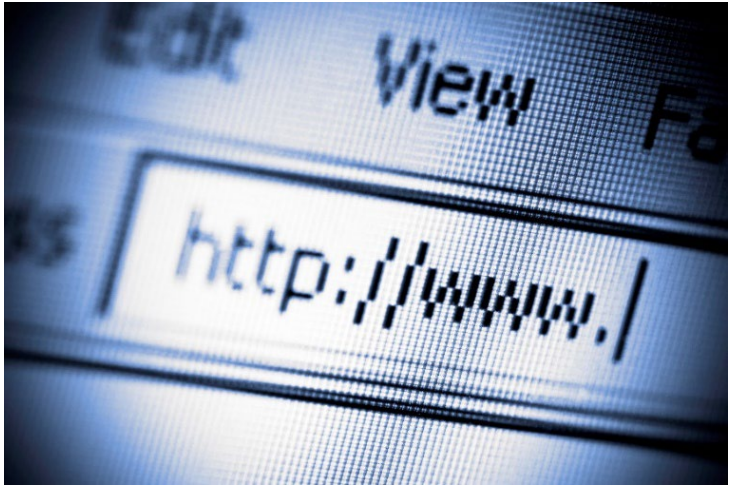
HSLU Hochschule Luzern

Web-Technologien

CSS-Layouts



- Absolute Positionierung
- Umfliessende Positionierung
- CSS Flexbox
- CSS Grid

Letzte Aktualisierung: 12. Oktober 2022

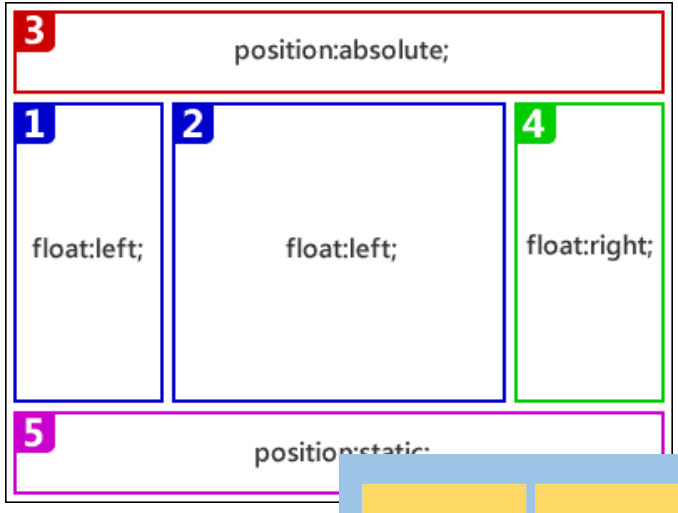
FH Zentralschweiz

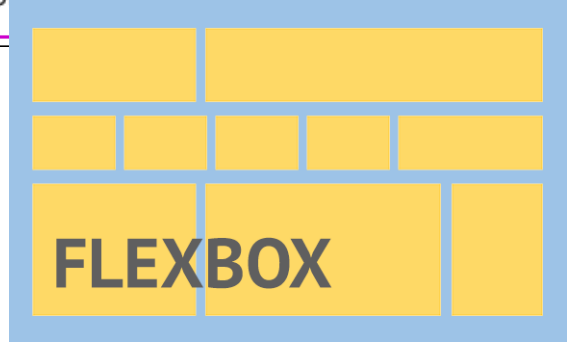
Lernziele

- Kennen und Anwenden können der CSS-Positionierungstechniken:
  - Absolut und relativ
  - Fliessend
  - Flexbox
  - Grid

CSS bietet diverse Möglichkeiten zur Positionierung

- Absolute Positionierung
- Fliessende Positionierung
- CSS-Flexbox
- CSS-Grid





Absolute und relative Positionierung

- Exakte Vorgabe der Stelle, an welcher ein Element dargestellt werden soll.

