

Web Engineering I - CSS

Elisabeth Kletsko – DHBW Mosbach WiSe 2024

Pseudoklassen Beispiele


- Pseudoklassen werden an einen Selektor direkt angehängt (ohne Leerzeichen)
- Fehlt der Selektor, wird automatisch `*` angenommen

Strukturelle Pseudoklassen

- Neben Zuständen wie `:hover`, `:visited` können auch Positionen von Elementen in CSS selektiert werden

Pseudoklasse	Beschreibung
<code>:first-child</code>	Elemente, die erstes Kind sind
<code>:last-child</code>	Elemente, die letztes Kind sind
<code>:nth-child(an+b)</code>	Elemente die $(a*n+b)$ -tes Kind sind (a, b sind ganze Zahlen ≥ 0)
<code>:nth-last-child(an+b)</code>	Elemente, die $(a*n+b)$ -letztes Kind sind
<code>:only-child</code>	Elemente, die einziges Kind des Elternelementes sind
<code>:first-of-type</code>	Elemente, die erstes Element dieses Typs sind
<code>:last-of-type</code> , <code>nth-of-type</code> , <code>:nth-last-of-type</code> , <code>:only-of-type</code>	Weitere Pseudoklassen

`nth-child(an+b)` genauer betrachtet

- `nth-child` selektiert jedes X-te Element
- Dabei können auch die Keywords `odd` und `even` als Argumente verwendet werden
- Die Formel `an+b`
 - `n` sind alle natürlichen Zahlen von 0 beginnend (0,1,2,3,4,5...)
 - `a` ist der Faktor mit welchem multipliziert wird `nth-child(2n)` jedes zweite Element usw.
 - `b` ist der Offset, welcher zu dem vorangehenden Produkt dazugaddiert wird
 - `:nth-child(2n+5)` → `2*0+5=5` , `2*1+5=7` ... es wird also das 5.,7., 9. Element usw. selektiert

Lab – Pseudoklassen vertiefen

- Öffne `lab-pseudoclasses/index.html` und notiere Dir für folgende Pseudoklassen-Selektoren, welche HTML Elemente markiert werden und warum

```
:first-child  
ul :first-child  
ul:first-child  
li:first-of-type  
li:nth-child(2n+1)
```

Lab – Pseudoklassen vertiefen Lösung

1. `:first-child` – Das `<h1>` Element und das erste `` Element, da dieses erste Kinder sind.

Da vor dem `:first-child` kein Selektor angegeben ist, wird der Universalselektor angewendet.

2. `ul:first-child` – Alle Nachfahren von einem `` Element – demnach das erste `` Element
3. `ul:first-child` – Keine Selektion, denn es sollen `` Elemente selektiert werden, die erste Kinder sind
4. `li:first-of-type` – Wieder das erste `` Element, da nur das erste Element eines Typs selektiert wird
5. `li:nth-child(2n+1)` – Das erste und dritte `` Element. Denn: $2 * 0 + 1 = 1$, $2 * 1 + 1 = 3$

Selektoren kombinieren – Nachfahrenselektor

- Kombinationen von CSS Selektoren werden in der Praxis gerne genutzt, um HTML Elemente *aufgrund Ihrer Position im DOM-Baum* auszuwählen
- **Fallbeispiel:** Wie selektieren wir das `span`-Element gezielt aus dem Header Bereich – ohne dabei `class` oder `id` zu verwenden?

```
<header id="header">  
  <h1>Some Header Text...</h1>  
  <p class="slogan">Was eine <span>Vorlesung</span>...</p>  
</header>
```

Selektoren kombinieren – Nachfahrenselektor

- Durch ein Leerzeichen wird ein **Nachfahrenselektor** gekennzeichnet

```
p.slogan span { /* Alle <span> Elemente innerhalb  
eines <p> mit der Klasse .slogan werden selektiert */  
  color: blue;  
}
```


Selektoren kombinieren – Nachfahrenselektor Beispiele

```
ul li {  
    /* Man möchte nur in ungeordneten Listen  
    die Aufzählungszeichen als Quadrat dargestellt haben */  
    list-style-type: square;  
}
```

- Nachfahrenselektoren sind nicht nur auf eine Ebene begrenzt!

```
ol li {  
    /* Alle li Elemente in einer geordneten Liste sollen Zahlen als Aufzählungszeichen haben*/  
    list-style-type: decimal;  
}  
ol ol li {  
    list-style-type: lower-alpha;  
}
```

Selektoren kombinieren – Kindselektoren

- Der Kindselektor in CSS ist durch ein `>` (bspw. `div > span`) gekennzeichnet
- Er ist präziser als der Nachfahrenselektor
 - `div span` würde *alle* `` Elemente selektieren innerhalb von `<div>` selektieren
 - `div > span` selektiert nur direkte Kindelemente von einem `div` (die 1. Ebene)
- Beispiel unter `lab-child-selector`
[➡ MDN Web Docs Child Selector](#)

Direkter Nachbarselektor (+)

- Werden zwei Selektoren durch den Kombinator + (Pluszeichen) verbunden, z. B. E + F, so wird das Element F nur dann angesprochen, wenn es im Elementbaum direkt auf ein E-Element folgt.

➡ Next Sibling Combinator

CSS

📋 </> Play

```
li:first-of-type + li {  
  color: red;  
  font-weight: bold;  
}
```

HTML

HTML

📋 </> Play

```
<ul>  
  <li>One</li>  
  <li>Two!</li>  
  <li>Three</li>  
</ul>
```

Result

📋 </> Play

- One
- **Two!**
- Three

Lab – Direkter Nachbarselektor (+)

```
div + p {  
  background-color: yellow;  
}
```

/* Welche Paragraphen werden selektiert?*/

```
<div>  
  <p>Paragraph 1 in the div.</p>  
  <p>Paragraph 2 in the div.</p>  
</div>
```

```
<p>Paragraph 3. After a div.</p>  
<p>Paragraph 4. After a div.</p>
```

```
<div>  
  <p>Paragraph 5 in the div.</p>  
  <p>Paragraph 6 in the div.</p>  
</div>
```

```
<p>Paragraph 7. After a div.</p>  
<p>Paragraph 8. After a div.</p>
```

Direkter Nachbarselektor (+)

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3. After a div.

Paragraph 4. After a div.

Paragraph 5 in the div.

Paragraph 6 in the div.

Paragraph 7. After a div.

Paragraph 8. After a div.

Geschwisterselektor (~)

- Werden zwei Selektoren durch den Kombinator ~ (Tilde) verbunden, z. B. E ~ F, so werden alle F-Elemente angesprochen, die im Elementbaum in derselben Ebene auf ein E-Element folgen – unabhängig davon, ob sich zwischen den Elementen weitere, im Selektor nicht genannte, Elemente befinden.

Geschwisterselektor (~)

➔ Was wird selektiert?

```
<p>The general sibling selector (~) selects all elements that are next siblings of a specified element.</p>  
<p>Paragraph 1.</p>  
<div>  
  <p>Paragraph 2.</p>  
</div>  
<p>Paragraph 3.</p>  
<code>Some code.</code>  
<p>Paragraph 4.</p>
```

Geschwisterselektor (~)

The general sibling selector (~) selects all elements that are next siblings of a specified element.

Paragraph 1.

Paragraph 2.

Paragraph 3.

Some code.

Paragraph 4.

