Cascading Style Sheets (CSS)

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de, Raum 149B
Version: 1.5.2

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

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

CSS und Resultat

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

---

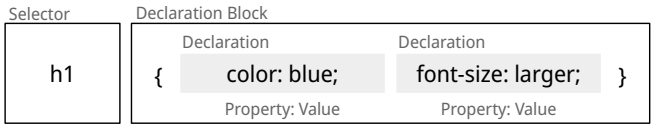
1. Absatz
2. Absatz
3. Absatz

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

```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.

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.og

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

```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

```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

[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

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

CSS

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

Resultat

Die Bedeutung des Seins.

To Be Or Not To Be

Play to win!

ICH BRAUCH NE

Kombinatoren

Nachfahren (bzgl. Dokumentenstruktur) (*Descendant Selector*)

`div p`	alle <p> Nachfahren von Elementen
`.important p`	alle <p> Nachfahren von `.important` Elementen

Alle direkten Kinder (*Child Selector*)

`div > p`	alle <p> Kinder von 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>
<div>D0</div>
<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 { background-color: bl
/*p + p { background-color: red
/*h1 ~ p { background-color: gr
/*div ~ div { margin-left: 1em;
/*div + div { font-size: 0.7em;
/*h1 ~ div { background-color:
```

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 Wurzelelement 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

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

CSS

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

```
<input type="email"
       placeholder="your email"
       required>
<input type="email"
       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;
}*/
```

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

Übung

2.1. Pseudo-class Selektoren

Gegeben sei

```html
<div class="slide">
  <h1>CSS</h1>
  <div class="version">1.0</div>
  <aside>
    <p>Cascading Style Sheets</p>
    <ol id="topics">
      <li>What is CSS?</li>
      <li>How to use CSS?<br></li>
      <li>Inline vs. Block</li>
      <li>Font Styling</li>
      <li class="todo">Lists</li>
      <li>Background Styling</li>
    </ol>
  </aside>
</div>
```

Aufgaben

1. Schreiben Sie einen CSS Selektor, um den ersten Buchstaben des ersten Wortes eines jeden Listenelements (`:first-letter`) in Kleinbuchstaben darzustellen mit `text-transform: lowercase`.
2. Schreiben Sie einen CSS Selektor, um jeder zweiten Zeile der Liste eine andere Hintergrundfarbe zu geben mit `background-color: plum`.
3. Selektieren Sie die Überschrift eines Blocks (`<div>`s) mit der Klasse `slide` wenn diese das erst Kind ist und geben Sie ihr die Schriftfarbe rot mit `color: red`.

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

```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; }*/
/*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 alls 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.

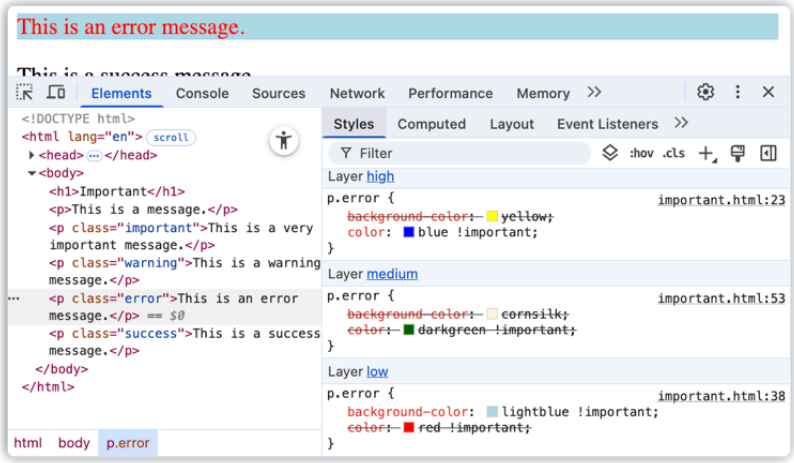
(Probleme bei der) Verwendung von `!important`[4]

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!

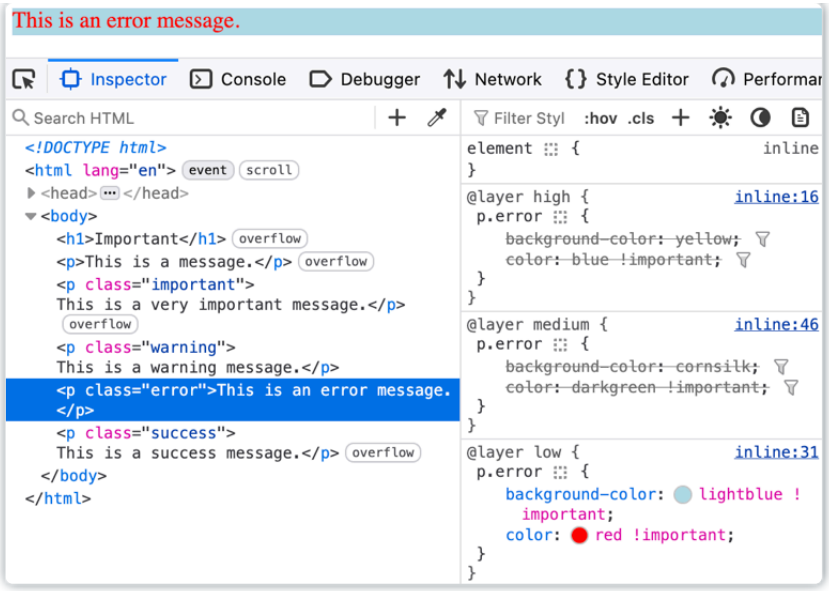
```
<head><style>   @layer low;       /* lowest priority */
                @layer medium;
                @layer high;      /* highest priority */

