

Inhaltsverzeichnis

Vorwort	3
Wie gehe ich mit dem Dokument um?	3
Feedback	3
Gruppen	3
Spieleentwickeln mit Phaser3	3
Eine Webapp programmieren mit NodeJS	4
Phaser3 - Projekt	4
Die Idee / Das Projekt	4
Phase 2	4
Phase 3	4
Was ist Phaser?	4
Erste Javascript Grundlagen	5
Ausgaben und Variablen	5
Rechnungen	6
Blöcke	7
Fallunterscheidung	8
Funktionen	10
Die Phaser Funktionen	11
Unser erstes Phaser Beispiel	11
NodeJS - Projekt	15
Die Idee / Das Projekt	15
Phase 1	16
NodeJS	16
Nodejs - Installation	16
Installation Testen	16
NodeJS verwenden	18
Unsere erste NodeJS Anwendung	18
Auf weitere Pfade reagieren	19
Unser HTML mit NodeJS ausgeben	19
Das HTML anpassen	21
Alles zusammensetzen	21

Exkurs - HTML	22
Aufbau eines Tags	23
Webseiten-Struktur	23
Head und Body	23
Ein paar weitere Tags	23
Kommentare	24
Quellcode-Strukturieren	24
Exkurs CSS	25
Selektoren	25
Erste Eigenschaften ändern	26
Einheiten und Werte	26
Werte	26
Maßeinheiten	27
Abstände	28
Abstand nach innen	28
Wie mache ich jetzt weiter?	29
Exkurs - HTTP und HTTPS	29
HTTP-Verbindungsaufbau	29
HTTPS-Verbindungsaufbau	30
Exkurs - Ports	30
Exkurs - JSON	31
Aufbau	31
Objekte als Wert	31
Arrays	32
Objekt-Arrays	32

Vorwort

Willkommen in dem Kurs des ersten Trimesters des Talentkolleg Ruhr im Jahr 2021. 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 oder anderen Themen haben.

Wie gehe ich mit dem Dokument um?

Das Dokument dient als Dokumentation für beide 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 angucken.

Feedback

Wenn ihr Feedback oder einen Fehler zu dem Dokument gefunden habt bitte unter meiner Email (dominikrobert@gmx.de) melden und mir bescheid geben.

Gruppen

Wir haben entschieden den Kurs zu teilen. Dafür haben wir zwei Gruppen gebildet die sich jeweils in einem anderen Thema bewegen und sich damit beschäftigen.

Spieleentwickeln mit Phaser3

Die erste Gruppe möchte gerne ein Spiel in Phaser3 entwickeln. Die Gruppe besteht aus:

- Sena
- Safa
- Maren
- Denis

Eine Webapp programmieren mit NodeJS

Die zweite Gruppe möchte gerne eine Webapp mithilfe von NodeJS bauen. Die Gruppe besteht aus:

- Daria
- Mert
- Yunes

Phaser3 - Projekt

Hier wird es alle Informationen zu dem Phaser3 - Projekt geben.

Die Idee / Das Projekt

Die Idee hinter dem Projekt ist es ein Jump n' Run zu entwickeln. Dabei wird das Projekt aufgrund des Umfangs und der begrenzten Zeit eingeteilt in mehreren Phasen. Die Phasen bilden die Ziele ab, die wir erreichen wollen um die Zeit besser einteilen zu können und besser die Stunden zu strukturieren für den Kurs. Wenn es von der Zeit her passt, können die Phasen und die Inhalten natürlich angepasst werden. ### Phase 1 In Phase 1 soll eine Spielfigur zu einer Tür (o.Ä. Objekt) mit der Tastatur bewegt werden können um das Level abzuschließen. In dem Level befinden sich verschiedene Plattformen auf denen man laufen kann.

Phase 2

In Phase 2 soll mehr Physik in das Spiel kommen und das Springen auf andere Plattformen ermöglicht werden. Außerdem soll mit der Umgebung interagiert werden (z.B. Münzen aufsammeln).

Phase 3

In Phase 3 kommen auch mindestens ein anderer Gegner in das Spiel um das Level etwas anspruchsvoller zu gestalten.

Was ist Phaser?

Phaser ist ein Spieleentwicklungs-Framework für Javascript mit dem man relativ leicht eigene Spiele für den Browser entwickeln kann aufgrund der leichten vordefinierten Funktionen.

Ein Framework ist eine Sammlung von Funktionen und Techniken, für jemand anderen um mit einem bestimmten Thema die Arbeit zu erleichtern (hier z.B. Spieleentwickeln).

