrund. Beide Bilder findest du im Internet.

Sobald die Webseite im Browser geöffnet wird, soll automatisch die Europahymne abgespielt werden. Die Controll-Elemente des Audio-Elements dürfen dabei nicht aufscheinen – jedoch soll ein Button zum Lautlos schalten bzw. Pausieren hinzugefügt werden.

- ❑ Sichtbar ist: Die EU-Flagge, Europa PNG und ein "Lautlos-Button".
- ❑ **JS** **Automatisches Abspielen eines MP3**
  Funktion zum Lautlos-Schalten bzw. Pause.
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**





Übung B: Sound-Design

Scripte ein Sound-Design für Buttons in einer Navigation. Soll heißen: Bei einem MouseOver, bei einem MouseClick usw. wird ein Geräusch abgespielt. Im Hintergrund soll ein Musik-Bett laufen! Suche im Internet nach passenden Sounds oder erstelle sie selbst (z. B. mit Audacity).

- ❑ **HTML** **mind. 3 Navigationsbuttons**
- ❑ **JS** **Diverse Events, Abspielen von Sounds**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**

Übung C: MP3 Player

Erstelle einen eigenen online MP3 Player.

- ❑ Einzige Vorgabe: keine Controls im <audio> Element.
- ❑ **HTML** **Freies Design**
- ❑ **CSS** **Freies Design**

17.2 Video

JSL172
croissants.mp4

Für das **<video>** *Element gibt es zahlreiche JavaScript Methoden und Eigenschaften. Die JavaScript Eigenschaften und Methoden vom* **<audio>** *Element (siehe 17.1 Audio) sind auch für das* **<video>** *Element anwendbar.*

JS

.load(); ← Ladet ein neues Video.

Die Wenn im **<video>** Element mit **<source>** mehrere Videofiles definiert wurden (z. b. auch bei einem Fallback), kann mit **.load()** und der ID ein bestimmtes Video geladen werden.

```
<button onclick="meinVideo();">Anderes Video</button>
 <video id="meinVideoElement" controls autoplay>
   <source id="mp4Video" src="clip.mp4" type="video/mp4">
   <source id="oggVideo" src="clip.ogg" type="video/ogg">
    Dein Browser unterstützt kein HTML5 video.
</video>
<script>
   function meinVideo() {
      document.getElementById("mp4Video").src = "movie.mp4";
      document.getElementById("oggVideo").src = "movie.ogg";
      document.getElementById("meinVideoElement").load();    }
</script>
```

JS

.canPlayType(Wert**);**

Prüft ob ein bestimmtes Video-Format (bzw. ein Codec) vom Browser unterstützt wird. Die Methode gibt "probably" ➔ der Browser unterstützt den Video-Typ, "maybe" ➔ der Browser unterstützt den Video-Typ aber nicht unbedingt den Codec. oder "" ➔ (leerer String) zurück, in diesem Fall wird der Video-Typ definitiv nicht unterstützt.

Typische Werte	Typische Wert mit Codecs
video/ogg	video/ogg; codecs="theora, vorbis"
video/mp4	video/mp4; codecs="avc1.4D401E, mp4a.40.2"
video/webm	video/webm; codecs="vp8.0, vorbis"
audio/ogg	audio/ogg; codecs="vorbis"
audio/mp4	audio/mp4; codecs="mp4a.40.5"
audio/mpeg	

```
function dasVideoStarten() {
    var meinVideo = document.createElement('video');
    var spieltEs = meinVideo.canPlayType('video/mp4');
    if (spieltEs == "") {
        alert("Kein Browser-Support für den Video-Typ");
    }
    else {meinVideo.src = "croissants.mp4";
          document.body.appendChild(meinVideo);
          meinVideo.play();  } }
```

JS

.preload = ""

Mit der `.preload` Eigenschaft wird ein Video unmittelbar nachdem die Seite geladen wurde ebenfalls geladen. Werte sind: `auto` (sofortiges Laden), `metadata` (nur die Metadaten) bzw. `none` (kein Laden).
z. B. `meinVideo.preload = "auto";`

17.3 Audio|Video Events

JSL173
croissants.mp4

*Die Audio bzw. Video Events können als Attribut im **<audio>** und **<video>** Element vorkommen oder als Eventlistener in Form einer Objekt-Vereinbarung. Selbstverständlich feuern auch die meisten anderen Events (z. B. Maus, Keyboard, Touch usw.) mit den AV-Elementen. Hier ein kleiner Auszug der Audio-Video-Events:*

loadstart	Zu Beginn des Ladeprozess, wenn der Browser nach dem AV-Element sucht!
JS	`objekt.addEventListener("loadstart", meineFunktion);`
HTML	`<video onloadstart="meineFunktion();">`
progress	Sobald der Download beginnt
JS	`objekt.addEventListener("progress", meineFunktion);`
HTML	`<video onprogress="meineFunktion();">`
canplay	Sobald das Video geladen ist und genug Buffer hat, um es zu starten!
JS	`objekt.addEventListener("canplay", meineFunktion);`
HTML	`<video oncanplay="meineFunktion();">`
waiting	Feuert, wenn das Video-Frame gebuffert werden muss.
JS	`objekt.addEventListener("waiting", meineFunktion);`
HTML	`<video onwaiting="meineFunktion();">`
playing	Wenn nach einer Pause (bzw. wegen Bufferung) wieder abgespielt wird.
JS	`objekt.addEventListener("playing", meineFunktion);`
HTML	`<video onplaying="meineFunktion();">`
seeked	Wenn der User die Position des Videos verändert hat.
JS	`objekt.addEventListener("seeked", meineFunktion);`
HTML	`<video onseeked="meineFunktion();">`