- **Verwendung:** Nur wenn keine andere Positionierung-Methode funktioniert.
  - Eigene programmatische Positionierung mittels JavaScript.
  - Überlappende Elemente (z.B. als immer sichtbarer Chat- oder Info-Link).

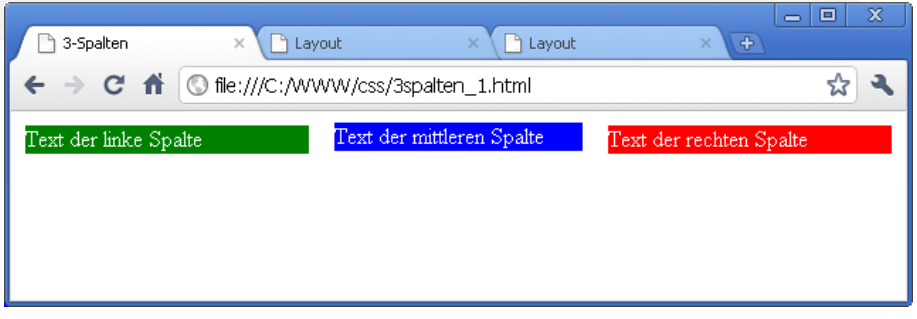
- Bei der absoluten Positionierung wird ein Element aus dem Dokumentenfluss entfernt => ungünstig bei Responsive Web Design.

Absolute Positionierung: Erstes Beispiel

3 Container, mittlere Spalte mit dynamischer Breite

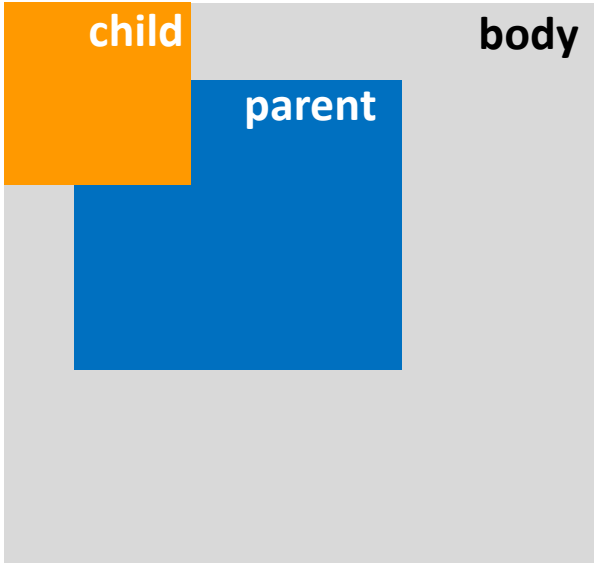
```
<style>
  #links  {position: absolute; left: 10px;  top: 10px; width: 200px;}
  #rechts {position: absolute; right: 10px; top: 10px; width: 200px;}
  #mitte  {margin-left: 220px; margin-right: 220px;}
</style>

<body>
  <div id="links">Text der linke Spalte</div>
  <div id="mitte">Text der mittleren Spalte</div>
  <div id="rechts">Text der rechten Spalte</div>
</body>
```



Nullpunkt der absoluten Positionierung

Elemente mit `position: absolute` werden in Bezug auf den nächsten positionierten Vorgänger (oder das Browserfenster) positioniert.



HTML:

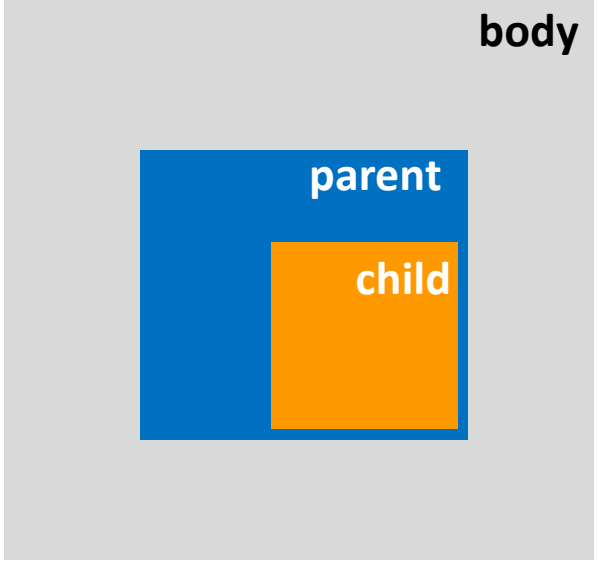
```
<div id="parent">
  <div id="child"></div>
</div>
```

