

## **Web Engineering**

# **Cascading Style Sheets**

Adrian Herzog

*(basierend auf der Arbeit von Michael Faes, Michael Heinrichs & Prof. Dierk König)*

# CSS

**HTML** vermittelt primär **Bedeutung**, nicht Präsentation

trotzdem ein wenig  
Präsentation...

*Dr. Eleanor Gaye*  
Awesome Science faculty  
University of Awesome  
Bobtown, CA 99999,  
USA  
**Tel:** 123-456-7890  
**Email:** [no\\_reply@example.com](mailto:no_reply@example.com)

20 January 2016

*Miss Eileen Dover*  
4321 Cliff Top Edge  
Dover, CT9 XXX  
UK

## **Re: Eileen Dover university application**

Dear Eileen,

Thank you for your recent application to join us at the University of Awesome's science faculty to study as part of your PhD next year. I will answer your questions one by one, in the following sections.

### **Starting dates**

We are happy to accommodate you starting your study with us at any time, however it would suit us better if you could start at the beginning of a semester; the start dates for each one are as follows:

- First semester: 9 September 2016
- Second semester: 15 January 2017
- Third semester: 2 May 2017

Please let me know if this is ok, and if so which start date you would prefer.

You can find more information about [important university dates](#) on our website.

### **Subjects of study**

At the Awesome Science Faculty, we have a pretty open-minded research facility — as long as the subjects fall somewhere in the realm of science and technology. You seem like an intelligent, dedicated researcher, and just the kind of person we'd like to have on our team. Saying that, of the ideas you submitted we were most intrigued by are as follows, in order of priority:

mit CSS

**Dr. Eleanor Gaye**  
Awesome Science faculty  
University of Awesome  
Bobtown, CA 99999,  
USA  
**Tel:** 123-456-7890  
**Email:** [no\\_reply@example.com](mailto:no_reply@example.com)

20 January 2016

**Miss Eileen Dover**  
4321 Cliff Top Edge  
Dover, CT9 XXX  
UK

## **Re: Eileen Dover university application**

Dear Eileen,

Thank you for your recent application to join us at the University of Awesome's science faculty to study as part of your PhD next year. I will answer your questions one by one, in the following sections.

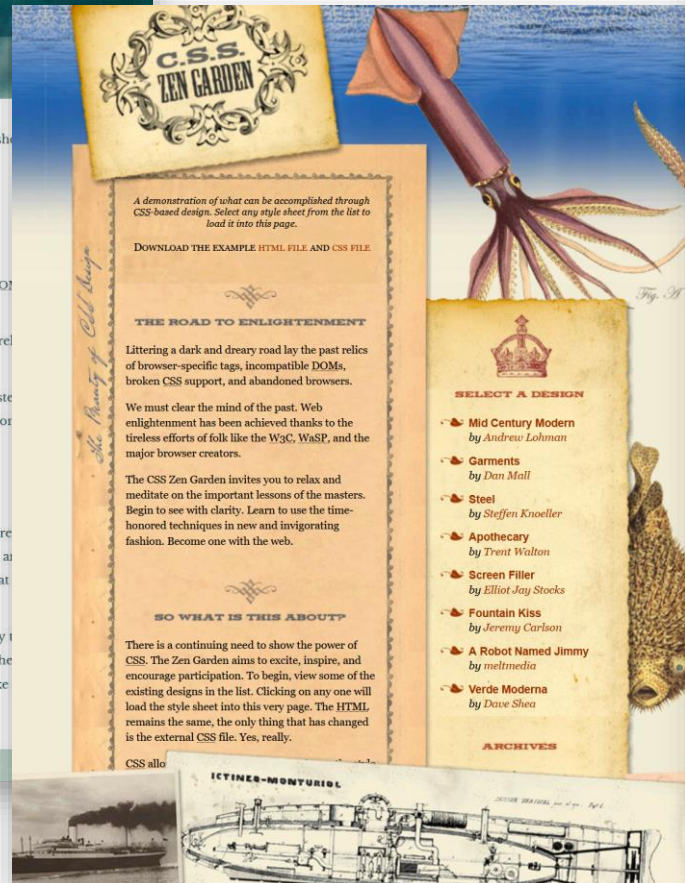
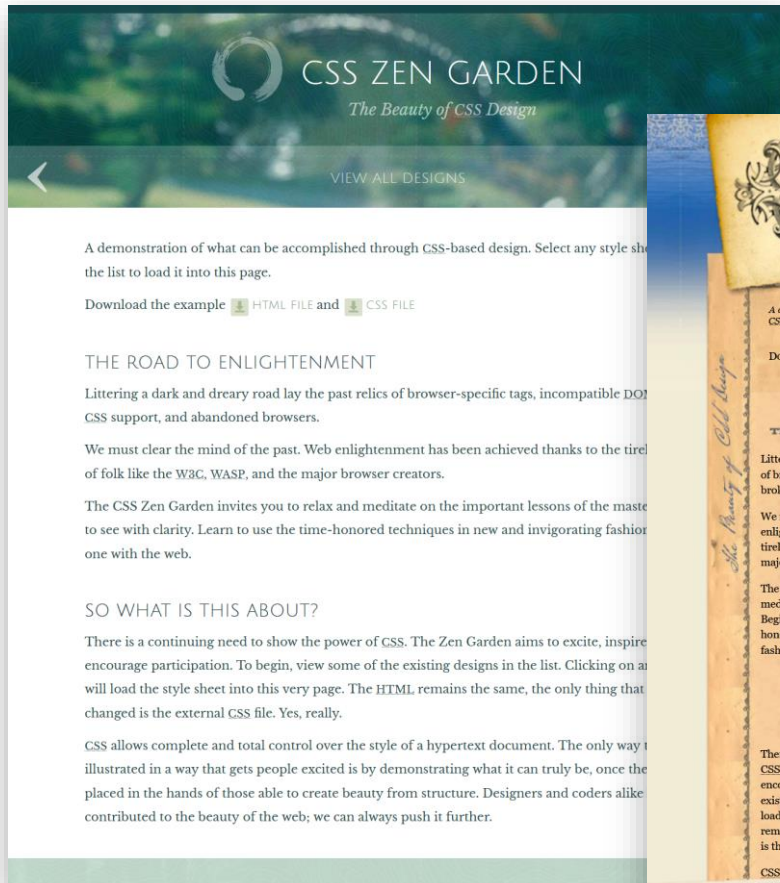
### **Starting dates**