Pseudo-Elemente Selektoren

Pseudo Elemente

Ein CSS Pseudo-Element wird genutzt, um bestimmte Bereiche eines CSS-Elements zu stylen

Bspw. den ersten Buchstaben eines Elements, Einfügen von Inhalten vor oder nach einem Element

[W3 Schools – Pseudo-Elemente Referenz](#)

Die Pseudo Elemente `::before` & `::after`

- Der `::before` und `::after` Selektor erlauben es Inhalte vor oder nach einem HTML-Element-Inhalt hinzuzufügen

```
p::after {  
  content: "Dieser Inhalt wird nach einem Text stehen;  
}
```

Lab – Pseudo Elemente

- Öffne `lab-pseudo-elements`
- Erstelle und importiere ein Stylesheet, welches
 - *vor* dem `cite` Element ein „Eine Weisheit aus dem Marvel-Universum“ setzt
 - Optional: *nach* dem `cite` Element das `svg` aus dem `lab-pseudo-elements/assets` einfügt.
- Nutze die eben gelernten Pseudo-Elemente

Lab – CSS Selektoren

- Öffne `lab-css-selectors` und beantworte welche Elemente selektiert werden und warum

```
h2
p li
ol li
h2 ~ p
h2 > p
p + p
h1 ~ h2
h2 > p
p + p
h1 ~ h2
ol *
ol > *
:first-child
:only-child
h2:first-of-type
.k
.p4
```

Lab – CSS Selektoren Lösung

h2 – Selektiert Elemente mit `#h2` und `#h3`
p li – Es wird nichts selektiert, da es keine li-Nachfahren bei einem p Element gibt
ol li – Selektiert alle li Nachfahren von einem ol Element #li2,li#3,li#4
h2 ~ p – Alle p Elemente die nach einem h2 Element auf der selben Ebene sind. #p2,#p3,#p4
h2 > p – Alle p Kinder von h2. Keine vorhanden.
p + p – Das p Element, was direkt auf ein anderes p Element folgt. Keine Selektion.
ol * – Alle Nachkommen von einem ol Element. #li1, #li2,#li3, #b
:first-child – Alle ersten Kinder in dem HTML Dokument; #h1,#li1
:only-child – Einzige Kinder eines Elements; Gibt es keine
h2:first-of-type – Erstes Element vom Typ h2; #h2
.k – Elemente mit Klasse „k“; #p4
.p4 – Elemente mit Klasse „p4“; IDs: Keine, da keine Klasse p4 existiert

Was kann ich mit CSS stylen?

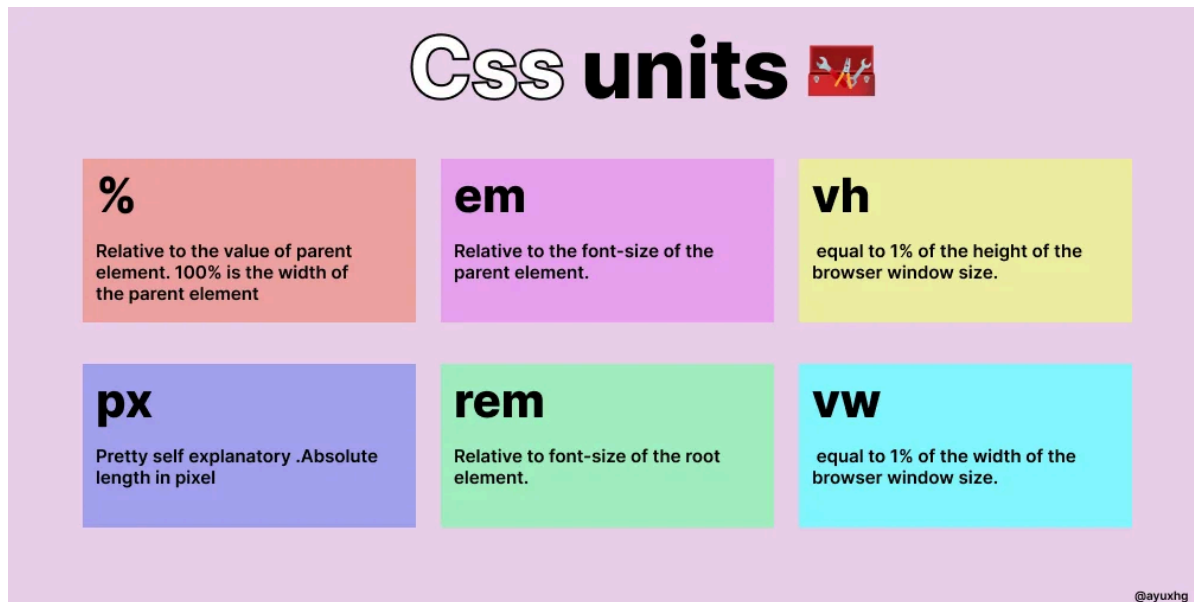
- Typografie [W3 Schools - CSS Text](#) [CSS Reference Typography](#)
- Farbe und Hintergründe [W3 Schools – CSS Backgrounds](#)
- Rahmen [W3 Schools – CSS Borders](#)
- Abstände [W3 Schools – CSS Margins](#), [W3 Schools – CSS Padding](#)
- Positionierung [W3 Schools – CSS Position](#)
- Listen [W3 Schools – Styling Lists](#)

Welche Werte gibt es in CSS?

- Maßeinheiten (wie `pt` oder `%`)
- Fest vorgeschriebene Werte (bspw. für die Attribute `font-family: sans-serif`, `font-size: large`)

Maßeinheiten in CSS

- CSS hat einige Maßeinheiten, im Allgemeinen *absolute* und *relative*
- Faustregel: Relative Einheiten eher für den Bildschirm, Absolute Einheiten für Druck



Vergleich von Maßeinheiten

Einheit	Art	Beschreibung	Verwendung
em	Relativ	Skaliert mit der Schriftgröße des Elternelements	Responsive Design
rem	Relativ	Bezieht sich auf die Schriftgröße des Root-Elements (<code><html></code>)	Konsistente Skalierung
%	Relativ	Basierend auf der Größe des Elternelements	Flexible Layouts
vh (Viewport Height)	Relativ	1vh entspricht 1% der Höhe des Ansichtsfensters	Layouts, die mit der Fensterhöhe skaliert werden
vw (Viewport Width)	Relativ	1vw entspricht 1% der Breite des Ansichtsfensters	Layouts, die mit der Fensterbreite skaliert werden
px (Pixel)	Absolut	Ein Pixel auf dem Bildschirm, abhängig von der Bildschirmauflösung	Standard für Weblayouts
cm (Zentimeter)	Absolut	Feste Maßeinheit, 1cm entspricht physisch 1cm	Printmedien
in (Zoll)	Absolut	1in entspricht 2.54cm	Printmedien
pt (Punkt)	Absolut	1pt = 1/72 Zoll, typischerweise für Schriftgrößen	Printmedien

CSS Vererbung – Vererbbare Eigenschaften

- In CSS gibt es *vererbbare* und *nicht vererbbare* Eigenschaften
- **Vererbbare Eigenschaften:**
 - Diese Eigenschaften werden automatisch von einem übergeordneten Element auf seine Kinder vererbt.
 - Beispiele: `color`, `font-family`, `line-height`, `text-align`, `visibility`.
 - Wenn zum Beispiel die Textfarbe (`color`) für ein übergeordnetes `<div>`-Element auf `blue` gesetzt ist wird diese Farbe automatisch auf alle Texte in den untergeordneten Elementen innerhalb dieses `<div>` angewendet, es sei denn, es wird eine andere Farbe speziell für die untergeordneten Elemente definiert.

