Lernzettel Klausur HTML & CSS

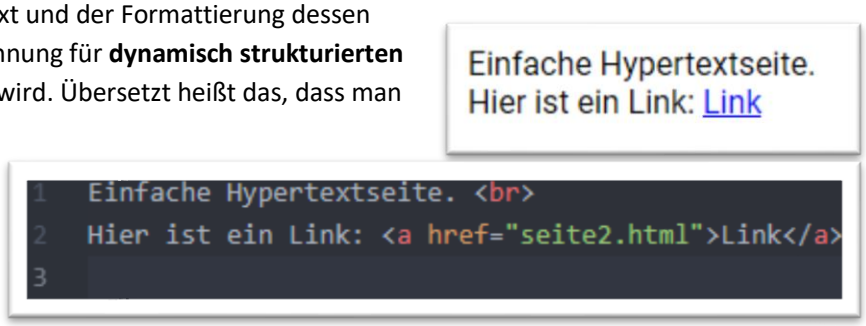
Grundlagen

In jedem Fall ist das verfolgte Ziel bei unserem Thema, das **Erstellen einer Website**. Heutzutage ist völlig klar, was eine Website ist und wie sie funktioniert, jedoch sollte man sich kurz mit der Herkunft dieser Informationsanzeige beschäftigen, um zu verstehen, warum sie so aufgebaut ist, wie sie aufgebaut ist.

Herkunft

Noch vor der Entwicklung eines globalen Informationsnetzwerkes („World Wide Web“), als der Datenaustausch zum Beispiel nur innerhalb von Firmengebäuden verwendet wurde, benötigte man eine Möglichkeit um **digitale Informationen grafisch übersichtlich darzustellen**. Zu dem Zeitpunkt war der Informationsaustausch auf Zeichenketten („Strings“) oder gar Binärcode beschränkt, oder wahlweise mussten die Unternehmen ihre eigenen Lösungen entwickeln, um digitale Informationen grafisch darzustellen.

Um dieses Problem zu lösen, entstand 1989 am **CERN** ein Projekt, das letzendlich am 3. November **1992** die erste Version des HTML Formates veröffentlichte. HTML steht dabei als Abkürzung für „**H**yper**t**ext **M**arkup **L**anguage“. Es ist allerdings KEINE Programmiersprache, sondern wie der Name schon sagt eine Sprache zur Darstellung von Hypertext und der Formattierung dessen („Markup“). Hypertext ist die Bezeichnung für **dynamisch strukturierten Text**, der technisch quasi „veredelt“ wird. Übersetzt heißt das, dass man diesen Hypertext nicht in einer speziellen Reihenfolge lesen muss, sondern durch Querverweise („Links“) oder andere Techniken zu den Informationen gelangen kann, die man braucht.



Schon ein Jahr später, 1993, gab es den ersten Vorschlag für für **Web-Stylesheets**. HTML-Dokumente hatten zwar die Möglichkeit geschaffen, Informationen darzustellen, es fehlte aber jegliche Möglichkeit, diese Darstellung zu personalisieren. Am 10. Oktober 1994 veröffentliche ein Mitarbeiter des CERN den ersten Vorschlag für „Cascading HTML Style Sheets“. Ab diesem Zeitpunkt arbeiteten mehrere Teams an dem gleichen Ziel, letzendlich setzte sich das CSS wie wir es heute kennen in einer frühen Version durch. CSS ist dabei genausowenig eine Programmiersprache wie HTML. **Cascading Style Sheets**, zu deutsch „Mehrstufige Formatvorlagen“ ist eine **Formattierungssprache**, vor allem für HTML, aber auch für SVG und XML Dokumente.

HTML Syntax

„Unter Syntax versteht man ein Regelsystem zur Kombination elementarer Zeichen zu zusammengesetzten Zeichen in natürlich oder künstlichen Zeichensystemen.“

Kurz gesagt: Der Syntax ist die **Struktur des Codes**, jede Programmiersprache hat in der Regel ihren eigenen Syntax, auch wenn viele Sprachen ihre Gemeinsamkeiten haben.

