Inhaltsverzeichnis

Vorwort

Willkommen in dem Kurs des ersten Trimesters des Talentkolleg Ruhr im Jahr 2022. Wie bereits besprochen schicke ich euch immer mal wieder ein paar Dateien und Informationen zu den im Kurs behandelten Themen zu. Ihr könnt mir natürlich jederzeit bescheid sagen, dass ihr das nicht mehr wollt. Auch ist noch zu erwähnen, dass ihr mir jederzeit eine Mail schreiben könnt solltet ihr eine Frage zu dem Kurs, den Themen im Kurs oder natürlich auch anderen Themen haben.

Wie gehe ich mit dem Dokument um?

Das Dokument dient als Dokumentation für das Projekte. Es reicht normalerweise, wenn ihr euch die relevanten Sachen raussucht und euch nur die anguckt, aber natürlich könnt ihr euch auch das komplette Dokument durchlesen.

Feedback

Wenn ihr Feedback oder einen Fehler zu dem Dokument gefunden habt bitte unter meiner Email melden und mir bescheid geben.

Github

Wir haben auch ein eigenes Repository, damit ihr den Code und die Doku immer beisammen habt: https://github.com/Talentkolleg-Herne/trimester-1-2022-mittwochskurs-dominik

Aufteilung des Projektes

Da das Projekt als ganzes recht umfangreich ist würde ich es gerne aufteilen, indem wir aus dem gesamten Projekt einzelne Phasen machen, die wir dann bei bedarf immer wieder abschließen können.

Phase 1

Phase 1 beinhaltet die komplette Verwaltung und Installation. Zusätzlich lernen wir in Phase 1 die Sprache Javascript kennen, so dass wir in Phase 2 erst mit dem eigentlichen Projekt anfangen können,

allerdings dafür alle Grundlagen gelegt sind.

Phase 2

In Phase 2 lernen wir wie wir in dem Browser etwas zeichnen können und was wir dafür brauchen

Phase 3

Hier versuchen wir den Spieler richtig abzubilden und ihn laufen lassen zu können

Phase 4

Hier bauen wir die Physik mit ein. Unser Spieler soll also mit anderen Objekten kollidieren können oder nicht mehr aus dem Level rausgehen können

Phase 5

Hier bauen wir die fahrt der Kamera ein und die generierung des Levels.

Phase 6

Ist reserviert für eigenständige Sachen, so dass wir hier noch weitere Sachen einbauen können, die wir gerne in dem Spiel haben möchten.

Erste Javascript Grundlagen

Als erstes haben wir uns mit dem Editor beschäftigt, dazu gibt es nächste Woche nochmal etwas mehr zu, bis dahin geh ich auf den nächsten Punkt ein.

Ausgaben und Variablen

Eine Ausgabe kann in Javascript auf mehrere Weisen erfolgen. Wir werden allerdings die Methode mit `console.log` nutzen. Dabei wird `console.log()` geschrieben um die Funktion aufzurufen und in den runden Klammern folgt nun die eigentliche Ausgabe. Das kann erstmal alles sein.

```
1 console.log('Ich bin eine Ausgabe');
2 console.log('Ich bin auch eine Ausgabe');
3 console.log(42);
```

Bei den obigen Ausgaben fällt auf, dass die 42 nicht mit Anführungsstrichen geschrieben wurde, dass liegt daran, dass Javascript zwischen Zahlen und Texten unterscheidet. Texte werden dabei immer mit Anführungszeichen geschrieben ‘ oder ” und Zahlen nicht. Der unterschiedt liegt in der Mathematik, denn mit Zahlen kann gerechnet werden und mit Text eben nicht. So ist es möglich auf eine Zahl alle Operationen (+,-,*,/) anzuwenden und auf einem Text ausschließlich das +, welches dann zwei Texte aneinander packt.

Eine Variable hält immer einen Wert (bei uns z.B. eine Zahl oder einen Text) um mit diesen zu arbeiten. Eine Variable kann wie folgt angelegt werden

```
1 let meineVariable = 17;
2 let nochEineVariable = 'Mein Text';
```

> Erstellung zweier Variablen mit einer Nummer und einem Text.