CSS Vererbung – Nicht vererbbare Eigenschaften

- **Nicht vererbbare Eigenschaften:**
 - Diese Eigenschaften werden nicht automatisch vererbt und gelten nur für das Element, dem sie explizit zugewiesen wurden.
 - Beispiele: margin, padding, border, width, height, background.
Diese Eigenschaften betreffen in der Regel das Layout und die Dimensionierung von Elementen und müssen für jedes Element einzeln definiert werden, wenn spezifische Anpassungen erforderlich sind

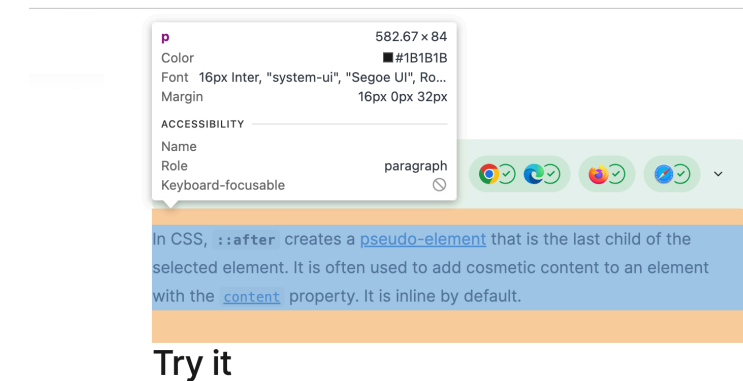
CSS Vererbung – `initial`

- Vererbung und Initialwerte: Wenn eine vererbbare Eigenschaft nicht für ein Element definiert ist und auch nicht durch Vererbung festgelegt wird, wird der Initialwert der Eigenschaft verwendet. Dieser Initialwert ist ein Standardwert, der vom W3C festgelegt wurde (z. B. `background-color: transparent`).
- `inherit` Schlüsselwort: In CSS kann man das Schlüsselwort `inherit` verwenden, um eine nicht vererbbare Eigenschaft von einem übergeordneten Element explizit zu vererben.

```
/* In diesem Fall übernimmt das <p>-Element die background-color des übergeordneten Elements, obwohl background-color normalerweise nicht vererbt wird */
```

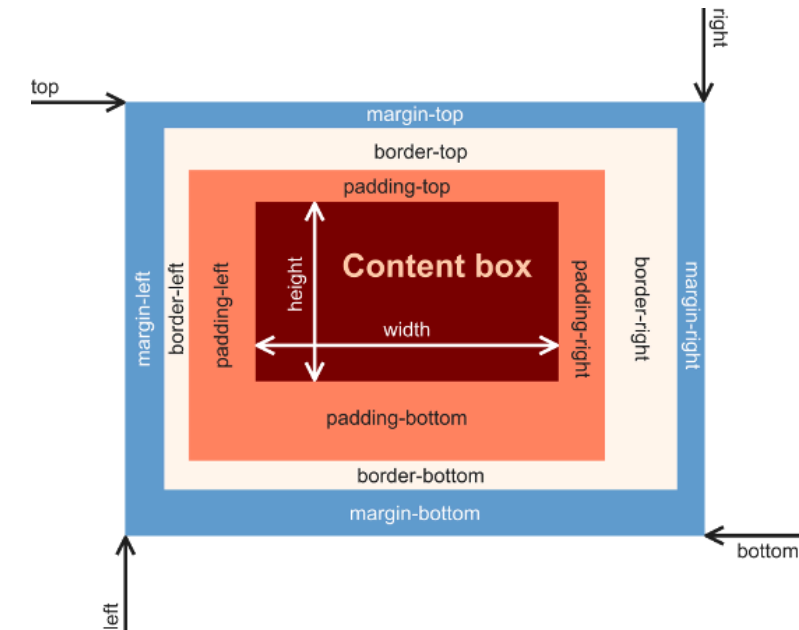
CSS Box Modell

- HTML Elemente können mit „Boxen“ verglichen werden
- Diese „Boxen“ sind nach dem selben Schema aufgebaut:
 - Es werden immer die Eigenschaften `width`, `height`, `border`, `padding`, `margin` deklariert
 - *Ausnahme*: Inline-Elemente wie `` kennen kein `width`, `height`, `margin-top`, `margin-bottom`



CSS Box Modell

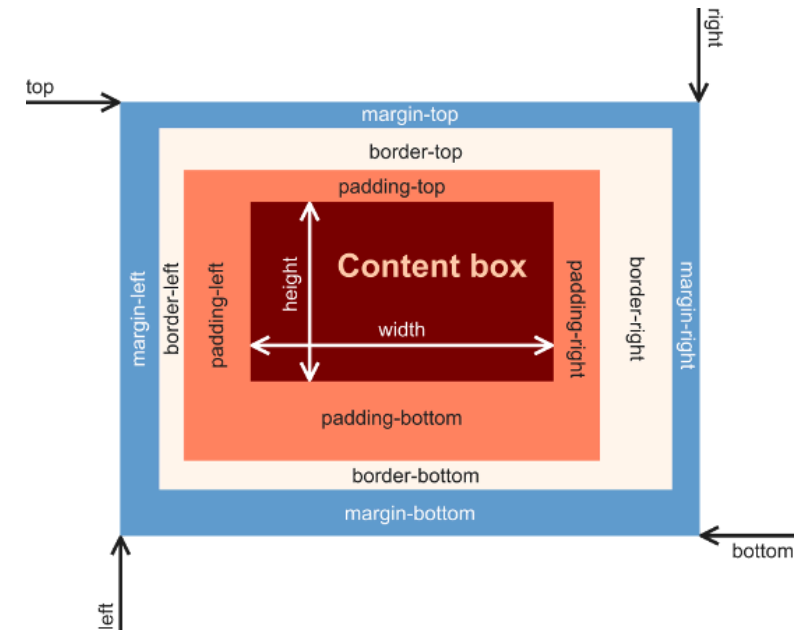
Englisch	Deutsch
box model	Box-Modell, Kastenmodell, Kästchenmodell
content	Inhalt, Inhaltsbereich
width	Breite, Inhaltsbreite
padding	Innenabstand, Polsterung, Auffüllung, Füllung
border	Rahmen, Rahmenlinie
margin	Außenabstand



CSS Box Modell – content

- Der Inhaltsbereich, die Größe wird durch die Eigenschaften `width` und `height` bestimmt
- **!!** Inline-Elemente (wie ``) kennen weder `width` noch `height`
- [MDN Docs](#) `width`
- [MDN Docs](#) `height`

```
div {  
  width: 100px;  
  height: 100px;  
}
```



CSS Box Modell – padding

- Der Inhaltsbereich (`content`) ist von einer „Polsterung“ umgeben (`padding`)
- Das `padding` übernimmt die Hintergrundfarbe des Inhaltsbereichs
- `padding` funktioniert bei Block- und Inline Elementen identisch
- Die Eigenschaften sind `padding-top`, `padding-bottom`, `padding-right`, `padding-left`
- [MDN Web Docs padding](#)

```
div {  
  padding: 10px 10px 10px 10px; /* top - right - bottom - left, abkürzbar mit padding: 10px */  
}
```

```
div {  
  padding: 10px 0; /* top & bottom hat 10px padding, right & left 0*/  
}
```


