

# Cascading Style Sheets (CSS)

---

Dozent: Prof. Dr. Michael Eichberg

Kontakt: [michael.eichberg@dhw.de](mailto:michael.eichberg@dhw.de), Raum 149B

Version: 1.2

---

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

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

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

---

# 1. Einführung

# CSS - Cascading Style Sheets

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

HTML

```
<section>
  <p>1. Absatz</p>
  <p>2. Absatz</p>
  <p>3. Absatz</p>
</section>
```

CSS und Resultat

```
p { color: red; }
p ~ p { color: blue; }
p:nth-child(3) { color: green; }
```

1. Absatz

2. Absatz

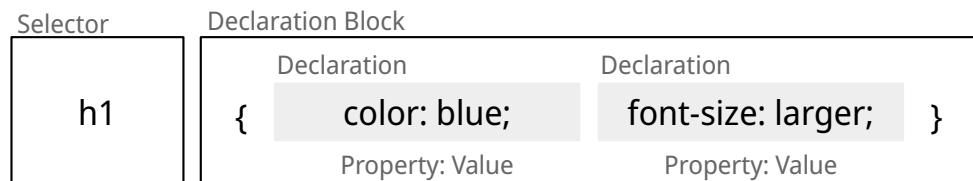
3. Absatz

# CSS - 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)

# Grundlagen

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



CSS

```
1 h1 {  
2   color: blue;  
3   font-size: larger;  
4 }  
5 body { /* the boss said so... */  
6   background-color:  
7   lightblue;  
8 }
```

Resultat

Überschrift  
Paragraph **in sehr wichtig!**.

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

Kommentare werden in /\* ... \*/ geschrieben.

# 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[1]: `<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.

[1] Siehe "@import" auf developer.mozilla.org

## 2. Selektoren

# Übersicht über Selektoren

## Zur Erinnerung

Im folgenden Beispiel ist `h1` der Selektor:

```
h1 { color: blue; font-size: larger; }
```

Der Selektor bestimmt, auf welche HTML Elemente die Regel angewendet werden soll.

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

## Class-Selectors - Beispiel[2]

HTML

```
1 <h1>Die Bedeutung des Seins</h1>
2 <h1 class="wip">
3     Die Bedeutung des Nicht-Seins
4 </h1>
5 <h1 class="todo future">
6     Das Sein und das Nicht-Sein
7 </h1>
```

CSS

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

Resultat

Die Bedeutung des Seins  
Die Bedeutung des Nicht-Seins

Das Sein und das Nicht-

[2] 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-Selectors[3]

- 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; }`
- 

[3] 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-Selectors - Beispiel

HTML

```
1 <h1 lang="de-DE">  
2   Die Bedeutung des Seins.</h1>  
3 <h1 lang="en-GB">  
4   To Be Or Not To Be</h1>  
5 <h1 lang="en-US">  
6   Play to win!</h1>  
7 <h1 lang="de-AT">  
8   Ich brauch ne Jause</h1>
```

CSS

```
1 [lang] { text-decoration: underline; }  
2 [lang$='US'] { color: orange; }  
3 [lang|= 'en'] { font-style: italic; }  
4 [lang="de-at" i] { text-transform: uppercase; }
```

Resultat

Die Bedeutung des Seins.

To Be Or Not To Be

Play to win!

ICH BRAUCH NE  
JAUSE

# Kombinatoren

## ■ Nachfahren (bzgl. Dokumentenstruktur) ( Descendant Selector):

div p	alle < <b>p</b> > Nachfahren von < <b>div</b> > Elementen
.important p	alle < <b>p</b> > Nachfahren von .important Elementen

## ■ Alle direkten Kinder ( Child Selector):

div > p	alle < <b>p</b> > Kinder von < <b>div</b> > Elementen
---------	---

## ■ Benachbarte Geschwister ( Adjacent Sibling Selector):

h1 + p	alle < <b>p</b> > Elemente, die direkt auf ein < <b>h1</b> > Element folgen und sich das gleiche Eltern-Element teilen
--------	--