*Beispiel **seeked**: Sobald der User die Zeit-Position im Video verändert hat, wird entweder 1. Abschnitt oder 2. Abschnitt im <h1> Element ausgegeben.*

```
<h1 id="status">Croissants backen!</h1>
<video controls src="croissants.mp4" id="meinVideo"></video>

<script>
  var dasVideoElement = document.getElementById("meinVideo");
  dasVideoElement.addEventListener("seeked", meineFunktion);

  function meineFunktion() {
    if (dasVideoElement.currentTime <= 30) {
       var ausgabeText = "1. Abschnitt";
    }
    else {var ausgabeText = "2. Abschnitt";}
    document.getElementById("status").innerHTML = ausgabeText;
  }
</script>
```

18.1 Canvas

Das **<canvas>** *Element ist ein Container für 2D und 3D Graphiken. Mit JavaScript kann man Pfade, Rechtecke, Kreise, Text und Bilder hinzufügen. Das* **<canvas>** *Element stellt eine Bitmap-basierte Leinwand mit einem kartesischen Koordinatensystem zur Verfügung. Der Ursprung* (0 | 0) *ist in der linken oberen Ecke.*

HTML

<canvas> … </canvas>

Wenn man die Höhe und Breite des Canvas als Attribute (**height, width**) definiert in Pixel und nicht über CSS bestimmt, werden die Standardeinstellungen des Browsers überschrieben.

Text und Content innerhalb des **<canvas>** Tags kann als Fallback verwendet werden falls der Browser keine Canvas unterstützt.

```
<canvas id="meinCanvas" height="300" width="600">
 Dein Browser unterstützt keine Canvas!</canvas>
```

JS

.getContext('2d');

Nachdem das **<canvas>** Element selektiert wurde, kann ein 2D Objekt vereinbart werden.

```
var leinwand = document.getElementById("meinCanvas");
var rahmen = leinwand.getContext('2d');
```

JS

.rect(x, y, Breite, Höhe) ← Erzeugt ein Rechteck

Im Beispiel wird ein Rechteck an der Position x = 30 und y = 100 (vom linken oberen Ursprung des canvas) mit einer Breite von 200 Pixel und einer Höhe von 80 Pixel gezeichnet.

```
rahmen.strokeStyle = "green";   // ← Definiert die Rahmenfarbe
rahmen.lineWidth = "5";         // ← Definiert die Rahmenbreite (5 Pixel)
rahmen.rect(30, 100, 200, 80);
rahmen.stroke();                // ← Zeichnet das Rechteck.
```

JS **.fillStyle** ← Bestimmt eine Hintergrundfarbe für das Rechteck

```
rahmen.fillStyle = "#FF0000";   // ← Roter Hintergrund
rahmen.fill();                  // ← Füllmethode
```

JS **.createLinearGradient(**x_0, y_0, x_1, y_1**);** ← Farbverlauf

Im Beispiel geht der Verlauf von oben (x_0 = 0 | y_0 = 0) nach unten (x_1 = 0 | y_1 = 300) mit den Farbverlauf von Blau nach Rot.

```
var verlauf = rahmen.createLinearGradient(0, 0, 0, 300);
    verlauf.addColorStop(0, "blue");
    verlauf.addColorStop(1, "red");
rahmen.fillStyle = verlauf;
rahmen.fill();
```

18.2 Canvas Pfade

Mit der `.beginPath()` *Methode kann ein Pfad in ein Canvas Element gezeichnet werden. Zur Erinnerung: Ein Canvas ist wie ein kartesisches Koordinatensystem mit dem Ursprung* (0 | 0) *in der linken oberen Ecke. Zuerst wird ein 2D-Objekt vereinbart. Am Objekt wird die* `beginPath()` *Methode ausgeführt. Mit* `.moveTo(x,y)` *wird der Startpunkt (Referenzpunkt) definiert und mit* `.lineTo(x,y)` *das Ziel.* `.closePath()` *schließt den Pfad – und kehrt wieder zurück zum Referenzpunkt.* `.stroke()` *zeichnet schlussendlich den Pfad.*

Ein grünes Dreieck

```
<canvas id="meinCanvas" width="300" height="300">
   Canvas wird nicht unterstützt!</canvas>
<script>
   var leinwand = document.getElementById("meinCanvas");
   var meinPfad = leinwand.getContext("2d");

   meinPfad.beginPath();                  // ← Pfad wird gestartet
   meinPfad.lineWidth = "5";              // ← Breite des Pfad in Pixel
   meinPfad.strokeStyle = "green";        // ← Farbe des Pfad (Grün)
   meinPfad.moveTo(10, 10);               // ← Referenzpunkt/Startpunkt
   meinPfad.lineTo(140, 140);             // ← Erstes Ziel
   meinPfad.lineTo(290, 10);              // ← Zweites Ziel
   meinPfad.closePath();                  // ← zurück zum Referenzpunkt
   meinPfad.stroke();                     // ← Pfad wird gezeichnet
</script>
```

Sobald der Pfad mit `.closePath()` *geschlossen wurde kann das Dreieck mit einer Farbe (bzw. einem Verlauf) gefüllt werden.*

```
meinPfad.fillStyle = "red";
meinPfad.fill();
```

Weitere Methoden

JS 	**`.save();`** **`.restore();`** Speichert das Abbild des Pfades (z. B. Farbe). Mit **`.restore()`** wird das Abbild wieder zurückgesetzt.
JS 	**`.clip();`** Sobald ein Pfad mit **`.clip()`** gestartet wird, werden weitere Zeichnungen nur mehr innerhalb des Pfades dargestellt.
JS 	**`.isPointInPath(x, y);`** Prüft ob der Punkt (x, y) innerhalb des Pfades ist. Rückgabe ist boolesch.

```
alert(meinPfad.isPointInPath(50, 60));
```

18.3 Canvas Text & Bild

In Canvas' können auch Texte und Bilder angezeigt werden. Für Texte muss über die `.font` *Methode die Schriftart und –größe definiert werden.*

Text hinzufügen

JS

`.strokeText(`text, x, y**`);`**

Die .strokeText Methode zeigt nur den Umriss des Text an.
Über die .font Methode wird zuerst Schriftgröße und Schriftart definiert.