CSS Box Modell – border

- `border` ist der Rahmen um das `padding`
- Der `border` kann mit Breite `width`, Linienart (`style`) und Farbe `color` gestyled werden
- Die Regeln für `border` gelten auch für Inline-Elemente
- [MDN Web Docs border](#)

```
div {  
  border: 1px solid black;  
}
```

CSS Box Modell - `margin`

- Jede „Box“ kann einen Außenabstand `margin` haben.
 - Dieser zeugt Abstand *zwischen* Elementen
- Die zu definierenden Eigenschaften sind `margin-top`, `margin-right`, `margin-bottom`, `margin-left`
 - **Erinnerung:** Inline-Elemente wie `` kennen kein `width`, `height`, `margin-top`, `margin-bottom`
- Die Eigenschaften können im Vergleich zu `padding` und `border` negative Werte haben
- Da der Abstand *außerhalb* der Box liegt, wird die Hintergrund-Farbe des umliegenden/ Eltern-Elements übernommen
- [MDN Web Docs](#) `margin`

CSS Box Modell – `margin` Collapse

- Manchmal kann es passieren, dass `margin` von HTML Elementen sich überlappen
- Diese werden dann zusammengefasst
- Wichtig dabei sind unter anderem folgende Regeln:
 - Nur **vertikale** `margins` können „collapsen“
 - Nur benachbart `margins` können „collapsen“
 - Das größere `margin` „gewinnt“
 - Negative `margin` können auch „collapsen“
- Mehr dazu: [Josh Comeau](#)

CSS Box Modell – padding & margin Kurzschreibweise

- Man muss nicht immer alle 4 Werte beim padding und margin ausschreiben.

```
margin: -3px; /* Alle vier Seiten */  
margin: 5% auto; /* top & bottom | left & right */  
margin: 1em auto 2em; /* top | left & right | bottom */  
margin: 2px 1em 0 auto; /* top | right | bottom | left */
```

CSS Box Modell - Gesamtbreite einer Box berechnen

- Wir haben gelernt, `width` bezieht sich nicht auf die Gesamtbreite einer Box
 - sondern nur auf den Inhalt
- Für die Gesamtbreite müssen wir `content`, `padding`, `border`, `margin` in Betracht ziehen

CSS Box Modell - Gesamtbreite einer Box berechnen

```
div {  
  width: 720px;  
  padding: 20px;  
  border: 0;  
  margin: 10px;  
}
```

- Was ist die Breite?
- Hilfestellung durch interaktives Box Diagram

CSS Box Modell - Gesamtbreite einer Box berechnen

- `width` und `height` sagen nicht automatisch aus, wie breit und hoch eine „Box“ ist (bis jetzt!)
- Die Gesamtbreite einer Box setzt sich aus `width`, `padding`, `border` und `margin` zusammen

CSS Box Modell - Gesamtbreite einer Box berechnen

Berechnung	Beispiel
<code>width</code>	720px
<code>padding-right</code>	20px
<code>padding-left</code>	20px
<code>border-right-width</code>	0px
<code>border-left-width</code>	0px
<code>margin-right</code>	10px
<code>margin-left</code>	10px
Gesamtbreite der Box	780px

➡ Für die Höhe gilt das selbe Prinzip

CSS Box Modell – border-box

- Es gibt ein Alternatives Modell zum herkömmlichen Box Modell (content-box) – border-box
- Definiert man eine width bei border-box , so ist das die Gesamtbreite der Box und nicht nur des Inhaltsbereichs
 - padding und border sind demnach schon in der width inkludiert
- Essentiell für responsive Layouts
- [MDN Web Docs box-sizing](#)

```
div {  
  box-sizing: border-box; /* Standard-mäßig content-box */  
}
```

Lab – CSS Box Modell

- Versuche ein Gefühl für `content-box` und `boder-box` zu erhalten mit dem interaktiven [Box Modell](#)

CSS `display` Attribut

- HTML Elemente sind Boxen
 - Diese Boxen können verschieden dargestellt (displayed) werden
- Wir behandeln erstmal: `inline`, `block`, `inline-block`, `none`

```
display: block;
```

CSS **display** Attribut

Eigenschaft	Beschreibung
inline	Elemente werden in einer Zeile dargestellt (auch wenn sie eigentlich Block-Elemente sind). Sie nehmen nur so viel Platz ein, wie ihr Inhalt benötigt. margin und padding wirken sich nicht auf die umgebenden Elemente aus.
block	Elemente beginnen immer auf einer neuen Zeile und nehmen die gesamte Breite ihres Containers ein. Sie können Höhe und Breite festlegen. margin und padding werden sowohl vertikal als auch horizontal respektiert.
inline-block	Elemente werden in einer Zeile dargestellt. Sie erlauben die Festlegung von Höhe und Breite. margin und padding wirken sich sowohl vertikal als auch horizontal aus.
none	Das HTML-Element wird aus dem Layout entfernt und ist nicht sichtbar. Es nimmt keinen Platz im Layout ein.

Lab – CSS `display` Attribut ausprobieren

- Öffne `lab-display/styles.css` und verändere bei den `p` Elementen die `display` Eigenschaft zu `block`, `inline`, `inline-block`, `none`
- Versuche die `width`, `height`, `margin` Eigenschaften zu setzen, kannst du die vorherigen Beschreibungen nachvollziehen?

CSS Positionierung

- Standardmäßig folgen HTML Elemente im sichtbaren Bereich des Browserfensters dem *document flow*
 - Die Elemente werden nach dem anderen Element platziert
- Wir erinnern uns: HTML Elemente kommen entweder als *Block*-Elemente, *Inline*-Elemente, oder *Inline-Block*-Elemente vor

Lab – Document Flow verstehen

- Für die nachfolgenden Slides öffne `lab-document-flow/index.html`
- Das CSS erzeugt 3 `div` Elemente mit weißem Hintergrund
- Diese nehmen die gesamte Breite an, da `<div>` Elemente Block Elemente sind.
 - Block Elemente nehmen immer die verfügbare Breite der umgebenden Box ein, in dem Fall also von `<body>`



Lab – Document Flow verstehen

- Setze die `width` der `div` Boxen auf 25%.
- Alles andere bleibt unverändert – $3 * 25 = 75$
 - *Stehen die Boxen nebeneinander?*

Lab – Document Flow verstehen

- Nein, da Block Elemente einen integrierten Zeilenumbruch beinhalten

```
div {  
  background-color: #fff;  
  padding: 10px; /* 10 px padding an allen Seiten */  
  border: 3px solid #8b0000;  
  border-radius: 5px;  
  margin: 5px;  
  width: 25%;  
}
```



CSS Positionierung – `position` -Attribut

- Mit dem `position` Attribut kann man in CSS in das *document flow* Verhalten eingreifen und die Positionierung der HTML Elemente ändern
- Default-mäßig ist der Wert für `position` `static`
- Bei `position: static` haben die Attribute `top`, `bottom`, `left`, `right` keine Auswirkung!

CSS Positionierung – `position: relative`

- Die relative Positionierung (`position: relative`) macht zwei Dinge in CSS
 1. Das HTML-Element (die „Box“) wird von dem normalen document flow verschoben
 2. Die ursprüngliche Position des Elements (der „Box“) wird als „geschützt“ markiert
- D.h. nachfolgende Elemente „rutschen“ nicht an diese Stelle
- Attribute `top`, `bottom`, `left`, `right` können nun gesetzt werden.