Tags

```
<tag>Inhalt</tag>
```

Jedes Element einer Website wird mithilfe der sogenannten **Tags** definiert. Jeder Tag wird innerhalb **dreieckiger Klammern** geschrieben („< >“). Und jedes Element wird jeweils durch einen Tag eingeleitet und beendet, das beendende Tag wird angezeigt durch ein „/“. Zwischen den Tags steht immer der Inhalt eines Elements. Das kann entweder Text sein, oder auch weitere Elemente.

Beispiel:

```
<p>Willkommen auf dieser Seite.</p>
```

Wie hier zu sehen, ist der Syntax, also der **Aufbau immer derselbe**. Der „p“-Tag steht für „Paragraph“, also „Absatz“ und definiert einen abgeschlossenen Textblock.

Ausnahme: Es gibt einige sogenannte „**Self-Closing-Tags**“. Einige Elemente können keine anderen Elemente in sich tragen. Diese Elemente sehen dann wie folgt aus:

```
<br>
```

```
<br />
```

In der neuesten HTML Version funktionieren beide Schreibweisen gleich, das Slash am Ende ist nur zur optischen Orientierung gut. Empfehlen tut sich allerdings die linke Schreibweise, da die rechte Version in einer älteren HTML Version zu
> konvertiert wird und dann Fehler ausgibt.
„br“ steht für „break“ und markiert einen Zeilenumbruch. Dieser kann logischerweise **keine weiteren Elemente enthalten** und ist daher ein „Self-Closing-Tag“.

Verschachtelung

Wie bereits erwähnt, können **Elemente** widerum **andere Elemente „enthalten“**, dieses Konzept nennt sich Verschachtelung. Beispiel:

```
<div> <p>Willkommen auf dieser Seite.</p> </div>
```

Es ist also problemlos möglich, weitere Objekte innerhalb anderer Objekte zu schreiben, in diesem Fall steht ein *p* innerhalb deines *div's*. Ein „div“ ist ein sogenannter „divider“ und teilt gewisse Abschnitte ein, um sie später gesondert zu formattieren, hat in diesem Beispiel aber noch keine wirklichen Auswirkungen.

Diese Verschachtelung funktioniert auch mit mehreren Elementen. Beispiel:

```
<div>
   <h1>Klassenarbeit</h1>
   <p>Willkommen auf dieser Seite.</p>
</div>
```

„h1“ ist ein sogenanntes Heading und definiert eine Überschrift. Bemerke, dass der Inhalt des Dividers hier **eingerückt geschrieben** ist, um die Verschachtelung besser darzustellen.

Es ist genauso gut möglich, alle Absätze, Tabstopps (zum Einrücken) und Leerzeichen wegzulassen und der Code liefert dasselbe Ergebnis. All diese Dinge sind einzig und alleine für die **gute Lesbarkeit des Codes** bestimmt, die allerdings sehr wichtig ist.

HTML Attribute

Eine Website besteht also aus einzelnen Elementen, die durch Tags definiert werden. Diese Elemente können nun aber auch bestimmte **Eigenschaften** haben. Das einfachste Beispiel dazu ist ein Bild:

```
<img src="bild.png" alt="Läd…">
```

Zunächst sieht man, dass Bilder „Self-Closing-Tags" sind, man braucht also nur ein Tag. Dieses Bild besitzt zwei Eigenschaften: Einmal das „src", was „source", bzw. „Quelle" bedeutet und den Dateipfad des Bildes angibt. Außerdem „alt", „alternative", der alternative Text, der angezeigt wird wenn das Bild (noch) nicht geladen wurde.