## ■ Allgemeiner Geschwister Selektor ( General Sibling Selector):

h1 ~ p	alle < <b>p</b> > Elemente, die auf ein < <b>h1</b> > Element folgen und sich das gleiche Eltern-Element teilen
--------	---

# Kombinatoren - Beispiele

## HTML

```
1 <h1>Ü1</h1>
2 Text
3 <p>P1</p>
4 <p>P2</p>
5 <div>D0</div>
6 <p>P3</p>
7
8 <h1>Ü2</h1>
9 <div>
10   D1
11   <div>D1.1</div>
12   <div>D1.2</div>
13 </div>
14 <div>D2</div>
15 <div>D3</div>
```

## Spielwiese

```
/*h1 + p { background-color: blue; }*/
/*p + p { background-color: red; }*/
/*h1 ~ p { background-color: green; }*/
/*div ~ div { margin-left: 1em; }*/
/*div + div { font-size: 0.7em; }*/
/*h1 ~ div { background-color: gray; }
```

## Ü1

Text  
P1  
P2  
D0  
P3

## Ü2

D1  
D1.1  
D1.2  
D2  
D3

Beim `div ~ div` Beispiel wurde eine CSS Eigenschaft gewählt, die nicht vererbt wird, da sonst der Effekt, dass D1.1 nicht gewählt wird, nicht sichtbar ist!

# Pseudo-class Selektors

- erlauben das Selektieren von Elementen basierend auf ihrem Zustand
  - können beliebig kombiniert werden: `a:link:hover { color: red; }` selektiert alle nicht-besuchten Links 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 aktuelle Element.
- 

- Bei `nth-child(n)` und `nth-of-type(n)` ist n eine Zahl oder ein Ausdruck ( $\alpha n + b$ ), 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 bei HTML Dokumenten oder das `<svg>` Element bei SVG Dokumenten. `:root` wird insbesondere zur Definition von CSS Variablen verwendet!
- `:only-child` und `:only-of-type` selektiert ein Element, das das einzige entsprechende Kind seines Eltern-Elements ist.

# Pseudo-class Selektors - Beispiel

## HTML

```
1 <div class="oma" id="Maria">
2   <div class="papa" id="Fritz">
3     Vater 1
4       <div class="kind" id="Elias">
5         Kind 1
6       </div>
7     </div>
8   <div class="papa" id="Hans">
9     Vater 2
10    <div class="kind" id="Tobias">
11      Kind 2
12    </div>
13  </div>
14 </div>
```

## CSS

```
1 .papa:first-child { color: red; }
2 .kind:first-child { color: green; }
```

### Frage

Welcher Selektor selektiert welches Element?

Zur Erinnerung: Pseudo-class Selektoren selektieren das Element, auf das sie sich beziehen.

Vater 1  
Kind 1  
Vater 2  
Kind 2

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*.

# Pseudo-class Selektors bzgl. Inputvalidierung

## HTML

```
1 <input type="email"  
2   placeholder="your email"  
3   required>  
4 <input type="email"  
5   placeholder="your friend's email">
```

## Spielwiese

```
input[type="email"]::valid {  
    color: green;  
    border: 6px solid green;  
}  
/*input[type="email"]::invalid {  
    color: red;  
    border: 6px 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.

# Spezifität 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.

## Beispiele

Selektor	Spezifität
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)

## HTML

```
1 <section>
2   <p id='this-is-it'>
3     Der erste Abschnitt!
4   </p>
5   <p class='obsolete'>
6     Ein alter Abschnitt.
7   </p>
8 </section>
9 <p>Der letzte Abschnitt.</p>
```

## Spielwiese

```
/*p[id*="this"] {color: green; }*/ /edit
/*section p { color: red; }*/
/*p { color: orange; }*/
/*p ~ p { color: aliceblue; }*/
```

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.  
Im Prinzip definiert ! important eine eigene Menge von Regeln und innerhalb dieser wird die Kaskadierung invertiert. Innerhalb eines Layers werden alle als ! important markierten Deklarationen nach den beschriebenen Regeln ausgewertet.

# (Probleme bei der) Verwendung von !important[4]

Mit Stand Mai 2025 setzen alle drei großen Browser (Chrome, Firefox und Safari) CSS *Layers* und !important beim Rendering korrekt um, aber nur Firefox zeigt es auch in den Developer Tools korrekt an!