                @layer high { p.error {
                        background-color: yellow;
                        color: blue !important;
                }   }
                @layer low { p.error {
                        background-color: lightblue !important;
                        color: red !important;
                }   }
                @layer medium { p.error {
                        background-color: cornsilk;
                        color: darkgreen !important;
                }   } </style></head>
<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.



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



> ⚠ 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`

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

    <style>
        @layer low;
```

```
        @layer medium;
        @layer high;

        /* The order is determined by the above declaration order! */
        @layer high {
            p.important {
                background-color: yellow;
                color: blue;
            }
            p.warning {
                background-color: yellow;
                color: blue;
            }
            p.error {
                background-color: yellow;
                color: blue !important;
            }
            p.success {
                background-color: lightgreen;
                color: rgb(61, 205, 200) !important;
            }
        }

        @layer low {
            p.important {
                background-color: lightblue !important;
                color: black;
            }
            p.warning {
                background-color: lightblue !important;
                color: black;
            }
            p.error {
                background-color: lightblue !important;
                color: red !important;
            }
        }

        @layer medium {
            p.important {
                background-color: cornsilk;
                color: darkgreen;
            }
            p.warning {
                background-color: cornsilk;
                color: darkgreen !important;
            }
            p.error {
                background-color: cornsilk;
                color: darkgreen !important;
            }
        }
    </style>
</head>
<body>
    <h1>Important</h1>
    <p>This is a message.</p>
```

```
    <p class="important">This is a very important message.</p>
    <p class="warning">This is a warning message.</p>
    <p class="error">This is an error message.</p>
    <p class="success" style="color:salmon">This is a success message.</p>
</body>
</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

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

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 HTML!

Vererbung von CSS Eigenschaften

- die meisten Eigenschaften (wie zum Beispiel `color`) werden vererbt
- Eigenschaften, die nicht vererbt werden, sind insbesondere: `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

> **Bemerkung**
>
> Benutzer-Stylesheets spielen heutzutage fast keine Rolle mehr. In den Anfangstagen war es möglich ein eigenes CSS zu definieren und dem Browser zu sagen, dass dieses angewendet werden soll.

Der Shadow-Dom kapselt CSS und JavaScript bgzl. 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 (zum Beispiel) "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

```
<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-throu
}
/*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. Die Spezifität ist bei :where() immer 0 und bei :is() gleich der die Spezifität des spezifischsten Selektors.

:has() - Beispiel

HTML

```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

```css
: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.

Übung

2.2. Spezifität von Selektoren

Bestimmen Sie die Spezifität der folgenden Selektoren:

Selektor	a	b	c
*			
#ld-menu:has(+ #ld-main[style*='display: none;'])			
#ld-menu:hover			
div#ld-menu			
#continuous-view div.supplemental img			

2.3. :is vs. :where

```
table :is(th, td.head) {
    font-weight: 900;
    color: green;
}

table :where(th, td.head):hover {
    background-color: lightgray;
    color: blue;
}
```

Header 1	Header 2	A Value
Row 1, Cell 1	Row 1, Cell 2	**Row 1, Cell 3**
Row 2, Cell 1	Row 2, Cell 2	Row 2, Cell 3

Was passiert, wenn Sie mit der Maus über die Tabelle fahren? Wird der Text blau/verändert sich der Hintergrund?

Übung

2.4. Einfache Selektoren

Gegeben sei

