

# CSS

---

**Dozent:** Prof. Dr. Michael Eichberg  
**Kontakt:** [michael.eichberg@dhbw-mannheim.de](mailto:michael.eichberg@dhbw-mannheim.de), Raum 149B  
**Version:** 2024-06-09



1

---

**Folien:** <https://delors.github.io/web-css/folien.de.rst.html>  
<https://delors.github.io/web-css/folien.de.rst.html.pdf>

**Fehler auf Folien melden:**  
<https://github.com/Delors/delors.github.io/issues>

# 1. EINFÜHRUNG

Prof. Dr. Michael Eichberg

# Hintergrund

CSS (Cascading Style Sheets) ist eine Stylesheet-Sprache, die verwendet wird, um das Aussehen von Dokumenten zu gestalten.

1

## Historie

- Entwicklung begann 1994; CSS 1 wurde 1996 veröffentlicht und war erst einmal ein Fehlschlag
- CSS 2 wurde 1998 veröffentlicht
- CSS 3 wurde modularisiert, um die Entwicklung zu beschleunigen
  - CSS Color Level 3 (2012)
  - CSS Namespaces Level 3 (2012)
  - CSS Selectors Level 3 (2012)
  - ...
  - CSS Flexbox Level 1 (2018) (nach 9 Jahren Entwicklungszeit)
  - CSS Selectors Level 4 (2024 noch Draft Status; insbesondere `:has()` hat breite Unterstützung)
- CSS Nesting (2024 noch Draft Status; dennoch bereits seit 2024 weit verfügbar)

2

3

# Grundlagen

Eine CSS-Datei besteht aus Regeln, die aus einem Selektor und einer oder mehreren Deklarationen bestehen:



## CSS

```
h1 {  
  color: blue;  
  font-size: larger;  
}  
body { /* the boss said so... */  
  background-color :  
    lightblue;  
}
```

## Resultat

### Überschrift

Paragraph in sehr wichtig!.

CSS ist im wesentlichen Whitespace insensitive, d.h., Leerzeichen, Zeilenumbrüche und Tabulatoren werden ignoriert.

Kommentare werden in `/* ... */` geschrieben.

# Elemente

- Wir unterscheiden zwischen *replaced elements* bei denen der Inhalt nicht Teil des Dokumentes ist (zum Beispiel `<img>`) und *non-replaced elements* (zum Beispiel `<p>` und `<div>`; d. h. die meisten HTML Elemente).
- Grundlegende Formatierungskontexte<sup>[1]</sup>: *block* (z. B. der Standard von `h1`, `p`, `div`, ...) und *inline* (z. B. der Standard von `strong`, `span`, ...).
- Block-Elemente generieren eine Box, welche den Inhaltsbereich des *Parent-Elements* ausfüllt.  
(*Replaced elements* können, müssen aber nicht Block-Elemente sein.)
- Inline-Elemente generieren eine Box innerhalb einer Zeile und unterbrechen den Fluss der Zeile nicht.
- Mittels CSS kann der Formatierungskontext geändert werden.

<sup>[1]</sup> Es gibt noch „viel mehr“ Kontexte für spezielle Anwendungsfälle.

# Block und Inline Elemente - Beispiel

```
h1 {  
  display: inline;  
}  
strong {  
  display: block;  
}
```

Folgendes Beispiel dient nur der Veranschaulichung:

```
Dies ist eine <strong><h1>Überschrift</h1>  
in sehr wichtig</strong>; wirklich!
```

## Warnung

(So nicht verwenden; dies ist kein gültiges HTML5!)

## Visualisierung

Dies ist eine

**Überschrift** in sehr wichtig  
; wirklich!

# Verknüpfung von CSS und HTML

- Inline CSS: `<p style="color: red;">`
- Externe CSS-Datei:
  - über Link: `<link rel="stylesheet" media="screen, print" href="style.css">`  
(Normalerweise im `<head>` deklariert.)
  - mittels `import` Direktive<sup>[2]</sup>: `<style>@import url(style2.css);</style>`
- im `<style>` Element: `<style> h1 { color: blue; } </style>`  
(Normalerweise im `<head>` deklariert.)
- Das Verwenden beliebig vieler CSS-Dateien und `style` Elemente ist möglich.

<sup>[2]</sup> `@import`

## 2. SELEKTOREN

Prof. Dr. Michael Eichberg



# Übersicht über Selektoren

<b>Typ:</b>	Selektoren basierend auf dem Typ des auszuwählenden Elements (z. B. <code>h1</code> , <code>div</code> , <code>span</code> , ...); meistens von HTML Elementen.
<b>IDs:</b>	Selektoren basierend auf den Werten der (einmaligen) <code>id</code> Attribute (z. B. <code>#core</code> , <code>#impressum</code> , ...).
<b>Klassen:</b>	Selektoren, die auf den Werten der <code>class</code> Attribute basieren (z. B. <code>.important</code> , <code>.highlight</code> , ...).
<b>Attributwerte:</b>	Selektoren, die auf einem Attribut bzw. dem Wert eines Attributs als solches basieren (z. B. <code>[href]</code> , <code>[type='text']</code> , ...).
<b>Pseudoklassen:</b>	Selektoren in Hinblick auf den Zustand eines Elements (z. B. <code>:hover</code> , <code>:active</code> , ...).
<b>Pseudoelemente:</b>	Selektoren eines Teils eines Elements (z. B. <code>::first-line</code> , <code>::first-letter</code> , ...).
<b>Gruppierung:</b>	Gruppierungen von durch Kommas getrennten Selektoren für die die selben Regeln angewandt werden sollen (z. B. <code>h1, h2, h3 { ... }</code> ).
<b>Kombinatoren:</b>	Selektoren, die auf der Beziehung zwischen zwei Elementen basieren (z. B. <code>div p { ... }</code> ).