```
1 <head><style>      @layer low;      /* lowest priority */
2   @layer medium;
3   @layer high;      /* highest priority */
4
5   @layer high { p.error {
6     background-color: yellow;
7     color: blue !important;
8   }
9   @layer low { p.error {
10    background-color: lightblue !important;
11    color: red !important;
12  }
13  @layer medium { p.error {
14    background-color: cornsilk;
15    color: darkgreen !important;
16  } } </style></head>
17 <body>          <p class="error">This is an error message.</p>      </body>
```

**Chrome 135** - falsche Darstellung in den Entwicklertools - Farbe von "error message" ist im Browser rot, wird in den Entwicklertools aber als blau angezeigt.

This is an error message.

This is a success message.

Elements    Console    Sources    Network    Performance    Memory    >

Styles    Computed    Layout    Event Listeners    >

Filter :hov .cls +, □

Layer high

p.error {  
 background-color: yellow;  
 color: blue !important;  
}

Layer medium

p.error {  
 background-color: cornsilk;  
 color: darkgreen !important;  
}

Layer low

p.error {  
 background-color: lightblue !important;  
 color: red !important;  
}

html    body    p.error

**Firefox 138** - korrekte Darstellung in den Entwicklertools - Farbe von "error message" ist im Browser rot und wird in den Entwicklertools auch als rot angezeigt.

## This is an error message.

The screenshot shows the browser's developer tools with the 'Inspector' tab selected. On the left, the DOM tree displays an HTML document structure with various elements and classes. On the right, the CSS panel shows the corresponding CSS rules, organized by layer (high, medium, low). The 'p.error' rule is highlighted, demonstrating how it applies to elements across different layers despite its specificity.

```
<!DOCTYPE html>
<html lang="en"> event scroll
  <head> ... </head>
  <body>
    <h1>Important</h1> overflow
    <p>This is a message.</p> overflow
    <p class="important">
      This is a very important message.</p>
      overflow
    <p class="warning">
      This is a warning message.</p>
    <p class="error">This is an error message.</p>
    <p class="success">
      This is a success message.</p> overflow
  </body>
</html>
```

```
element :: { inline }
@layer high { inline:16
  p.error :: {
    background-color: yellow; ▾
    color: blue !important; ▾
  }
}
@layer medium { inline:46
  p.error :: {
    background-color: cornsilk; ▾
    color: darkgreen !important; ▾
  }
}
@layer low { inline:31
  p.error :: {
    background-color: lightblue !important;
    color: red !important;
  }
}
```

### Achtung!

Die Verwendung von KI Assistenten im Zusammenhang mit neueren CSS Features ist sehr problematisch, da diese ggf. noch nicht genug neuen Code gesehen haben und dann schlicht falsche Aussagen treffen!

### Unsinnige Antwort von ChatGPT auf einen Prompt bzgl. **!important** und CSS Layers:

#### 📌 The Role of **!important** with Cascade Layers

#### 🔥 Key Rule:

***!important** breaks out of layer ordering and competes only with other **!important** rules — regardless of layer. [...]*

#### ✅ That also means:

*If multiple **!important** rules exist, specificity and layer order determine the winner — but only within the **!important** set.*

—ChatGPT 5. Mai 2025

Hier gilt, dass "regardless of layer" nicht korrekt ist und auf die Inversion der Layer wird gar nicht eingegangen!

### Hinweis

Eine Verwendung von **!important** ist ein Zeichen dafür, dass die CSS Regeln nicht gut strukturiert sind!

### Vollständiges Beispiel bzgl. **!important**

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Important</title>
7
```

```
8 <style>
9     @layer low;
10    @layer medium;
11    @layer high;
12
13    /* The order is determined by the above declaration order! */
14    @layer high {
15        p.important {
16            background-color: yellow;
17            color: blue;
18        }
19        p.warning {
20            background-color: yellow;
21            color: blue;
22        }
23        p.error {
24            background-color: yellow;
25            color: blue !important;
26        }
27        p.success {
28            background-color: lightgreen;
29            color: rgb(61, 205, 200) !important;
30        }
31    }
32
33    @layer low {
34        p.important {
35            background-color: lightblue !important;
36            color: black;
37        }
38        p.warning {
39            background-color: lightblue !important;
40            color: black;
41        }
42        p.error {
43            background-color: lightblue !important;
44            color: red !important;
45        }
46    }
47
48    @layer medium {
49        p.important {
50            background-color: cornsilk;
51            color: darkgreen;
52        }
53        p.warning {
54            background-color: cornsilk;
55            color: darkgreen !important;
56        }
57        p.error {
58            background-color: cornsilk;
59            color: darkgreen !important;
60        }
61    }
62 </style>
63 </head>
64 <body>
65 <h1>Important</h1>
66 <p>This is a message.</p>
67 <p class="important">This is a very important message.</p>
```

```
68 |     <p class="warning">This is a warning message.</p>
69 |     <p class="error">This is an error message.</p>
70 |     <p class="success" style="color:salmon">This is a success message.</p>
71 | </body>
72 | </html>
```