Lab – position: relative verstehen

- Setze `top` und `right` Attribute des ersten `div` Elements (Tipp: Nutze `first-child`)
- Verleihe dem `div` die `position: relative`
- Verändern sich Box 2 und 3? Warum (nicht)?

Lab – position: relative verstehen

- Box 2 und 3 verändern sich nicht, denn bei relativer Positionierung bleibt die ursprüngliche Position des Elements (in diesem Fall dem ersten `div` / Box 1) geschützt und darf nicht von den nachfolgenden Elementen beansprucht werden.

```
div:first-child {  
  background-color: #e3c0c0;  
  position: relative;  
  top: 25px; /* An die normale Position werden 25px von oben hinzugefügt */  
  right: 25px;  
}
```

CSS Positionierung – `position: absolute`

- Im Gegensatz zu `position: relative` nimmt `position: absolute` das Element aus dem document flow „heraus“.
 - Alle anderen Elemente verhalten sich so, als wäre das „absolute“ Element nicht da
 - Sie können demnach auch dessen Position einnehmen

CSS Positionierung – `position: absolute`

- Die genaue Position eines HTML Elements bei `position: absolute` wird mit `top`, `right`, `bottom`, `left` bestimmt.
- Die absolute Positionierung eines Elements bezieht sich dann dabei auf das nächste umgebende Element mit `position relative`, `absolute` oder `fixed`
 - Sollte dieses nicht gegeben sein, ist dies das oberste `html` Element.
- Es kann passieren, dass sich `absolut` positionierte Elemente überlappen. Mit der Eigenschaft `z-index` kann man festlegen, welche Elemente vorne und welche hinten liegen.

Lab – position: absolute verstehen

- Ändere vom `div:first-child` die position von `relative` zu `absolut`
- Was passiert?

Lab – position: absolute verstehen Lösung

- Box 1 ist rechts außen, Box 2 und 3 rutschen auf dessen ursprüngliche Stelle hoch

```
div:first-child {  
  background-color: #e3c0c0;  
  position: absolute;  
  top: 25px; /* An die normale Position werden 25px von oben hinzugefügt */  
  right: 25px;  
}
```

Lab – `position: absolute` anwenden

- Gehe zu `lab-position-notepad` und passe die `styles.css` so an, so dass das Element mit `.stock_badge` rechts oben in der Ecke des Elements mit `.container_card` erscheint.

Lab – position: absolute Lösung

```
.container_card {  
    position: relative;  
}  
  
.stock_badge {  
    position: absolute;  
    top: 0px;  
    right: 0;  
}
```

CSS Positionierung – `position: fixed`

- Verhalten wie `position: absolute`, nur, dass es nicht mitscrollt
- Darüber hinaus sind `position: absolute` Elemente *relativ zu einem umgebenden Element* im Dokument und scrollen mit
- Das umgebende Element für `position: fixed` Elemente ist *immer* das Browserfenster („Viewport“) und nicht das Stammelement `html` innerhalb des Fensters (wie bei `position: absolut`, sobald es kein „umgebendes“ Element gibt)

Lab – `position: fixed` ausprobieren

- Öffne die `lab-position-fixed/index.html` im Browser
- Scrolle und schaue dir das Verhalten der fixierten Box an
- Verändere die `position` zu `absolute` – Was fällt dir auf?
 - Warum ist die Box nicht in der vertikale „Mitte“ vom Text?

CSS Spezifität – Wer styled was?

- Stylesheets werden über die Zeit länger
 - Das Ergebnis: Es gibt CSS Regeln, welche sich zum Teil widersprechen
 - *Wie entscheidet der Browser in solchen Konflikten?*

CSS Spezifität (eng. „specificity“)

Das „Punktesystem“ eines Browsers welches bestimmt, welcher Selektor der wichtigste ist

➔ Weitere Infos [Kulturbanause_ CSS Spezifität](#)

➔ Weitere Infos [Web Dev Simplified YouTube](#)

CSS Spezifität – Punkteverteilung

- Die Punkte werden jeweils addiert
- Ein `<p class="infobox">...</p>` hat somit eine Spezifität von $1 + 10 = 11$

Selektor-Typ	Beispiel	Punkte Summe
Einfacher Typselektor	<code>p</code>	1
Klasse	<code>.infobox</code>	10
Pseudoklasse	<code>:visited</code>	10
ID	<code>#navibereich</code>	100
Attribut <code>style=""</code>	<code>style="color:red;"</code>	1000

Die Annotation !important;

- Wenn der Browser eine bestimmte CSS Regel anwenden will, egal welche Spezifität berechnet wurde, so kann man die Eigenschaft `!important;` verwenden;

```
h2 { color: red!important ; } /* Die Leerstelle ist wichtig */
```


Beispiele für die Punktebewertung

- Sollte die Punktzahl *gleich* sein, so wird die Regel priorisiert, die weiter unten im Stylesheet steht

Selektor	Beschreibung	Punkte	Gesamt
<code>body</code>	Typselektor	1	1
<code>h1, h2</code>	gruppierter Typselektor	1	1
<code>a:visited</code>	Typ+Pseudoklasse	1+10	11
<code>.infobox</code>	Klasse	10	10
<code>p.infobox</code>	Typselektor + Klasse	1 + 10	11
<code>#navibereich</code>	ID	100	100
<code>nav#navibereich</code>	Typselektor + ID	1 + 100	101
<code>#navibereich a</code>	ID + Typselektor	1+100	101
<code>#navibereich a:visited</code>	ID + Typselektor + Pseudoklasse	100 + 1 + 10	111
<code><p style="color: red;"></code>	Inline Attribut <code>style</code>	1000	1000

CSS Spezifität – Gleiche Punktzahl Beispiel

```
h1 { /* Spezifität 1 */  
    color: blue;  
}  
h1,h2 { /* Spezifität 1 */  
    color: red; /* Diese Regel wird angewandt, da sie weiter unten steht */  
}
```

CSS Spezifität – Best Practices

- Spezifität möglichst gering halten – keine ID (`#id`) dort verwenden, wo sie nicht notwendig ist
- Inline Styles vermeiden – diese können nur mit einem `!important;` überschrieben werden
- `!important` sparsam nutzen – erschwert irgendwann das Debugging

Lab – Specifity Wars

- Ein Spiel zur Abwechslung :)
- [Specifity Wars](#)
- [Specifity Calculator](#)

Die CSS Kaskade – Wer „malt“ zuerst?

- Nachdem der Browser einen DOM Baum erstellt hat, müssen die enthaltenen Elemente gestyled werden
 - Hierfür muss für jedes Element eine CSS-Eigenschaft mit **eindeutigem** Wert gefunden werden
- Hierbei helfen dem Browser 3 Konzepte: **Kaskade**, **Vererbung** und **Standardwert**

CSS Kaskade

The cascade is an algorithm that defines how user agents combine property values originating from different sources.