# Klassen ( *class-Selector*) - Beispiel[3]

## HTML

```
<h1>Die Bedeutung des Seins.</h1>
<h1 class="wip">
  Die Bedeutung des Nicht-Seins
</h1>
<h1 class="todo future">
  Das Sein und das Nicht-Sein
</h1>
```

## CSS

```
h1 { color: black; }
h1.wip { color: green; }
*.todo { color: red; }
.future { text-decoration: underline;}
```

## Resultat

**Die Bedeutung des Seins.**

**Die Bedeutung des Nicht-Seins**

**Die Bedeutung des Nicht-Seins**

[3] ID basierte Selektoren funktionieren vergleichbar, jedoch wird ein **#** anstatt eines **.** verwendet. (In CSS müssen IDs nicht eindeutig sein; dies ist aber eine Verletzung von HTML und eindeutige IDs sind eine *Best Practices*.)

## Attribute (🇩🇪 *Attribute-Selector*) [4]

- basierend auf der Existenz eines Attributs: `h1[lang] { color: red; }`
- basierend auf dem *exakten* Wert eines Attributs: `h1[lang="de-DE"] { color: red; }`
- basierend auf einem partiellen Match:
  - enthält als eigenständiges **de**: `h1[lang~="de"] { color: red; }`
  - beginnt mit **de**: `h1[lang^="de"] { color: red; }`
  - substring **de**: `h1[lang*="de"] { color: red; }`
  - endet mit **de**: `h1[lang$="de"] { color: red; }`
  - beginnt mit **de** und wird dann gefolgt von einem Bindestrich oder steht alleine:  
`h1[lang|="de"] { color: red; }`
- durch ein **i** am Ende wird der **Selektor für den Wert** *case-insensitive*:  
`h1[lang="de-de" i] { color: red; }`

[4] Im Allgemeinen sind Attribut-basierte Selektoren vergleichsweise fragil und werden deswegen nur spärlich eingesetzt. Im Zusammenhang mit **data-\*** Attributen ist dies jedoch eine sehr mächtige Technik.

# Attribute ( *Attribute-Selector*) - Beispiel

## HTML

```
<h1 lang="de-DE">Die Bedeutung des Seins.</h1>  
<h1 lang="en-US">To Be Or Not To Be</h1>  
<h1 lang="de-AT">Ich brauch ne Jause</h1>
```

## CSS

```
[lang] { text-decoration: underline; }  
[lang|"de"] { font-size: larger; }  
[lang="de-at" i] { text-transform: uppercase; }
```

## Resultat

Die Bedeutung des Seins.

To Be Or Not To Be

ICH BRAUCH NE JAUSE

# Kombinatoren

- Nachfahren (bzgl. Dokumentenstruktur) (🇺🇸 *Descendant Selector*):

**div p:** alle <p> Nachfahren von <div> Elementen

**.important[lang='de-de' i] p:**

alle <p> Nachfahren von **.important** Elementen, die ein **lang** Attribut mit dem Wert **de-DE** haben.

- Alle direkten Kinder (🇺🇸 *Child Selector*):

**div > p:** alle <p> Kinder von <div> Elementen.

- Benachbarte Geschwister (🇺🇸 *Adjacent Sibling Selector*):

**h1 + p:** alle <p> Elemente, die *direkt* auf ein <h1> Element folgen und sich das Gleiche Eltern-Element teilen.

- Allgemeiner Geschwister Selektor (🇺🇸 *General Sibling Selector*):

**h1 ~ p:** alle <p> Elemente, die auf ein <h1> Element folgen und sich das Gleiche Eltern-Element teilen.

# Kombinatoren - Beispiele

## HTML

```
<h1>Ü1</h1>
```

Text

```
<p>P1</p>
```

```
<p>P2</p>
```

```
<p>P3</p>
```

```
<h1>Ü2</h1>
```

```
<div>
```

D1

```
<div>D1.1</div>
```

```
<div>D1.2</div>
```

```
</div>
```

```
<div>D2</div>
```

```
<div>D3</div>
```

## Spielwiese

```
/* h1 + p { color: blue; } */  
/* p + p { color: red; } */  
/* h1 ~ p { color: green; } */  
/* div ~ div { color: yellow; } */  
/* div + div { color: purple; } */  
/* h1 ~ div { color: orange; } */
```

# Ü1

Text

P1

P2

P3

# Ü2

D1

D1.1

D1.2

D2

D3

# Pseudo-class Selektors

- erlauben das Selektieren von Elementen basierend auf ihrem Zustand
- können beliebig kombiniert werden: `a:link:hover { color: red; }`  
selektiert alle Links, die noch nicht besucht wurden und über denen sich die Maus befindet
- Ausgewählte Beispiele:
  - Bzgl. der Struktur: `:first-child`, `:last-child`, `:nth-child(n)`, `:nth-of-type(n)`, `:root`, `:only-child`, `:only-of-type`, `:link`, `:visited`
  - Basierend auf Nutzerinteraktionen: `:hover`, `:active`, `:focus`
  - Zustand des Elements: `:enabled`, `:disabled`, `:checked`, `:valid`, `:invalid`
  - Sprache und Lokalisierung: `:lang(de)`, `:dir(ltr)`
  - Logische Selektoren: `:not(selector)`, `:is(selector)`, `:where(selector)`, `:has(selector)`
- Pseudo-class Selektoren beziehen sich immer auf das Element auf das sie sich

beziehen.