We are happy to accommodate you starting your study with us at any time, however it would suit us better if you could start at the beginning of a semester; the start dates for each one are as follows:

- First semester: 9 September 2016
- Second semester: 15 January 2017
- Third semester: 2 May 2017

# Trennung Inhalt und Präsentation

Genau das gleiche HTML-Dokument, mit unterschiedlichem CSS:



[www.csszengarden.com](http://www.csszengarden.com)

# ...oder doch nicht?



"Tailwind CSS is the only framework that I've seen scale on large teams. It's easy to customize, adapts to any design, and the build size is tiny."

[Sarah Dayan](#)  
Staff Engineer, Algolia

```
1 <figure class="md:flex bg-slate-100 rounded-xl p-8 md:p-0 dark:bg-slate-900">
2   <img class="w-24 h-24 md:w-48 md:h-auto md:rounded-none rounded-full" src="" alt="Profile picture of Sarah Dayan" data-bbox="118 316 242 602"/>
3   <div class="pt-6 md:p-8 text-center md:text-left space-y-4">
4     <blockquote>
5       <p class="text-lg font-medium">
6         "Tailwind CSS is the only framework that I've seen scale
7         on large teams. It's easy to customize, adapts to any design,
8         and the build size is tiny."
9       </p>
10    </blockquote>
11    <figcaption class="font-medium">
12      <div class="text-sky-500 dark:text-sky-400">
13        Sarah Dayan
14      </div>
15      <div class="text-slate-700 dark:text-slate-500">
16        Staff Engineer, Algolia
17      </div>
18    </figcaption>
19  </div>
20 </figure>
21
```

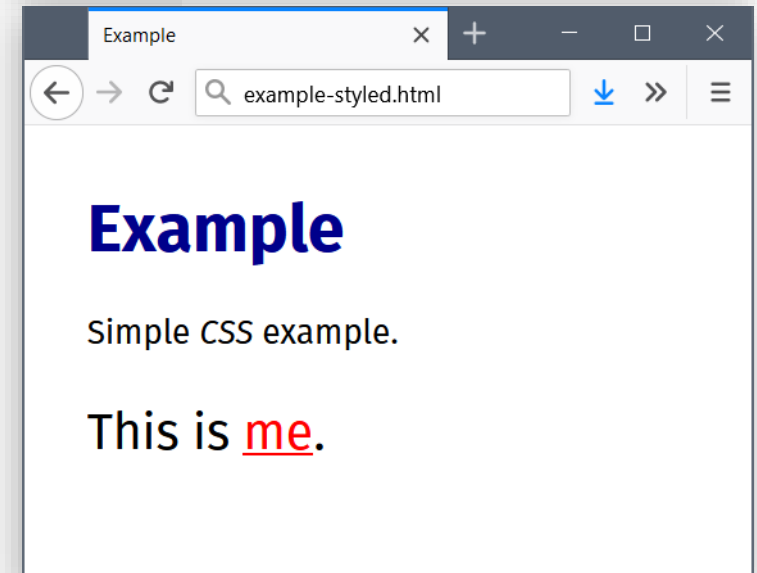
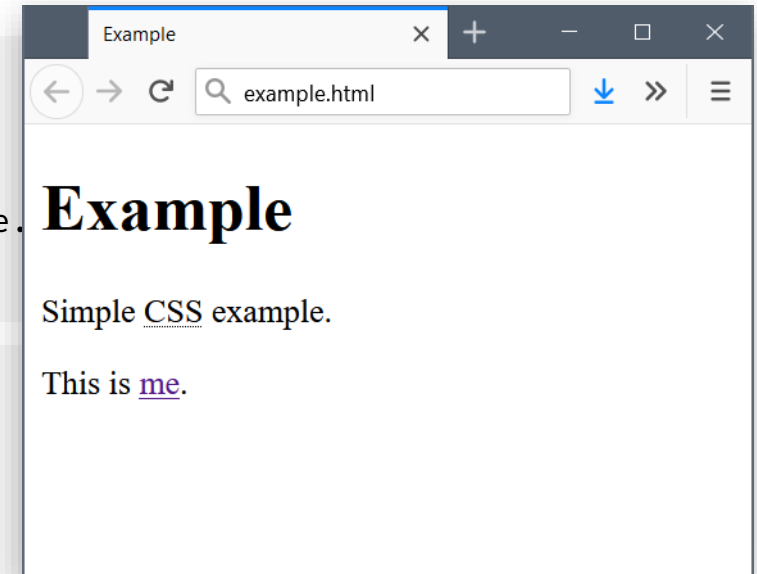


Mit Frameworks wie *Bootstrap* oder *Tailwind* definiert man Darstellung doch wieder im HTML-Code... Aber trotzdem basierend auf CSS!