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 Unterschied 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(nocheinVariable);
```

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:

```
1 {
2   let meineVariable = 5;
3
4   // Hier natürlich noch gültig
5   console.log(meineVariable);
6
7   {
8     // hier auch, da es ein innerer Block ist
9     console.log(meineVariable);
10  }
11 }
12 // Fehler: meineVariable ist nicht bekannt
13 console.log(meineVariable);
```

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

```
1 {
2   let meineVariable1 = 5;
```

```
3 // meineVariable2 ist jetzt in jedem Block gültig und steht global
  zur Verfügung (d.h. auch in jeder Funktion usw.)
4 var meineVariable2 = 10;
5
6 // Hier natürlich noch gültig
7 console.log(meineVariable1);
8
9 console.log(meineVariable2);
10 {
11     // hier auch, da es ein innerer Block ist
12     console.log(meineVariable1);
13     console.log(meineVariable2);
14 }
15 }
16 // Fehler: meineVariable ist nicht bekannt
17 console.log(meineVariable1);
18 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)

Operator	Wort	Beispiel
!=	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** Anweisung den Wert FALSCH liefert.

Als Beispiele:

```
1 // definiere eine Variable die mein Alter speichert
2 let meinAlter = 23;
3
4 if (meinAlter >= 18) {
5   console.log('Du bist volljährig');
6 }
```

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

```
1 let meinAlter = 23;
2
3 if (meinAlter >= 18) {
4   console.log('Du bist volljährig');
5 } else {
6   console.log('Du bist noch minderjährig');
7 }
```

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

```
1 let meinAlter = 23;
2
3 if (meinAlter >= 18) {
4   console.log('Du bist volljährig');
5 } else if (meinAlter >= 16) {
6   console.log('noch nicht volljährig, aber nah dran');
7 } else {
8   console.log('Du bist noch minderjährig');
9 }
```

Dieses Programm gibt ein „Du bist volljährig“ wenn der Benutzer älter oder gleich 18 Jahre alt ist. Zudem wird noch ein anderer Text ausgegeben 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 ausgegeben 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.

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');
```

Die Phaser Funktionen

Phaser stellt mehrere Funktionen zur Verfügung, die verschiedene Aufgaben haben:

- preload
 - Die Preload-Funktion wird ausgeführt beim starten von Phaser. Diese Funktion kann benutzt werden um z.B. Bilder und Sounds zu laden.
- create
 - Ist die 2. Funktion die ausgeführt wird. Hier werden normalerweise die Bilder platziert und an die richtige Position mit der richtigen größe geschoben
- render
 - Die render-Funktion wird als letztes von unseren hier genannten Funktionen aufgerufen und wird immer so oft es geht ausgeführt.

Unser erstes Phaser Beispiel

Unser erstes Beispiel erläutere ich mal mit den Zeilenangaben und dem Aufbau des Codes in kleineren Schritten.

Zuerst, da es sich dabei um ein Spiel handelt welches im Browser dargestellt wird und der Browser üblicherweise mit HTML Dateien arbeitet definieren wir das Grundgerüst für eine HTML Datei:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4   </head>
5   <body>
6   </body>
7 </html>
```

In Zeile 1 definieren wir also, dass es sich dabei um eine HTML-Datei handelt. Nun wird alles was zu der HTML-Datei gehört in den `<html>` Tag geschrieben (für weitere Erläuterungen => siehe Exkurs HTML). Nun können wir mittels eines CDN das Phaser-Framework mit den Funktionen die wir brauchen hinzufügen.

Ein CDN oder auch Content Delivery Network stellt Inhalte über das Netzwerk (Internet) zur Verfügung um mal schnell etwas zu testen oder Inhalte zentral verwalten zu können. In unserem Fall können wir es verwenden um mit Phaser schnell anfangen zu können ohne vorher die Phaser-Dateien downloaden zu müssen.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <script src="https://cdn.jsdelivr.net/npm/phaser@3.15.1/dist/phaser-arcade-physics.min.js"></script>
5   </head>
6   <body>
7   </body>
8 </html>
```

Nun können wir in den Body-Tag unser Javascript für Phaser schreiben. Da HTML eigentlich kein Javascript unterstützt, müssen wir den Code dafür innerhalb eines Script-Tags schreiben:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <script src="https://cdn.jsdelivr.net/npm/phaser@3.15.1/dist/phaser-arcade-physics.min.js"></script>
5   </head>
6   <body>
7     <script>
8     </script>
9   </body>
10 </html>
```

Ab jetzt werde ich auch ausschließlich den Code aus dem Script-Tag schreiben und nur zum Schluss noch einmal den vollständigen Code zeigen.