```html
<div class="slide">
  <h1>CSS</h1>
  <div class="version">1.0</div>
  <aside>
    <p>Cascading Style Sheets</p>
    <ol id="topics">
      <li>What is CSS?</li>
      <li>How to use CSS?<br></li>
      <li>Inline vs. Block</li>
      <li>Font Styling</li>
      <li class="todo">Lists</li>
      <li>Background Styling</li>
    </ol>
  </aside>
</div>
```

Aufgaben

1. Selektieren Sie `<div>` Elemente mit Klassen.
2. Selektieren Sie alle `<div>` Elemente, die eine Klasse haben und ein `<h1>` Element enthalten.
3. Selektieren Sie `<li>` Elemente die die Klasse `todo` haben oder auf ein `<li>` folgen, das die Klasse `todo` hat.
4. Selektieren Sie die geordnete Liste mit der id `topics`.
5. Selektieren Sie ein `<h1>` Element, wenn es ein direktes Geschwister-Element hat, dass die Klasse `version` hat.
6. Selektieren Sie `<li>` Elemente, die keine Klassen haben.

Kopieren Sie den HTML Code in eine Datei und fügen Sie ein `<style>`-Element hinzu. Testen Sie Ihre Selektoren zum Beispiel durch das Setzen der Farbe des Textes oder des Hintergrunds (background-color). Insbesondere für den Hintergrund bietet es sich ggf. an eine Farge zu verwenden, die einen Alpha Kanal hat (z. B. `background-color: rgba(255, 155, 0, 0.6);`).

Nesting

- 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.

```
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).

Nesting - Beispiel

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-throu
    background-color: lightgray

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

~~1. Überschrift~~

Ein alter Absatz

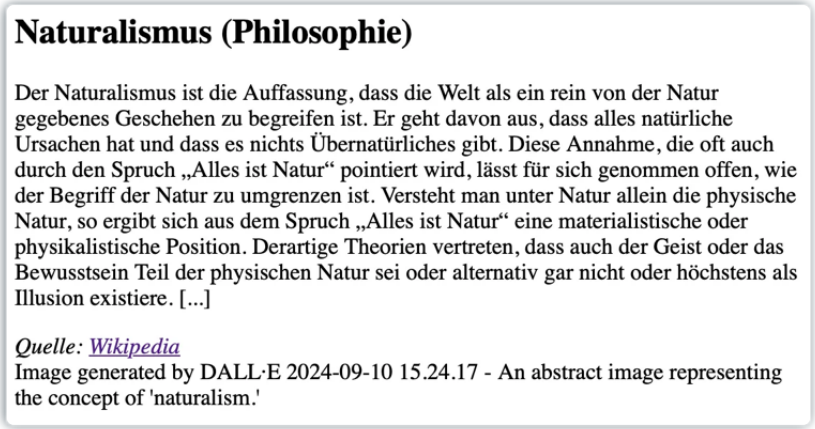
2. Überschrift

Ein neuer, besserer Absatz

Übung - Einbinden von CSS in HTML

2.5. Einbinden von CSS

Gegeben sei die folgende (unformatierte) Webseite:



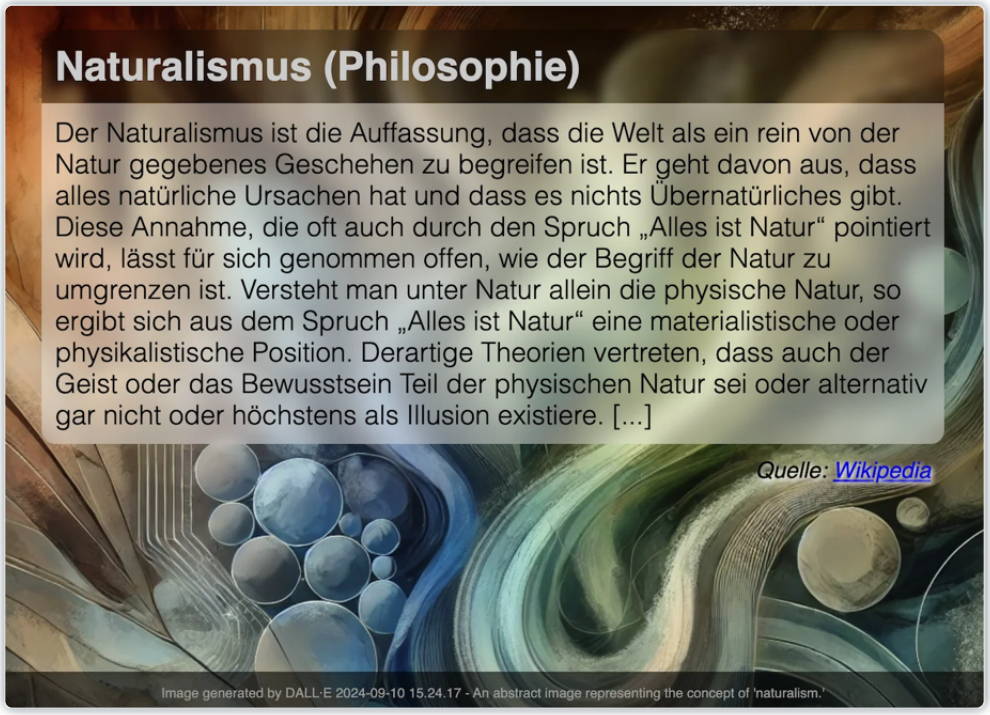
Naturalismus (Philosophie)