`text`	Text der angezeigt werden soll
`x`	x-Position vom linken Rand des Canvas in Pixel
`y`	y-Position vom oberen Rand des Canvas in Pixel

```
<canvas id="meinCanvas" width="300" height="200">
    Ihr Browser unterstützt keine Canvas!
</canvas>
<script>
    var dasCanvas = document.getElementById("meinCanvas");
    var ctx = dasCanvas.getContext("2d");
    ctx.font = "40px Arial";
    ctx.strokeText("Willkommen",40,100);
</script>
```

Mit `.fillText()` *wird Textfüllung angezeigt.*

```
    ctx.fillStyle = "blue";
    ctx.fillText("Willkommen",40,100);
```

Ein Bild darstellen

JS

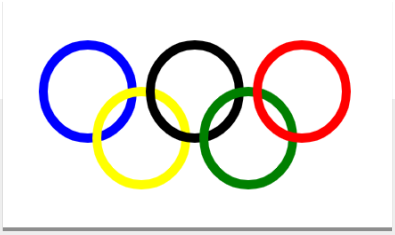
`.drawImage(img, x, y);`

Zuerst wird das Canvas-Objekt vereinbart und ein Image-Objekt dimensioniert. Um die `.drawImage` Methode auszuführen, sollte das Image-Objekt (`src`) vollständig geladen sein. Entweder man selektiert also ein img-HTML Element oder man führt die `.drawImage` Methode in einer eigenen Funktion aus, die über ein `.onload` Event gestartet wird. Genauer genommen, das vollständige Laden des Image-Objekts.

```
<canvas id="meinCanvas" width="500" height="500"
    Ihr Browser unterstützt keine Canvas!
</canvas>
<script>
    var dasCanvas = document.getElementById("meinCanvas");
    var ctx = dasCanvas.getContext("2d");
    var meinBild = new Image();

    meinBild.onload = function() {
      ctx.drawImage(meinBild, 50,50);
    }
    meinBild.src = "pointer.png";
</script>
```

Ü 18.3 Canvas



Übung A: Olympische Ringe

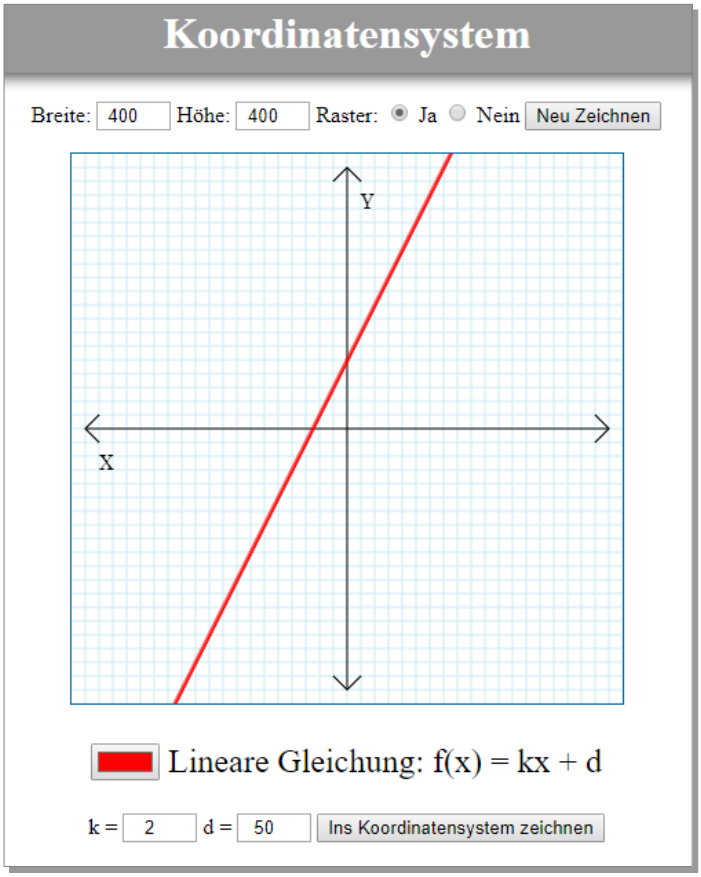
Auf einem Canvas Element sollen die Olympischen Ringe dargestellt werden.

- ❑ Recherchiere im Internet, wie man auf einem Canvas Kreise zeichnet
- ❑ Die Ringe sollen in unterschiedlichen Farben dargestellt werden.
- ❑ **Animiere die Darstellung der Ringe**

Übung B: Koordinatensystem

Zeichne ein kartesisches Koordinatensystem in ein Canvas Element. Zusätzlich soll es noch möglich sein, eine lineare Gleichung f(x) = kx + d ins Koordinatensystem zu zeichnen.

- ❑ **HTML** **Canvas Objekt, Buttons, Input-Felder**
- ❑ **JS** **Ein Koordinatensystem zeichnen**
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Erweitere das Koordinatensystem nach eigenen Vorstellungen (z. B. ein Raster).**



Tipp: Ebenen

Wenn man einmal mehrere Ebenen benötigt (ähnlich wie bei Photoshop) dann kann man zwei oder mehrere `<canvas>` Elemente mit der CSS Eigenschaft `position:absolute;` übereinanderlegen.