Zuerst können wir uns ein Konfigurationsobjekt erstellen, in dem festgelegt wird die Größe unser Spiel z.B. ist (die Angabe erfolgt in der Einheit Pixel). Hier könnten wir auch direkt eine Hintergrundfarbe für das Spiel festlegen.

```
1 var config = {
2   width: 800,
3   height: 600,
4   backgroundColor: '#4488aa',
5 }
```

Als nächstes muss in dieses Objekt auch die Funktionen die wir von Phaser benutzen wollen mit rein. Dafür definieren wir das Attribut `scene` innerhalb unserer Konfiguration und schreiben die Funktionen rein die wir benutzen möchten mit der Funktion wie Sie bei uns heißt.

```
1 var config = {
2   width: 800,
3   height: 600,
4   backgroundColor: '#4488aa',
5   scene: {
6     preload: preload,
7     create: create
8   }
9 }
```

Wir können jetzt auch sofort diese Funktionen definieren.

```
1 var config = {
2   width: 800,
3   height: 600,
4   backgroundColor: '#4488aa',
5   scene: {
6     preload: preload,
7     create: create
8   }
9 };
10
11 function preload() {
12 }
13
14 function create () {
15 }
```

Hier haben wir die Funktions-Namen nicht geändert. Dazu gibt es meistens auch keinen Grund und der Standardname kann einfach beibehalten werden.

Als nächstes müssen wir unser Spiel erstellen. Das geht ganz einfach mittels

```
1 var game = new Phaser.Game(config);
```

Damit legen wir eine neue Variable an, die alle Informationen und Daten für das Spiel enthält. Dazu benutzen wir eine Funktion die von Phaser bereitgestellt wird `Game` und übergeben dieser Funktion unser Konfigurationsobjekt als Parameter.

```
1 var config = {
2   width: 800,
3   height: 600,
4   backgroundColor: '#4488aa',
5   scene: {
6     preload: preload,
7     create: create
8   }
9 }
10
```

```
11 var game = new Phaser.Game(config);
12
13 function preload() {
14 }
15
16 function create () {
17 }
```

Unser Spiel würd nun schon funktionieren und die Funktionen ausführen. Das können wir auch testen, indem wir in diese Funktionen ein `console.log` schreiben.

Wir fangen nun an, ein Bild hinzuzufügen.

```
1 var config = {
2   width: 800,
3   height: 600,
4   backgroundColor: '#4488aa',
5   scene: {
6     preload: preload,
7     create: create
8   }
9 }
10
11 var game = new Phaser.Game(config);
12
13 function preload() {
14   this.load.image('sky', 'assets/bild1.png');
15 }
16
17 function create () {
18   this.add.image(500, 200, 'sky');
19 }
```

Mittels **this** kann auf Werte und Funktionen eines Objekts zugegriffen werden. Dieses Objekt stellt die Funktion **image** zur Verfügung an die wir mittels eines weiteren Objekts **load** kommen. Damit können wir ein Bild laden und ihn einen Namen geben (erster Parameter ist der Name und der zweite ist der Pfad zum Bild).

In der Create Funktion wird das Bild nun dargestellt mittels dem Objekt **add** und der Funktion **image**. Da geben wir die Position auf der X-Achse und der Y-Achse an um zum Schluss den Namen des Bildes angeben zu können. Die Position 0/0 ist in Phaser in der linken oberen Ecke.

Unser kompletter Code sieht also nun wie folgt aus:

```
1 <!DOCTYPE html>
2 <html>
```

```
3
4   <head>
5     <script src="https://cdn.jsdelivr.net/npm/phaser@3.15.1/dist/phaser-arcade-physics.min.js"></script>
6   </head>
7
8   <body>
9     <script>
10      var config = {
11        width: 800,
12        height: 600,
13        backgroundColor: '#4488aa',
14        scene: {
15          preload: preload,
16          create: create
17        }
18      };
19
20      var game = new Phaser.Game(config);
21
22      function preload() {
23        this.load.image('sky', 'assets/bild1.png');
24      }
25
26      function create () {
27        this.add.image(500, 200, 'sky');
28      }
29    </script>
30  </body>
31 </html>
```

NodeJS - Projekt

Hier wird es alle Informationen zu dem NodeJS - Projekt geben.

Die Idee / Das Projekt

Das Projekt ist es einen Blog mit Tagebucheinträgen zu entwickeln, in dem die entwicklung von diesem Projekt festgehalten und dokumentiert werden kann. Dafür muss man einen Artikel schreiben (erstmal als Datei) und später können auch Bilder eingefügt werden.

Phase 1

Die erste Phase ist das fertigstellen des Blogs, wobei die Artikel in einer Datei stehen und nicht in einer Datenbank. Die Datei wird quasi als Datenbank benutzt. Das Format der Datei steht zu diesem Zeitpunkt noch nicht fest. ### Phase 2 Die zweite Phase ist es sich mit einer Datenbank zu beschäftigen und die Artikel in diese zu verlagern.

NodeJS

NodeJS ist eine Laufumgebung für Javascript mit dem man Anwendungen in Javascript schreiben kann ohne einen Webserver aufsetzen zu müssen. Mit NodeJS ist es also möglich serverseitig Javascript zu benutzen und ist somit nicht mehr zwingend auf einen Webserver angewiesen. Zudem ermöglicht NodeJS den Zugriff auf lokale Ressourcen (wie Dateien usw.) und macht es somit nützlich für viele Projekte.