# Mini-Beispiel

```
<!DOCTYPE html>
<title>Example</title>
<h1>Example</h1>
<p>Simple<abbr title="Cascading Style Sheets">CSS</abbr> example.
<p class="special">This is <a href="example.html">me</a>.
```

```
body {
  margin: 30px;
  font-family: Fira Sans, sans-serif;
}
h1 {
  color: darkblue;
}
abbr {
  font-style: italic;
  text-decoration: none;
}
a {
  color: red;
}
.special {
  font-size: 150%;
}
```



# Aufbau von CSS

CSS besteht aus *Regeln*:



**Selektor:** bestimmt, auf welche HTML-Elemente Regel zutrifft

**Deklarationen:** bestimmen Aussehen dieser Elemente

# CSS zu HTML hinzufügen

**Inline:** direkt auf einem HTML Tag

```
<h1 style="font-size: 36pt; color: red;">Example</h1>
```

**Intern:** im <head>-Element des HTML-Dokuments

```
<head>
  <style>
    h1 {
      font-size: 36pt;
      color: red;
    }
  </style>
</head>
```



# CSS zu HTML hinzufügen

**Extern:** in separater Datei

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

example.html

in diesem Fall im  
selben «Ordner»

```
h1 {
  font-size: 36pt;
  color: darkblue;
}
```

style.css



# Übung 1: Etwas CSS für den Letter

- a) Das Dokument «letter.html» enthält bereits CSS, direkt in einem `<style>`-Element. Verschieben Sie diese CSS-Regeln in eine separate Datei «letter.css» und verlinken Sie diese im HTML.
- b) Erweitern Sie das Stylesheet um einige Regeln:
- Der ganze Brief soll in einer *serifenlosen* Schriftart (`sans-serif`) dargestellt werden.
  - Die Überschriften im Brief sollen einen dezenten, nicht allzu gesättigten Blauton haben.
  - Die Links sollen einen hellen, gesättigten Blauton haben.
  - Begriffe, die definiert werden, sollen fett und kursiv sein.
  - Alles, was fett ist, soll dunkelblau sein (ausser Überschriften).

# CSS zu HTML hinzufügen

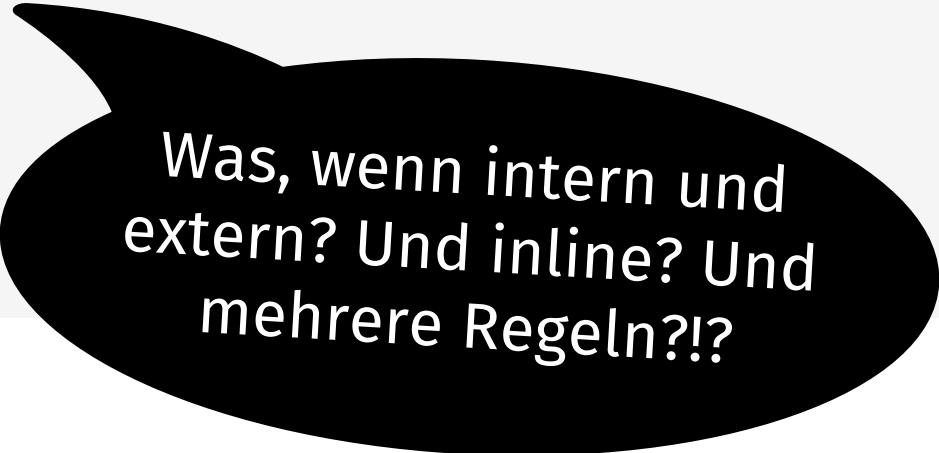
**Extern:** in separater Datei

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

example.html

```
h1 {
  font-size: 36pt;
  color: darkblue;
}
```

style.css



Was, wenn intern und  
extern? Und inline? Und  
mehrere Regeln?!?

# **Selektoren und Priorität**

# Selektoren

**Elementtyp:** gilt für alle Elemente des entsprechenden Typs

```
h1 {  
    color: red;  
}
```

```
<h1>Example</h1>
```

**ID:** gilt für alle Element mit einem bestimmten `id`-Attribut

```
#foo {  
    color: red;  
}
```

```
<h1 id="foo">Example</h1>
```

eindeutig innerhalb  
eines HTML-Dokuments!

**Klasse:** gilt für alle Elemente mit entsprechendem `class`-Attribut

```
.special {  
    color: red;  
}
```

```
<h1 class="special">Example</h1>
```

Ein Element kann mehrere Klassen haben:

```
.special {  
    color: red;  
}  
.spaced {  
    line-height: 1.5;  
}
```

```
<p class="special spaced">  
    Example  
</p>
```

**Pseudoklasse:** gilt für Elemente in einem besonderen Zustand...

```
a: hover {  
  color: red;  
}
```

```
<a href="...">Link</a>
```

Link

Link



... oder in/mit bestimmter Dokumentstruktur:

```
p: first-child {  
  color: red;  
}
```

```
<article>
```

```
  <p>First paragraph</p>
```



```
  <p>Second paragraph</p>
```



```
</article>
```

First paragraph

Second paragraph

Weitere: :last-child, :nth-child(...), ...

**Attribut:** gilt für Elemente mit einem bestimmten Attribut

```
[title] {  
    color: red;  
}
```

```
<h1 title="Foo">Example</h1>
```

```
[title="Bar"] {  
    color: red;  
}
```

```
<h1 title="Bar">Example</h1>
```



# Selektoren kombinieren

Elementtyp, Klasse, Pseudoklasse und Attribut kann man kombinieren:

```
h1.special {  
    color: red;  
}
```

```
<h1 class="special">Example</h1>
```

**beide** Selektoren  
müssen gelten

```
a.special:hover {  
    color: red;  
}
```

```
<a class="special" href="...">Me</a>
```

Me

Me



```
h1[title="Foo"] {  
    color: red;  
}
```

```
<h1 title="Foo">Example</a>
```

# Kombinatoren

Zusätzlich *Kombinatoren*, um Auswahl weiter einzuschränken.