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. Diese Annahme, die oft auch durch den Spruch „Alles ist Natur“ pointiert wird, lässt für sich genommen offen, wie der Begriff der Natur zu umgrenzen ist. Versteht man unter Natur allein die physische Natur, so ergibt sich aus dem Spruch „Alles ist Natur“ eine materialistische oder physikalistische Position. Derartige Theorien vertreten, dass auch der Geist oder das Bewusstsein Teil der physischen Natur sei oder alternativ gar nicht oder höchstens als Illusion existiere. [...]

Quelle: Wikipedia
Image generated by DALL·E 2024-09-10 15.24.17 - An abstract image representing the concept of 'naturalism.'

Code (HTML): exercise-template.html
Hintergrundbild: image.webp

1. Binden Sie den angehängten CSS Code ein, um grundlegend das folgende Layout zu erhalten.



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 ;
```

Grundlegender CSS Code

```css
:root {
    background-size: cover;
    background-image: url('web-css/code/1st-exercise/image.webp');
}
body {
    max-width: 60ch;
    padding: 0;
    margin: 0;
    margin-right:auto;
    margin-left:auto;
}
h1 {
    padding:0.5rem;
    margin-bottom: 0;
    backdrop-filter: blur(5px);
    -webkit-backdrop-filter: blur(10px);

}
p {
    position: relative;
    margin-top:0;
    margin-bottom:0;
    padding: 0.5rem;
    font-weight: 100;
    text-wrap: pretty;
    -webkit-backdrop-filter: blur(10px);
    backdrop-filter: blur(10px);
}
cite {
    display: block;
    padding: 0.5rem;
}
footer {
    position: fixed;
    bottom: 0;
    left: 0;
    right: 0;
    padding: 0.5rem;
}
```

Übung - CSS Selektoren

2.6. CSS Selektoren

Gegeben:

```
<body><h1>Country Info</h1>
      <ul><li>Germany
            <ul><li>Berlin</li>
                <li>Hamburg</li>
                <li>Munich</li>
            </ul></li>
          <li>France</li>
          <li>Spain</li>
          <li>Sweden</li>
          <li>Finland</li>
          <li>Norway</li>
          <li>Italy</li>
          <li>Albania</li>
          <li>Portugal</li></ul>
</body>
```

Realisieren Sie folgenden Effekt nur mit CSS:

Sie benötigten folgende Selektoren: `:nth-child(2n-1)`, `:hover`, `+`, `:has`, `h1`, `ul`, `li`

Verwenden Sie CSS Nesting, wenn möglich.

3. Werte und Einheiten

Grundlagen

Schlüsselworte und einfache Werte

- 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
- 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`)
- Funktionswerte: `calc()`, `min()`, `max()`, `clamp(<min_value>,<preferred_value>, <max_value>)`, `attr` und über 90 weitere Funktionen

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)
- Container bezogene Größen: cqw (container query width)

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

`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 verschiedene 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.)

`initial` vs. `inherit` vs. `unset` vs. `revert`[6]

`initial`:	setzt den Wert auf den Standardwert der CSS Spezifikation
`inherit`:	setzt den Wert auf den Wert des Elternelements
`unset`:	setzt den Wert auf den Standardwert der CSS Spezifikation, wenn die Eigenschaft nicht vererbt wird (vgl. `initial`), ansonsten auf den Wert des Elternelements (`inherit`)
`revert`:	setzt den Wert auf den Wert, der durch die frühere Kaskade bestimmt wird (Typischerweise der Wert der sich aus der dem User-Agent ergibt, wenn keine anderen Regeln zutreffen.)
`revert-layer`:	setzt den Wert auf den Wert, der durch die frühere Kaskade bestimmt wird, aber nur für die Layer, die vor dem aktuellen Layer liegen. (Siehe auch `@layer`.)

[6] https://www.w3.org/Style/CSS/all-properties.en.html bzw. https://www.w3.org/TR/CSS2/propidx.html

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

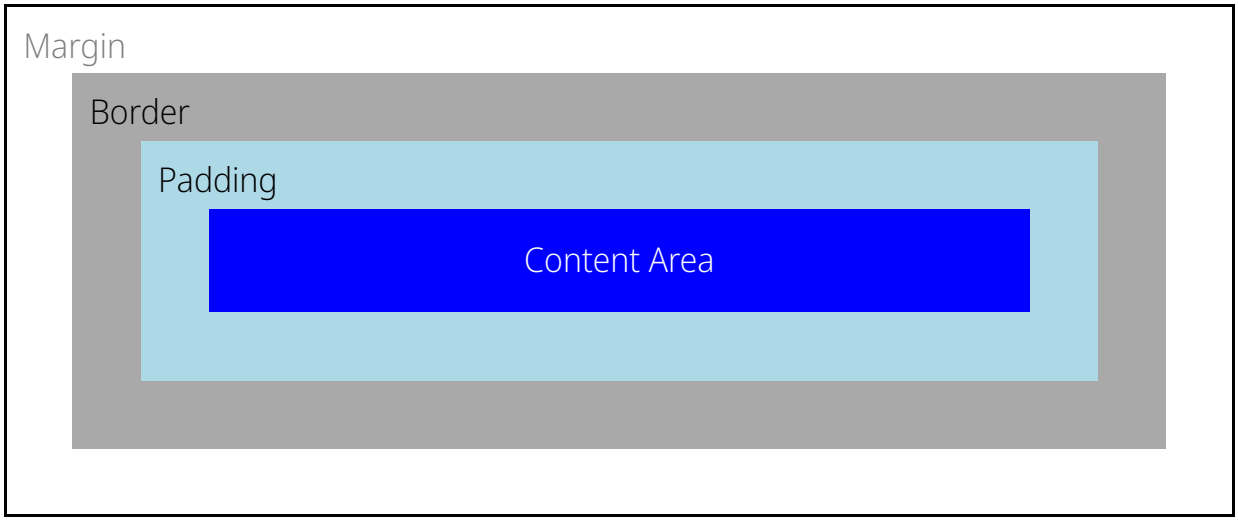
- jedes Element erzeugt eine Box (*Element Box*):
  - entweder eine *Block Box*
  - oder eine *Inline Box*
- Es ist möglich den Typ der Box zu ändern.
- Es ist möglich die Größe der Box zu ändern.
  - Basierend auf der Größe des Inhalts: `max-content`, `min-content`, `fit-content`

    (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` (CSS 2.1).)

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

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

Anzeige des Inhalts, der nicht in die umgebende Box passt, kontrollieren

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, hidde
    /* overflow-x: hidden; */
}
div > div {
    width: 100%
    height: 40px;
}
```