Die CSS Kaskade – Algorithmus

- Um den eindeutigen CSS Wert zu ermitteln, beginnt der Browser damit, alle **relevanten** Deklarationen zu sammeln für ein HTML Element. Hierbei können folgende Fälle eintreten:
 1. **Keine relevanten Deklarationen:** Man schaut, ob das Element einen Wert erben kann. Falls nicht, wird der Standardwert genommen
 2. **Eine relevante Deklaration:** Der Browser nimmt die gefundene Deklaration
 3. **Mehrere relevante Deklarationen:** Weitere Sortierungen in den Bereichen **Wichtigkeit, Spezifität und Reihenfolge**

Die CSS Kaskade – Wichtigkeit, Spezifität und Reihenfolge

- Die Kaskade kann man in drei Bereiche unterteilen – Wichtigkeit, Spezifität und Position der Deklaration.
- **Wichtigkeit** – handelt es sich um eine `animation`, `transition`, `!important` Regel? Woher stammen die Stylesheets?
 - i. Deklarationen in Autoren (= Web Designer)-Stylesheets haben für den Algorithmus die höchste Priorität
 - ii. Benutzer-Stylesheets überschreiben Eigenschaften von Browser-Stylesheets (bspw. Dark Mode, LRS-freundliche Schriftarten usw)
 - iii. Browser-Stylesheets besitzen vorgegebene Formatierungen für HTML-Dokumente. Dieses Stylesheet hat die niedrigste Priorität und bildet die Basis für jedes HTML Dokument

Die CSS Kaskade – Bereiche

- **Spezifität:** Siehe vorherige Folien
- **Reihenfolge:** Sollte die gleiche Spezifität vorliegen, „gewinnt“ das letzte definierte Element

Die CSS Kaskade – Ablauf

- Sammeln aller Deklarationen für das zu stylende Element
- Sortieren der Deklarationen nach Herkunft und Wichtigkeit (entnommen aus [W3 Cascade](#))

1. Deklarationen im Browser-Stylesheet (niedrigste Priorität)
2. Deklarationen des Benutzers
3. Deklarationen des Autors
4. animations
5. !important-Deklarationen des Autors
6. !important-Deklarationen des Benutzers
7. !important-Deklarationen im Browser-Stylesheet
8. transitions (höchste Priorität)

Responsive Web Design (RWD)

Responsive Web Design

Responsive Frontends passen sich zu *jedem* beliebigen Zeitpunkt an die Umgebung an, wo sie genutzt werden

- Pionier: Ethan Marcotte mit Blog Eintrag [Responsive Web Design](#)
- Wichtiger denn je, aufgrund der steigenden Anzahl und Diversität von Geräten, mit welchen man Inhalte im WWW aufruft
- [Patterns und Resources für RWD](#)
- Im Folgenden zwei Techniken, um RWD näher zu kommen

CSS @media queries

- Um die sehr große Bandbreite an Endgeräten und deren Bildschirmgrößen optimal bedienen zu können, gibt es die sogenannte Media Queries (Medienabfragen)
- Mittels Media Queries lassen sich Layouts erstellen, die an die spezifischen Eigenschaften der Endgeräte angepasst sind (bspw. landscape, portrait, screen, print)
- [CSS Tricks Media Queries Guide](#)

Syntax von Media Queries

```
@media (Bedingung) {  
    /* CSS-Regeln */  
}  
/* Typisches Beispiel */  
@media (min-width:100px) and (max-width: 500px) {  
    /* ... */  
}  
@media only screen and (orientation: landscape) {  
    /* ... */  
}
```

Lab CSS @media queries

- In `lab-mediaqueries`

Erstelle eine einfache Webseite, die folgende Anforderungen erfüllt:

Layout für Mobilgeräte:

Verwende eine maximale Breite von 600px.
Setze den Hintergrund auf hellblau.

Layout für Tablets:

Verwende eine minimale Breite von 601px und eine maximale Breite von 900px.
Setze den Hintergrund auf hellgrün.

Layout für Desktops:

Verwende eine minimale Breite von 901px.
Setze den Hintergrund auf hellorange.

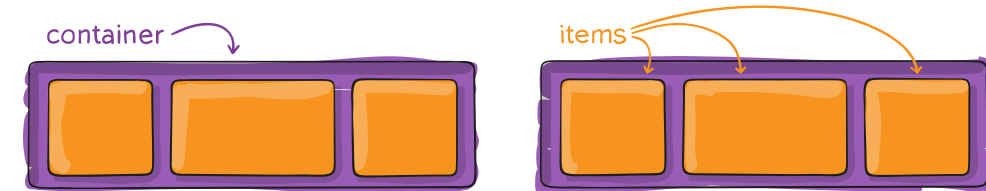
CSS flexbox

Flexbox

Flexboxen (aktiviert mit `display: flex`) erlauben in CSS Elemente in *einer* Dimension anzuordnen (`row` oder `column`).

Dabei gibt es einen **Flex Container** und **Flex Elementen** (die Kinder des Containers)

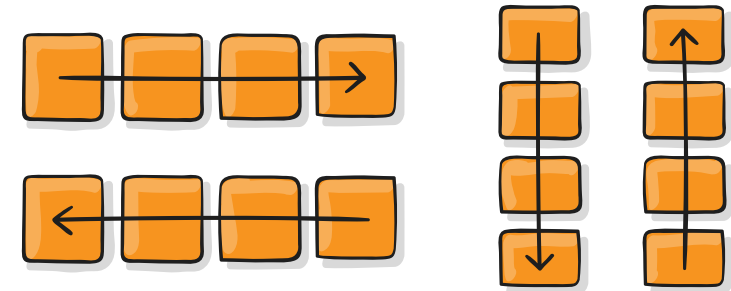
- „flex“ kommt daher, dass die Kinderelemente ihre Breite / Höhe dem Platz entsprechend anpassen
- Der Container ist dynamisch veränderbar



Flexbox Container – Anordnung

- Um die Anordnungs-Richtung der Elemente zu bestimmen, verwendet man die `flex-direction` Eigenschaft
- Wir erinnern uns: `row` oder `column`

```
.flex-container {  
  display: flex;  
  flex-direction: column; /* Default-mäßig "row" */  
}
```



Flexbox – justify-content

- Die **justify-content** Eigenschaft bestimmt, wie die Flex-Elemente entlang der Hauptachse (horizontal bei **row**, vertikal bei **column**) innerhalb des Flex-Containers verteilt werden.

```
.flex-container {  
  display: flex;  
  justify-content: center; /* Zentriert die Elemente */  
}
```