## HTML

```
<div class="oma" id="Maria">
  <div class="papa" id="Fritz">
    <div class="kind" id="Elias">
      Kind 1
    </div>
  </div>
  <div class="papa" id="Hans">
    <div class="kind" id="Tobias">
      Kind 2
    </div>
  </div>
</div>
```

## CSS

```
.papa:first-child { color: red; }
```

Selektiert welches Element?

Selektiert wird ein Element mit der Klasse **papa**, wenn es das erste Kind seines Eltern-Elements ist. Es wird *nicht das erste Kind des Elements* selektiert.

- Bei `:nth-child(n)` und `:nth-of-type` ist  $n$  eine Zahl oder ein Ausdruck ( $a \cdot n + b$ ), der eine Zahl ergibt (z. B.  $2n+1$  oder aber `even`). Das Zählen der Elemente beginnt bei 1.
- `:root` selektiert das Wurzelement des Dokuments, also das `<html>` Element.
- `:only-child` und `:only-of-type` selektiert ein Element, das das einzige entsprechende Kind seines Eltern-Elements ist.





# Pseudo-class Selektors bzgl. Inputvalidierung

## HTML

```
<input type="email"
        placeholder="your email"
        required>
<input type="email"
        placeholder="your friend's email">
```

## Spielwiese

```
input[type='email']:valid {
    color: green;
    border: 2px solid green;
}
/*input[type='email']:invalid {
    color: red;
    border: 2px solid red;
}*/
```

your email

your friend`s email

Da das zweite Eingabefeld nicht als **required** markiert ist, wird es auch dann als **:valid** betrachtet, wenn es leer ist.

# Spezifizität von Selektoren

- Die Spezifität eines Selektors bestimmt, welcher Stil auf ein Element angewendet wird, wenn mehrere Regeln auf ein Element zutreffen und diese bzgl. der gleichen Eigenschaften in Konflikt stehen.

Die Spezifität wird durch einen Vektor (**a**, **b**, **c**) dargestellt:

- **a**: Anzahl der ID Selektoren
- **b**: Anzahl der Klassen-, Attribut- und Pseudo-Klassen Selektoren
- **c**: Anzahl der Element- und Pseudo-Element Selektoren

Die Spezifität wird in der Reihenfolge **a**, **b**, **c** verglichen.

- Konzeptionell wird die Spezifität pro Deklaration betrachtet.

1

- Beispiele:

Selektor	Spezifizität
p { color: black; }	0, 0, 1
section p { color: orange; }	0, 0, 2
section > p { color: orange; }	0, 0, 2
p.warning { color: red; }	0, 1, 1
p[id*='this'] {color: green; }	0, 1, 1
#main { color: yellow; }	1, 0, 0
* { color: yellow !important; }	0, 0, 0 (Important)

2

## HTML

```
<section>
  <p id='this-is-it'>
    Der erste Abschnitt!
  </p>
  <p class='obsolete'>
    Ein alter Abschnitt.
  </p>
</section>
<p>Der letzte Abschnitt.</p>
```

## Spielwiese

```
/*p[id*='this'] {color: green; }*/
/*section p { color: red; }*/
/*p { color: orange; }*/
```

Der erste Abschnitt!  
Ein alter Abschnitt.  
Der letzte Abschnitt.

- Kombinatoren haben keine Spezifität.
- \* hat die Spezifität (0,0,0)
- eine Deklaration mit **!important** hat eine höhere Spezifität als jede Deklaration ohne **!important**. Alle als **!important** markierten Deklarationen werden nach den beschriebenen Regeln ausgewertet.

# Nesting

## HTML

```
<h1 class="obsolete">1. Überschrift</h1>
  <p>Ein alter Absatz</p>
<h2>2. Überschrift</h2>
  <p>Ein neuer, besserer Absatz</p>
```

## Spielwiese

```
h1.obsolete {
  color: red;
  text-decoration: line-through;
  background-color: lightgray;

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

### ~~1. Überschrift~~

Ein alter Absatz

### 2. Überschrift

Ein neuer, besserer Absatz

CSS Nesting ist erst seit 2024 in CSS verfügbar. Nesting findet bzgl. der Selektoren statt. Häufig(er) in Kombination mit *At-Regeln* (🇺🇸 *at-rules*; z. B. **@media**) verwendet.

# Nesting - & Operator

Der & Operator kann immer verwendet werden, ist aber oft optional.