---

[4] Stand Mai 2025

# 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[5]: *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.

[5] Es gibt noch „viel mehr“ Kontexte für spezielle Anwendungsfälle.

# Block und Inline Elemente - Beispiel

Code

```
1 h1 {  
2   display: inline;  
3 }  
4 strong {  
5   display: block;  
6 }
```

Visualisierung

Dies ist eine  
**Überschrift** in sehr wichtig  
; wirklich!

Folgendes Beispiel dient nur der Veranschaulichung:

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

Warnung

Dies ist kein gültiges HTML5!

# Vererbung von CSS Eigenschaften

- 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
- 

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

# CSS und nicht-unterstützte Eigenschaften

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:

```
1 | div {  
2 |   height: 100vh;  
3 |   height: 100svh; /* Kennt der Browser "svh" nicht,  
4 |                         dann wird die vorhergehende gültige Definition verwendet. */  
5 | }
```

---

`vh (Viewport Height):

1% der Höhe des Viewports

svh (Small Viewport Height):

1% der Höhe des *effektiven* Viewports. Dies ist insbesondere für Mobilgeräte relevant, da hier die Adresszeile und die Navigationsleiste nicht immer sichtbar sind.

## :not() - Beispiel

### HTML

```
1 <hr>
2 <p class="new">
3   Neuer Absatz
4 </p>
5 <p class="new">
6   Noch ein neuer Absatz
7 </p>
8 <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

```
1 <hr>
2 <ol>
3   <li>Aufgezählt</li>
4 </ol>
5 <ul>
6   <li>Ein Punkt</li>
7 </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. Die Spezifität ist bei :where() immer 0 und bei :is() gleich der die Spezifität des spezifischsten Selektors.

## :has() - Beispiel

HTML

```
1 <ol>
2   <li class="important">Aufgezählt</li>
3   <li>Aufgezählt</li>
4 </ol>
5 <ul>
6   <li>Ein
7     <span class='important'>Punkt</span>
8   </li>
9   <li>Semicolon</li>
10 </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

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

### Bemerkung

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

# Nesting

## HTML

```
1 <h1 class="obsolete">1. Überschrift</h1>
2   <p>Ein alter Absatz</p>
3 <h2>2. Überschrift</h2>
4   <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 breit 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.

```
1 | p {  
2 |     .obsolete {  
3 |         text-decoration: line-through;  
4 |     }  
5 | }
```

ist äquivalent zu:

```
1 | p .obsolete {  
2 |     text-decoration: line-through;  
3 | }
```

```
1 | p {  
2 |     &.obsolete {  
3 |         text-decoration: line-through;  
4 |     }  
5 | }
```

ist äquivalent zu:

```
1 | p.<span>.obsolete {  
2 |     text-decoration: line-through;  
3 | }
```

---

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).