CSS:

```
#parent {
  margin: 20px;
}
#child {
  position: absolute;
  width: 55%;
  top:    0;
  left:   0;
}
```

Relative Positionierung

Ein Element mit `position: relative` ist in Bezug zu seiner vorhergesehenen Position positioniert.



HTML:

```html
<div id="parent">
  <div id="child"></div>
</div>
```

CSS:

```css
#parent {
  position: absolut;
  margin: 20px;
  top:    20px;
  left:   20px;
}
#child{
  position: relative;
  width:  55%;
  bottom: 5px;
  right:  5px;
}
```

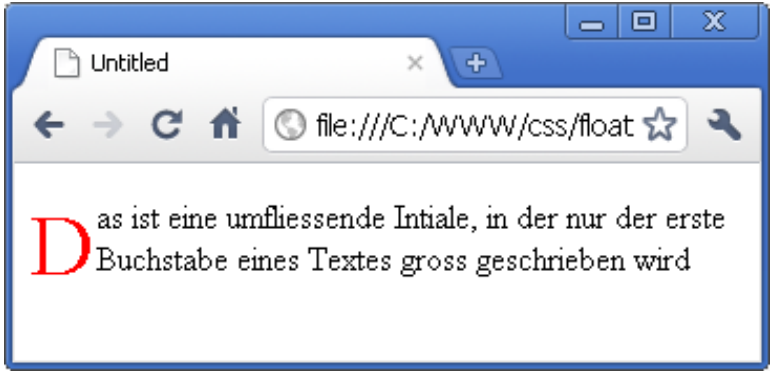
Fliessende Positionierung

Mittels der `float`-Eigenschaft kann man Text um Bereiche herumfliessen lassen (oder genau das verbieten).

Beispiel:

CSS:

```
#initial {
    color:red;
    float:left;
    font-size:2.5em;
    padding-right:2px;
}
```

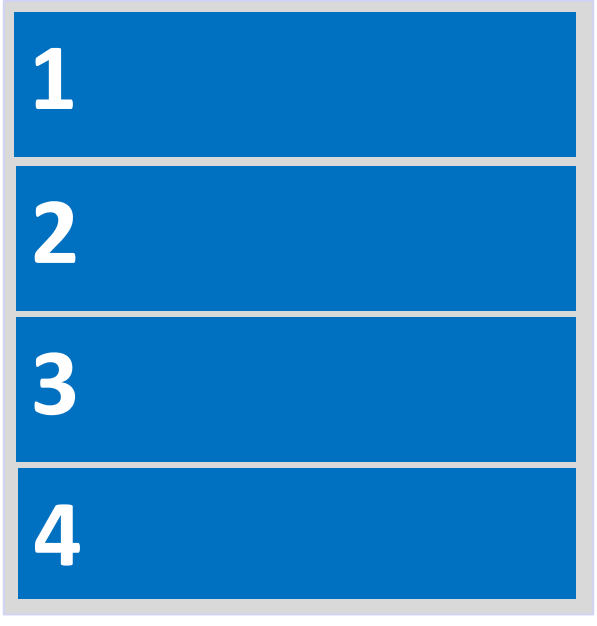


HTML:

```
<p><span id="initial">D</span>as ist eine umfliessende
Initiale, in der nur der erste Buchstabe eines Textes gross
geschrieben wird</p>
```

(Kann ebenso für Bilder verwendet werden)