**Nachkomme** (*descendant*): für Elemente innerhalb anderer Elemente



```
p em {  
    color: red;  
}
```

```
<p>  
    <em>Very</em> important.  
    <span>Or <em>not</em>?</span>  
</p>
```

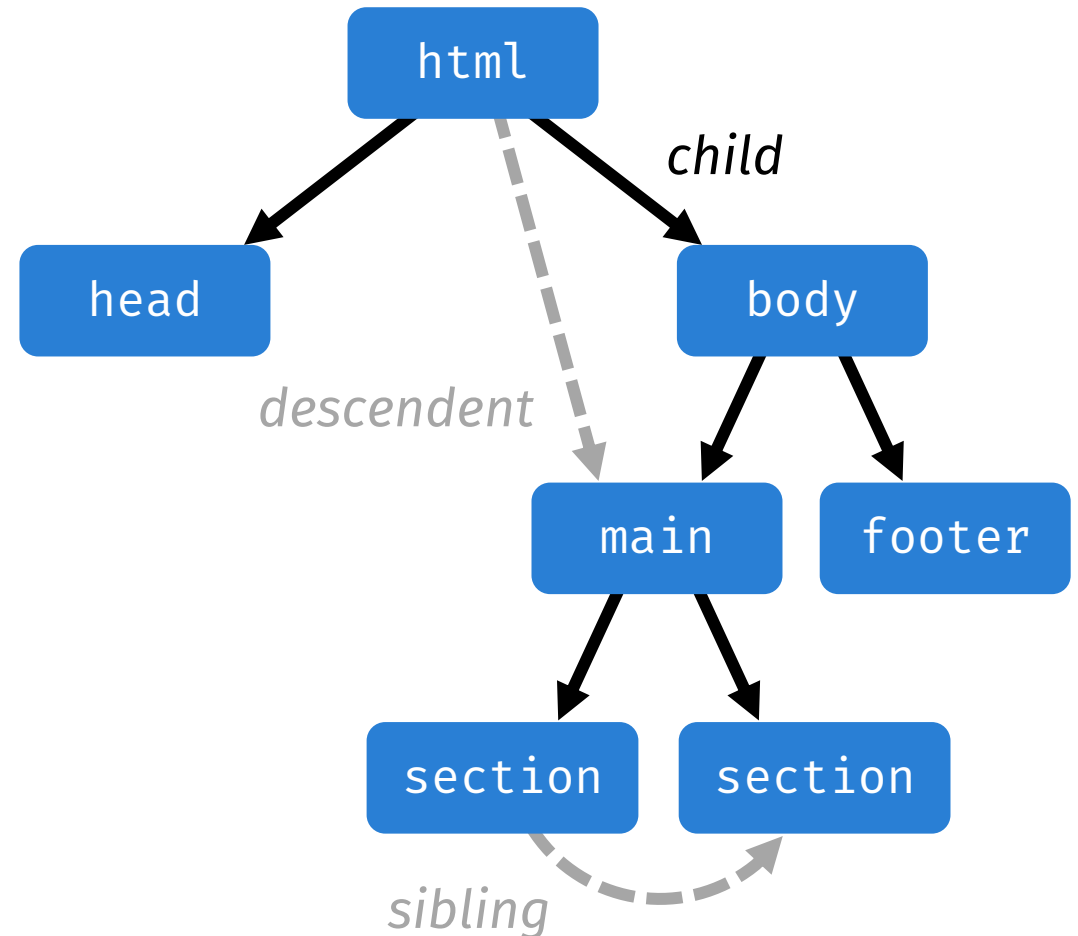
**Kind** (*child*): für Elemente, die *direkt* innerhalb anderer Elemente sind

```
p > em {  
    color: green;  
}
```

```
<p>  
    ✓ <em>Very</em> important.  
    <span>Or <em>not</em>?</span>  
    ✗ </p>
```

# Rückblick: DOM

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
  </head>
  <body>
    <main>
      <section>...</section>
      <section>...</section>
    </main>
    <footer>...</footer>
  </body>
</html>
```



*descendent, child, usw.* beziehen sich auf **(Stamm-)Baum!**

## Nächstes Geschwister (*next sibling*): Elemente neben einem anderen

```
h1 + p {  
    color: green;  
}
```

```
<h1>Heading</h1>  
<p>First paragraph</p> ✓  
<p>Second paragraph</p> ✗
```

## Mehrere Selektoren (*grouping*):

```
h1, p {  
    color: green;  
}
```

```
<h1>Heading</h1>  
<p>Paragraph</p>
```

**mindestens ein**  
Selektor muss gelten

# Priorität

Wenn mehrere Regeln zutreffen, was gilt?

```
<!DOCTYPE html>
<head>
  <title>Example</title>
  <style>
    p    { color: red;   }
    #foo { color: green; }
    .foo { color: blue;  }
  </style>
</head>
<p>One</p>
<p class="foo">Two</p>
<p id="foo" class="foo">Three</p>
<p style="color: purple;">Four</p>
<p class="foo" style="color: purple;">Five</p>
```



# Prioritäts-Tabelle

Priorität	Merkmal	Beschreibung
1	Wichtigkeit	Mit <b>!important</b> kann alles überschrieben werden <i>Sollte aber möglichst nie bzw. höchstens als Notlösung verwendet werden!</i>
2	Inline	Inline-Regeln überschreiben andere Regeln
3	Selektoren-Spezifität	Verschiedene Arten von Selektoren haben unterschiedliche Priorität
4	Reihenfolge	Spätere Sheets / Regeln überschreiben frühere
5	Vererbung	Gewisse undefinierte Eigenschaften werden von Eltern-Elementen geerbt

wieder  
Stammbaum-  
Analogie

# Selektoren-Spezifität

Wichtiger als Reihenfolge der Regeln ist die *Spezifität des Selektors*.

A

Anzahl #IDs

B

Anzahl .classes, [attr], :pseudo

C

Anzahl Typen

Beispiele:

Selektor	Spezifität		
p	0	0	1
main p	0	0	2
.foo	0	1	0
p.foo	0	1	1
p.foo a[alt]	0	2	2
#baz	1	0	0

niedrigere Priorität



höhere Priorität



# Übung 2: Priorität & Spezifität

Was gilt denn jetzt?