Danach kann mit der Variable umgegangen werden als ob es ein Text oder eine Ziffer wäre:

```
1 let meineVariable = 17;
2 let nochEineVariable = 'Mein Text';
3
4 console.log(meineVariable);
5 console.log(nochEineVariable);
```

Eine andere Möglichkeit eine Variable zu definieren und mehrere Werte beieinander zu halten ist ein Objekt. Ein Objekt kann mehrere Werte beinhalten. So kann z.B. die Konfiguration für ein Spiel in einem Objekt festgehalten werden. Ein Objekt wird erstmal wie eine Variable behandelt und umfasst geschweifte Klammern `{}`. Innerhalb dieser Klammern können nun die Felder reingeschrieben werden die in dem Objekt zur Verfügung stehen. Dabei wird ein Feldname immer mit einem Doppelpunkt von dem Wert getrennt und mit einem Komma zum nächsten geleitet.

```
1  let meinObjekt = {
2    name: 'Hans Dieter',
3    alter: 17,
4    groesse: 200,
5    istSchueler: true
6  }
7
8  // der Zugriff erfolgt über die Variable mit einem Punkt und dem
       Feldnamen:
9  console.log(meinObjekt.name);
10 console.log(meinObjekt.alter);
11 console.log(meinObjekt.groesse);
```

```
12 console.log(meinObjekt.istSchueler);
```

Rechnungen

Um mit einer Variable oder einem Wert zu rechnen kann man einfach das Operationszeichen schreiben:

```
1 let meineVariable = 17 + 3;
2 let nochEineVariable = 'Mein Text';
3
4 meineVariable = 28 / 2 + 5;
5 nochEineVariable = 'Hallo' + ' Welt';
6 console.log(meineVariable);
7 console.log(nochEineVariable);
```

> Wichtig: Nur bei der ersten Verwendung einer Variablen muss das Wort let davor geschrieben werden, danach darf es nicht mehr. Wichtig: Die Werte der Variablen werden dabei immer überschrieben und sind somit nicht mehr nutzbar. Um mit einer Variablen weiterzurechnen kann folgendes gemacht werden:

```
1 let test = 17;
2 test = test + 5;
```

> Hier wird eine Variable erzeugt mit dem Namen test der der Wert 17 zugeordnet wird. In der nächsten Zeile wird ihr ein neuer Wert zugewiesen. Dafür wird wieder die Variable benutzt um den Wert von ihr mit 5 zu addieren und der Variablen erneut zuzuweisen.

Kommentare

Kommentare werden in Javascript ignoriert (nicht ausgeführt) und dienen der Übersicht für den Programmierer. Kommentare werden mit einem `//` eingeleitet um einen Kommentar in einer Zeile zu haben oder aber über mehrere Zeilen sieht es dann wie folgt aus

```
1 // Ich bin ein Einzeiliger-Kommentar
2
3 /*
4   Ich gehe
5   über
6   mehrere
7   Zeilen
8 */
```

Blöcke

Blöcke werden in Javascript mit den geschweiften Klammern beschrieben `{}`. Diese Blöcke bilden eine Einheit und sorgen in Verbindung mit Schlüsselwörtern dazu, dass alles ausgeführt wird was in dem Block steht => z.B. bei späteren Funktionen oder Fallunterscheidungen. Diese Blöcke haben auch die Besonderheit, dass Variablen immer nur in dem Block gültig sind, in dem Sie erstellt wurden und in den Blöcken, die innerhalb dieses Blockes erstellt wurden.

Als Beispiel:

```
{
  let meineVariable = 5;

  // Hier natürlich noch gültig
  console.log(meineVariable);

  {
    // hier auch, da es ein innerer Block ist
    console.log(meineVariable);
  }
}
// Fehler: meineVariable ist nicht bekannt
console.log(meineVariable);
```

Eine Ausnahme bildet hier das Schlüsselwort `var`. Wenn ihr damit eine Variable erstellt anstatt mit `let` ist diese Variable `global` gültig.

```
{
  let meineVariable1 = 5;
  // meineVariable2 ist jetzt in jedem Block gültig und steht global
     zur Verfügung (d.h. auch in jeder Funktion usw.)
  var meineVariable2 = 10;

  // Hier natürlich noch gültig
  console.log(meineVariable1);

  console.log(meineVariable2);
  {
    // hier auch, da es ein innerer Block ist
    console.log(meineVariable1);
  console.log(meineVariable2);
  }
}
// Fehler: meineVariable ist nicht bekannt
console.log(meineVariable1);
console.log(meineVariable2);
```

Fallunterscheidung

Eine Fallunterscheidung dient dazu zwischen zwei Zuständen zu unterscheiden und verschiedenen Code für die Zustände auszuführen. Als Beispiel möchte man z.B. zusätzlichen Code ausführen wenn jemand über 18 ist (z.B. Zugang zu einer Webseite). Auch in der realen Welt gibt es natürlich auch Fallunterscheidungen die wir intuitiv fällen. So gibt es z.B. die Möglichkeit sich wenn es kalt ist eine Jacke anzuziehen.