# Übung - Einbinden von CSS in HTML

Gegeben sei die folgende (unformatierte) Webseite:

## Naturalismus (Philosophie)

Der Naturalismus ist die Auffassung, dass die Welt als ein gegebenes Geschehen zu begreifen ist. Er geht davon aus, Ursachen hat und dass es nichts Übernatürliches gibt. Dies durch den Spruch „Alles ist Natur“ pointiert wird, lässt für der Begriff der Natur zu umgrenzen ist. Versteht man unter Natur, so ergibt sich aus dem Spruch „Alles ist Natur“ eine physikalistische Position. Derartige Theorien vertreten, dass Bewusstsein Teil der physischen Natur sei oder alternativ es Illusion existiere. [...]

*Quelle: [Wikipedia](#)*

Image generated by DALL·E 2024-09-10 15.24.17 - An abstract illustration of the concept of 'naturalism.'

Code (HTML): [exercise-template.html](#)

Hintergrundbild: [image.png](#)

- Binden Sie den CSS Code (siehe Anhang) ein, um grundlegend das folgende Layout zu erhalten:

# Naturalismus (Philosophie)

Der Naturalismus ist die Auffassung, dass die Welt aus Natur gegebenes Geschehen zu begreifen ist. Es gibt nichts außer alles natürliche Ursachen hat und dass es nichts übernatürliches gibt. Diese Annahme, die oft auch durch den Spruch „Alles ist Natur“ ausgedrückt wird, lässt für sich genommen offen, wie der Begriff Natur umgrenzen ist. Versteht man unter Natur allein die Materie, ergibt sich aus dem Spruch „Alles ist Natur“ eine physikalistische Position. Derartige Theorien vertreten die Meinung, dass Geist oder das Bewusstsein Teil der physischen Welt sind und gar nicht oder höchstens als Illusion existiere. [...]



Image generated by DALL-E 2024-09-10 15.24.17 - An abstract image representing the concept of naturalism.

2. Erweitern Sie den CSS Code, um das finale Layout zu erhalten. Dazu müssen sie die folgenden CSS Eigenschaften passend „einfügen“.

```
text-align: center;  
text-align: right;  
  
font-family: sans-serif;  
font-size: smaller;  
font-size: 0.5em;  
font-size: 25px;  
text-shadow: 2px 2px 4px white;  
  
color: #999; /* defines the font color */  
color: #ccc;
```

```

background-color: rgba(0, 0, 0, 0.3);
background-color: rgba(0, 0, 0, 0.6);
background-color: rgba(255, 255, 255, 0.4);

/* Corners: top-left; top-right; bottom-right; bottom-left */
border-radius: 0.5em 0.5em 0 0;
border-radius: 0 0 0.5em 0.5em ;

```

## 2.1. Einbinden von CSS

---

### Grundlegender CSS Code

```

1  :root {
2    background-size: cover;
3    background-image: url('image.png');
4  }
5  body {
6    max-width: 60ch;
7    padding: 0;
8    margin: 0;
9    margin-right:auto;
10   margin-left:auto;
11  }
12  h1 {
13    padding:0.5rem;
14    margin-bottom: 0;
15    backdrop-filter: blur(5px);
16    -webkit-backdrop-filter: blur(10px);
17  }
18  p {
19    position: relative;
20    margin-top:0;
21    margin-bottom:0;
22    padding: 0.5rem;
23    font-weight: 100;
24    text-wrap: pretty;
25    -webkit-backdrop-filter: blur(10px);
26    backdrop-filter: blur(10px);
27  }
28  cite {
29    display: block;
30    padding: 0.5rem;
31  }
32  footer {
33    position: fixed;
34    bottom: 0;
35    left: 0;
36    right: 0;
37    padding: 0.5rem;
38  }
39