Diese Eigenschaften nennt man **Attribute**. Attribute werden immer im **ersten Tag** eines Elementes geschrieben, also „`<tag attributes></tag>`" and not „`<tag ></tag attributes >`". Im obigen Beispiel ist das egal, da es ja nur einen Tag gibt.

Attribute wie „src" und „alt" sind speziell für Bilder gedacht, es gibt allerdings auch ein paar Attribute, die **allgemeingültig** für alle Elemente gelten. Die wichtigen sind:

Attributname	Beschreibung
id	Eindeutige Identifikation eines Elements für CSS & Javascript. Es können keine mehreren Elemente dieselbe ID haben.
class	Identifikation der Elemente für CSS & Javascript. Klassen werden einmal definiert und können unbegrenzt oft wiederverwendet werden, was viele Vorteile mit sich bringt.
style	Ermöglicht CSS Code innerhalb der Elemente. Mehr dazu im CSS Teil.

Wie diese Elemente angewendet werden, wird im CSS Teil beispielhaft erklärt.

HTML Dokumente

Doctype

In einer HTML Datei sollte man immer zu Beginn noch einmal den „Doctype" angeben, damit der Browser sicher weiß, dass **HTML Code** folgt. Man könnte meinen, dass der Browser automatisch erkennt, wenn er eine .html Datei ließt, dass es sich um HTML-Code handelt, was auch der Fall ist. Das Problem ist ein anderes, denn wenn man z.B. auf eine Website geht wie https://www.google.de/google.html dann erhält der Browser **nur den Inhalt** der google.html **Datei** übertragen, anstatt die gesamte Datei mitsamt Endung. Deshalb kann der Browser unter Umständen nicht erkennen, dass es sich um Html Code handelt.

```
<!doctype html>
```

Dieser Tag sollte daher zu Beginn eines jeden HTML Dokumentes stehen.

Head

Typischerweise nach dem Doctype folgt der „Head“, also der **Kopf** des Dokumentes. Der Kopf enthält normalerweise alle Informationen die normalerweise **unsichtbar** für den Nutzer sind und die **Rahmeninformationen** für den Browser. Diese Art der Informationen nennt man **Metadaten**.

Folgende Metadaten sind wichtig:

```
<!doctype html>
<head>
   <title>Titel</title>
   <link rel=“stylesheet“ href=“styles.css“>
   <script src=“script.js“>
   <meta name=“Meta Name“ content=“Meta Settings“>
</head>
```

Title

Der **Titel des Dokuments**, der im Browser auf dem Tab der Seite angezeigt wird.

Link

Wird verwendet, um eine externe Resource zu verlinken. In der Regel stylesheets (CSS).
Das Attribut „rel“ gibt die Art der Resource an und „href“ den Dateipfad/Link.

Script

Wird verwendet, um Javascript zu verlinken. Entweder wie oben gezeigt als externe Datei über das „src“-Attribut, oder zwischen den <script></script> Tags.

Meta

Alle anderen Metadaten werden über den den „meta“ tag definiert. Beispielsweise:

```
<meta name="description" content="Lernzettel für IV">
```

Dies ist das **allgemeine Format** für Metadaten, der Name ist in diesem Fall „description“, es definiert also die Beschreibung und die Einstellung dessen steht unter „content“. Wichtige Ausnahme ist das Folgende:

```
<meta charset=“UTF-8“>
```

Diese Einstellung sollte in jedem Dokument stehen, es definiert den **Zeichensatz**, der im Dokument verwendet wird. Das ist wichtig, um **Umlaute und Sonderzeichen** verwenden zu können.

WELCHE METADATEN SIND WICHTIG?