In der Programmierung ist es allerdings nicht ganz so einfach mit den Bedingungen. Hier muss jede Bedingung nach WAHR oder FALSCH beurteilt werden können. Diese Werte WAHR oder FALSCH werden auch als boolsche Werte bezeichnet. Für diese Aussagen (boolsche Bedingung => Eine Bedingung die nach einer boolschen Wert auswertbar ist) stehen uns folgende Operatoren zur Verfügung:

Operator	Wort	Beispiel
>	größer	5 > 10 5 ist kleiner als 10 (WAHR)
<	kleiner	10 < 5 10 ist kleiner als 5 (FALSCH)
>=	größer gleich	16 >= 17 16 ist größer oder gleich 17 (FALSCH)
<=	kleiner gleich	16 <= 16 16 ist kleiner oder gleich 16 (WAHR)
!=	nicht gleich	0 != 1 0 ist nicht gleich 1 (WAHR)
==	gleich	1 == 1 1 ist gleich 1 (WAHR)
===	typengleich	1 === „1“ Der Typ und der Wert von 1 ist gleich dem Typ und der Wert von 1 (FALSCH => da links eine Zahl und rechts ein Text)

Mithilfe dieser Operatoren lassen sich nun die Bedingungen schreiben. Um so eine Bedingung zu schreiben, müssen wir vorher das Schlüsselwort für eine Fallunterscheidung voran schreiben `if` um in Klammern dahinter die Bedingung zu schreiben. Sollte nun die Bedingung wahr sein, wird alle was in den geschweiften Klammern dahinter kommt ausgeführt. Wenn man möchte kann man hinter der `if` noch ein `else` schreiben um einen block nur Auszuführen, wenn die Bedingung in der `if` Anwei-

sung den Wert FALSCH liefert.

Als Beispiele:

```
// definiere eine Variable die mein Alter speichert
let meinAlter = 23;

if (meinAlter >= 18) {
  console.log('Du bist volljährig');
}
```

> Programm welches das Alter speichert und „Du bist volljährig“ auf der Konsole schreibt wenn man älter oder gleich 18 Jahre alt ist.

```
let meinAlter = 23;

if (meinAlter >= 18) {
  console.log('Du bist volljährig');
} else {
  console.log('Du bist noch minderjährig');
}
```

> Programm welches das Alter speichert und „Du bist volljährig“ auf der Konsole schreibt wenn man älter oder gleich 18 Jahre alt ist. Sollte man jedoch jünger als 18 Jahre alt sein wird „Du bist noch minderjährig“ auf die Konsole geschrieben

```
let meinAlter = 23;

if (meinAlter >= 18) {
  console.log('Du bist volljährig');
} else if (meinAlter >= 16) {
  console.log('noch nicht volljährig, aber nah dran');
} else {
  console.log('Du bist noch minderjährig');
}
```

> Dieses Programm gibt ein „Du bist volljährig“ wenn der Benutzer älter oder gleich 18 Jahre alt ist. Zudem wird noch ein anderer Text ausgebenen wenn das alter größer oder gleich 16 ist. Als alternative wenn nichts anderes zutrifft wird „Du bist noch minderjährig“ ausgegeben. In dem Beispiel wird das **`if else`**-Konstrukt verwendet, welches nur die erste Bedingung ausführt. Das ist hier besonders Sinnvoll, da ansonsten immer beide Texte ausgebeben werden würden (da jemand der über 18 Jahre ist auch immer über 16 Jahre alt ist) und wir mit dem **`if else`**-Konstrukt nun nach der ersten Bedingung den rest des Konstrukts überspringen können, sollte jemand über oder gleich 18 Jahre alt sein.