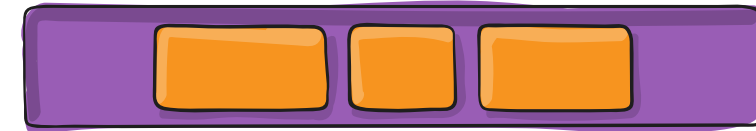
flex-start



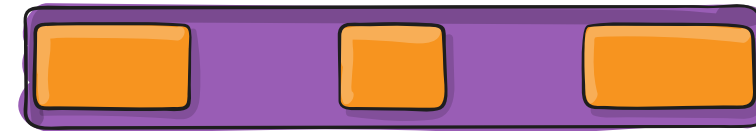
flex-end



center



space-between



space-around



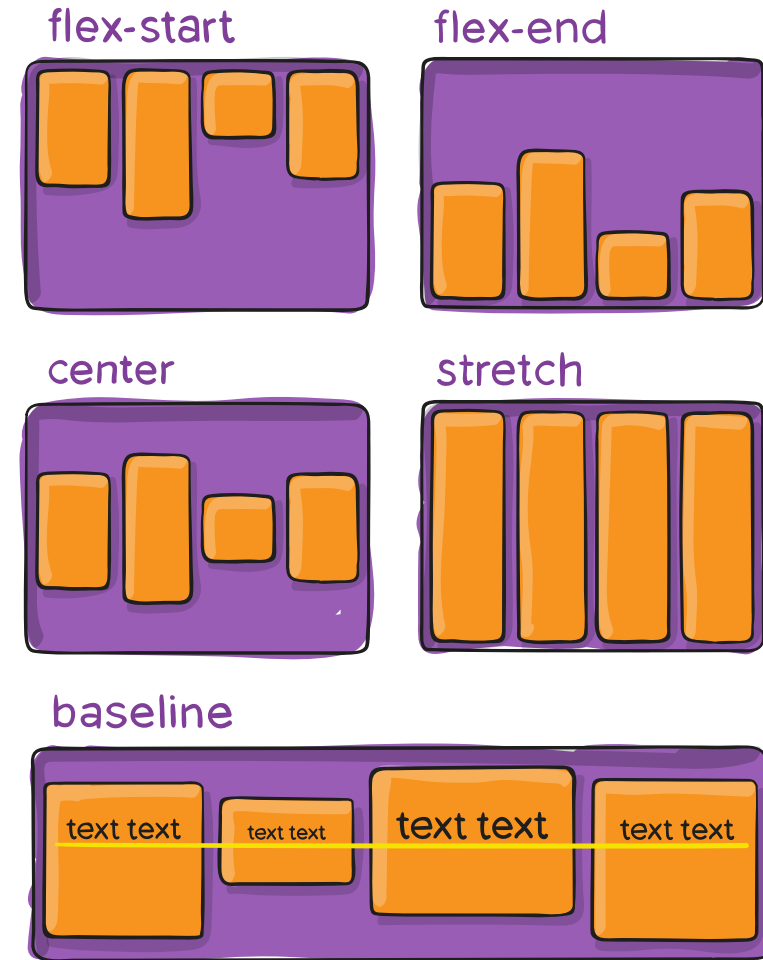
space-evenly



Flexbox – align-items

- Die **align-items** Eigenschaft bestimmt, wie die Flex-Elemente entlang der Querachse (vertikal bei **row**, horizontal bei **column**) innerhalb des Flex-Containers ausgerichtet werden.

```
.flex-container {  
  display: flex;  
  align-items: stretch;  
  /* Elemente dehnen sich, um den Container auszufüllen */  
}
```



Lab – Zentriere das `div`

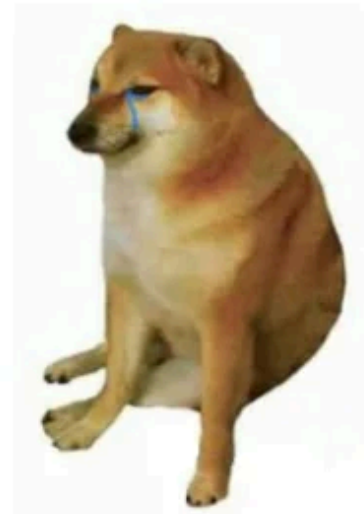
- Gehe zu `lab-center-div`
- Zentriere das `div` mit Hilfe von Flexbox Eigenschaften

Software Engineer



I write Software
that helps the People

Web Developer



How to center div?

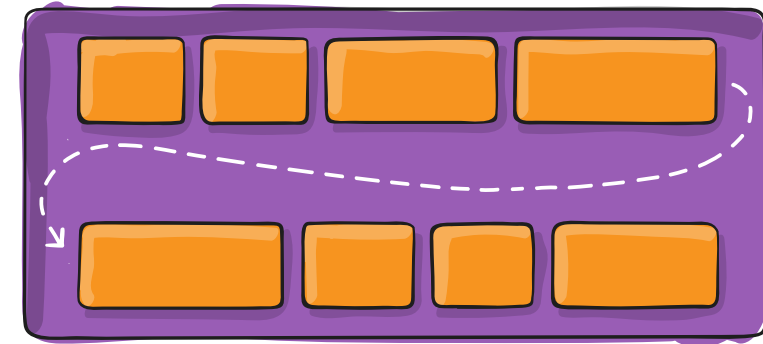
Lab – Zentriere das `div` Lösung

```
#container {  
    /* ... */  
    display: flex;  
    align-items: center;  
    justify-content: center;  
}
```

Flexbox – flex-wrap

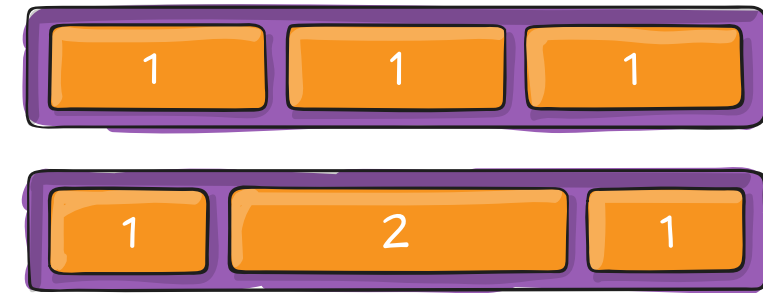
- Per default wird der Flexcontainer versuchen, alle Elemente in eine Reihe zu packen
- Durch die `flex-wrap` Eigenschaft kann man einen Umbruch erlauben, sollte der Platz nicht ausreichen

```
.container {  
  /* nowrap: Alle Elemente auf einer Zeile  
  wrap: Überschüssige Elemente in neuen Zeilen (top to bottom)  
  wrap-reverse: Überschüssige Elemente in neuen Zeilen (bottom to top)  
  */  
  flex-wrap: nowrap | wrap | wrap-reverse;  
}
```



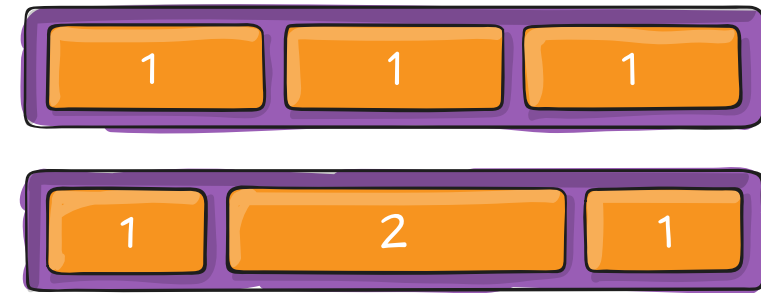
Flexbox – `flex-grow`

- `flex-grow` beschreibt das *relative* Wachstums-Verhältnis eines Elements zu den anderen Elementen im selben Flexbox-Container (bspw. ein Wert von `0` sagt aus, dass das Element sich nicht ausdehnen soll, obwohl Platz vorhanden ist)
- `flex-grow: 0` (Standardwert): Das Element wächst nicht, wenn zusätzlicher Platz im Container vorhanden ist.
- `flex-grow: 1`: Das Element wächst, um den verfügbaren Platz gleichmäßig mit anderen Flex-Items zu teilen, die ebenfalls `flex-grow: 1`



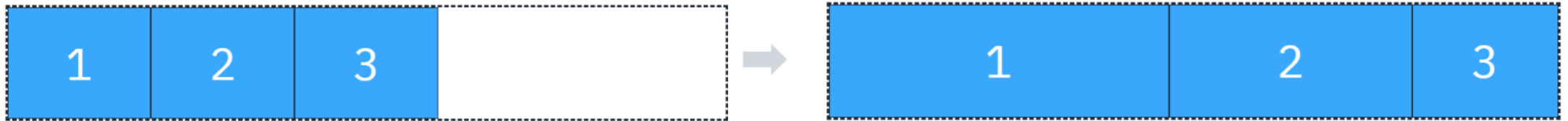
Flexbox – flex-grow

- flex-grow: 2 oder höher: flex-grow bezieht sich nur auf den verbleibenden Platz, nachdem alle Elemente dem Container hinzugefügt wurden. Da unsere 3 Elemente standardmäßig 60 % des Containers einnehmen, bleibt nur 40 % des Platzes übrig, um diesen zwischen den Elementen zu verteilen.