```
p {  
  .obsolete {  
    text-decoration: line-through;  
  }  
}
```

ist äquivalent zu:

```
p .obsolete {  
  text-decoration: line-through;  
}
```

```
p {  
  &.obsolete {  
    text-decoration: line-through;  
  }  
}
```

ist äquivalent zu:

```
p.obsolete {  
  text-decoration: line-through;  
}
```

D. h. sollten nur solche Paragraphen durchgestrichen werden, die als *obsolete* markiert sind (d. h. `<p class='obsolete'>`) und nicht alle darunter liegenden Elemente, dann muss der & Operator verwendet werden (& ist dann nicht optional).

# Vererbung

- die meisten Eigenschaften (wie zum Beispiel **color**) werden vererbt
- Eigenschaften, die nicht vererbt werden sind zum Beispiel: **border**, **margin**, **padding** und **background**
- vererbte Eigenschaften haben **keine Spezifität**

(D. h. ein `:where()` Selektor oder der Universal-Selektor `*` gewinnen.)

# Kaskadierung

Die Entscheidung welche Regeln bzw. Deklarationen Anwendung finden, wird durch die Kaskadierung bestimmt:

1. Bestimme alle Regeln, die auf ein Element zutreffen.
2. Sortiere die Regeln nach Gewicht des Selektors (d.h. **!important** oder *normal*)
3. Sortiere alle Deklarationen basierend auf der Quelle:
  - Autor (höchste Priorität)
  - Benutzer (mittlere Priorität; z. B. *User-Stylesheets*)
  - *User Agent* (niedrigste Priorität; z. B. Browser Standard Styles)
4. Sortiere nach *Encapsulation Context* (cf. Shadow-DOM)
5. Sortiere danach ob die Deklarationen *Element Attached* sind (d. h. mittels **style** Attribut)
6. Sortiere nach *Cascade Layer*
7. Sortiere nach Spezifität
8. Sortiere nach Reihenfolge der Deklarationen

21

---

Der Shadow-Dom kapselt CSS und JavaScript bzgl. eines Elements. Dies ist insbesondere für Web-Komponenten relevant.

Sollte eine Deklaration möglicherweise nicht unterstützt werden, es jedoch einen vernünftigen Fallback geben, dann ist es möglich, die Deklarationen untereinander zu schreiben. Der Browser wird die unterstützte Deklaration verwenden und die anderen ignorieren.

Beispiel:

```
div {  
  height: 100vh;  
  height: 100svh;  
}
```



# :not() - Beispiel

## HTML

```
<hr>
<p class="new">
  Neuer Absatz
</p>
<p class="new">
  Noch ein neuer Absatz
</p>
<p>Alter text.</p>
```

## Spielwiese

```
p:not(.new) {
  text-decoration: line-through;
}
/*hr ~ *:not([class]) {
  font-size: smaller;
  color: red;
}*/
```

Neuer Absatz  
Noch ein neuer Absatz  
~~Alter text.~~

- :not(<list of selectors>) erlaubt die logische Und-Verknüpfung:  
:not(<selector\_a>, <selector\_b>) ≙ nicht selector\_a und nicht selector\_b.
- die Spezifität ergibt sich aus der Spezifität des spezifischsten Selektors

# :is() und :where() - Beispiel

Erlauben das Gruppieren von Selektoren innerhalb eines (komplexen) Selektors.

HTML

```
<hr>
<ol>
  <li>Aufgezählt</li>
</ol>
<ul>
  <li>Ein Punkt</li>
</ul>
```

Spielwiese

```
:is(ol, ul) li {
  font-style: italic;
}
/* :where(ol, ul) li {
  font-weight: bold;
  font-style: normal;
}*/
```

*1. Aufgezählt*

• *Ein Punkt*

- **:is()** und **:where()** unterscheiden sich nur in der Spezifität. (0 bei **:where()**; die Spezifität des spezifischsten Selektors bei **:is()**).

# :has() - Beispiel

## HTML

```
<ol>
  <li class="important">Aufgezählt</li>
  <li>Aufgezählt</li>
</ol>
<ul>
  <li>Ein
    <span class='important'>Punkt</span>
  </li>
  <li>Semikolon</li>
</ul>
```

## Spielwiese