Schleifen

Schleifen dienen in der Programmierung dazu Aufgaben zu bewältigen die öfters getan werden müssen. So ist bei einem Spiel immer so eine Schleife dabei die z.B. unseren Bildschirm immer neu lädt um so z.B. Dinge bewegen zu können. So kann man mit dem Schlüsselwort `while` so eine Schleife einleiten und dann in runden Klammern dahinter sofort die Bedingung definieren.

```
1 while(x < 1000) {
2   // Wird solange ausgeführt wie die Bedingung (in den Klammern) wahr
        ist.
3 }
```

Also um ein kleines Beispiel zu machen, wollen wir nun die Zahlen von 1 bis 1000 ausgeben. Mit der alten weise müssten wir leider 1000x `console.log()` schreiben. Wir wollen das aber natürlich etwas vereinfachen und nehmen daher eine Schleife.

```
1  let x = 1; // wir wollen bei 1 anfangen
2
3  while(x <= 1000) { // solange x kleiner oder gleich ist als 1000 gehen
       wir in die Schleife
4    // wird solange ausgeführt wie die Variable x kleiner oder gleich
        1000 ist.
5    // Ausgabe der Variablen x
6    console.log(x);
7
8    // erhöhe x um eins um die Schleife auch irgendwann beenden zu können
        .
9    x = x + 1;
10 }
```

Funktionen

Mittels Funktionen können Codeteile ausgelagert werden und zentral an einer Stelle zur Verfügung gestellt werden. Das ist von Vorteil, wenn man Code in einem Programm oft verwendet und daher den Code nur an einer Stelle updaten möchte. Desweiteren ist es mit Funktionen einfacher funktionalitäten zu teilen (wie es z.B. Phaser3 macht bzw. alle anderen Frameworks). Um in Javascript eine Funktion zu erstellen, wird das Schlüsselwort `function` vor dem Namen der Funktion geschrieben. Danach kommen Klammern `()`. In diesen Klammern können nun Parameter definiert werden um bestimmte Variablen auch innerhalb der Funktion zur Verfügung zu haben. Die Funktion kann nach der definition aufgerufen werden, indem man den Namen mit den beiden Klammern schreibt.

```
1 // definition der Funktion
2 function meineFunktion() {
3   console.log('Ich bin eine Funktion');
```

```
4  }
5
6  // Aufruf der Funktion.
7  meineFunktion();
8
9  // definition der Funktion mit einem Parameter
10 function meineFunktion2(meinParameter) {
11   console.log(meinParameter);
12 }
13
14 // Aufruf der Funktion mit Parameter
15 // Hier wird in den Klammern der Wert eingetragen und kann in der
       Funktion benutzt werden
16 meineFunktion('hello');
```

Summe berechnen zwischen X und Y

Das ganze kann man einmal iterativ machen und rekursiv.

Primzahl berechnen durch Prüfung

Fibonacci Zahlen

Rekursiv und Iterativ 0 1 1 2 3 5 8 13 21 34

Fakultät

Rekursiv und Iterativ

Quersumme

Quersumme berechnen von einer beliebigen Zahl

Harshad-Zahl

Durch die Quersumme teilbar

Sieb des Erastothenes

1 2 3 5 7
11 13 17 19

Canvas

Ein Canvas ist im Javascript-Context quasi ein Zeichenbrett. Auf diesem Zeichenbrett können wir nun unsere verschiedenen Figuren zeichnen und animieren.

Ein Canvas anlegen

Um auf ein Canvas zu zeichnen, brauchen wir zuerst natürlich ein Canvas. Dafür öffnen wir unsere `.html` Datei und schreiben folgende Zeile in dem body:

```
<canvas id="game"></canvas>
```

> Ein neues leeres Canvas mit der id: game

Diese Zeile erstellt ein neues Canvas mit der Identifikation (kurz: id) game. Die ID vergeben wir damit wir Sie in Javascript leichter referenzieren können.

Das canvas stylen

Um das Canvas zu stylen um z.B. die größe und breite zu ändern, können wir das ganze canvas mit CSS weiter anpassen. Dafür schreiben wir in den Head-Bereich einen neuen Tag mit dem Namen style und fügen dort den Canvas ein.

```
<style>
  body {
    margin: 0px;
    padding: 0px;
  }

  canvas {
    background: grey;
    width: 1600px;
    height: 800px;
  }
</style>
```

Damit wird der Abstand vom body (dem ganzen Dokument) auf 0 gesetzt und zusätzlich der canvas auf einen grauen Hintergrund und die Breite auf 1600px und die höhe auf 800px.