2

3

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

```
<div><aside style='
            height: 4lh; padding: 1em;
            background-color: black;
            color: white'>
        Rechtspopulismus</aside>
  [...] Dabei verhält sich der Rechts-
  populismus durchaus ambivalent:
  Während er in einigen Bereichen der
  Politik [...] einen starken Staat
  fordert, lehnt er ihn in anderen
  Bereichen ab [...] weil er dem repräs-
  sentativen Charakter von Parlamenten
  misstraut. [...]
  <cite>Wikipedia: Rechtspopulismus</cite>
</div>
```

Spielwiese

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

- 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 (vorhergehendes) Float in ein anderes Element hineinragt, kann `clear` verwendet werden.

Positioning - `relative` und `absolute`

HTML

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

Spielwiese

```css
.page {
    width: calc(100% - 20px); height:
    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`

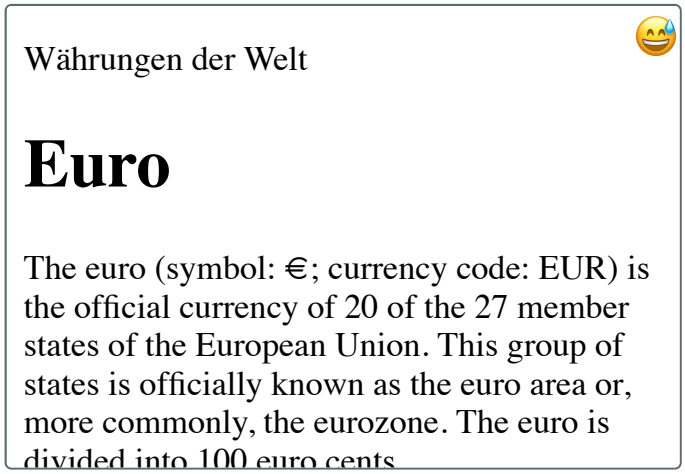
fixed: Das Element wird relativ zum Viewport positioniert.

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

HTML

```
<div class="fixed">😅</div>
<p>Währung der Welt</p>
<h1>Euro</h1>
<p>The euro (symbol: €; currency
   code: EUR) is the official ...</p>
<style>
    .fixed { position: fixed;
             top: 0;
             right: 0; }
    h1 {     position: sticky;
             top: 0; }
</style>
```

Spielwiese



Flexbox

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

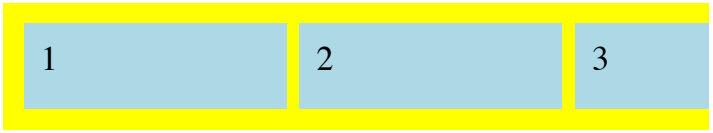
(Aktiviert mit `display: flex;` oder `display: inline-flex`.)

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: 25em; }
div.flex-container {
    display: flex;
    flex-direction: row; /* column */
    flex-wrap: wrap;
    justify-content: space-evenly;
    .flex-item {
        flex-basis: 5em;
        flex-grow: 1;
        height: 1.5em;
}   }
```



- 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

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

```css
<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:

```html
<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.

```css
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;
        }
    }
}
```

CSS Trick mit Flexbox - Beispielhaft

```html
<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>
```

```html
<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>
```

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.)