```

# Übung - CSS Selektoren

## 2.2. CSS Selektoren

Gegeben sei folgendes HTML Dokument:

```
1 <body>
2   <h1>Country Information</h1>
3   <ul>
4     <li>Germany
5       <ul>
6         <li>Berlin</li>
7         <li>Hamburg</li>
8         <li>Munich</li>
9       </ul>
10      </li>
11    <li>France</li>
12    <li>Spain</li>
13    <li>Sweden</li>
14    <li>Finland</li>
15    <li>Norway</li>
16    <li>Italy</li>
17    <li>Albania</li>
18    <li>Portugal</li>
19  </ul>
20 </body>
```

Schreiben Sie CSS Code, um folgende Formatierung zu erreichen:



00:00

Sie benötigten folgende Selektoren:

:nth-child(2n-1), :hover, +, :has, h1, ul, li

Verwenden Sie CSS Nesting, wenn möglich.

### 3. Werte und Einheiten

# Grundlagen

- Einige Eigenschaften haben Schlüsselwörter, 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 Schlüsselwörter, 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`, `ch`
    - 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 Schlüsselwörtern: (`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 Einige 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

`1h` computed line-height

`ch` Breite des Zeichens „0“ (ZERO, U+0030) (Ein Wert von 60ch entspricht bei vielen Fonts einer

effektiven Breite von ca. 80 Zeichen im Durchschnitt.)

`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)