Ganzer HTML-Code

Der ganze Code in der `index.html` sieht damit wie folgt aus:

```
<!DOCTYPE html>
<html>

<head>
  <title>Mein erstes Spiel</title>
  <style>
    body {
      margin: 0px;
      padding: 0px;
    }

    canvas {
      background: grey;
      width: 1600px;
      height: 800px;
    }
  </style>
</head>

<body>
  <canvas id="game"></canvas>
  <script src="game.js"></script>
</body>

</html>
```

Auf dem Canvas zeichnen

Um auf dem Canvas zu zeichnen brauchen wir zuerst eine Referenz auf den Canvas. Dafür können wir eine neue Variable anlegen und ihr die Referenz auf das Canvas mit der Id `game` geben. Dafür gibt es im globalen Objekt `document` die Funktion `getElementById`.

Danach müssen wir den Context des Canvas definieren. Hier gibt es 2d und 3d. Der Code dafür sieht also nun so aus:

```
var canvas = document.getElementById('game');
var ctx = canvas.getContext('2d');
```

Jetzt definieren wir noch um damit später leichter zu arbeiten die höhe und breite des canvas in zusätzlichen Variablen:

```
var WINDOWS_WIDTH = canvas.width;
var WIDOWS_HEIGHT = canvas.height;
```

Als nächstes können wir unser erstes Objekt auf dem canvas zeichnen.

Dafür wird zuerst die Methode vom unserem Canvas `beginPath` benutzt. Als letztes muss `closePath` geschrieben. Alles was nun dazwischen liegt wird zu dem Objekt gezählt welches gezeichnet werden soll. Das ist wichtig, damit der Browser weiß welche Farbe, Form und Style das jeweilige Objekt haben soll ohne durcheinander zu geraten.

```
ctx.beginPath();
// rect ist eine Funktion, die folgende Werte bekommt.
// 1. die x Position des Elements
// 2. die y Position des Elements
// 3. die breite des Elements
// 4. die höhe des Elements
ctx.rect(10,10 50, 50);
// fillStyle gibt die Farbe wieder, die das Element annehmen soll
ctx.fillStyle = '#FFFFFF';
ctx.fill();
ctx.closePath();
```

Das canvas leeren

Das Canvas kann mittels folgender Funktion wieder geleert werden. Das ist dafür wichtig um später die Bewegung darstellen zu können. Damit wird dann quasi die vorherige Position wieder gelöscht.

```
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

Hier wird die Position eingegrenzt worin alles geleert wird. Dafür geben wir die x und y Koordinate des Anfangs an und die x und y Koordinate des Endes an. Das können wir ganz einfach machen, indem wir den kompletten Bildschirm nehmen, welches dann von 0,0 bis zur maximalen Breite und Länge unseres Canvas geht.

Exkurs - HTML

HTML (Hypertext Markup Language) wird benutzt um eine Webseite zu beschreiben. Das heißt mittels HTML wird der Aufbau einer Webseite beschrieben und nicht deren eigentlichen Design. Das Design wird anschließend in CSS gemacht. In der HTML gibt es sogenannte Tags, die für verschiedene Struk-

turen stehen und von dem Englischen Begriff abgeleitet werden. Ein Tag wird mit den folgenden Klammern eingeleitet <> und meistens wie folt beendet </>. Der Tagname steht zwischen den Klammern bzw. zwischen dem Schrägstrich und der schließenden Klammer. Hier sind ein paar Beispiele:

- <html></html>
-
- <head></head>
- <body></body>

Man kann diese Tags auch als Container oder Behälter betrachten, da diese auch Daten beinhalten können. So wird ein Paragraphen-Tag p am Anfang eines Paragraphen geschrieben um diesen auch mit dem schließenden Tag zu beenden.

```
<p>Ich bin ein neuer Paragraph</p>
<p>Paragraphen können auch
mehrzeilig sein</p>
```

Aufbau eines Tags

```
                            +--- HTML-Attribut
                            |   +--- Wert in Hochkommas
                            |   |
<p>ein Text mit einem <a href="…">Link</a></p>
 |                    |                |  |
 |                    |                |  +--- schließendes
 |                    |                |       HTML-Tag mit
 |                    |                |       Schrägstrich
 |                    |                |
 |                    |                +--- schließendes Tag
 |                    +--- öffnendes Tag
 +--- öffnendes HTML-Tag
```

Webseiten-Struktur

Als ein Beispiel wird hier ein sehr simple Webseite herhalten.