Ist der Viewport kleiner als 900px - also zum Beispiel 800px, dann ist der Wert für `flex-basis` 900px * 999 - 800px * 999 = 99.900 und somit extrem groß. Damit wird der Wert von :css:`flex-basis „ignoriert“ und jedes Elemente wird in einer Zeile dargestellt. Ist der Viewport zum Beispiel 1000px, dann ist der Wert für `flex-basis` 900px * 999 - 1000px * 999 = -99.000 und damit negativ. Somit wird der Wert von `flex-basis` wieder „ignoriert“ und alle Elemente werden neben einander in einer Zeile dargestellt.

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 ) { ... }`

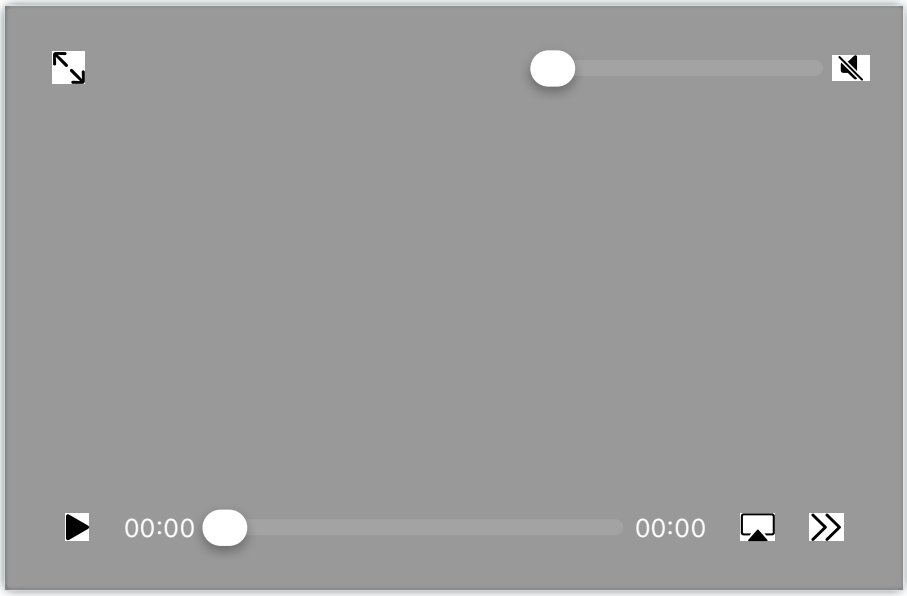
(Eine) Vorgehensweise: Definition des Farbschemas über *Custom Properties*

```
: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;
    }
}
```

Übung - Wo Licht ist, ist auch Schatten

Bauen Sie Unterstützung für den Dark und Light Mode nach.[7]



HTML-Datei

```html
<!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

```css
/* The following CSS does not define any colors/color scheme. */
:root {
    --font-size: 18px;
    --font-family: sans-serif;
}

body {
    max-width: 60ch;
    padding: 20px;
```

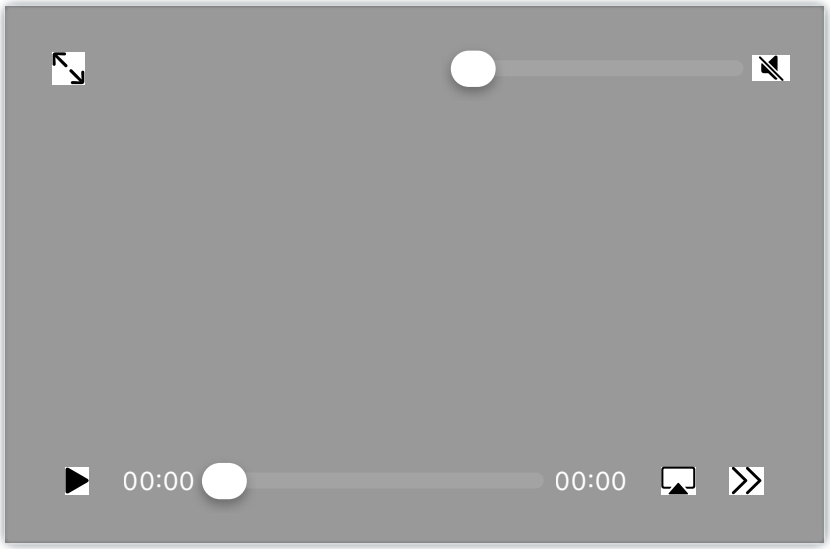
```
    font-size: var(--font-size);
    font-family: var(--font-family);
    padding: 0;
    margin: 0;
    margin-right:auto;
    margin-left:auto;
}
h1 {
    padding:0.75rem;
    margin-bottom: 0;
    border-radius: 0.5em 0.5em 0 0;
    backdrop-filter: blur(5px);
    -webkit-backdrop-filter: blur(10px);
}
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;
}
```

[7] Den Rumpf der HTML-Datei finden Sie im Anhang.

Übung

5.1. 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;

※ Hinweis

Nutzen Sie ggf. die Tricks aus dem Foliensatz!

Rumpf der HTML-Datei