Nodejs - Installation

Da ich oben schon beschrieben habe was NodeJS überhaupt ist, gehe ich jetzt hier also lediglich auf die Installation ein. Heruntergeladen werden kann NodeJS von folgender Webseite: <https://nodejs.org/en/>. Hier kann nun entweder die LTS Version oder die Current Version gewählt, heruntergeladen und installiert werden.

Installation Testen

Dazu drücken wir die **Windows**-Taste und die Taste **R**. Damit rufen wir ein Fenster auf (siehe Abbildung 1), indem wir Programme ausführen können. In dieses Fenster schreiben wir nun `cmd` rein und drücken Enter. Damit öffnen wir die Eingabeaufforderung (siehe Abbildung 2) in der wir nun ebenfalls wieder Befehle ausführen können. In dieses Fenster können wir nun das Programm `node` (welches von NodeJS kommt) ausführen und uns z.B. die Version anzeigen lassen. Dafür schreiben wir in die Eingabeaufforderung folgenden Befehl rein.

```
1 node -v
```

Mittels `node` rufen wir das Programm `nodejs` auf und mit dem Parameter `-v` weisen wir `node` an uns nur die Version anzuzeigen. Das ganze habe ich mal als Abbildung 3 hier reingepackt. Wichtig ist, es kann durchaus sein, dass die Version nicht die gleiche ist wie bei mir.

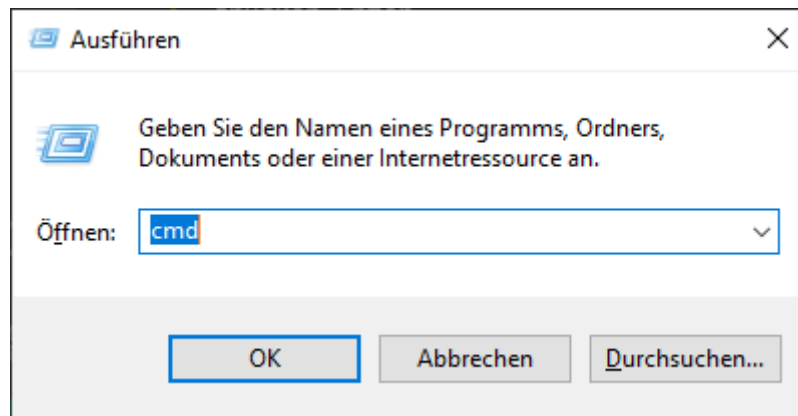


Abbildung 1: In dem Ausführen Fenster können wir Programme oder Befehle ausführen