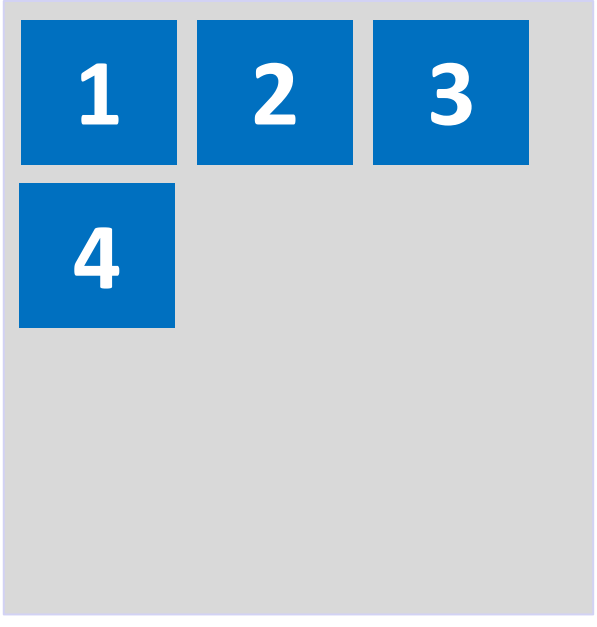
Standard-Position von Blockelementen (Wiederholung)



HTML:

```
<div>1</div>
<div>2</div>
<div>3</div>
<div>4</div>
```

Float-Positionierung in Aktion: Von Links nach Rechts



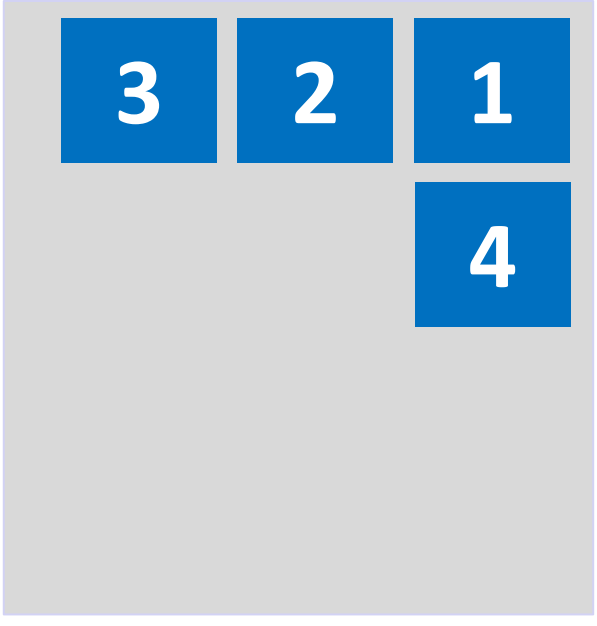
HTML:

```
<div>1</div>
<div>2</div>
<div>3</div>
<div>4</div>
```

CSS:

```
div {
  float: left;
  width: 30%;
  margin: 1% 1% 0;
}
```

Float-Positionierung in Aktion: Von Rechts nach Links



HTML:

```
<div>1</div>
<div>2</div>
<div>3</div>
<div>4</div>
```

CSS:

```
div {
  float: right;
  width: 30%;
  margin: 1% 1% 0;
}
```

Flexbox-Positionierung

Konzept: Gibt dem Container die Möglichkeit die Grösse der enthaltenen Items zu ändern:

- Ein Flex-Container vergrössert seine Items um Platz zu füllen oder verkleinert diese um Platz zu schaffen.

- Ein Flex-Container kann seine Items entweder horizontal oder vertikal ausrichten.

- **Vorteil:** Viele Standard-Layouts lassen sich mit einigen wenigen Einstellungen erzeugen. Viel weniger Code zu schreiben.

Flex-Positionierung: Grundlagen

Ziel: Die Kinder eines Elternelements (`parent`) sollen mittels Flexbox positioniert werden

- (1) Einstellen der Property `display` des Elternelements (`parent`) auf `flex`. Die Kinder verwenden nun die Flex-Positionierung.