```
1  <!DOCTYPE html>
2  <html lang="de">
3
4  <head>
5      <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>
```

```
63
64              <h1 id="rep_demokratie">Repräsentative Demokratie</h1>
65              <p>
66                  In einer repräsentativen Demokratie, in der gewählte
67                  Repräsentanten zentrale politische Entscheidungen
68                  treffen, haben oft Parteien maßgeblichen Anteil an der
69                  politischen Willensbildung und an der durch
70                  Wahlen
71                  legitimierten Regierung. Die Opposition ist fester
72                  Bestandteil eines solchen demokratischen Systems, zu
73                  dem
74                  auch die freie Meinungsäußerung samt Pressefreiheit, die
75                  Möglichkeit friedlicher Regierungswechsel und
76                  der
77                  Minderheitenschutz gehören.
78              </p>
79              <h1 id="dir_demokratie">Direkte Demokratie</h1>
80              <p>
81                  In einer direkten Demokratie trifft das Stimmvolk
82                  politische Entscheidungen direkt.
83              </p>
84          </blockquote>
85      </main>
86
87  </body>
88
89  </html>
```

6. CSS-Layers

CSS-Layers

- CSS Layers sind insbesondere in komplexen Projekten hilfreich, um die Kaskadierung von CSS-Regeln zu steuern und sicherzustellen, dass bestimmte Regeln Vorrang haben.
- `@layer` ermöglicht das deklarative Festlegen einer expliziten Kaskaden-Reihenfolge:
- Eine *später* deklarierte Schicht überschreibt frühere Schichten unabhängig von der Spezifität der Selektoren.
- Beim Import von CSS-Dateien kann die Schicht ebenfalls angegeben werden:

Beispiel

```
@layer basis, overrides, theme;

@import url("modularization/basis.css") layer(basis);
@import url("modularization/overrides.css") layer(overrides);
@import url("modularization/theme.css") layer(theme);
```

- Importiert/deklariert eine CSS Datei, die bereits einer Schicht zugeordnet ist, weitere Schicthen so bilden diese automatisch eine Hierarchie.

Insbesondere in großen Projekten sind CSS Layers wichtige, um ein Kampf um die höchste Spezifität zu vermeiden. Weiterhin erlaubt es die problemlose Nutzung von CSS-Frameworks, die in eigenen Schichten organisiert sind. Ggf. kann auch ein Import in eine Schicht erfolgen, um die Kaskadierung zu steuern.

CSS-Layers - Beispiel

HTML

```
<p id="the-one">
    Dies ist ein Absatz.
</p>
```

Spielwiese

```
/* @layer overrides; */
@layer basis {
    p#the-one{color:green;}
}
@layer overrides {
    p{color:red;}
}
/* @layer basis{p#the-one{color:blue;}} */'
```

Dies ist ein Absatz.

Verschachtelte CSS-Layers

- `@layer` könne beliebig tieft verschachtelt werden:
- Deklarationen einer höheren Schicht (hier `base`) überschreiben Deklarationen einer tieferen Schicht (hier: `base.components`).

Beispiel