```
:is(ol, ul):has(>.important) li {
  font-style: italic;
  color: red;
}
```

1. Aufgezählt  
2. Aufgezählt

- Ein Punkt
- Semikolon

- bei **:has()** werden die Selektoren relativ zum Element ausgewählt, welche den Anker für **:has()** bilden
- **:has(<list of selectors>)** verknüpft die Selektoren mittels logischem Oder.  
**:has(<selector\_a>, <selector\_b>) ≡ selector\_a oder selector\_b passt.**
- die Spezifität ergibt sich aus der Spezifität des spezifischsten Selektors

25

Mittels **:has** können wir (hier) eine Liste als ganzes selektieren, wenn ein Element in der Liste eine bestimmte Klasse hat (z. B. **important**).

CSS Selektoren werden auch von der JavaScript API für HTML Dokumente verwendet, um Elemente zu selektieren.

# 3. WERTE UND EINHEITEN

Prof. Dr. Michael Eichberg

# Kategorien

- Einige Eigenschaften haben Schlüsselworte, die spezielle Werte repräsentieren (z. B. **none** bei **text-decoration**)
- Das gleiche Schlüsselwort kann verschiedene Bedeutungen haben (z. B. **normal** bei **letter-spacing** und **font-style**)
- Es gibt fünf globale Schlüsselworte, die immer verwendet werden können: **inherit**, **initial**, **unset**, **revert**, und **revert-layer**.
- Strings können in ' oder " eingeschlossen werden
- Identifikatoren (z. B. **checked**)
- URLs werden mittels **url(...)** angegeben
- Ganzzahlen, Fließkommazahlen und Prozente
- Ausgewählte Distanzen:
  - Absolute Längen: **cm**, **mm**, **in**, **pt**, **pc**, **px**
  - Relative Längen:
    - Charakter bezogene Längen: **em**, **ex**, **lh**
    - Root bezogene Längen: **rem** (*root-em*)
    - Relation: **fr** (Anteil vom Leerraum)
  - Viewport bezogene Längen: **vw** (viewport width), **vh** (viewport height), **dvh** (dynamic viewport height), **dvw** (dynamic viewport width), **svh** (small viewport height), **svw** (small viewport width)
- Funktionswerte: **calc()**, **min()**, **max()**, **clamp(<min\_value>, <preferred\_value>, <max\_value>)**, **attr** und über 90 weitere Funktionen
- Farben werden spezifiziert mittels Schlüsselworte: (**red**, **green**, etc.), RGB-Werte: **rgb(<red>, <green>, <blue>)** oder **rgb(<red> <green> <blue> [/ <alpha>])**; oder ...
- Zeitangaben: **s** und **ms**
- Verhältnisse: **<number> / <number>** (z. B. **16/9**)
- Benutzerdefinierte Eigenschaften (*CSS Variables*):  
Beispiel:

## 1. Deklaration

```
html { --main-color: red;}
```

(Häufig `:root {...}` statt `html`.)

## 2. Verwendung inkl. Fallback-Wert:

```
p {color: var(--main-color, black)}
```

Der Scope ergibt sich aus dem Element, in dem die Variable definiert wurde.

Bei Verwendung findet einfaches (textuelles) Ersetzen statt.

**px** ist ein Pixel ist die Größe, die man benötigt, wenn man 96 Pixel pro Zoll hat; **px** ist die Einzige absolute Längeneinheit, die von Webseiten typischerweise verwendet wird. Ein Pixel ist somit unabhängig von der Größe eines Pixels auf dem Bildschirm!

**em** der Wert der Font-Größe des aktuellen Fonts.


**ex** ist die größe eines kleinen x im aktuellen Font

**lh** computed line-height


**calc** erlaubt verschiedenste Berechnungen ist aber an einigen Stellen *Whitespace-sensitive* und unterliegt bestimmten Einschränkungen welche Arten von Werten verrechnet werden können. (+ und - müssen immer mit Leerraum umgeben sein.)

# CSS - Berechnung von Werten


Der Wert einer CSS Eigenschaft wird wie folgt bestimmt:

1. der spezifizierte Wert wird basierend auf der Auswertung der Kaskadierung bestimmt
2. der berechnete Wert ( *computed value*) wird bestimmt basierend auf der CSS Spezifikation

(Dieser Wert lässt sich mittels JavaScript abfragen.)

3. der verwendete Wert ( *used value*) wird bestimmt basierend auf dem berechneten Wert und den Eigenschaften des Ausgabemediums

(Größen sind zum Beispiel in Pixel.)

4. der tatsächliche Wert ( *actual value*) wird bestimmt basierend auf dem verwendeten Wert (z. B. durch Rundung auf ganze Zahlen)



## 4. GRUNDLEGENDE FORMATIERUNG

Prof. Dr. Michael Eichberg

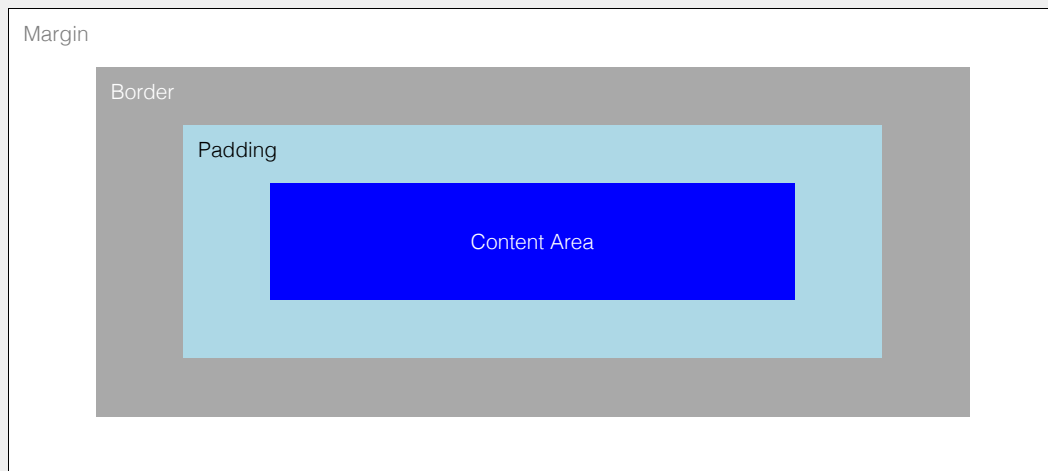
# Box-Modell - Einführung

- jedes Element erzeugt eine Box (*Element Box*):
  - entweder eine *Block Box*
  - oder eine *Inline Box*
- Es ist möglich den Typ der Box zu ändern.
- Es ist möglich die Größe der Box zu ändern.
  - Basierend auf der Größe des Inhalts: `max-content`, `min-content`, `fit-content`  
(Insbesondere - aber nicht ausschließlich - genutzt bei Grid-Layouts.)
  - Explizite Angabe der Größe: `width`, `height`, `min-width`, `max-width`, `min-height`, `max-height`
    - absolute Werte: insbesondere `px`
    - relative Werte: `width: x%` setzt die Breite auf `x%` der Größe des *Containing Block*. `height: y%` setzt die Höhe auf `y%` der Größe des *Containing Block* - wenn dieser eine explizite Höhe hat!
    - `auto` ist der Standardwert
  - Die Größe wird bei *Inline-Replaced Elements* ignoriert.
- Die Größe der Box berechnet sich „nur“ aus der Größe des Inhalts (d. h. der `content` Bereich); dies kann geändert werden durch: `box-sizing: border-box;`

`box-sizing: border-box;` setzt die Größe der Box auf die Größe des Inhalts plus Padding und Border. (Der Standardwert ist `content-box`.)

# Darstellung des Box-Modells

Im Zentrum ist der Content-Bereich (*Content Area*)



- Das Layout erfolgt relativ zum *Containing Block*.

32

Eine Block Box generiert vor und nach ihrer Box einen Leerraum entlang des normalen Flusses des Dokuments.  
Eine Inline Box, die länger als eine Zeile ist, wird in mehrere Zeilen umgebrochen - außer bei *Replaced Elements*.

Padding und Border können nicht negativ sein. Margin kann negativ sein.

**outlines** belegen keinen Platz und sind nicht Teil des Box-Modells.

# Inhalt, der nicht in die umgebende Box passt

## HTML