CSS:

```
.meinCan {position:absolute;}
```

HTML:

```
<div>
  <canvas id="meinCanvas1"
          class="meinCan"
          height="300"
          width="300">
  </canvas>
  <canvas id="meinCanvas2"
          class="meinCan"
          height="300"
          width="300">
</canvas>
</div>
```

19.1 Geolocation

Die Geolocation API liefert die geographische Position eines Users bzw. Device. Dabei greift die API auf IP-Basis, WLAN-Netzwerkdaten, Funk-Signale oder GPS zu und ermittelt so die Position des Device (z. B. Tablett, Smartphone). Wegen dem Datenschutz wird der_die User_in oft aufgefordert, den Zugriff auf den Standort zu erlauben. Darüber hinaus erlauben moderne Browser den Zugriff auf den Standort nur in einem "sicheren Kontext" (d. H. HTTPs, lokale Dateien, localhost). `Geolocation` *ist ein Objekt von* `window.navigator.` → `navigator.geolocation.`

JS

`.getCurrentPosition(`*zeigePositionFunktion, zeigeFehlerFunktion***`);`**

Im Beispiel wird zuerst geprüft, ob der Browser das `.geolocation` Objekt unterstützt. Ist diese Prüfung `true`, dann ruft die `.getCurrentPosition()` Methode eine Funktion auf welche die Positionsdaten übergibt und eine Funktion die etwaige Fehler auswertet.

```
function ermittlePosition() {
     var meinePos = navigator.geolocation;
     if (meinePos) {
         meinePos.getCurrentPosition(zeigePosition, zeigeFehler);}
   else {alert("Ihr Browser unterstützt keine Geolocation.");}
}
```

JS

`.coords.`

Die `zeigePosition` Funktion übernimmt die Positionsdaten welche über die `.coords` Eigenschaft ausgelesen werden können.

Folgende Eigenschaften können ausgelesen werden:

`.coords.latitude`.....Breitenangabe als Dezimalzahl
`.coords.longitude`.....Längenangabe als Dezimalzahl
`.coords.accuracy`.....Genauigkeit der Koordinaten (in Meter)
`.coords.altitude`.....Höhenangabe (über dem Meeresspiegel)
`.coords.altitudeAccuracy`.....Genauigkeit der Höhenangabe
`.coords.heading`.....Richtung
`.coords.speed`.....Geschwindigkeit (in m/s)
`.timestamp`.....Zeit der Positionsangabe

```
function zeigePosition(position) {
    alert("Breite: " + position.coords.latitude);
    alert("Länge: " + position.coords.longitude);
    alert("Höhe: " + position.coords.altitude); }
```

Etwaige Fehler werden in einer Funktion abgefangen und über eine Switch-Verzweigung ausgewertet.

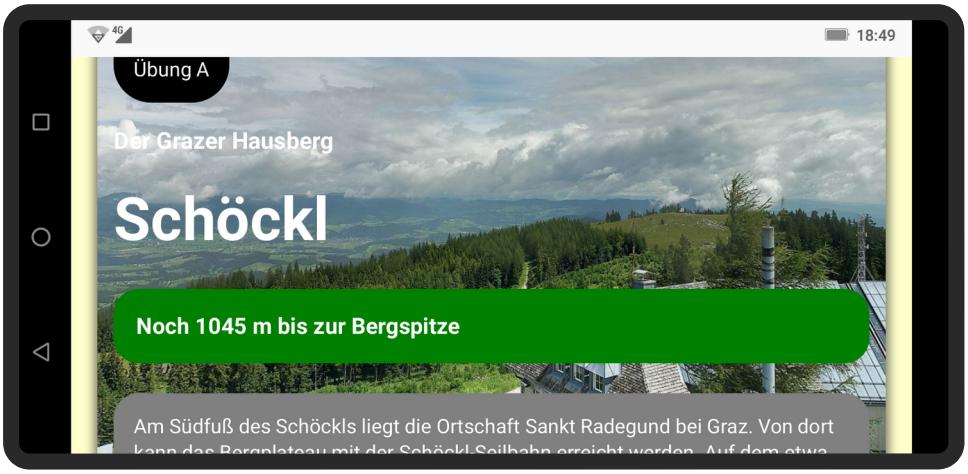
```
function zeigeFehler(fehler) {
  switch(fehler.code) {
    case fehler.PERMISSION_DENIED:
         alert("Benutzer lehnte Standortabfrage ab."); break;
    case fehler.POSITION_UNAVAILABLE:
         alert("Standortdaten sind nicht verfügbar."); break;
    case fehler.TIMEOUT:
         alert("Die Standortabfrage dauerte zu lange."); break;
    case fehler.UNKNOWN_ERROR:
         alert("unbekannter Fehler."); break;}}
```

Ü 19.1 Geolocation

Übung A: Berg-App

Schreibe eine Webseite über einen österreichischen Berg mit seinen Attraktionen und Besonderheiten. Zusätzlich soll ein Höhenmesser eingefügt werden, der die Entfernung von der aktuellen Position bis zur Bergspitze anzeigt.

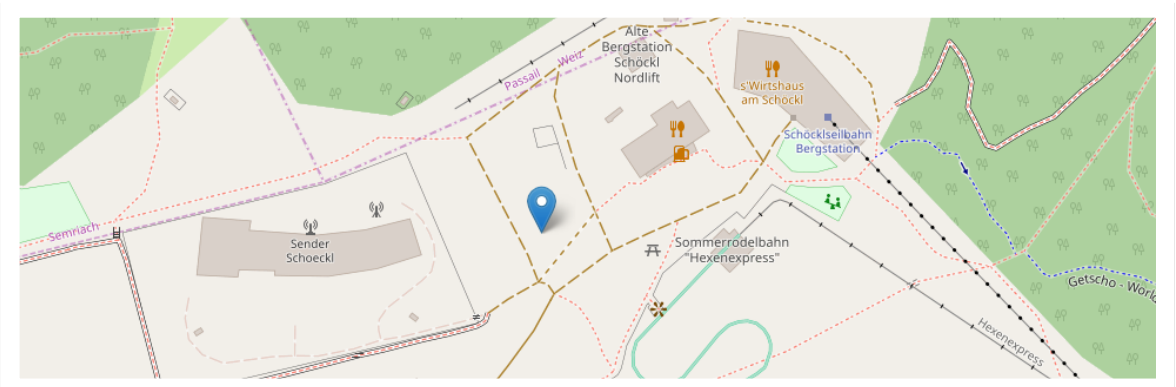
- ❑ Recherchiere im Internet die Funktionsweise der `.watchPosition()` Methode und nutze diese in deiner Webseite!
- ❑ **HTML** **Freie Wahl der Inhalte**
- ❑ **JS** **Geolocation Script (mit Error-Handler)**
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Teste deine Seite bei deiner nächsten Bergwanderung bzw. Ski-Urlaub!**



Übung B: Online Karte auf der Berg-App

Öffne deinen Lösung von Übung A: Berg-App und füge einen Button hinzu, der die aktuelle Position des Device in einer Online-Karte anzeigt.