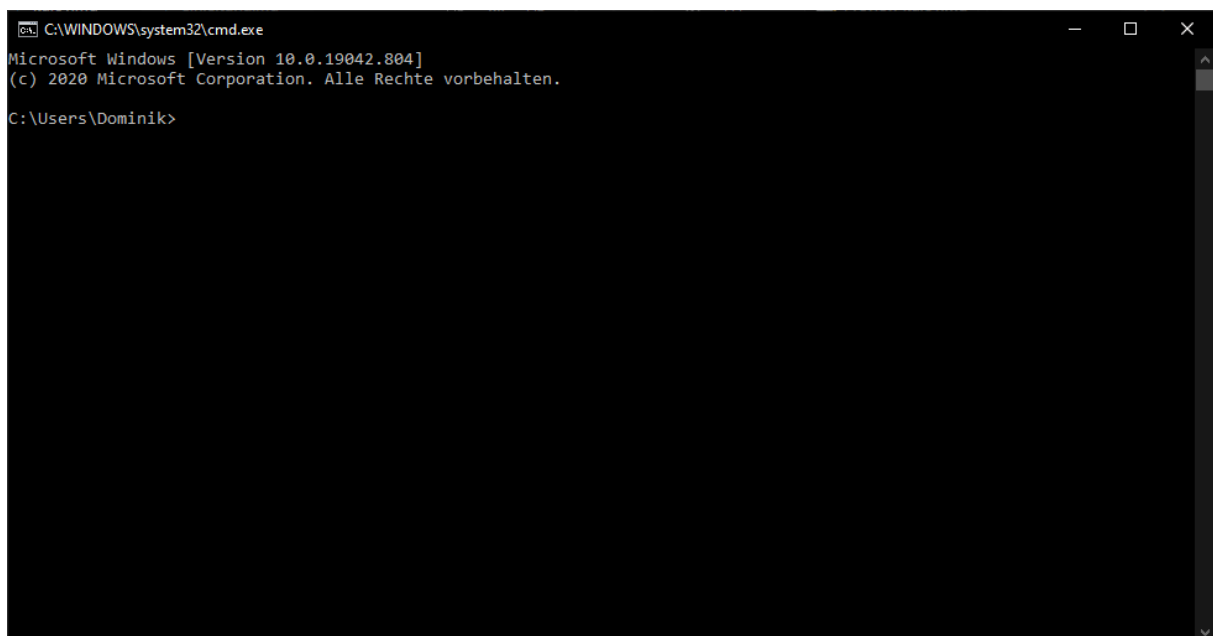
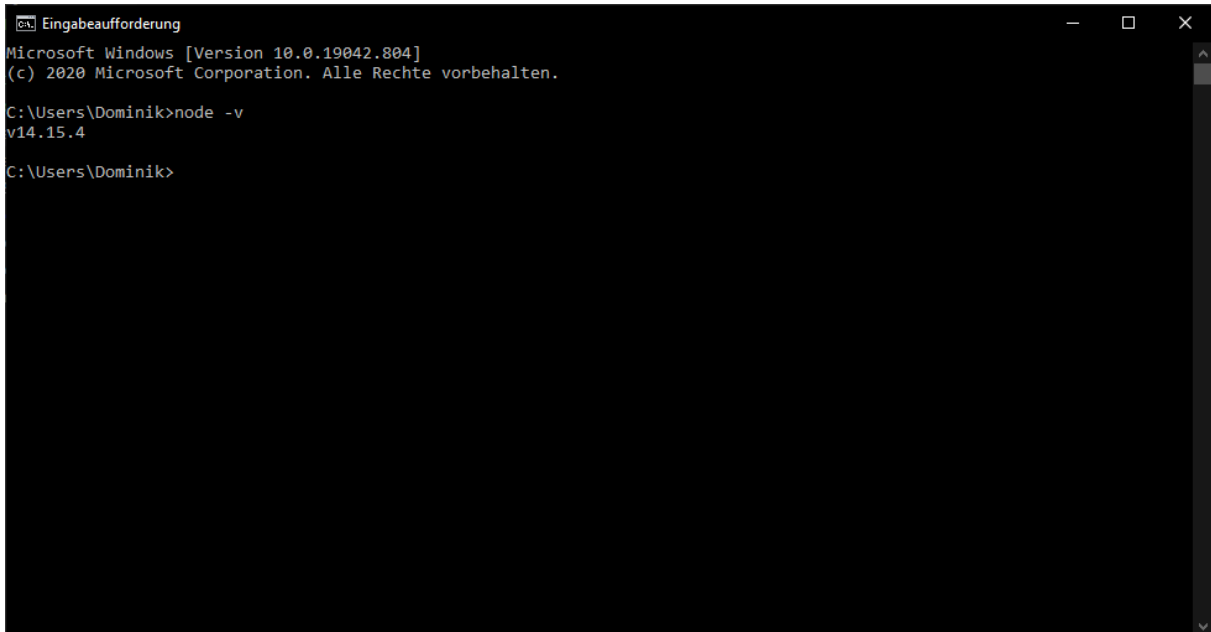


Abbildung 2: Unsere Eingabeaufforderung für Befehle unter Windows



```
Eingabeaufforderung
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Dominik>node -v
v14.15.4

C:\Users\Dominik>
```

Abbildung 3: `node -v` gibt die verwendete Version von dem Programm nodejs aus

NodeJS verwenden

Um NodeJS nun zu verwenden, können wir in unserem Projektordner eine neue Datei namens `indx.js` erstellen. Die Dateierweiterung `.JS` steht für Javascript und der Name kann tatsächlich frei ausgewählt werden. Um nun die Datei auszuführen, müssen wir das Node-Programm ausführen, mittels `node` gefolgt von der Datei, die wir ausführen wollen. In unserem Fall als `node index.js`

Unsere erste NodeJS Anwendung

Für unsere erste NodeJS-Anwendung müssen wir uns, da wir ja auch eine Browseranwendung schreiben wollen, von NodeJS bestimmte `http` (Protokoll für Browser) Funktionen importieren. Das kann mittels folgendem Code gemacht werden.

```
1 const http = require('http');
```

packt die nodejs internen `http`-Funktionen in die Variable `http`. Damit sind die Funktionen über diese Variable aufrufbar.

Als nächstes können wir unser Server definieren.

```
1 const server = http.createServer((req, res) => {
2   res.end('Hello World');
```

```
3 });
```

Erstellt ein Objekt (unseren Server) mit den zuvor importierten http-Funktionen. Das req und res sind der Request (Anfrage) an den Server und der Response (Antwort von unserem Server). Diese lassen sich nun verarbeiten und bearbeiten.

Mittels der Zeile `res.end('Hello World')` wird als Response an unserem Browser die Nachricht `Hello World` geschickt. Damit wird die Schrift `Hello World` in unserem Browser angezeigt, wenn wir den Server aufrufen.