```
<!DOCTYPE html>
<head>
  <title>Example</title>
  <style>
    p    { color: red;    }
    #foo { color: green;  }
    .foo { color: blue;   }
  </style>
</head>
<p>One</p>
<p class="foo">Two</p>
<p id="foo" class="foo">Three</p>
<p style="color: purple;">Four</p>
<p class="foo" style="color: purple;">Five</p>
```

One

Two

Three

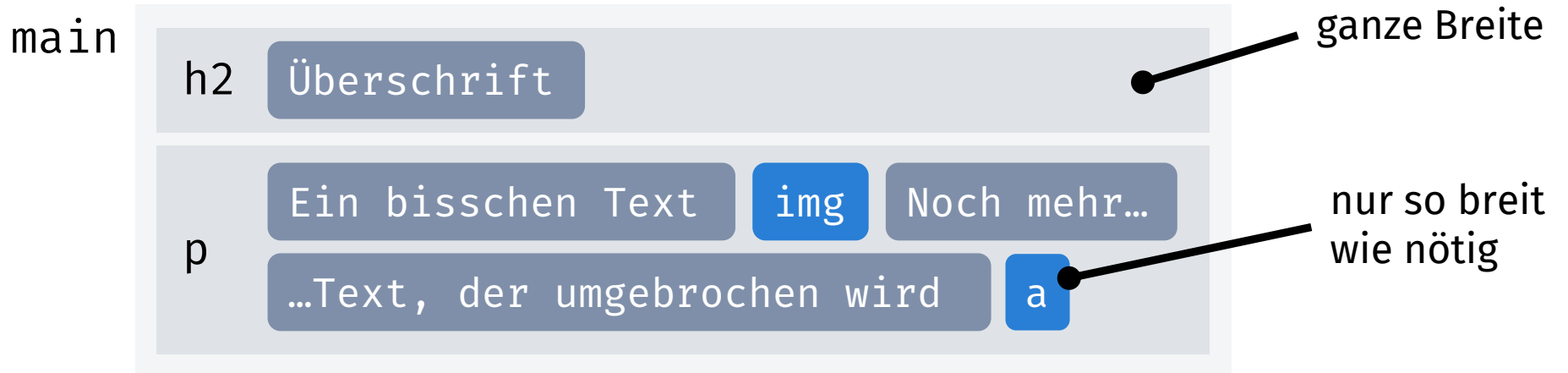
Four

Five

# Layout mit CSS

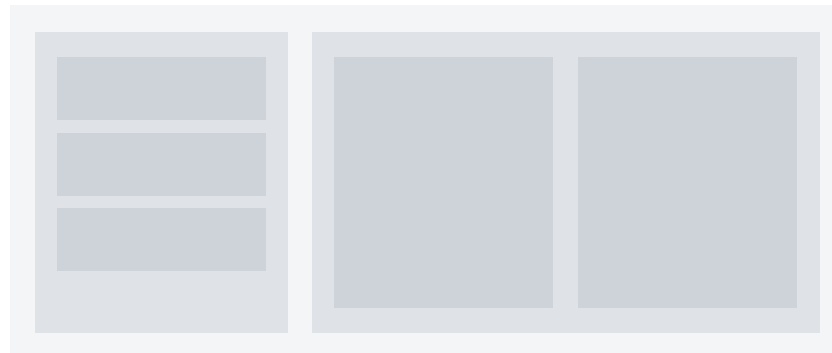
# Box Model

**Rückblick:** *Block-* und *Inline-Elemente* in HTML.

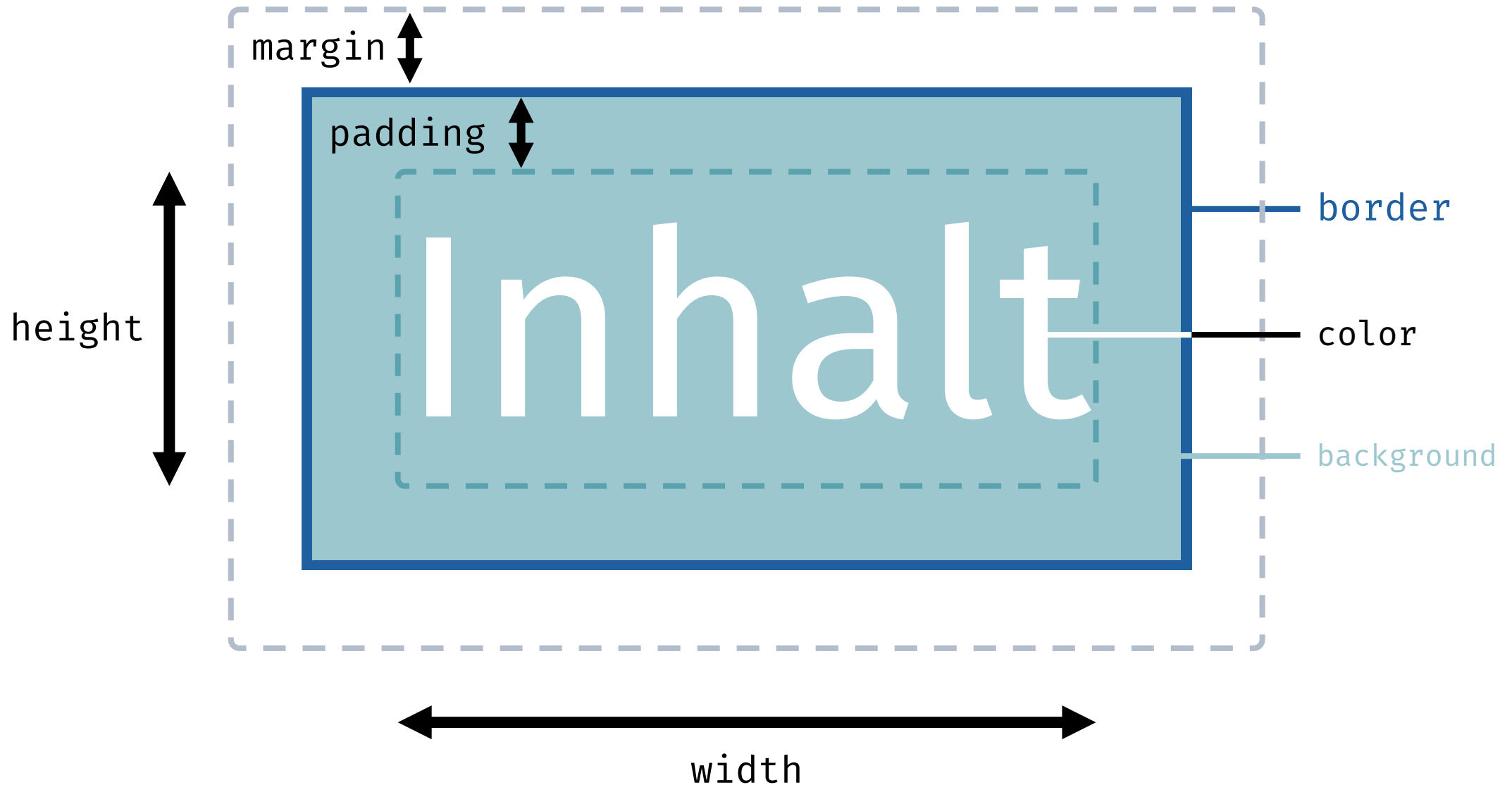


Egal, ob Block oder Inline, jedes Element wird als *Box* angezeigt.

Layout mit CSS: **Grösse und Anordnung** der Boxen ändern.



Die Box eines Elements besteht aus mehreren Teilen:



## Beispiel:

Hello, World!