```
<html>
  <head>
    <title>Meine Webseite</title>
  </head>
  <body>
    Lorem ipsum
  </body>
</html>
```

Head und Body

Eine HTML-Seite besteht immer aus einem <html>-Bereich, indem der eigentliche Seitenaufbau beschrieben wird. Darin wird nun ein <head> und einen <body>-Bereich definiert. Innerhalb des <head> kommen nun Metadaten wie z.B. der Author, das Zeichenformat und dort kann man den Titel der Seite beschreiben. In den <body> Bereich kommt nun alles, was für den Benutzer sichtbar wird rein.

Ein paar weitere Tags

Um ein paar weitere Tags und einen ersten einfachen Aufbau zu bekommen schreiben wir hier folgende Zeilen, die ich anschließend erläutern werde. Dabei schreibe ich hier nur den <body>:

```
<body>
   <h1>Lorem ipsum </h1>
   <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. </p>
   <p>Mauris ante neque, vehicula quis, convallis sit amet,
      vestibulum ornare, arcu. </p>
   <p>Donec at lorem et elit congue dictum. Cras sapien ligula,
      rutrum quis, posuere id, faucibus id, risus. Maecenas sed est
      volutpat arcu adipiscing tempus. Sed dictum mauris euismod
      mauris.</p>
</body>
```

- <h1>
  - Eine h1 ist eine Überschrift1 (engl. Headline1). Die Überschriften gehen von 1 (die größte) bis 6 (die kleinste). Nach einer Überschrift folgt auch immer ein Zeilenumbruch.
- <p>
  - Ein p ist ein Paragraph. Ein Paragraph enthält meistens Text und anschließend ein Zeilenumbruch.
- <ul, ol>
  - erzeugen ungeordnete bzw. geordnete (nummerierte) Listen. Ungeordnete Listen werden mit einem führenden Listenpunkt dargestellt, geordnete Listen mit einem Index. In beiden Listen wird jedes Listenelement in li-Tags eingeschlossen.

```
<ol>
  <li>HTML strukturiert den Inhalt </li>
</ol>
```

-

- In einem DIV können andere Elemente hineingefügt werden. Das div fungiert nur als eine Art Container.

Kommentare

Kommentare werden in HTML wie folgt beschrieben:

```
<!-- Dies ist ein Kommentar -->
```

> Kommentare dienen dazu den Quelltext besser zu Strukturen und für den Leser annehmlicher zu gestalten.

Quellcode-Strukturieren

Es ist üblich den Quellcode mit bestimmten Einrückungen besser lesbar zu machen. Dazu gehört z.B. dass verschachtelte Tags meistens in einer neuen Zeile (außer z.B. Links oder andere kurze Tags) und mit einem vorangegangenen Tab-Zeichen eingerückt werden.

Beispiel hier mit einer Tabelle:

```
<table>
   <tr>
      <td>Zeile 1 Spalte 1</td>
      <td>Zeile 1 Spalte 2</td>
   </tr>
   <tr>
      <td>Zeile 2 Spalte 1</td>
      <td>Zeile 2 Spalte 2</td>
   </tr>
</table>
```

> Wie man sieht sidn die `<tr>`und die `<td>` jeweils in einer neuen Zeile und weiter eingerückt als z.B. das `<table>`

Exkurs CSS

Mittels CSS (Cascade Style Sheets) wird die HTMl-Seite ansehnlicher gemacht und letzlich wird hier das Design festgelegt. Dafür kann man bestimmte HTML-Tags oder aber alle Tags ansprechen und mit den CSS-Eigenschaften ein bestimmtes styling zuweisen.

Selektoren

Mithilfe dieser Selektoren werden die Tags ausgewählt, wo man gerne das aussehen verändern möchte. Ein Selektor kann einfach der gewählte Tag sein:

```
1 div {}
```

> dieser Selektor wählt alle vorhandenen divs aus und würde die Regeln in den geschweiften Klammern {} auf diese divs anwenden.

Zwei weitere Selektoren sind der Klassen-Selector und der ID-Selector. Diese Selektoren müssen allerdings vorher im HTML versehen werden:

```
1 <body>
2   <div id="inhalt">
3     <p class="meinAbsatz">Mein erster Absatz</p>
4   </div>
5 </body>
```

> Unser HTML mit einer ID und einer class (Klasse)

Um nun die ID anzusprechen benutzen wir den Namen der id mit einer Raute # voran und für die Klasse ein `.`.