- (2) Bestimmen wie sich die Kinder verhalten sollen (CSS-Property `flex`)

HTML:

```html
<div id="parent">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

CSS:

```css
#parent {
  display: flex;
}


#parent div {
  flex: 1 0 auto;
}
```

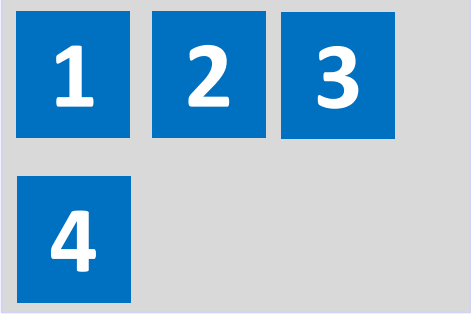
CSS-Property flex: Bestimmt die Länge eines Flex-Elementes relativ zu den anderen Flex-Elementen in dem gleichen Container

Flexbox: Flussrichtung und Umbruch

Elementfluss des Inhalts: `flex-flow: row;`

=> Die Grösse der Items hat sich dem Container angepasst

Elementfluss mit Unterbruch: `flex-flow: row wrap;`



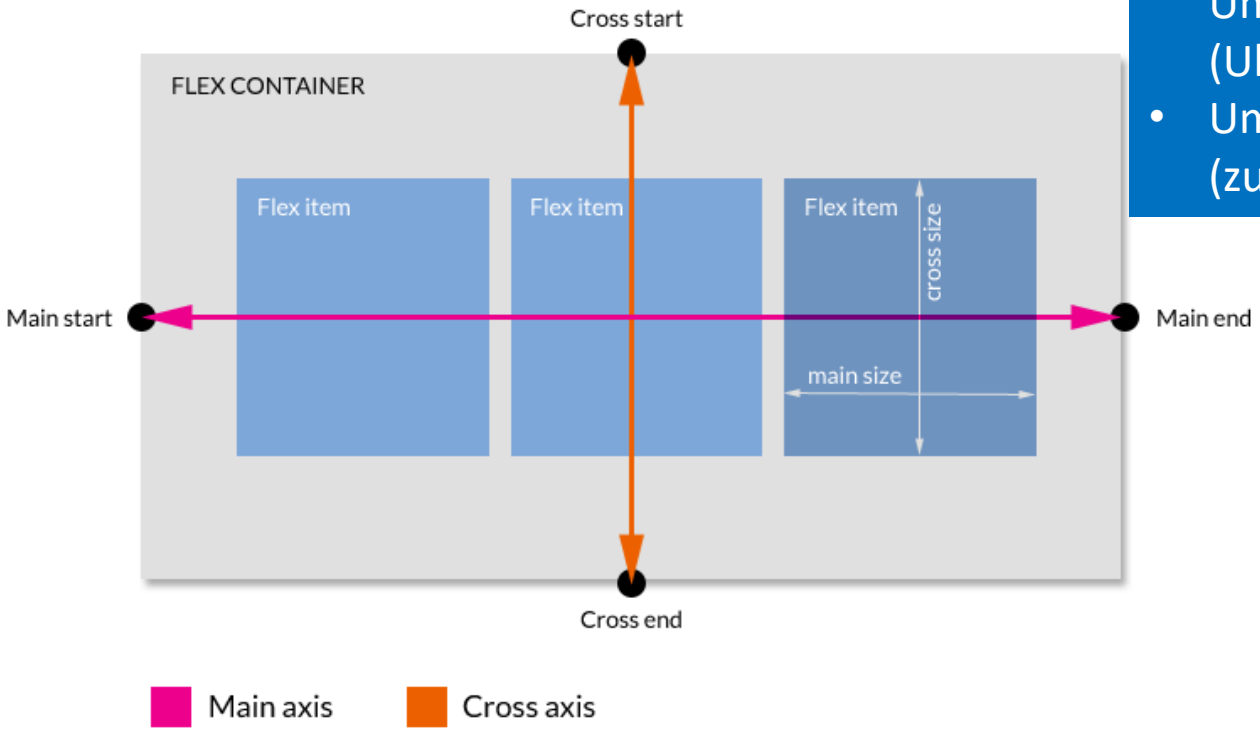
=> Die Grösse der Items bleibt und ein Zeilenumbruch findet statt