```
<p class="warning">  
  <strong>Warning:</strong>  
  The server has experienced...  
</p>
```

More Text

```
.warning {  
  margin: 40px;  
  padding: 20px 30px;  
  background-color: #fff6e7;  
  border: solid 2px orange;  
  color: darkorange;  
}
```

Hello, World!

**Warning:** The server has experienced an unexpected shutdown.  
Please check the log files for more information.

More Text

# Display-Typ: `block` & `inline`

«Block» und «Inline» sind eigentlich CSS-Begriffe. Zwei mögliche *Display-Typen*; definieren, wie die Box eines Elements gelayoutet wird.

Display-Typ von HTML-Elementen ist per Default durch Elementtyp definiert. Beispiele: `<p>` sind Block, `<strong>` sind Inline.

Display-Typ kann durch CSS-Eigenschaft `display` geändert werden.

## Beispiel:

```
/* Rest wie vorher */  
  
.warning strong {  
    display: block;  
    padding-bottom: 5px;  
}
```

Hello, World!

### Warning:

The server has experienced an unexpected shutdown. Please check the log files for more information.

More Text

Display-Typ beeinflusst Layout auf verschiedene Arten:

`display: block;`

- Box landet auf neuer Zeile (und folgendes Element ebenfalls)
- Box füllt **ganze Breite**
- Grösse kann durch Eigenschaften `width` & `height` geändert werden
- `padding`, `border` & `margin` schieben andere Elemente von Box weg

`display: inline;`

- Box landet **nicht** auf neuer Zeile
- Box ist so breit wie durch Inhalt vorgegeben
- Eigenschaften `width` & `height` werden **ignoriert**
- `padding`, `border` & `margin` schieben andere Inline-Elemente **nur in horizontaler Richtung** von Box weg



# Innerer Display-Typ

`block` und `inline` definieren nur, wie Element sich «gegen aussen» verhält, d.h. wie es im Bezug auf Nachbarn gelayoutet wird.

*Innerer Display-Typ* definiert das Layout der *Inhalte des Elements*. Wird an «äusseren» Display-Typ in `display`-Eigenschaft angehängt.

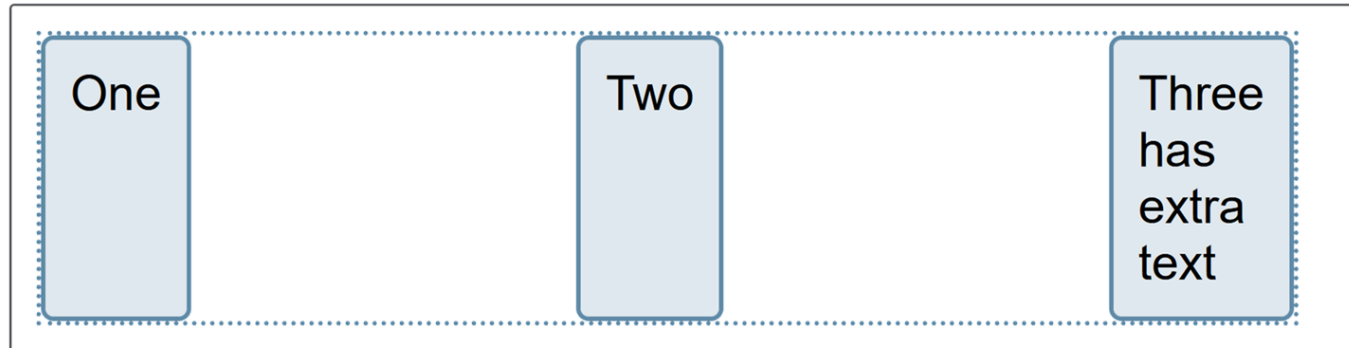
## Mögliche Werte

(`flow`): Normales «fliessendes» Layout (von oben nach unten, bzw. von links nach rechts). Default, normalerweise weggelassen.

`flex`: Eindimensionales Layout in beliebige Richtung

`grid`: Zweidimensionales Raster-Layout. Elemente können explizit oder automatisch platziert werden.

# Beispiel flex-Layout