```
1 #inhalt {
2
3 }
4
5 .meinAbsatz {
6
7 }
```

Erste Eigenschäften ändern

Um nun mal auf die verschiedenen Eigenschaften einzugehen würde ich gerne dieses Beispiel zeigen:

```
1 p {
2   color: red;
3   width: 500px;
4   border: 1px solid black;
5 }
```

Diese Selektor addressiert erstmal alle Paragraphen in unserem HTML. Anschließend wird die Textfarbe mittels `color` auf rot gesetzt, die breite des PAragraphen wird auf 500px begrenzt und zum Schluss

erhält er noch ein Rahmen mit 1px dicke in Schwarz mittels `border: 1px solid black`.

Einheiten und Werte

Werte

In CSS ist erstmal jede numerische Zahl gestattet (jede Rationale-Zahl ist in CSS ein geeigneter Wert). Dabei wird nur in Ganzzahl und Fließkommazahl unterschieden, wobei eine Fließkommazahl mit einem Punkt getrennt wird. Eine 0 wird vor einem Komma ist immer optional. Hier sind ein paar Beispiele für richtige Werte:

- 1
- -5
- .5
- 192
- -50
- 49.5
- 10.09

Maßeinheiten

Ein Maß (Maßeinheit) wird definiert, indem direkt nach dem Wert ohne ein Leerzeichen die Maßeinheit geschrieben wird. Hierbei gibt es viele unterschiedliche Arten von Maßeinheiten. Wir wollen einmal die zwei häufigsten hier erwähnen:

Feste Maßeinheiten Hier sind Maßeinheiten, die sich nicht an die Bildschirmgröße anpassen.

Einheit	CSS	Beschreibung	Größe in px
Zoll	in	Ein Zoll ist 2,54cm lang	96 Pixel
Pica	pc	1/6 eines Zolls	16 Pixel
Punkt	pt	1/72 eines Zolls	1,33 Pixel
Zentimeter	cm	Hunderste Teil eines Meters	37,8 Pixel
Millimeter	mm	Tausendste Teil eines Meters	3,78 Pixel

Einheit	CSS	Beschreibung	Größe in px
Pixel	px	Sichtbare Bereich eines Pixel bei 96 DPI und einer Armlänge Abstand	0,75 Punkt

Diese Maßeinheiten sind besonders für den Druck auf Papier geeignet, da dort das Format eines Papieres (z.B. A4) immer gleich ist und man nicht das Problem hat, dass es sich um ein kleines Mobiles Display handelt.

Relationale Maßeinheiten Als relativ werden Längenmaße bezeichnet, deren Umrechnungsfaktor zu einem absoluten Maß variabel ist. So kann ein bestimmtes Maß je nach Definition (auch innerhalb eines Dokuments) verschiedene Größen annehmen.

Einheit	CSS	Beschreibung
em	em	Bezieht sich auf die Schriftgröße
Wurzel-em	rem	Bezieht sich auf das Wurzelelement der Schrift
Prozent	%	Ist die Prozentangabe zu dem Elternelement
Viewport-Höhe	vh	Ist die Höhe des Anzeigegerätes
Viewport-Breite	vw	Ist die Breite des Anzeigegerätes

Abstände

Bei den Abständen gibt es zwei Möglichkeiten innerhalb von CSS. ### Abstand nach außen Mit Abstand nach außen ist der Abstand zwischen zwei Elementen (wie z.B. zweier Buttons) gemeint. Das heißt damit kann man den Abstand, der zwischen einem Element ist ändern. Die CSS-Eigenschaft hierfür ist: `margin` Das margin bietet die Möglichkeit den Abstand mittels margin-{Richtung} beliebig für jede Position festzusetzen. So kann z.B.