- ❑ Verwende OpenStreetMaps.org oder Google Maps
- ❑ **HTML** **Iframe oder Link zum Mapanbieter**
- ❑ **JS** **Geolocation**
- ❑ **CSS** **Ansprechende Gestaltung (Freies Design)**



20.1 OOP Einführung

*Auch mit JavaScript ist **O**bjekt**o**rientiertes **P**rogrammieren möglich, wenn auch nicht in dem Ausmaß wie es in anderen Programmiersprachen möglich ist. Problemlos lassen sich aber eigene Objekte erstellen, die eigene Methoden und Eigenschaften zulassen. Ebenfalls ist auch eine Objekterstellung mit einer Konstruktorfunktion möglich.*
Am einfachsten wird ein Einzelobjekt mit dem Wort **`new`** *erzeugt.*

Zuerst vereinbaren wir ein neues Objekt!

```
var sparbuch = new Object;
```

Dem neuen Objekt werden die Eigenschaften `.name`, `.zinssatz`, `.laufzeit` und `.kapital` zugewiesen.

```
sparbuch.name = "Hans Berger";
sparbuch.zinssatz = 3.55;
sparbuch.laufzeit = 10;
sparbuch.kapital = 10000;
```

Auf die Objekteigenschaften kann problemlos zugegriffen werden:

```
console.log(sparbuch.kapital);
```

Methoden werden mit einer `function()` zugewiesen. Hier im Beispiel wird nach dem Methodenaufruf ein alert-Fenster mit den Eigenschaften angezeigt!

```
sparbuch.zinsmeldung = function() {
     var ausgabe =  "Kontoinhaber: " + sparbuch.name;
         ausgabe += "Zinsatz: " + sparbuch.zinssatz;
     alert (ausgabe);}
```

Der Aufruf der neuen Objektmethode verläuft wie gewohnt.

```
sparbuch.zinsmeldung();
```

Eine Methode zur Errechnung von Zinseszinsen sieht dann so aus – mit `return` wird das Rechenergebnis zurückgegeben!

```
sparbuch.rechne = function() {
var zw = Math.pow(1 + sparbuch.zinssatz / 100, sparbuch.laufzeit);
return sparbuch.kapital * zw;}
```

*Die Methode wird mit **`sparbuch.rechne();`** ausgeführt. Natürlich können den Eigenschaften andere Werte zugewiesen werden!*

```
sparbuch.laufzeit = 10;
alert("Das Endkapital beträgt: " + sparbuch.rechne());
```

20.2 OOP Literale

Es gibt noch eine zweite Schreibweise, wie man ein neues Objekt definieren kann. Dabei wird einfach eine Objektvariable vereinbart. `var obj = { … }` *Eigenschaften und Methoden werden nach dem Key-Value Prinzip erstellt. Der Key wird mit einem Doppelpunkt vom Value getrennt. Die Zuweisung wird mit einem einfachen Kommazeichen getrennt.*

Objektliteral
Es wird eine Eigenschaft (**girokonto.nutzer**) und eine Methode (**girokonto.anzeige**) mit einer alternativen Schreibweise erstellt.

```
var girokonto = {
    nutzer  : "Petra Müller",
    anzeige : function() {alert(girokonto.nutzer)}
}
girokonto.anzeige();
```

Unser Sparbuch Objekt kann also auch wie folgt gescriptet werden:

```
var sparbuch = {
    name        : "Hans Berger",
    zinssatz    : 4,
    laufzeit    : 5,
    kapital     : 1200,
    zinsmeldung : function() {
    var ausgabe =  "Kontoinhaber: " + sparbuch.name;
        ausgabe += "Zinsatz: " + sparbuch.zinssatz;
        alert (ausgabe);}
    }
```

Wenn in einer Methode auf eine Eigenschaft zugegriffen werden soll, empfiehlt es sich das mit der Anweisung **this** *zu tun. Die* **rechne()** *Methode muss innerhalb des eingezäunten* **sparbuch** *Objekt sein damit* **this** *darauf zugreifen kann!*

```
    rechne : function() {
    var zw = Math.pow(1 + this.zinssatz / 100, this.laufzeit);
             return this.kapital * zw;
    }
```

Natürlich kann innerhalb eines Objekts auch ein Array definiert werden!

```
var sparbuch = {auszahlung : [300, 100, 550, 300]}
  /* bzw. */
sparbuch.auszahlung = [300, 100, 550, 300];
```

Das Array wird dann wie gewohnt abgerufen:
Hier die Stelle **[2]** *also der Wert 550*

```
alert(sparbuch.auszahlung[2]);
```

20.3 OOP Konstruktor

Man kann sich einen Konstruktor als eine Art Container vorstellen, der alle Eigenschaften und Methoden bei der Vereinbarung mit **new** *an das neue Objekt übergibt. Im Gegensatz zu anderen Programmiersprachen, wo ein Konstruktor meist über eine Klasse definiert wird, benötigt man in JavaScript eine einfache Funktion → eine Konstruktorfunktion. Erst beim instanziieren mit* **new** *wird die normale Funktion zu einer objektorientierten Konstruktorfunktion.*

```
function GiroKonto() { this.form = "Giro" }
var meinKonto = new GiroKonto;
```

Konstruktorfunktion: Im Beispiel wird der Funktion ein String-Wert (`besitzer`) übergeben die der Eigenschaft `.name` zugewiesen wird.

```
function meinSparbuch(besitzer){
   this.name = besitzer;
   this.kapital = 1200;

   this.meldung = function() {alert("Kontoinhaber: " + this.name);}
}
```