```
.box {  
  display: flex;  
  justify-content: space-between;  
}
```

```
<div class="box">  
  <div>One</div>  
  <div>Two</div>  
  <div>Three  
    <br>has  
    <br>extra  
    <br>text  
  </div>  
</div>
```

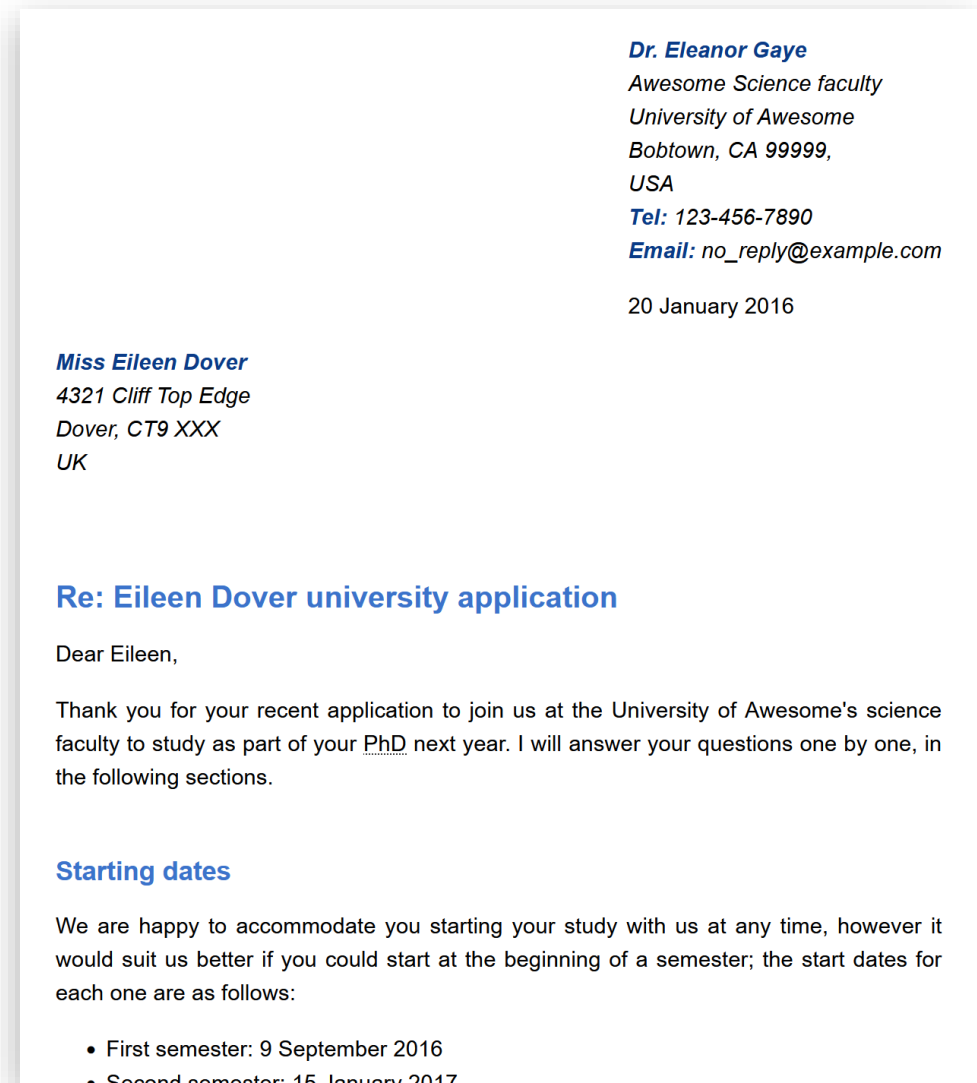
demo time!

[https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Flexible\\_Box\\_Layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout)  
or <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

# Übung 3: Letter fertig stylen

Erweitern Sie das Stylesheet für den Letter erneut, sodass folgende Darstellung erreicht wird:

- Adressat links-bündig, aber eingerückt
- Mehr Abstand zwischen Abschnitten (oberhalb von Überschriften)
- Platz für Unterschrift
- Blocksatz



# **Engineering-Aspekte**

# Organisation

Anzahl CSS-Regeln wächst schnell auf Dutzende/Hunderte. Wie organisieren?

1. Stylesheet in Abschnitte unterteilen, mit Leerzeilen und Kommentaren
2. Regeln thematisch auf mehrere Stylesheets aufteilen
3. Konsistente Klassennamen und IDs
  - Typischerweise wird `kebab-case` verwendet
  - Namenskonvention verwenden, z. B. *BEM*: [getbem.com/naming/](https://getbem.com/naming/)

```
/* Typography */  
  
h1, h2, h3 {  
  font-family: Arial, sans-serif;  
  line-height: 1.3;  
}  
  
/* Colors */  
  
h1, a, strong {  
  color: peachpuff;  
}
```

# CSS-Variablen

Um Duplizierung von Werten wie Farben, Schriftarten, -größen, usw. zu vermeiden:

1. Selektoren gruppieren:

```
h1, a, strong {  
  color: peachpuff;  
}
```

2. CSS-Variablen verwenden:

Variablen werden geerbt und können auch überschrieben werden

(`:root` entspricht eigentl. `html`-Element)