Den Server aufrufen Um den Server aufzurufen müssen wir definieren, worauf unser Server hören soll. Das können wir mittels folgendem Code machen:

```
1 server.listen(port, hostname, () => {
2   console.log(`Server is running`);
3 });
```

Lässt unser Server-Objekt auf den Port und den Hostnamen hören.

Hier wird festgelegt, dass der Server auf den spezifizierten Port und dem festgelegten Hostnamen hört. Als Beispiel gebe ich hier meine locale Adresse (127.0.0.1) und den Port 3000 an.

```
1 server.listen(3000, '127.0.0.1', () => {
2   console.log(`Server is running`);
3 });
```

Auf weitere Pfade reagieren

Bis jetzt können wir auf folgende Pfade reagieren `http://localhost:3000` reagieren. Meistens ist es jedoch so, dass man auch noch auf andere URL reagieren möchte. In unserem Beispiel wäre also als Beispiel / um alle Blogs anzuzeigen und vielleicht `/blog/1` um z.B. einen bestimmten Blog anzuzeigen.

Aufrufen kann das ganze nun im Browser unter `http://localhost:3000/test` und `http://localhost:3000`

Unser HTML mit NodeJS ausgeben

Möchte man nun HTML mittels nodeJS ausgeben, muss man erstmal die Datei an sich in NodeJS laden. Das ganze geht, indem man ebenfalls die Funktionen für Dateien importiert.

```
1 const fs = require("fs");
```

Importiert die Filesystem (Dateisysteme auf Deutsch) Funktionen und „speichert“ sie in der Variablen fs.

Damit kann man nun das HTML in eine andere Variable reinladen.

```
1 fs.readFile("index.html", "utf8", (err, data) => {
2   if (err) {
3     console.log(err);
4   }
5   index = data;
6 });
```

Mittels der Filesystem Funktionen können wir die Funktion `readFile` aufrufen und dort die Dateinamen und die Kodierung (wie die Datei genau abgespeichert wurde (ist meistens utf8)) angeben um Sie so zu laden. Der dritte Parameter ist eine Funktion. In dieser Funktion gibt es einmal ein Fehler (error) und daten (data). Diese Daten können wir benutzen und einer anderen Variablen zuweisen. Vorher prüfen wir nach ob die Variable err einen Fehler enthält.

```
1 if (err) {
2   console.log(err);
3 }
4 index = data;
```

Als nächstes können wir recht einfach unser html auf einen bestimmten Pfad ausgeben. Das HTML liegt nämlich nun in unserer Variable und kann mittels `res.end(index)` ausgegeben werden. Wichtig ist hierbei, dass es zuerst als Text ausgegeben wird und nicht als HTML dargestellt wird. Das liegt an den Header des Response, welchen wir normalerweise nicht sehen, er allerdings ein paar Standardwerte mitbekommt. Dieser ist für den Inhaltstyp (content-type) eben text/plain also einfach normaler text. Um das Verhalten zu ändern und alles als HTML darzustellen muss das nun geändert werden auf den Inhaltstyp text/html. Das ganze geht mittels

```
1 res.setHeader("Content-Type", "text/html");
```

Unser vollständiger Pfad sieht also nun so aus:

```
1 else if (req.url.startsWith("/blog")) {
2   res.setHeader("Content-Type", "text/html");
3   res.end(index);
4 }
```

Setzt den Inhaltstyp (Content-Type) auf text/html um das Ergebnis als HTML darzustellen.

Das HTML anpassen

Das HTML kann mittels eines einfachen Tricks angepasst werden und zwar dem des Suchen und Ersetzen. In Javascript kann man auf Texte Funktionen anwenden und so z.B. die Funktion `replace` aufrufen. Diese Funktion überschreibt alle Vorkommnisse des ersten Parameters mit den des zweiten in dem ausgewählten Text. Dafür schreiben wir als Beispiel in unser HTML bestimmte Tags oder auch Titel die wir kennen um Sie später ersetzen zu können. Als Beispiel setze ich hier einmal den Titel (im Header) als `{{TITLE}}` um ihn einfach identifizieren und ändern zu können.

```
1 <!DOCTYPE html>
2 <html lang="de">
3   <head>
4     <title>{{TITLE}}</title>
5   </head>
6   ...
```

... steht für den weiteren Text, da hier erstmal nur der `<title>` Tag interessant ist.