Es gibt zahlreiche Metadaten, die folgenden Einstellungen sind jedoch **empfohlen**:

```
<head>
  <meta charset="UTF-8">
  <meta name="description" content="Lernzettel">
  <meta name="keywords" content="IV, Klausur, Lernzettel">
  <meta name="author" content="DeWil">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
</head>
```

Und sie haben die folgenden Funktionen:

Name	Funktion
description	Beschreibung der Seite, wird u.a. von Suchmaschinen genutzt
keywords	Stichwörter die die Seite beschreiben, wird u.a. von Suchmaschinen genutzt
author	Author der Seite,
viewport	Einstellungen für verschiedene Endgeräte, speziell für Smartphones wichtig.

Body

Der wichtigste Part eines Dokuments, hier werden all die Elemente einer Website platziert, die man am Ende sehen kann. Beispiel für eine „fertige“ Website:

```
<!doctype html>
<head>
  <meta charset="UTF-8">
  <meta name="description" content="Lernzettel">
  <meta name="keywords" content="IV, Klausur, Lernzettel">
  <meta name="author" content="DeWil">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
</head>
<body>
  <h1>Lernzettel für IV</h1>
  <p>Dies ist die finale Website.</p>
</body>
```

HTML Elemente

Jetzt, wo die Struktur einer Website klar ist, aus welchen Elementen kann man sie zusammensetzen?

<a> - Link

Verlinkungen werden mithilfe des „a“-Tags definiert, zwischen den Tags kann der angezeigte Text stehen und als Attribut dann der Link. Beispiel:

```
Hier ist ein <a href=“https://www.google.de“>Link</a>
```

Hier gelten die Regeln für Dateipfade, wie in einem späteren Kapitel erklärt. „href“ heißt das Attribut.

<p> - Absatz

Ein Fließtext in HTML sollte in Absätze unterteilt werden. Ein solcher Textabsatz steht zwischen p-Tags.

```
<p>Einfach nur ein kleiner <b>Absatz.</b></p>
```

Außerdem im Beispiel verwendet wird hier ein „b“-Tag, („bold“ also fett), einer von folgenden:

- `<b>` - Bold text
- `<strong>` - Important text
- `<i>` - Italic text
- `<em>` - Emphasized text
- `<mark>` - Marked text
- `<small>` - Small text
- `<del>` - Deleted text
- `<ins>` - Inserted text
- `<sub>` - Subscript text
- `<sup>` - Superscript text

Diese Formattierungen können nicht nur in Absätzen verwendet werden, sondern überall wo Text angezeigt wird. Allerdings werden sie meistens in Absätzen verwendet.

<h1>, <h2> … - Überschrift

Zur besseren Struktur können Überschriften verwendet werden, mithilfe der Tags <h1> bis <h6>, 1 ist das größte und 6 ist das kleinste.

```
<h2>Überschrift</h2>
```

<img> - Bilder

Wie bereits gezeigt, gibt es die Möglichkeit, Bilder einzubinden. Beispiel:

```
<img src=“bild.png“ alt=“Läd…“>
```

Bilder benötigen zunächst den Dateipfad mithilfe von „src“ und hier ist noch „alt“ als Alternativtext angegeben, falls das Bild noch nicht geladen hat.

Bild haben prinzipiell Eigenschaften wie Breite und Höhe, diese können mithilfe von „height“ und „width“ Attributen innerhalb des Tags angegeben werden. Es empfiehlt sich allerdings, die Bildergrößen in CSS zu definieren, damit ein responsive Design erreicht wird, es also auf allen Gerätegrößen gut funktioniert.

<table> - Tabellen

HTML unterstützt auch Tabellen, diese werden durch die folgenden Tags definiert:

- <table> - Die Tabelle
  - <tr> - Eine Reihe der Tabelle – *„tr“ = „Table Row“*
    - <td> - Ein Datensatz der Tabelle (Eine Zelle) – *„td“ = „Table Data“*
    - <th> - Tabellenkopfzelle, genau wie TD, nur dass man es wegen dem anderen Namen einzeln mit CSS formattieren kann. – *„th“ = „Table Header“*