```
@layer base {
    h1 { color: navy; }

    @layer components {

        h1 { color: red; }

        .highlight {
            background-color: yellow;
        }
    }
}
```

Übung - Modularisierung von CSS und CSS-Layers

6.1. CSS Layers

Nehmen Sie die Lösung von der vorherigen Übung und modularisieren Sie diese mit Hilfe von CSS-Layers. Nutzen Sie zum Normalisieren von CSS die Datei normalize.css[8].

Legen Sie danach zwei weitere Dateien an, die die grundlegenden CSS Eigenschaften (Position etc.) enthalten und eine Datei, die sich nur um das Theming kümmert.

[8] Original: normalize.css @ GitHub

7. Web Komponenten (Shadow DOM und CSS)[9]

[9] JavaScript spezifische Funktionalitäten von Web Komponenten werden hier nicht behandelt.

Shadow DOM und Light DOM

- Web Komponenten ermöglichen die Kapselung von HTML, CSS (und JavaScript) in einem eigenen Kontext, dem Shadow DOM.
- Der Shadow DOM ermöglicht eine klare Trennung von Stilen und Verhalten der Komponente von dem umgebenden Dokument.
- *normale* Kindelemente des Shadow Host bilden den Light DOM und können aus dem Shadow DOM via `<slot>` referenziert werden.
- Struktur:

Der Light DOM sind die Kindelement des *Shadow Host*. Diese werden nicht gerendert.

(Declarative) Shadow/Light DOM - Anwendungsbeispiel

Code

```
<div>   <!-- Das h2 Element wird "slotted"! -->
        <h2 style="color: var(--c)">Demo</h2>
  <template shadowrootmode="open">
    <style>
      style[contenteditable] {
        display: block;
        font-family: var(--code-font-family); }
    </style>
    <style contenteditable
           onkeydown="event.stopPropagation()">
      /* :host { --code-font-family: Serif; }*/
      p#the-one{color:green;}</style>
    <slot></slot>
    <p>Dies ist der erste Absatz.</>
    <p id="the-one">Dies ist ein Absatz.</p>
  </template></div>
```

Eingebettet

```
/*:host {
     --code-font-family: Serif;
}*/
/*:host { --c: red; }*/
p#the-one{color:green;}
```

⚠ keine Spiegelung

Demo

Dies ist der erste Absatz.

Dies ist ein Absatz.

Slotted Elemente befinden sich im Light DOM, werden aber visuell im Shadow DOM über `<slot>` angezeigt. Ihr Layout kann nicht direkt vom Shadow DOM beeinflusst werden.

CSS-Variablen (`--var`) hingegen können vom Shadow DOM nach außen vererbt und vom Light DOM genutzt werden.

Für ein slotted Element gilt grundsätzlich das CSS des Light DOMs, außer es wird über `::slotted(...)` gestylt.

Wenn ein `<template>`-Element das Attribut `shadowrootmode="open"` hat, erzeugt der Browser beim Parsen automatisch einen Shadow Root für das direkt übergeordnete Element. Der Inhalt des `<template>` wird in diesen Shadow Root übernommen und automatisch gerendert, sofern der Browser deklarative Shadow DOMs unterstützt.

Ausgewählte vererbte CSS Eigenschaften

Die folgenden Eigenschaften werden vererbt, und gelten auch im Shadow DOM von Web Komponenten; solange diese nicht überschrieben werden.

Kategorie	Vererbte CSS Eigenschaften
Text und Font	■ color ■ font, font-family, font-kerning, font-optical-sizing, font-size, font-size-adjust, font-stretch, font-style, font-variant, font-variant-caps, font-variant-ligatures, font-variant-numeric, font-variant-position, font-variation-settings, font-weight ■ letter-spacing, line-height ■ text-align, text-align-last, text-indent, text-justify, text-transform ■ visibility ■ white-space, word-break, word-spacing
Listen und Zähler	■ list-style, list-style-image, list-style-position, list-style-type
Table Layout	■ border-collapse, border-spacing, caption-side, empty-cells
Other	■ tab-size ■ quotes

Übung - Shadow und Light DOM

7.1. Shadow DOM und Slots verstehen

1. Welche Rolle spielt das `<template>`-Element in diesem Beispiel?
2. Was passiert beim Parsen dieses Dokuments mit dem Inhalt des *template*-Elements?
3. Warum wird der Text „Hallo Welt!“ in grün dargestellt?
4. Wie müsste man den Code ändern, damit der Text **nicht** mehr sichtbar ist, obwohl er im DOM vorhanden bleibt?

```
<x-demo>
  <template shadowrootmode="open">
    <style>
      ::slotted(p) { color: green; }
    </style>
    <slot></slot>
  </template>
  <p>Hallo Welt!</p>
</x-demo>
```

5. Ergänzen Sie den Code so, dass mehrere Slots mit Namen verwendet werden. Dazu geben Sie bei den Element im Light DOM den Namen mittels des slot attributes an (z. B. `<p slot="footer">`) und referenzieren diesen dann durch die Angabe des Namens im `<slot>` Elements (z. B. `<slot name="header">` und `<slot name="footer">`).

Nicht Behandelte Themen (Auszug)

- Counter
- Transformations (skalieren, drehen, ...)
- Animation
- (bisher nur grob) Flexbox (A guide to flex-box)
- Grid-Layout (A complete guide to CSS Grid)
- (mehr) CSS Tricks
- 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 in Kombination mit einer CSS Variable. Dieser Index kann dann für „die Berechnung“ von weiteren Werten verwendet werden.