```
:root {  
  --highlight-color: peachpuff;  
}  
  
h1, a, strong {  
  color: var(--highlight-color);  
}
```

«custom property»

```
.button-highlight {  
  color: var(--highlight-color);  
}
```

# CSS Nesting

Seit neuem in den wichtigsten Browsern unterstützt:

<https://caniuse.com/css-nesting>

```
.notice {  
  width: 90%;  
  &.warning {  
    /* equivalent to `.notice.warning` */  
    background-color: #d81b60;  
    border-color: #d81b60;  
    color: white;  
  }  
  &.success {  
    /* equivalent to `.notice.success` */  
    background-color: #004d40;  
    border-color: #004d40;  
    color: white;  
  }  
}
```



# Sinnvolle Selektoren

Für jedes Element in einem HTML-Dokument gibt es diverse «passende» Selektoren. Wann welche verwenden?

**Grundidee:** Selektor soll künftige Änderungen (an HTML oder CSS) möglichst einfach machen.

1. Selektor möglichst nahe an Design-Regel anlehnen:

- «Buttons, die *‘gefährlich’* sind, sind rot»

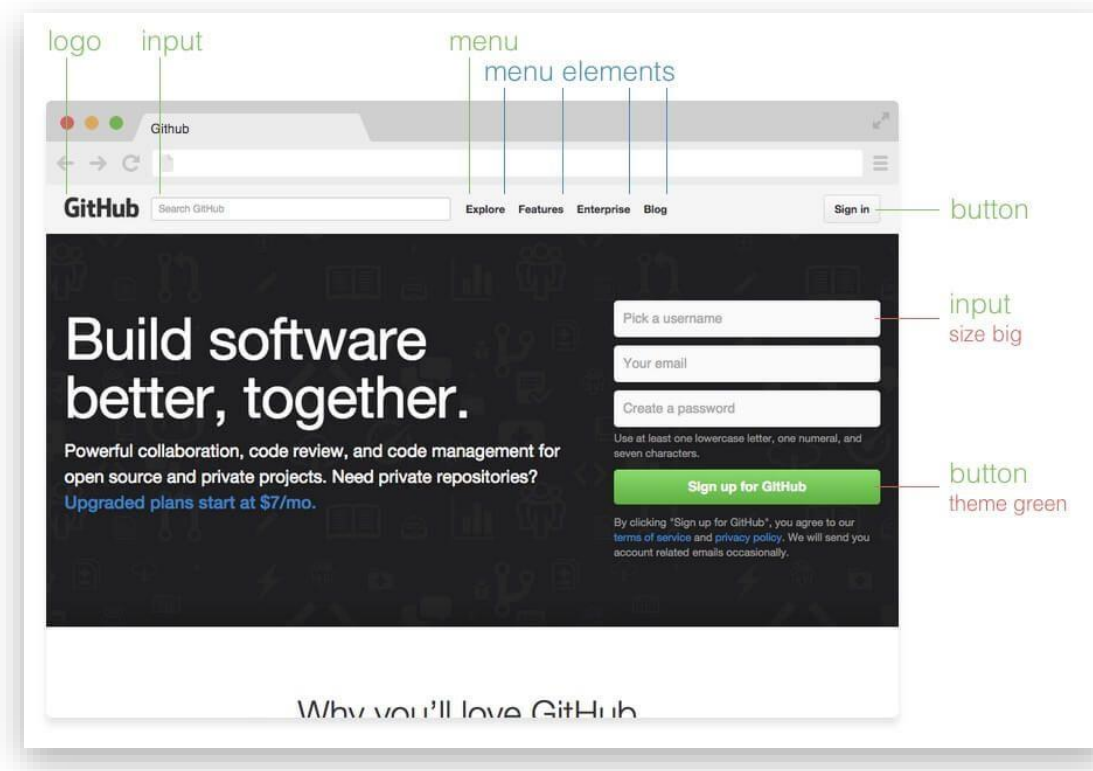
```
.button-dangerous { ... }
```

- «Erster Absatz von jedem Artikel ist fett»

```
article > p:first-of-type { ... }
```

2. Für grosse, komplexe Web-Applikationen: HTML-Struktur und CSS-Selektoren anhand einer *Methodologie* entwerfen.

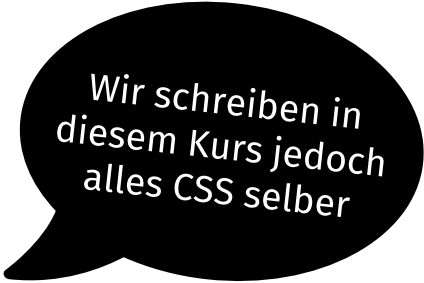
**Beispiel:** *BEM (Blocks, Elements, Modifiers)*



<https://getbem.com/introduction/>

# CSS / Component Frameworks

- Bootstrap
  - “Erstes”, sehr populäres Framework, seit 2011
  - Grundgerüst (Header, Menus, etc.)
  - Styling (austauschbar mit Themes)
  - Wiederverwendbare Komponenten
- Tailwind CSS
  - “utility-first CSS Framework”
- Sehr viele mehr
  - Jedes Framework hat seine Stärken und Schwächen
  - Evaluation anhand vom Einsatzkontext ist nötig



Wir schreiben in  
diesem Kurs jedoch  
alles CSS selber

# Details

Eigenschaften, Einheiten, Funktionen, Transitions, ...

→ Zu viele Details, um alles zu erklären

→ *Learning by doing* ist angesagt



Übungen

## Ressourcen:

MDN CSS Referenz

<https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

MDN Flexbox Guide

[https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\\_layout/Flexbox](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox)

MDN Grid Guide

[https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\\_layout/Grids](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids)

# Fragen?