Flexbox – **flex-grow** Beispiel

- Um den verbleibenden Platz zu berechnen, addieren wir die **flex-grow** - Werte der Elemente im Container ($1 + 2 = 3$) und teilen dann den flex-grow jedes Elements durch diese Summe. (1.child $2/3$ und 2. child $1/3$)



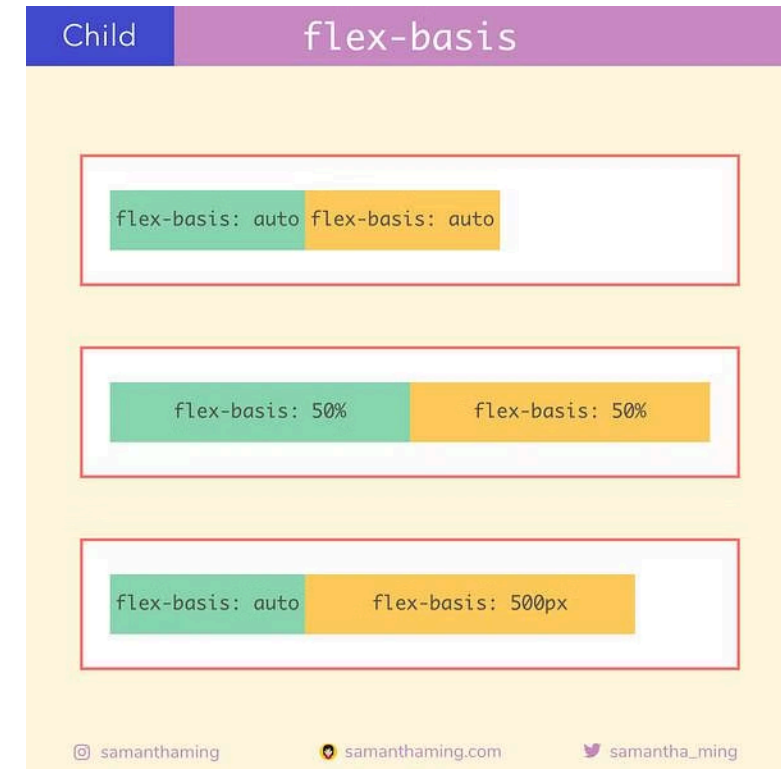
```
.flex-item:nth-child(1) {  
  flex-grow: 2;  
}  
.flex-item:nth-child(2) {  
  flex-grow: 1;  
}
```


Flexbox – `flex-shrink`

- `flex-shrink` beschreibt das *relative* "Schrumpf"-Verhältnis eines Elements zu den anderen Elementen im selben Flexbox-Container
- `flex-shrink: 0` : Das Element wird nicht schrumpfen, auch wenn der verfügbare Platz im Container knapp wird.
- `flex-shrink: 1` : Das Element wird schrumpfen, wenn es nötig ist, und zwar proportional zu anderen Elementen, die ebenfalls schrumpfen dürfen.
- `flex-shrink: 2` oder mehr: Das Element wird doppelt (oder entsprechend dem Wert) so stark schrumpfen wie ein Element mit `flexshrink: 1`.

Flexbox – flex-basis

- `flex-basis` legt die Basisgröße eines Flex-Elements fest, bevor der verbleibende Platz im Container verteilt wird.
- `auto` : Die Größe wird durch Inhalt oder `width` / `height` bestimmt.
- 0 oder 0%: Das Element hat keine anfängliche Größe, es wird nur durch flex-grow und flex-shrink beeinflusst.
- `<Längenwert>`: Ein fester Wert definiert die Ausgangsgröße, unabhängig vom Inhalt.



Flexbox – flex

- `flex-grow`, `flex-shrink`, `flex-basis` können zusammengefasst werden mit der Eigenschaft `flex`
- Der default ist dabei `0 1 auto`

```
.item {  
  flex: 0 1 auto; /* <flex-grow> <flex-shrink> <flex-basis> */  
  /* flex-shrink und flex-basis sind optional */  
}
```

- [MDN Web Docs](#) `flex`

Lab – flex

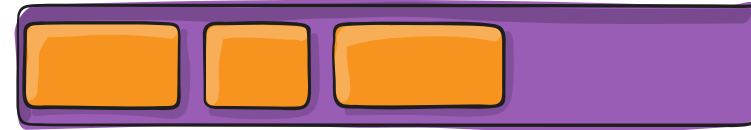
- Gehe zu `lab-flex-basis` und passe das CSS so an, dass der Menüpunkt „Weihnachtszubehör 🎄“ immer 2/5 der Gesamtbreite des Menüs einnimmt

Flexbox gap

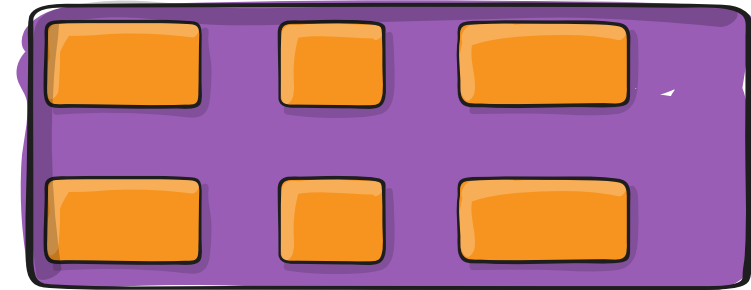
- Die `gap` Eigenschaft ist vergleichbar mit `margin`, bezieht sich jedoch nur auf den Abstand zwischen den Elementen

```
.container {  
  display: flex;  
  ...  
  gap: 10px;  
  gap: 10px 20px; /* row-gap column gap */  
  row-gap: 10px;  
  column-gap: 20px;  
}
```

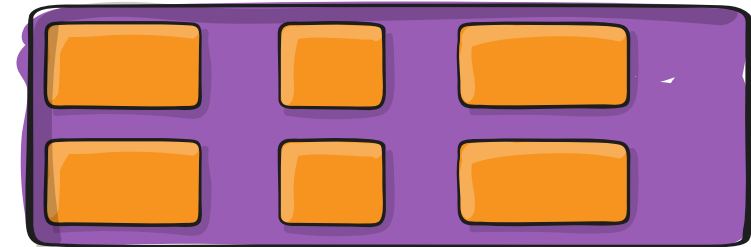
gap: 10px



gap: 30px



gap: 10px 30px



Abschließendes zu Flexbox

- Kein einfaches Thema und es braucht Zeit, reinzukommen
- [CSS Tricks](#) gibt eine schöne Übersicht über die Syntax