Diese Struktur sieht in der Praxis wie folgt aus:

```
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

, <ol> - Listen

„ul“ steht für „unordered list“, während „ol“ für „ordered list“ steht. Beides sind Listen, allerdings haben geordnete Listen z.B. Zahlen oder Buchstaben vor den einzelnen Elementen, während ungeordnete Listen z.B. Punkte davor haben.

In beiden typen steht <li> für „List Item“, also für die einzelnen Punkte der Liste. Beispiel:

```
<ul>
  <li>ASUS</li>
  <li>MSI</li>
  <li>EVGA</li>
  <li>ZOTAC</li>
</ul>
```

Jegliche weitere Einstellung der Listen erfolgt mithilfe von CSS.

<!-- ... --> - Kommentare

Jeder gut strukturierte Code sollte ab und zu Kommentare enthalten, um anderen Leuten das Verständnis über den eigenen Code zu erleichtern. Das Format für Kommentare sieht so aus:

```
<!-- Kommentar -->
```

Kommentare können außerdem gut zu Testzwecken verwendet werden, indem bestimmte Bereiche des Codes in <!-- --> gesetzt werden, um zu testen wie eine Alternative dazu aussehen würde.

Loading Order

„Loading Order is Important“ ist quasi das Grundmotto einer jeden Website. Für eine einfache Website ist das noch nicht unbedingt relevant, aber sobald man mit Skripten arbeitet, sollte man dieses Konzept kennen.

Objekte werden in der Reihenfolge geladen, in der sie auf der Seite auftauchen!

Das ist deshalb wichtig, da man z.B. nicht im Head Datei ein Skript einbindet, das sofort versucht auf ein HTML Objekt im Body zuzugreift, das aber noch gar nicht existiert.

Nichts, desto trotz sollte man Skripte immer am Anfang der Datei einbinden und in diesem speziellen Fall vom „onload“ Attribut in HTML gebrauch machen, das dann später das Skript ausführt, sobald das Objekt geladen wurde.

CSS Grundlagen

Gut, also die Website ist mithilfe unseres HTML Codes strukturiert aber wirklich gut aussehen tut das alles irgendwie noch nicht.. Also können wir die einzelnen Elemente jetzt mithilfe von CSS Code formattieren!

Extern vs. Inline

Prinzipiell gibt es 3 Möglichkeiten, CSS Code in eine Website einzubinden:

```
<link rel=“stylesheet“ href=“styles.css“>
```

```
<style> CSS CODE </style>
```

```
<div style=“CSS CODE“> HTML Element </div>
```

Das erste Beispiel ist mit Abstand das **eleganteste**: Man trennt HTML und CSS code sauber, indem man eine **externe CSS Datei** verlinkt. Der große Vorteil außerdem: Man kann eine globale CSS Datei über mehrere HTML Dateien hinweg verwenden.

Das zweite Beispiel eignet sich vor allem für Tests und extrem kleine Websites. Man definiert einfach ein Style Element **innerhalb** des HTML Dokuments und schreibt dort den CSS Code hinein. Problem ist die **fehlende Wiederverwendbarkeit** über mehrere Dokumente hinweg. Nutzbar trotzdem, falls eine spezielle Seite sich z.B. von der globalen Norm unterscheidet.

Das dritte Beispiel sollte **nie** verwendet werden, da es **mitten im HTML Code** steht und diesen unnötig unübersichtlich macht. Im Zweifel sollte man, wenn man einzelne Elemente formattieren möchte, immer Klassen oder IDs verwenden. (Später mehr dazu)

Selektoren

Um ein Element mithilfe von CSS zu formattieren, muss man logischerweise erst auf das Objekt zugreifen. Dazu gibt es 3 Möglichkeiten.

Elementname

```
body {

}
```

Indem man einfach den Name des Elementtyps nutzt, lässt sich auf dessen Eigenschaften zugreifen. In diesem Fall auf den gesamten Body des Dokumentes, beispielsweise um eine globale Hintergrundfarbe zu definieren. Dieser Selektor sollte allerdings mit vorsicht verwendet werden, da es wirklich auf **ALLE Elemente** des entsprechenden Typs zutrifft.

Ids