```
<div class="container">
  <div style='width:1000px;
    text-align:center;'>
    1
  </div>
  <div>2</div>
  <div>3</div>
</div>
<p>Der Test ist zu lang.</p>
```

## Spielwiese

```
div.container {
  height: 160px;
  overflow: scroll; /*visible, hidden*/
  /* overflow-x: hidden; */
}
div > div {
  width: 100%
  height: 40px;
}
```

1

2

Der Test ist zu lang.

# ******Collapsing Block-Axis Margins******

HTML

```
<div class="container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

Spielwiese

```
div.container {
  padding: 0;
}
div > div {
  width: 100%;
  height: 1.2em;
  margin: 1.2em;
  /*margin-bottom: 0;*/
}
```

1

2

3

# Floating

Elemente können mit **float** aus dem normalen Fluss genommen werden:

## HTML

```
<div>
  <aside style='
    height: 5lh; padding: 1em;
    background-color: black; color: white'>
    Rechtspopulismus
  </aside>
  [...] Dabei verhält sich der Rechtspopulismus
  durchaus ambivalent: Während er in einigen
  Bereichen der Politik, wie der Kriminalitäts-
  bekämpfung, einen starken Staat fordert, lehnt
  er ihn in anderen Bereichen ab und fordert
  stattdessen Volksabstimmungen, weil er dem
  repräsentativen Charakter von Parlamenten
  misstraut und durch sie den Volkswillen
  verfälscht sieht. [...]

  <cite> Wikipedia – Rechtspopulismus </cite>
</div>
```

## Spielwiese

```
aside {
  /*display: inline;*/
  float: right;
  box-sizing: border-box;
}
cite { display: block;}
```

[...] Dabei verhält sich der Rechtspopulismus durchaus ambivalent: Während er in einigen Bereichen der Politik, wie der Kriminalitätsbekämpfung, einen starken Staat fordert, lehnt er ihn in anderen Bereichen ab und fordert stattdessen Volksabstimmungen, weil er dem repräsentativen Charakter von Parlamenten misstraut und durch sie den Volkswillen verfälscht sieht. [...] <https://de.wikipedia.org/wiki/Rechtspopulismus>

Rechtspopulismus

35

- Varianten:
  - **left**: Element wird links ausgerichtet
  - **right**: Element wird rechts ausgerichtet
  - **none**: Element wird nicht ausgerichtet
- Standardansatz für das Erstellen von Layouts in den Anfangstagen (totaler Hack!)
- Um zu verhindern, dass ein Float in ein anderes Element hineinragt, kann **clear** verwendet werden.

# Positioning - relative und absolute

## HTML

```
<div class="page">
  Ein erster Text.
</div>
<div class="page">
  Hier kommt mehr text.
</div>
```

## Spielwiese

```
.page {
  width: calc(100% - 20px); height: 100px;
  background-color: yellow;
  position: relative;
  padding: 10px;
  margin: 10px;
  box-sizing: border-box;
}
.page::after{
  content: '<Page>';
  font-size: 0.8em;
  position: absolute;
  bottom: 10px;
  right: 10px;
}
```

Ein erster Text.

<Page>

Hier kommt mehr text.

<Page>

36

Die Positionierung erfolgt dann über die *Offset Eigenschaften*:

- top:** Abstand zum oberen Rand des *Containing Block*
- right:** Abstand zum rechten Rand des *Containing Block*
- bottom:** Abstand zum unteren Rand des *Containing Block*
- left:** Abstand zum linken Rand des *Containing Block*

**relative** positionierte Elemente verhalten sich wie **static** positionierte Elemente; bilden jedoch den *Containing Block* für **absolute** positionierte Elemente.