```
margin-top: 5px;
margin-left: 5px;
margin-right: 5px;
margin-bottom: 5px;
```

> Das Element erhält zu jedem anderen Element 5px Abstand

Einzeln vergeben werden oder aber zusammenfassend geschrieben werden:

```
margin: 5px 5px 5px 5px;
```

> Das Element erhält zu jedem anderen Element 5px Abstand. Die Aufteilung ist hier wie folgt: top, rigth, bottom, left Hier können auch nur drei oder zwei Werte angegeben werden. Bei dreien wäre links und rechts das gleiche und bei zweien würde zuerst top|bottom und danach left|right definiert werden.

Abstand nach innen

Mit Abstand nach innen ist der Abstand vom Inhalt zum Rand des Elementes gemeint. So kann man z.B. einen Button deutlich größer erscheinen lassen, wenn man das Padding von den Seiten erhöht (der Abstand von der Schrift des Buttons zum Rand hin wird vergrößert). Die CSS-Eigenschaft hierfür ist: `padding` Das padding bietet die Möglichkeit den Abstand mittels padding-{Richtung} beliebig für jede Position festzusetzen. So kann z.B.

```
padding-top: 5px;
padding-left: 5px;
padding-right: 5px;
padding-bottom: 5px;
```

> Das Element erhält in jede Richtung 5px mehr zum Rand

Einzeln vergeben werden oder aber zusammenfassend geschrieben werden:

```
margin: 5px 5px 5px 5px;
```

> Das Element erhält in jede Richtung 5px mehr zum Rand. Die Aufteilung ist hier wie folgt: top, rigth, bottom, left Hier können auch nur drei oder zwei Werte angegeben werden. Bei dreien wäre links und rechts das gleiche und bei zweien würde zuerst top|bottom und danach left|right definiert werden.

Wie mache ich jetzt weiter?

Da es hier nahezu unmöglich ist alle CSS-Regeln und die verschiedenen Selektoren zu erläutern würde ich gerne auf die CSS-Referenz von Mozilla hinweisen. Dort sind quasi alle Selektoren und Eigenschaften aufgeführt. Um wirklich weiter zu machen und besser zu werden, eignet sich allerdings ein kleines Projekt wie z.B. den eignenen Lebenslauf mal in eine HTML-Seite zu beschreiben.

https://developer.mozilla.org/de/docs/Web/CSS/CSS_Referenz

Exkurs - HTTP und HTTPS

HTTP-Verbindungsaufbau

Vorgang	Beschreibung
Verbindungsaufbau	Der Client schickt eine Anfrage zum Server auf Port 80
Dokument Anfordern	Der Client fordert über das Kommando GET das gewünschte Dokument vom Server an
Übertragung	Der Server fängt an das Dokument zum Client zu übertragen
Verbindungsabbau	Der Verbindungsabbau erfolgt nach der Übertragung vom Server.

HTTPS-Verbindungsaufbau

Vorgang	Beschreibung
Client Anfrage	Der Client schickt eine Anfrage an den Server mit Verschlüsselungsoptionen.
Server Anfrage	Der Server schickt eine Antwort mit seinem Öffentlichen-Schlüssel.
Server überprüfung	Der Client überprüft nun den Schlüssel des Servers. Ist dieser ungültig wird die Verbindung abgebrochen ist sie gültig wird fortgefahren. Dafür generiert der Client einen Sitzungsschlüssel nur für diese Verbindung und verschlüsselt ihn mit dem Öffentlichen-Schlüssel des Servers
Client Annahme	Der Server kann die Anfrage mit dem Sitzungsschlüssel nun mit seinem privaten-Schlüssel entschlüsseln.
Verbindung aufgebaut	Die Verbindung ist nun aufgebaut und es können nun alle HTTP-Anfragen verschlüsselt übertragen werden, bis die Übertragung abgebrochen wird.

Exkurs - Ports

In einem System gibt es mehrere Dienste. Möchte man diese Dienste auch nach außen hin benutzbar machen muss man ihn eine eindeutige Nummer zuweisen. Das hat den Sinn, damit man mehrere Dienste adressieren kann und nicht jedes System nur ein Dienst unterstützen kann.

Bei den Portnummern gibt es unterschiedliche Bereiche.

Name	Bereich	Erläuterung
System Ports / Well-Known-Ports	0 - 1023	Sind standardisierte Ports für Dienste. Hier findet man zum Beispiel Ping (7) HTTP (80) und HTTPS (443)
Registered Ports	1024 - 49151	Sind für registrierte Dienste die nicht in den Standard aufgenommen werden. Hier finden sich diverse Programme von Unternehmen wie Adobe, Microsoft, Google aber auch kleinere Unternehmen.
Dynamic Ports	49152 - 65535	Dynamische Ports werden vom Betriebssystem an einen Client adressiert. So kann zum Beispiel der Webbrowser für eine Anfrage aus dem Bereich einen Port verwenden.