```
#element {

}
```

```
<div id="element"></div>
```

Diese Methode ist sehr nützlich, wenn ein Element wirklich so **nur einmal** auf einer Seite existiert. Wichtig ist, dass eine ID nur einmal verwendet werden kann. Außerdem muss die ID im HTML Code als Attribut angegeben werden. IDs werden in CSS mit einer #Raute geschrieben.

Klassen

```
.element {

}
```

```
<div class="element"></div>
```

Diese Methode ist ebenfalls empfehlenswert, allerdings kann ein Element in diesem Fall nicht nur einmal existieren sondern **beliebig oft**. Eine Klasse ist dazu da, dass bestimmte Eigenschaften über mehrere Objekte hinweg verändert werden. Klassen müssen ebenfalls als Attribut im HTML Code angegeben werden und in CSS werden sie mit einem .Punkt geschrieben.

Selektoren Mixen

In der Praxis ist es hilfreich, mehrere Selektoren zu vermischen, um der Übersichtlichkeit halber so wenig Code wie möglich schreiben zu müssen.

Zugriff auf mehrere Elemente

```
p, #main-header {
}
```

Durch ein Komma getrennt lässt sich auf die Eigenschaften **mehrerer Elemente** zugreifen, beispielsweise lässt sich so generell die Textfarbe ändern und die des „Main-Headers".

Zugriff auf verschachtelte Elemente

```
#hauptartikel h1 {
}
```

```
<div id="hauptartikel">
  <h1>Überschrift</h1>
</div>
```

Nützlich, um nicht jedem **Subelement** einzeln eine ID / Klasse geben zu müssen.

Weitere Selektoren

Es gibt viele viele weitere Selektoren, die verwendet werden können, beispielsweise kann man HTML Elemente durch ihre Attribute auswählen oder durch die Mausposition. Eine **Liste** aller weiteren Selektoren gibt es hier.

CSS Kommentare

In CSS lassen sich ebenfalls sehr einfach Kommentare schreiben, genauso empfehlenswert wie in HTML.

```
body {
  /* Kommentar */
}
```

In CSS, Kommentare stehen zwischen **/*** und ***/**.

CSS Eigenschaften

Nach dem Selektor eines Elementes, lassen sich dessen Eigenschaften ändern. Beispiel:

```
p {
  font-size: 40px;
}
```

In diesem Beispiel wird die Schriftgröße verändert, aller Textabsätze auf der Seite.

Es hängt immer von dem gewählten Element ab, welche Eigenschaften veränderbar sind. Eine Liste aller wichtigen Eigenschaften folgt später.

Farben

Grundfarben gibt es in CSS vordefiniert, z.B. „red“ oder „black“ kennt jeder Browser. Außerdem können Farben wie folgt definiert werden:

- „HEX“-Werte, Bsp. `#ff6347` für Tomatenrot.
- „RGB“-Werte, Bsp. `rgb(60, 179, 113)` für ein Hellgrün.
- „HSL“-Werte, Bsp. `hsl(300, 76%, 72%)` für ein helles Violett.

Größen

Jegliche Abstände und Größen werden mithilfe verschiedener Einheiten definiert. Diese Einheiten lauten:

Absolute Größen

Diese Größen sind für Websites eher ungeeignet, da sie (relativ) unabhängig von der Bildschirmgröße immer gleich sind und daher nicht skalieren.

cm	Centimeter
mm	Millimeter
in	Inch
px	Pixel - Wird für Websites gerne verwendet aber sollte mit Bedacht gewählt werden, ggf. lassen sich auch für verschiedene Bildschirmgrößen verschiedene px-Werte definieren.
pt	Punkte
pc	Picas

Relative Größen

Diese Größen skalieren mit der Bildschirmgröße (oder mit anderen Größen) und sind daher indeal, um ein gut skalierendes Webdesign zu erreichen. (Responsiveness)

em	Relativ zu der Schriftgröße des Elementes
ex	Relativ zu der x-Höhe der Schriftgröße des Elementes
ch	Relativ zur Breite von „0“
rem	Relativ zur Schriftgröße der Wurzel des Dokumentes (Root)
vw	Relativ zu 1% der Breite des Fensters
vh	Relativ zu 1% der Höhe des Fensters
vmin	Relativ zu 1% der kurzen Seite des Fensters
vmax	Relativ zu 1% der langen Seite des Fensters
%	Relativ zum übergeordneten Element

Auch möglich: „auto“, z.B. bei margin-right & margin-left, um ein Objekt zu zentrieren.

Liste der wichtigsten Eigenschaften

Name	**Beschreibung**
background-color	Hintergrundfarbe
background-image	Hintergrundbild, wird als `url(„bild.png“)` angegeben.
background-repeat	Wiederholung des Hintergrunds: `repeat|repeat-x|repeat-y|no-repeat`
background-attachment	Position zum Fenster hin: `scroll|fixed|local|initial|inherit`
background-position	Position innerhalb des Dokumentes: Link
border-style	Rahmen eines Elements: `solid|dotted|dashed` etc…
border-width	Rahmendicke
border-color	Farbe des Rahmens
margin	Abstand nach außen
margin-top […]	Abstand nach außen in eine Richtung (margin-top, -bottom, -right, -left)
padding	Abstand nach innen
padding-top […]	Abstand nach innen in eine Richtung (padding-top, -bottom, -right, -left)
height	Höhe eines Elements
width	Breite eines Elements
max-height	Maximale Höhe eines Elements, begrenzt relative Höhe durch absolute Werte. Auch möglich: min-height
max-width	Maximale Breite eines Elements, begrenzt relative Breite durch absolute Werte. Auch möglich: min-width
color	Farbe z.B. von Text
text-align	Linksbündig, Zentriert… `left|right|center|justify`
font-family	Schriftart. Durch Kommata getrennt sollten immer mehrere Schriftarten definiert werden, falls der Browser die Schrift nicht kennt, probiert er die nächste.
list-style-type	Aufzählungszeichen in einer Liste: `circle|square|none` …
display	Anzeigbarkeit eines Elements: `inline|block|flex|none` …
position	Position des Elements: `static|relative|fixed|absolute`
float	Lässt ein Objekt z.B. rechts innerhalb des übergeordneten Objekts anliegen `right|left|none|inherit`

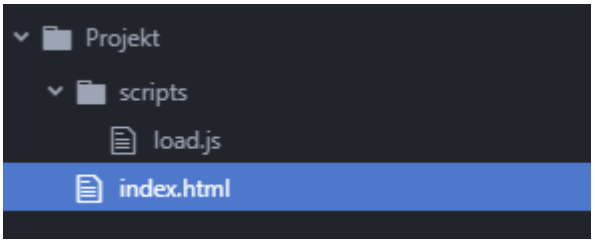
Projektstruktur

Es ist natürlich möglich, einfach alle Dateien in einen Ordner zu schmeißen und fertig ist. Jede Datei findet jede andere Datei durch ihren Dateinamen und alles funktioniert, aber das wird schnell sehr unübersichtlich und sollte vermieden werden.

Dateipfade

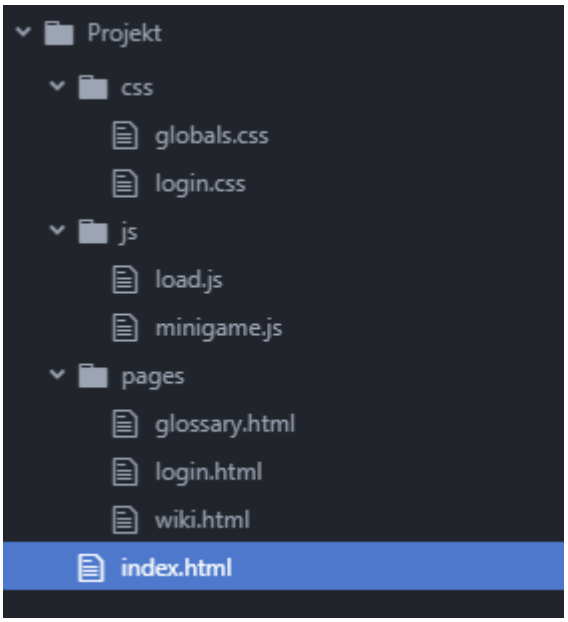
Um überhaupt Dateien in Ordnern verschachteln zu können, muss man wissen, wie man relative Dateipfadangaben verwendet. Natürlich könnte man überall „C:/Benutzer/Noob/Desktop/daj.html“ schreiben, aber spätestens wenn man die Website auf einen Server hochläd, oder wenn man die Website auf einem anderen Rechner starten möchte, kann der Browser mit dem Pfade nichts mehr anfangen. Die Lösung: Relative Dateipfade.

Relative Dateipfade gehen immer von der Datei aus, in der sie stehen. Beispiel:




```
<script src="scripts/load.js">
```

Dieses Konzept funktioniert prinzipiell immer. Ordername/Datei oder z.B. Ordner/Ordner/Datei etc. Das kann beliebig weitergeführt werden. Solange man die HTML Dateien alle im obersten Ordner behält, funktioniert das auch. Optimal ist dabei eine solche Struktur:



Wie man sieht, existiert eine minimale Anzahl an Ordnern, aber möglichst effektiv. Egal, in welchem der Ordner man sich befindet, es gibt immer ein paar wenige Dateien / Ordner und es ist übersichtlich, aber trotzdem vollkommen logisch.

Aber um so eine Struktur erreichen zu können, muss man z.B. aus der `wiki.html` Datei heraus auf `globals.css` zugreifen können, was mit den bisherigen Mitteln nicht möglich war. In diesem Fall ginge das wie folgt:

```
<link rel="stylesheet"
href="../css/globals.css">
```

Mithilfe von `../` kann man einen Ordner nach oben gehen in der Struktur. Dies kann ebenfalls mehrmals in einem Pfad verwendet werden, sodass eine saubere Ordnerstruktur möglich ist.

Best Practices

Die Krone auf all dem Wissen bisher, ist wie man es **optimal** einsetzt. Wenn der Code nicht nur funktioniert, sondern schön aussieht, übersichtlich ist und durch Kommentare auf Anhieb zu verstehen ist. Drei Mottos, die man im Hinterkopf behalten sollte stehen hier:

- **Desto weniger, desto besser**. Man sollte immer versuchen, so wenig Code und so wenig Dateien wie möglich zu generieren, denn weniger ist immer **übersichtlicher**. Und wenn etwas übersichtlicher ist, dann besteht weniger Buggefahr, es ist angenehmer zu lesen und man kann es einfacher bearbeiten.
- **Vererbung erspart Arbeit**. Es ist immer sinnvoll, wirklich jede Funktion, jedes Format, welche(s) an mehreren Stellen verwendet werden(/wird), extern an einer Stelle zu definieren und dann einfach mehrmals zu verwenden. So erreicht man ein stimmiges globales Design und man spart einiges an Code.
- **Eigene Strukturregeln**. Beispielsweise in HTML bietet es sich an, an bestimmten Stellen etwas Freiraum bzw. einen zusätzlichen Absatz zu setzen, damit zusammengehörende Objekte auch im Code so aussehen. Außerdem gilt wieder: Möglichst übersichtlich.

Ein interessanter Link zu diesem Thema kommt von Google.