**absolute** positionierte Elemente werden relativ zum nächsten *positionierten* Elternelement positioniert. Sollte ein solches Element nicht existieren, dann wird das Element relativ zum *Initial Containing Block* positioniert.

# Positioning - **fixed** und **sticky**

**fixed:**

Das Element wird relativ zum Viewport positioniert.

**sticky:**

Das Element bleibt im normalen Fluss, bis der Zeitpunkt erreicht ist, an dem es fixiert wird (d. h. absolut positioniert wird).



# Flexbox

Layout-Modell, das es ermöglicht Elemente einfach innerhalb eines Containers anzuordnen.

(Aktiviert mit `display: flex;` oder `display: inline-flex`)

HTML

```
<div id="main">
  <div class="flex-container">
    <div class="flex-item">1</div>
    <div class="flex-item">2</div>
    <div class="flex-item">3</div>
  </div>
</div>
```

Spielwiese

```
#main {width: 850px;}
div.flex-container {
  display: flex;
  flex-direction: row; /* column */
  flex-wrap: wrap;
  justify-content: space-evenly;
}
div.flex-item {
  flex-basis: 150px;
  flex-grow: 1;
  height: 30px;
}
```



38

- Flexbox ist ein „ganzes CSS-Modul“, dass aus mehreren Eigenschaften besteht.
- Eigenschaften des Container: `flex-direction`, `flex-wrap`, `justify-content`, `align-items`, `align-content`, `(row-|column-)gap`
- Eigenschaften der Elemente des Containers: `align-self`, `flex-grow`, `flex-shrink`, `flex-basis`, `order`
- Flexbox unterscheidet zwischen der *Main Axis* und *Cross Axis*. `flex-direction` legt die Hauptachse fest.

## 5. RESPONSIVE DESIGN

Prof. Dr. Michael Eichberg

# Responsive Design - Grundlagen

- Ziel ist es sicherzustellen, dass eine Webseite auf verschiedenen Geräten mit (sehr) unterschiedlichen Auflösungen gut aussieht.
- Durch unterschiedliche Techniken umsetzbar
  - Media-Queries
  - Flexbox
  - Grid-Layout

# Media-Queries - Beispielhaft

```
<h1>Überschrift</h1>
<p>
  Ein Absatz.
</p>
```

```
<style>
  @media screen and (600px <= width < 1200px) {
    body { background-color: lightblue; }
    html { font-size: 16px; }
  }
  @media screen and (width < 600px) {
    body { background-color: red; }
    html { font-size: 12px; }
  }
  @media screen and (width >= 1200px) {
    body {
      background-color: whitesmoke;
      transition: all 2.5s;
    }
    html { font-size: 24px; }
  }
</style>
```

41

Der Type kann für referenzierte Stylesheets direkt angegeben werden:

```
<link rel="stylesheet" media="screen and (max-width: 600px)" href="small.css">
<link rel="stylesheet" media="print" href="print.css">
```

# Media-Queries und CSS Nesting - Beispielhaft

Kombination von Media-Queries und CSS Nesting, um *Drop Caps* nur auf großen Bildschirmen anzuzeigen.

```
p {  
  font-size: 0.9rem;  
  font-style: italic;  
  min-height: 3lh;  
  
  @media (width >= 1200px) {  
    &::first-letter {  
      float: left;  
      font-size: 2lh;  
      line-height: 2lh;  
      font-weight: bold;  
    }  
  }  
}
```

# Flexbox - Beispielhaft

```
<section>
  <p>
    D-Day bezeichnet im Englischen
    den Stichtag militärischer
    Operationen.
  </p>
  <p>
    Die Europawahl 2024 ist die
    zehnte Direktwahl zum
    Europäischen Parlament.
  </p>
  <p>
    Demokratie ist ein Begriff für
    Formen der Herrschaftsorgani-
    sation auf der Grundlage der
    Partizipation aller.
  </p>
</section>
```

```
<style>
  section {
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    gap: 1em;
  }

  section p {
    flex-basis:
      calc(900px * 999 - 100% * 999);
    flex-grow: 1;
    flex-shrink: 1;
    background-color: whitesmoke;
    padding: 1em;
    margin: 0;
  }
</style>
```

43

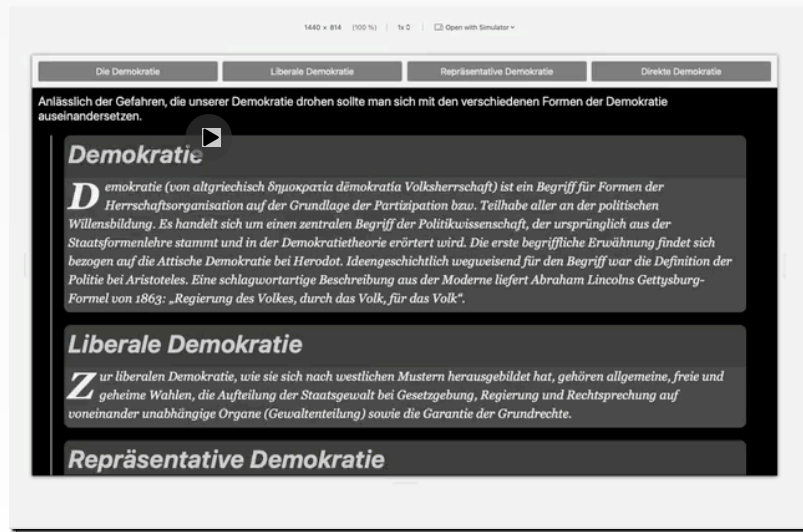
Der „Trick“ ist, dass die Berechnung für **flex-basis** so gewählt ist, dass ab einer bestimmten Größe der Wert für flex-basis entweder sehr groß ist (und damit nur noch ein Element in die Zeile passt oder eben sehr klein ist und damit alle Elemente in eine Zeile passen.)