Vereinbart wird das neue Objekt (`sBuch1`) mit `new` und dem Namen der Konstruktorfunktion (`meinSparbuch`). Übergeben wurde der Besitzernamen als String. In Folge wird die Eigenschaft `.kapital` in der Console ausgegeben und die Methode `.meldung()` aufgerufen.

```
var sBuch1 = new meinSparbuch("Hans Berger");
console.log(sBuch1.kapital);
sBuch1.meldung();
```

Eine weitere Objekt-Vereinbarung ist problemlos möglich!

```
var sBuch2 = new meinSparbuch("Claudia Klein");
sBuch2.meldung();
```

Mit `.prototype.` lässt sich der Konstruktor erweitern! Diese Anweisungen sollten außerhalb der Konstruktorfunktion definiert werden.
Hier um die Eigenschaften `.art` und `.dauer`.

```
meinSparbuch.prototype.art = "Prämiensparen";
meinSparbuch.prototype.dauer = 24;
var sBuch3 = new meinSparbuch();
alert("Art des Sparbuch: " + sBuch3.art);
```

Mit `.prototype` lässt sich sogar eine Konstruktorfunktion um eine zusätzliche erweitern. Man spricht dann von Vererbung:
`meinSparbuch.prototype = new GiroKonto;`

Ü 20.3 OOP Konstruktor

Übung A: Anhalteweg

Der Anhalteweg eines KFZ setzt sich zusammen aus dem Reaktionsweg und dem Bremsweg – also vom Erkennen einer Gefahr bis zum Stillstand des KFZ.

$$Reaktionsweg = \frac{Geschwindigkeit}{10} \times 3$$

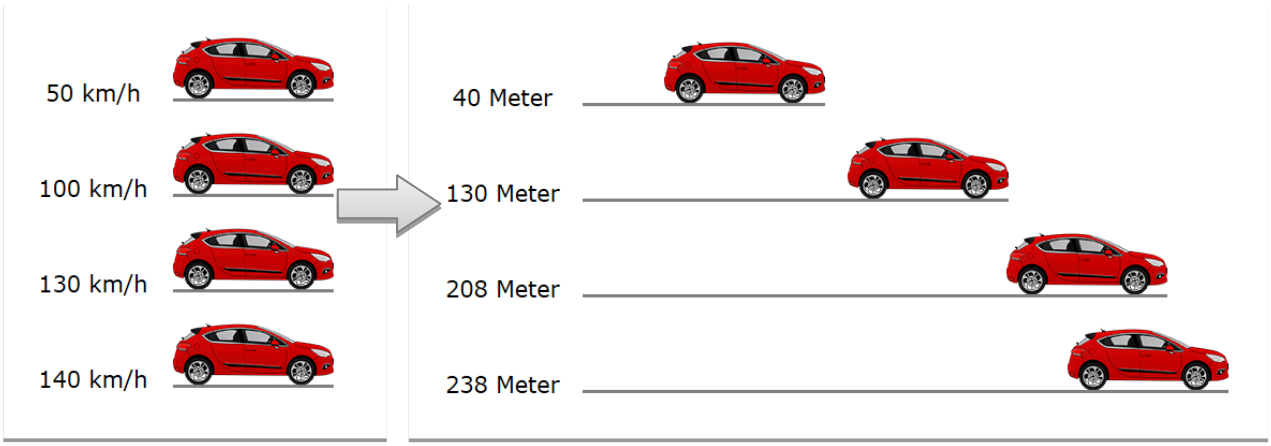
$$Bremsweg = \frac{Geschwindigkeit}{10} \times \frac{Geschwindigkeit}{10}$$

$$Anhalteweg = Reaktionsweg + Bremsweg$$

Scripte eine Webseite, die unterschiedliche Anhaltewege gegenüberstellt. Arbeite hierfür Objektorientiert mit einer Konstruktorfunktion mit dem Ziel, dass beliebig viele Objekte dimensioniert werden. Per Klick auf einen "Abbremsen-Button" soll der Anhalteweg errechnet und dementsprechend dargestellt werden.

- ❑ **JS** **Konstruktorfunktion, Eigenschaften und eine Methode**
- ❑ **CSS** Ansprechende Gestaltung (Freies Design)
- ❑ **Vereinbarung/Dimensionierung mit `new` und der Geschwindigkeit als Übergabe. Die Konstruktorfunktion heißt `meinAuto()`**

```
var auto1 = new meinAuto(50);      // ← auto1 fährt 50 km/h
var auto2 = new meinAuto(100);     // ← auto2 fährt 100 km/h
var auto3 = new meinAuto(130);     // ← auto3 fährt 130 km/h
var auto4 = new meinAuto(140);     // ← auto4 fährt 140 km/h
```



Übung B: OOP Präsentation

Erstelle eine Präsentation über weitere Feinheit der Objektorientierten Programmierung in JavaScript.

- ❑ **Themen**: Verschachtelung von Objektliteralen, Zugriffsrechte der Konstruktorfunktion, Anonyme Funktionen, Prototypen (Vererbung, Zugriffsrechte), Aggregation, Closures und Überschreiben …
- ❑ **Freie Wahl des Präsentationsmedium (z. B. eine Webseite, PowerPoint, ein Plakat, Mind-Map, am Whiteboard, einen Film usw.)**

20.4 JSON

*Mit JSON (**J**ava**S**cript **O**bject **N**otation) verfügt man über einen Syntax um Daten zu speichern, bzw. Daten auszutauschen. Diese Notation wandelt Objekte, Variablen und Arrays in einen reinen Text um, der dann z. B. mit* **`localStorage`** *gespeichert oder an ein PHP Skript übergeben werden kann. JSON ist ähnlich aufgebaut wie ein OOP Literal, nur werden hier die Keys immer unter doppelte Anführungszeichen gestellt.*

```
{ "Name" : "Karl König", "Alter" : 33 }
```

Folgende Datentypen werden durch JSON unterstützt.