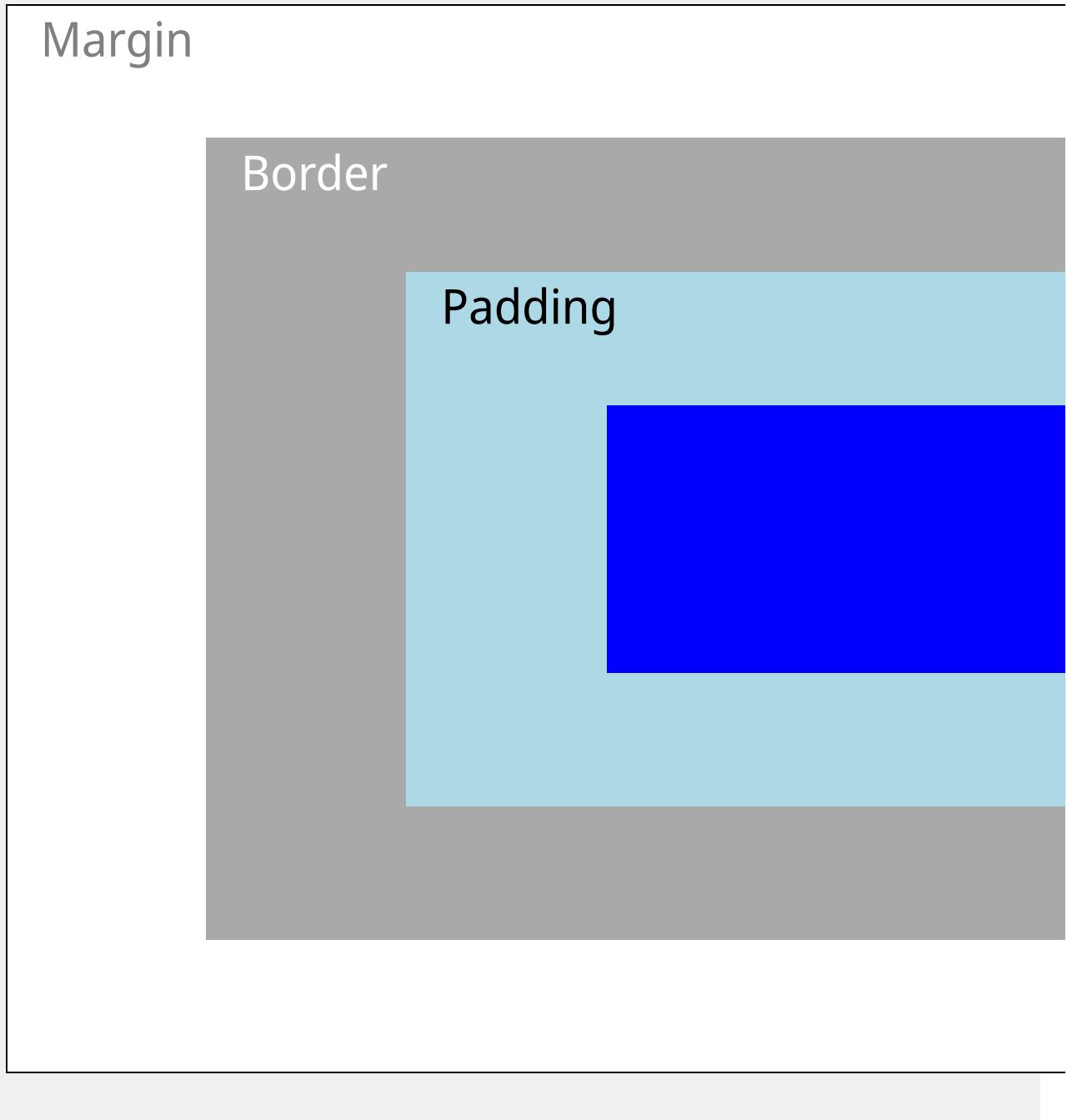
## 4. Grundlegende Formatierung

# 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*.

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.

#### Hinweis

`outline`s belegen keinen Platz und sind nicht Teil des Box-Modells.

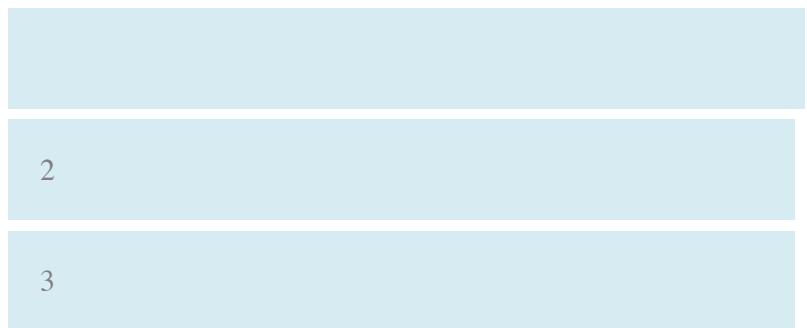
# Inhalt, der nicht in die umgebende Box passt

HTML

```
1 <div class="container">
2   <div style='width:1000px;
3     text-align:center; '>
4     1
5   </div>
6   <div>2</div>
7   <div>3</div>
8 </div>
9 <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;
}
```



Der Test ist zu lang.

# Collapsing Block-Axis Margins

HTML

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

Spielwiese

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

1

2

3

Text

# Floating

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

## HTML

```
1 <div>
2   <aside style='
3     height: 5lh; padding: 1em;
4     background-color: black; color: white'>
5     Rechtspopulismus
6   </aside>
7   [...] Dabei verhält sich der Rechtspopulismus
8   durchaus ambivalent: Während er in einigen
9   Bereichen der Politik, wie der Kriminalitäts-
10  bekämpfung, einen starken Staat fordert, lehnt
11  er ihn in anderen Bereichen ab und fordert
12  stattdessen Volksabstimmungen, weil er dem
13  repräsentativen Charakter von Parlamenten
14  misstraut und durch sie den Volkswillen
15  verfälscht sieht. [...]
16
17   <cite> Wikipedia - Rechtspopulismus </cite>
18 </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. [...]

Rechtspopulismus

## Varianten:

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

# Positioning - relative und absolute

## HTML

```
1 <div class="page">
2   Ein erster Text.
3 </div>
4 <div class="page">
5   Hier kommt mehr text.
6 </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>

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

## Zusammenfassung

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

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

### HTML

```
1 <div id="main">
2   <div class="flex-container">
3     <div class="flex-item">1</div>
4     <div class="flex-item">2</div>
5     <div class="flex-item">3</div>
6   </div>
7 </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;
}
```