Jetzt kann in der Funktion, die das HTML zurück gibt unsere Variable angepasst werden.

```
1 index = index.replace("{{TITLE}}", "Meine Geschichte");
```

Überschreibt alle Vorkommnisse von `{{TITLE}}` mit „Meine Geschichte“

Alles zusammensetzen

Nun möchten wir alles zusammen setzen und unser HTML an die Blogseite anpassen. Dafür können wir uns erstmal eine URL aussuchen auf die wir dann reagieren möchten, wenn Sie angesprochen wird und wir für unseren Blog benutzen wollen. Ich würde jetzt einfach mal festlegen, dass wir die Seiten des blogs auf `/blog/0` als Beispiel legen um den ersten Eintrag unseres Blogs anzuzeigen. Wieso der Blog bei 0 anfängt erkläre ich später noch einmal. Jetzt müssen wir uns die letzte Zahl unserer URL besorgen. Da es hier wichtig ist immer die letzte komplette Zahl zu bekommen können wir uns alles geben lassen vom Ende bis zu der Position des letzten `/`-Zeichens.

```
1 else if (req.url.startsWith("/blog")) {
2   // Gibt mir die Position des letzten / Zeichens
3   var position = req.url.lastIndexOf("/");
4
5   // Gibt mir die eigentliche Zahl von unserer oben errechneten
   Position und der gesamten länge
6   var zahl = req.url.substring(position + 1 , req.url.length);
7
8   // Setzt nochmal den Inhaltstyp auf text/html
9   res.setHeader("Content-Type", "text/html");
```

```
10
11 // Greift auf unsere Geschichten Variable zu und überschreibt den
    entsprechenden Text mit der anderen Variable aus unserer vorher
    definierten Datei
12 index = index.replace("{TITLE}", geschichten[zahl].zusammenfassung)
    ;
13 index = index.replace("{UBERSCHRIFT}", geschichten[zahl].
    ueberschrift);
14
15 // Gibt den aktualisierten Text wieder zurück um ihn darzustellen im
    Browser
16 res.end(index);
17 }
```

Die Funktion ist für alle urls gültig die mit /blog anfangen. Berechnet anschließend die letzte Zahl und nimmt diese Zahl als index für unsere JSON-Datei. In Javascript fangen solche Listen von mehreren Elementen immer mit 0 an, weswegen unser erster Eintrag auch den Index mit der Nummer 0 zugeschrieben bekommt. Alternativ muss mit einer 1 angefangen werden und immer 1 abgezogen werden um wieder den richtigen Index zu bekommen.

Exkurs - HTML

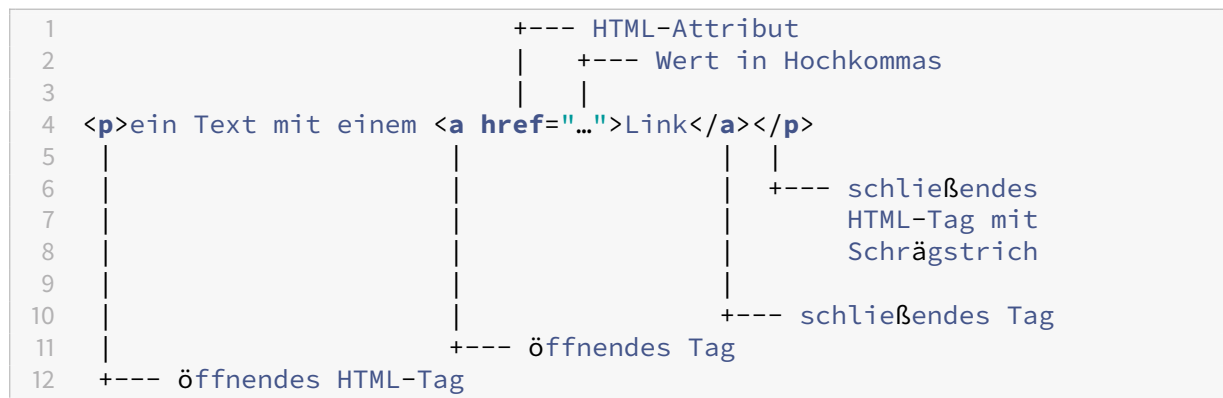
HTML (Hypertext Markup Language) wird benutzt um eine Webseite zu beschreiben. Das heißt mittels HTML wird der Aufbau einer Webseite beschrieben und nicht deren eigentlichen Design. Das Design wird anschließend in CSS gemacht. In der HTML gibt es sogenannte Tags, die für verschiedene Strukturen stehen und von dem Englischen Begriff abgeleitet werden. Ein Tag wird mit den folgenden Klammern eingeleitet `<>` und meistens wie folgt beendet `</>`. Der Tagname steht zwischen den Klammern bzw. zwischen dem Schrägstrich und der schließenden Klammer. Hier sind ein paar Beispiele:

- `<html></html>`
- `<p></p>`
- `<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.