String	`{ "wohnort":"Graz" }`
Zahlen	`{ "PLZ":8010 }`
JSON Objekt	`{ "Person":{"name":"Peter", "alter":25} }`
Arrays	`{ "Besucher":["Karl","Eva","Friedrich"] }`
Boolean	`{ "zugriff":true }`
JSON null	`{ "akademischerGrad":null }`

Gegenwärtig ist es nicht möglich eine Funktion oder ein Datum in eine JSON Notation zu wandeln. Möglich ist es aber, das Datum oder die Funktion als String-Typ anzugeben.

JS

`JSON.stringify(`objekt**`);`**

Die Methode wandelt ein JavaScript Objekt (oder Array bzw. Variable) in die JSON Notation. Das Beispiel hat eine Konstruktorfunktion `mitglied()`. Ein `xxlGruppe` Objekt wird vereinbart – die Eigenschaften `.name` und `.pincode` wurden verändert. Im Anschluss wird das `xxlGruppe` mit `stringify` in einen JSON Text gewandelt.

```
function mitglied() {
      this.name = "",
      this.frei = [3,6,4,22],
      this.pincode = 9999
}
var xxlGruppe = new mitglied;
    xxlGruppe.name = "Herbert Trinker";
    xxlGruppe.pincode = 3532;
var speichern = JSON.stringify(xxlGruppe);
console.log(speichern);
```

JS

`JSON.parse(objekt);`

In die andere Richtung geht es mit `JSON.parse();`
Dieses konvertiert den JSON Text in ein JavaScript Objekt.

```
var eineGruppe = JSON.parse(speichern);
console.log(eineGruppe);
```

21.1 Projekte

Projektplanung

Bevor die Programmierarbeit beginnt, soll eine kurze Planung des Projekts erstellt werden. Folgende Inhalte soll die Projektplanung beinhalten:

- ❑ Thema und Arbeitstitel des Projekts
- ❑ Beschreibung des Projekts
- ❑ Ziele des Projekts (aufgeteilt in Meilensteine → Grobziele → Feinziele)
- ❑ Aufgaben der Teammitglieder (bei Partner bzw. Gruppenarbeit)
- ❑ Beschreibung der Projektphasen (Einsatz der Technologien, Dauer)
- ❑ Aufstellung der benötigten Ressourcen (Software, Hardware, geschätzte Kosten in € usw.)
- ❑ Verwendung nach Projektfertigstellung (z. B. Veröffentlichung im Web, App, wirtschaftliche Verwertbarkeit, usw.)

Prozessdokumentation

Die Prozessdokumentation beschreibt die realen Gegebenheiten des Projekts.
Das "Projekttagebuch" soll folgende Inhalte erfassen:

- ❑ Zeiterfassung bzw. Mitschrift der effektiven Arbeitszeit am Projekt. Das schließt jede Projektbesprechung, Schreibarbeit und Recherche sowie Arbeiten an Planung und Dokumentation mit ein.
- ❑ Aufstellung der verwendeten Mittel (Ressourcen inkl. Kosten in €)
- ❑ Soll-Ist Vergleich (Analyse der Projektplanung)
- ❑ Quellenverzeichnis (Literatur, Internetquellen, Bildquellen, usw.)
- ❑ Read-Me File mit technischen Details (insbesondere bei Versionsänderungen)
- ❑ How-To Dokument (Benutzerhandbuch)

Rechtliches

Für jedes Projekt müssen die rechtlichen Rahmenbedingungen beachtet werden. Insbesondere Datenschutz, Urheberrecht und Kennzeichnungspflicht.

- ❑ Sichtbare Kennzeichnung der Projektmitglieder (Name, Kontaktmöglichkeit, Impressum) auf dem Produkt. Zusätzlich auch in den `<meta>` Tags.
- ❑ Jedes fremde Werk mit Schöpfungstiefe (Bilder, Videos, Sounds, Text, Scripte usw.) ist durch das Urheberrecht geschützt. Verwende also nur externe Quellen die über die Creative Commons bzw. als 'frei Verwendbar' gekennzeichnet sind. Auf jeden Fall sollen externe Quellen mit Link, Name des Urhebers und einem Datum gekennzeichnet werden.
- ❑ Datenschutz: Wenn personenbezogene Daten gespeichert und|oder verarbeitet werden (z. B. in einem Formular) muss eine Datenschutzerklärung vorliegen. Diese beschreibt was mit den Daten passiert und welche Möglichkeiten es zur Einsicht und Löschung der Daten gibt. *!!! Eine Speicherung als Cookie und|oder localStorage benötigt immer die Erlaubnis des User !!!*

Ü 21.1 Projekte

Für alle Projekte gilt das Prinzip des freien Design für das HTML Layout und die CSS Anweisungen. Erweitere dein JavaScript um Feinheiten, wie z. B. Eingabeüberprüfungen oder Fehlerhandling.

 Leicht
 Mittel
 Schwer

Projekt A: Kryptografie

Schreibe ein Script für eine cleveren Kryptografie. Ein String bzw. Objekt soll verschlüsselt und wieder entschlüsselt werden.

Projekt B: Alphanumerisches Sortieren

Scripte eine Funktion zum alphabetischen Sortieren von Wörtern bzw. einer Tabelle.

Projekt C: Ampelanlage

Simuliere mit JavaScript, HTML und CSS eine Straßenkreuzung mit einer Ampelanlage. Erweitere die Kreuzung um Abbiegestreifen, Fussgängerstreifen, Fahrradstreifen usw.

Projekt D: Triangulation

Erstelle eine umfangreiche Webseite zum Thema Dreiecke und Pyramiden. Scripte dafür alles von Pythagoras bis zu den Winkelfunktionen. Erweitere es um graphische Darstellungen.

- ❑ **HTML** **Canvas bzw. SVG für die graphische Darstellung**
- ❑ **JS** **Objektorientiertes Programmieren**

Projekt E: Koordinatensystem und Kurvendiskussion

Erstelle eine Website für eine Kurvendiskussion. Funktionsgraph, Nullstellen, Wendepunkt und Extremwerte sollen in einem Koordinatensystem dargestellt werden.