- Flexbox ist ein „ganzes CSS-Modul“, dass aus mehreren Teilen 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

# 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
  - Container 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>
```

---

Der Type kann für referenzierte Stylesheets direkt angegeben werden:

```
1 | <link rel="stylesheet" media="screen and (max-width: 600px)" href="small.css">
2 | <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 / 1000 + 1px);
    flex-grow: 1;
    flex-shrink: 1;
    background-color: whitesmoke;
    padding: 1em;
    margin: 0;
  }
</style>
```

---

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.)

# Dark and Light Mode

- Die Unterstützung sowohl von Dark und Light-Mode ist mittlerweile Standard.
- Der aktuelle Modus kann mittels `prefers-color-scheme` abgefragt werden:

- `@media ( prefers-color-scheme: dark ) { ... }`
- `@media ( prefers-color-scheme: light ) { ... }`

(Eine) Vorgehensweise: Definition des Farbschemas über *Custom Properties*

```
:root {  
    /* Here, the default theme is the "light theme" */  
    --background-color: white;  
    --text-color: black;  
}  
  
@media ( prefers-color-scheme: dark ) {  
    :root {  
        --background-color: black;  
        --text-color: white;  
    }  
    a:link {  
        color: lightcoral;  
    }  
}
```

# Nicht Behandelte Themen

- Cascade Layers
  - Counter
  - Transformation (skalieren, drehen, ...)
  - Animation
  - (bisher nur grob) Flexbox ([A guide to flex-box](#))
  - Grid-Layout ([A complete guide to CSS Grid](#))
  - CSS Tricks
  - Shadow-DOM (und HTML Custom Elements)
  - Dokumente mit alternativen Flussrichtungen (rechts nach links / oben nach unten)
  - CSS bzgl. Printing
- 

Es gibt sehr, sehr viele CSS Tricks die Dinge ermöglichen, die nicht unmittelbar zu erwarten gewesen wären. Z. B. kann man einem Element einen Index zuordnen basierend auf dem ":nth-child()" Selektor. Dieser Index kann dann für „die Berechnung“ von weiteren Werten verwendet werden.

# Übung - Wo Licht ist, ist auch Schatten

Bauen Sie Unterstützung für den Dark und Light Mode nach. Den Rumpf der HTML-Datei finden Sie im Anhang.



00:00

---

**HTML-Datei**

```

<!DOCTYPE html>
<html lang="de">
<body>
  <main>
    <h1>Naturalismus (Philosophie)</h1>
    <p>
      Der Naturalismus ist die Auffassung, dass die Welt als ein
      rein von der Natur gegebenes Geschehen zu begreifen ist.
      Er geht davon aus, dass alles natürliche Ursachen hat und
      dass es nichts Übernatürliches gibt.[...]
    </p>
    <cite>
      Quelle:
      <a href="https://de.wikipedia.org/wiki/Naturalismus_(Philosophie)">
        Wikipedia
      </a>
    </cite>
  </main>
</body>
</html>

```

## Grundlegendes CSS Gerüst

```

/* The following CSS does not define any colors/color scheme. */
:root {
  --font-size: 1em;
  --font-family: sans-serif;
}

body {
  max-width: 60ch;
  padding: 20px;
  font-size: var(--font-size);
  font-family: var(--font-family);
  padding: 0;
  margin: 0;
  margin-right: auto;
  margin-left: auto;
}

h1 {
  padding: 0.75rem;
  margin-bottom: 0;
  border-radius: 0.5em 0.5em 0 0;
  backdrop-filter: blur(5px);
  -webkit-backdrop-filter: blur(10px);
}

p {
  position: relative;
  margin-top: 0;
  margin-bottom: 0;
  padding: 0.75rem;
  border-radius: 0 0 0.5em 0.5em;
  font-weight: 100;
  text-wrap: pretty;
}

cite {
  display: block;
  padding: 0.5rem;
  text-align: right;
  font-size: smaller;
}

```



# Übung - Komplexeres Layout

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;

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">Liberale 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">Liberale 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>

</html>