# Nicht Behandelte Themen



- Animation
- Counter
- CSS bzgl. Printing
- Transformation (scaling, rotating, ...) .. scaling using `scale` vs. using `transform: scale`
- (bisher nur grob) Flexbox ([A guide to flex-box](#))
- Grid-Layout ([A complete guide to CSS Grid](#))
- Cascade Layers
- CSS Tricks .. e.g. assigning an index to an element using "nth-child"
- Shadow-DOM (und HTML Custom Elements)
- Styling von Dokumenten bei denen die Flussrichtung nicht links nach rechts ist

Versuchen Sie das Layout der folgenden HTML-Datei mittels CSS nachzubauen. Der HTML Code darf nicht verändert werden. JavaScript darf auch nicht verwendet werden. Den Rumpf der HTML-Datei finden Sie im Anhang.



## Hinweise

Mit Hilfe der folgenden CSS Eigenschaften können Sie das Layout nachbauen. Es gibt aber viele Wege, die zum Ziel führen!

### Verhalten (zum Beispiel mit Flexbox)

- display: flex, flex-direction, flex-wrap, flex-basis, flex-grow, gap, height, overflow-y

### Größen und Abstände

- margin(-right|-left), border, padding, font-size, line-height

### Optik

- box-shadow, font-style, font-family, color, background-color, border-radius, text-decoration

### Animation

- transition: all 0.6s;

### Trick

Nutzen Sie ggf. die Tricks aus dem Foliensatz!

## Rumpf der HTML-Datei

```
<!DOCTYPE html>
<html lang="de">

<head>
  <style>
    html {
      margin: 0;
      border: 0;
      padding: 0;
      font-size: 24px;
    }

    /* TODO */
  </style>
```



```

</head>

<body>
  <header>
    <nav>
      <a href="#einfuehrung">Die Demokratie</a>
      <a href="#lib_demokratie">Liberales Demokratie</a>
      <a href="#rep_demokratie">Repräsentative Demokratie</a>
      <a href="#dir_demokratie">Direkte Demokratie</a>
    </nav>
  </header>
  <main>
    Anlässlich der Gefahren, die unserer Demokratie drohen, sollte man sich mit den verschiedenen Formen der Demokratie auseinandersetzen.

    <blockquote cite="https://de.wikipedia.org/wiki/Demokratie">
      <h1 id="einfuehrung">Demokratie</h1>
      <p>
        Demokratie (von altgriechisch δημοκρατία dēmokratía Volksherrschaft) ist ein Begriff für Formen der Herrschaftsorganisation auf der Grundlage der Partizipation bzw. Teilhabe aller an der politischen Willensbildung. Es handelt sich um einen zentralen Begriff der Politikwissenschaft, der ursprünglich aus der Staatsformenlehre stammt und in der Demokratietheorie erörtert wird. Die erste begriffliche Erwähnung findet sich bezogen auf die Attische Demokratie bei Herodot. Ideengeschichtlich wegweisend für den Begriff war die Definition der Politie bei Aristoteles. Eine schlagwortartige Beschreibung aus der Moderne liefert Abraham Lincolns Gettysburg-Formel von 1863: „Regierung des Volkes, durch das Volk, für das Volk“.
      </p>

      <h1 id="lib_demokratie">Liberales Demokratie</h1>
      <p>
        Zur liberalen Demokratie, wie sie sich nach westlichen Mustern herausgebildet hat, gehören allgemeine, freie und geheime Wahlen, die Aufteilung der Staatsgewalt bei Gesetzgebung, Regierung und Rechtsprechung auf voneinander unabhängige Organe (Gewaltenteilung) sowie die Garantie der Grundrechte.
      </p>

      <h1 id="rep_demokratie">Repräsentative Demokratie</h1>
      <p>
        In einer repräsentativen Demokratie, in der gewählte Repräsentanten zentrale politische Entscheidungen treffen, haben oft Parteien maßgeblichen Anteil an der politischen Willensbildung und an der durch Wahlen legitimierten Regierung. Die Opposition ist fester Bestandteil eines solchen demokratischen Systems, zu dem auch die freie Meinungsäußerung samt Pressefreiheit, die Möglichkeit friedlicher Regierungswechsel und der Minderheitenschutz gehören.
      </p>

      <h1 id="dir_demokratie">Direkte Demokratie</h1>
      <p>
        In einer direkten Demokratie trifft das Stimmvolk politische Entscheidungen direkt.
      </p>
    </blockquote>
  </main>
</body>

```



Versuchen Sie das Layout der folgenden HTML-Datei mittels CSS nachzubauen. Der HTML Code darf nicht verändert werden. JavaScript darf auch nicht verwendet werden. Den Rumpf der HTML-Datei finden Sie im Anhang.