Projekt F: Suchfunktion

Entwickle eine clevere Such-Funktion für HTML Seiten.

Projekt G: Navigation mit Sitemap

Scripte ein cooles Navigationslayout für ein Website-Projekt. Überrasche durch smarte Animationen und einer automatisch generierten Sitemap.

Projekt H: Freies Projekt

Hast du eine eigene Idee? Dann mach es zu deinem Projekt – alles ist erlaubt!

Ü 21.2 Projekte Wirtschaft

Für alle Projekte gilt das Prinzip des freien Design für das HTML Layout und die CSS Anweisungen. Erweitere dein JavaScript um Feinheiten, wie z. B. Eingabeüberprüfungen oder Fehlerhandling.

 Leicht
 Mittel
 Schwer

Projekt A: Bezugs- und Absatzkalkulation

Erstelle eine Webseite mit einer progressivien und retrograden Bezugs- und Absatzkalkulation.

Projekt B: Direct Costing

Scripte eine Webseite für eine einstufige Deckungsbeitragsrechnung und kombiniere diese mit einer Break-Even Analyse.

Projekt C: Anlageverzeichnis

Die Anlagegüter sollen erfasst und in Folge soll Buchwert und AfA errechnet werden. Verbinde das Anlagezeichnis auch mit einem Buchungssatzgenerator, der alle AfA Buchungen ausgibt.

Projekt D: Rechnungsgenerator

Die Rechnungsbestandteile werde über ein Eingabeforumlar ermittelt und in Folge wird eine automatische USt-Rechnung ausgegeben.

Projekt E: Einkommensteuer

Nach der Eingabe der Einkünfte und aller weiteren steuerrelevanten Posten, soll das JavaScript die Höhe der Einkommenssteuer ermittlen.

Projekt F: Lohn- und Gehaltsrechnung

Scripte ein Tool für die automatisierte Lohn- und Gehaltsrechnung.

Projekt G: Betriebliche Kennzahlen

Erstelle eine Webseite, die nach Eingabe von Unternehmensdaten aus dem Rechnungswesen und der Kostenrechnung unterschiedliche betriebliche Kennzahlen ausrechnet und übersichtlich (wenn möglich sogar graphisch) ausgibt. Beispiele:

- ❑ Gesamt- und Eigenkapitalrentabilität, ROI, Cash Flow, Anlagendeckung, Verschuldungsgrad und Kapitalbindung, Kapitalumschlagshäufigkeit uvm.

Projekt H: Freies Projekt

Hast du eine eigene Idee? Dann mach es zu deinem Projekt – alles ist erlaubt!

Ü 21.3 Projekte Entertainment

Für alle Projekte gilt das Prinzip des freien Design für das HTML Layout und die CSS Anweisungen. Erweitere dein JavaScript um Feinheiten, wie z. B. Eingabeüberprüfungen oder Fehlerhandling.

 Leicht
 Mittel
 Schwer

Projekt A: Medienplayer

Erstelle einen Medienplayer auf einer Webseite. Der Medienplayer soll die gängigen Video und Sound-Formate abspielen können.

Projekt B: Schiffe versenken

Scripte eine Website mit dem beliebten "Koordinatenspiel" Schiffe versenken. Eine KI soll der Computer-Gegner sein – überlege dir auch eine Möglichkeit um Mensch gegen Mensch zu spielen.

Projekt C: Mastermind

Das beliebte "Code-Knacker" Spiel mit den fünf Farben.

Projekt D: Würfelpoker

Würfelpoker bzw. Yatzy – die Herausforderung ist die Punkteliste im digitalen "Spielblock".

Projekt E: Wissens-Quiz

Ein spannendes Wissens-Quiz mit Fragen und Antworten.

- ❑ Die Fragen und Antworten sollen als JSON, externes JS oder XML gespeichert werden.

Projekt F: Ein Brettspiel

Checkers, Chess und Mensch-Ärgere-Dich-Nicht. Jedes 2-D-Brettspiel lässt sich im Browser realisieren. Suche dir eines aus, beachte aber die Urheberrechte des Spieles.

Projekt G: Ein Kartenspiel

Erstelle eine eigene Spielkarten API und realisiere sie in einem beliebigen Kartenspiel (z. B. Schnapsen, Black-Jack, Rummy odgl.)

Projekt H: Freies Projekt

Hast du eine eigene Idee für ein spannendes Spiel oder eine coole Entertainment-Anwendung? Dann mach es zu deinem Projekt – alles ist erlaubt!

Quellenverzeichnis

Internetquellen und online Tools

Background Image Generator
http://bg.siteorigin.com/

css4.at Lernplattform
https://www.css4.at

dict.cc Wörterbücher
https://www.dict.cc/

Duden onine
https://www.duden.de/woerterbuch

ExtractMetaData – Meta Daten auslesen
https://www.extractmetadata.com/de.html

flickr Fotocommunitiy
https://www.flickr.com/

Fundamentum – pädagogisches Forum
http://fundamentum.xobor.de/

Google (nahezu alle Dienste)
https://www.google.at

MDN web docs – Mozilla für Entwickler
https://developer.mozilla.org/de/

Mediaevent – CSS, HTML und JS mit {stil}
https://www.mediaevent.de/

openstreetmap
https://www.openstreetmap.org

Peter Kropff Tutorials
https://www.peterkropff.de/

pixabay Bilderplattform
https://pixabay.com/de/

php.net – Benutzerhandbuch
https://www.php.net

PHP-Einfach.de Tutorials
https://www.php-einfach.de

selfhtml.org – Die Energie des Verstehens
https://selfhtml.org/

stackoverflow – developers empowering
https://stackoverflow.com/

W3C CSS Validation Service
http://jigsaw.w3.org/css-validator

W3C Markup Validation Service
https://validator.w3.org

w3schools.com – Web Developer Site
https://www.w3schools.com/

Wikimedia Commons
https://commons.wikimedia.org

Wikipedia – die freie Enzyklopädie
https://de.wikipedia.org