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

Aufbau eines Tags



Webseiten-Struktur

Als ein Beispiel wird hier ein sehr simple Webseite erhalten.

```
1 <html>
2   <head>
3     <title>Meine Webseite</title>
4   </head>
5   <body>
6     Lorem ipsum
7   </body>
8 </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>`:

```
1 <body>
2   <h1>Lorem ipsum </h1>
3   <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. </p>
```

```
4   <p>Mauris ante neque, vehicula quis, convallis sit amet,  
5   vestibulum ornare, arcu. </p>  
6   <p>Donec at lorem et elit congue dictum. Cras sapien ligula,  
7   rutrum quis, posuere id, faucibus id, risus. Maecenas sed est  
8   volutpat arcu adipiscing tempus. Sed dictum mauris euismod  
9   mauris.</p>  
10 </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.
- ``, ``
 - erzeugen ungeordnete bzw. geordnete (nummerierte) Listen. Ungeordnete Listen werden mit einem führenden Listenelement dargestellt, geordnete Listen mit einem Index. In beiden Listen wird jedes Listenelement in li-Tags eingeschlossen.

```
1 <ol>  
2   <li>HTML strukturiert den Inhalt </li>  
3 </ol>
```

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

Kommentare

Kommentare werden in HTML wie folgt beschrieben:

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

Kommentare dienen dazu den Quelltext besser zu strukturieren und für den Leser angenehmer 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:

```
1 <table>
2   <tr>
3     <td>Zeile 1 Spalte 1</td>
4     <td>Zeile 1 Spalte 2</td>
5   </tr>
6   <tr>
7     <td>Zeile 2 Spalte 1</td>
8     <td>Zeile 2 Spalte 2</td>
9   </tr>
10 </table>
```

Wie man sieht sind 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 letztlich 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 gesehen 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 Eigenschaften 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 adressiert 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
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 Papiers (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 Wurzelement 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.

```
1 margin-top: 5px;  
2 margin-left: 5px;  
3 margin-right: 5px;  
4 margin-bottom: 5px;
```

Das Element erhält zu jedem anderen Element 5px Abstand

Einzeln vergeben werden oder aber zusammenfassend geschrieben werden:

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

Das Element erhält zu jedem anderen Element 5px Abstand. Die Aufteilung ist hier wie folgt: top, right, 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.

```
1 padding-top: 5px;  
2 padding-left: 5px;  
3 padding-right: 5px;  
4 padding-bottom: 5px;
```

Das Element erhält in jede Richtung 5px mehr zum Rand

Einzeln vergeben werden oder aber zusammenfassend geschrieben werden:

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

Das Element erhält in jede Richtung 5px mehr zum Rand. Die Aufteilung ist hier wie folgt: top, right, 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 eigenen 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 ihnen 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.

Name	Bereich	Erläuterung
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.

Exkurs - JSON

JSON steht für Javascript Object Notation und ist ein Textbasiertes Datenformat, welches normalerweise dazu verwendet wird Daten von einem Webservice zum Client zu übertragen. Zur Datenvisualisierung verwendet JSON das Key Value Prinzip, welches darauf beruht zu jedem Eintrag ein bestimmten Wert zu haben. Ein großer Vorteil von JSON ist, dass es in nahezu jeder Programmier- oder Skriptsprache entsprechende Funktionen für den Umgang mit JSON gibt.

Aufbau

Der Aufbau von JSON sind Key-Value-Paare, wo immer links der Schlüssel (Key) und rechts der Wert (Value) steht. Dabei werden Texte (strings) in Hochkommata eingeschlossen und Zahlen nicht. Jedes Key-Value paar wird von einem Komma getrennt, bis auf das letzte. Nach dem letzten Element kommt kein Komma. Als ungewöhnlicher Wert gibt es hier den Wert null. Dieser bedeutet es gibt kein Wert (es ist also weder eine Zahl noch ein String).

```
1 {  
2   "name": "Max",  
3   "nachname": "Mustermann",  
4   "strasse": "Musterstrasse",  
5   "hausnummer": 7  
6 }
```

Objekte als Wert

In JSON ist es ebenfalls möglich komplette Objekte (bzw. mehrere Zuweisungen unter einen Key (Schlüssel) laufen zu lassen):

```
1 {  
2   "person": {  
3     "vorname": "Max",  
4     "nachname": "Mustermann",  
5     "alter": 32  
6   }
```

```
6   }  
7 }
```

Arrays

Wenn man in dem Datenformat JSON ein Array vereinbaren möchte kann man die Daten in eckigen Klammern einschließen.

```
1 {  
2   "namen": ["Max", "Gabi", "Bettina", "Frank"],  
3   "zahlen": [5, 7, 9, 10, 13]  
4 }
```

Objekt-Arrays

In JSON ist es auch möglich Arrays zu definieren. Hierfür wird der Schlüssel hingeschrieben und die eigentlichen Arraydaten werden in eckigen Klammern (`[]`) eingeschlossen.

```
1 {  
2   "personen": [  
3     {  
4       "name": "Max",  
5       "nachname": "Muster"  
6     },  
7     {  
8       "name": "Gabi",  
9       "nachname": "Mustermann"  
10    }  
11  ]  
12 }
```