```html
<div id="parent">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>

#parent {
  display: flex;
  flex-flow: row;
}
```

- **Flussrichtungen:** `row`, `column`, `row-reverse`, `column-reverse`
- **Umbrüche:** `nowrap`, `wrap`, `wrap-reverse`

Orientierung im Flexbox-Container



- Flussrichtung column:
  Um 90 Grad gedreht
  (Uhrzeigerrichtung)
- Umgekehrte Flussrichtung:
  (zusätzlich) gespiegelt

Flexbox: Ausrichtung entlang der Hauptachse (main-axis)

CSS-Property: ***justify-content***

HTML:

```
<div id="parent">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

CSS:

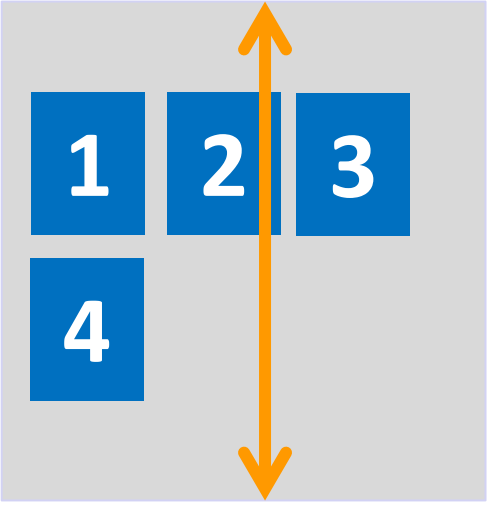
```
#parent {
  display: flex;
  flex-flow: row wrap;
  justify-content: center;
}
```

Eigenschaften von *justify-content*:

- flex-start: Anfang des Containers
- flex-end: Ende des Containers
- center: Im Zentrum des Containers
- space-between: Abstand zwischen den Items
- space-around: Abstand zwischen den Items und dem Rand

Flexbox: Ausrichtung entlang der Kreuzachse (cross-axis)

CSS-Property: ***align-content***



HTML:

```
<div id="parent">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

CSS:

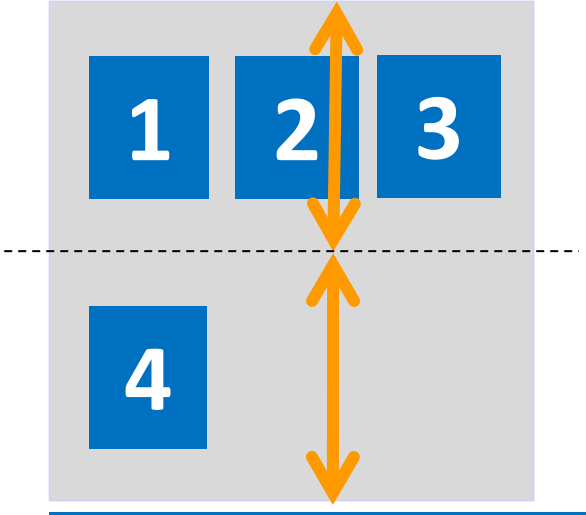
```
#parent {
  display: flex;
  flex-flow: row wrap;
  align-content: center;
}
```

Eigenschaften von *align-content*:

- stretch: Zeilen maximieren
- flex-start: Anfang des Containers
- flex-end: Ende des Containers
- center: Im Zentrum des Containers
- space-between: Abstand zwischen den Items
- space-around: Abstand zwischen den Items und Rand

Flexbox: Zeilenweise Ausrichtung des Inhalts (cross-axis)

CSS-Property: ***align-items***



HTML:

```
<div id="parent">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

CSS:

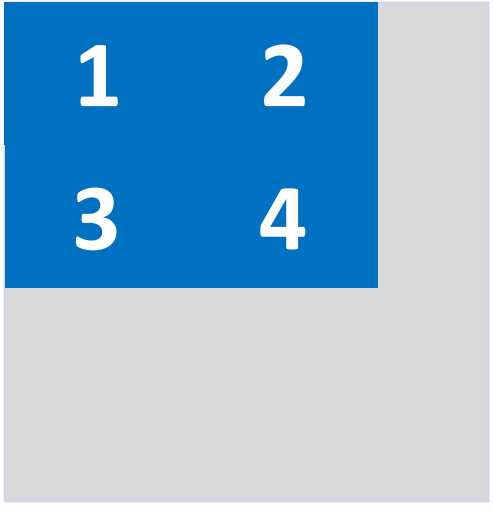
```
#parent {
  display: flex;
  flex-flow: row wrap;
  align-items: center;
}
```

Eigenschaften von *align-items*:

- stretch: Zeilen maximieren
- flex-start: Anfang des Containers
- flex-end: Ende des Containers
- center: Im Zentrum des Containers
- baseline: Entlang der Baseline des Containers

Grösse der Items: Initiale Grösse

flex-basis: bestimmt initiale Grösse eines Items



HTML:

```
<div id="parent">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

CSS:

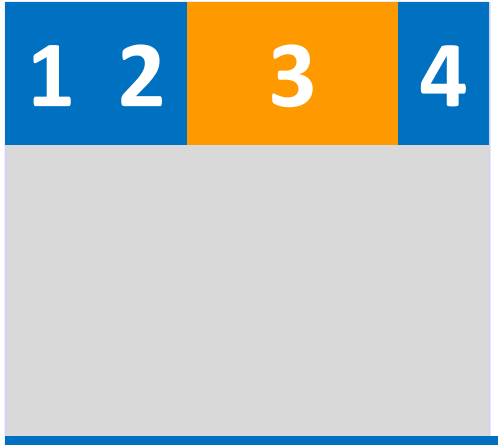
```
#parent {
  display: flex;
  flex-flow: row wrap;
}
#parent div {
  flex-basis: 35%;
}
```

Eigenschaften von *flex-basis*:

- auto: Initiale Grösse entspricht width bzw. height
- content: Initiale Grösse entspricht dem Inhalt
- CSS-Grössenangabe, z.B. 5%

Grösse der Items: Vergrösserung

flex-grow: bestimmt Vergrösserung eines Items im Verhältnis zu den anderen Items



Beispiel:

1: basis: 5%; grow: 1 → width: 5% + 80% * 1 / 6
2: basis: 5%; grow: 1 → width: 5% + 80% * 1 / 6
3: basis: 5%; grow: 3 → width: 5% + 80% * 3 / 6
4: basis: 5%; grow: 1 → width: 5% + 80% * 1 / 6

HTML:

```
<div id="parent">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

CSS:

```
#parent { display: flex; }
#parent div {
   flex-basis:5%;
   flex-grow: 1;
}
#parent div:nth-child(3) {
  flex-grow: 3;
}
```

Grösse der Items: Verkleinern

flex-shrink: bestimmt Verkleinerung eines Items im Verhältnis zu den anderen Items

Beispiel:
1: basis 40%; shrink: 1 → width: 40% - 60% * 1 / 6
2: basis 40%; shrink: 1 → width: 40% - 60% * 1 / 6
3: basis 40%; shrink: 3 → width: 40% - 60% * 3 / 6
4: basis 40%; shrink: 1 → width: 40% - 60% * 1 / 6

HTML:

```
<div id="parent">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

CSS:

```
#parent { display: flex; }
#parent div {
  flex-basis:40%;
  flex-shrink: 1;
}
#parent div:nth-child(3) {
  flex-shrink: 3;
}
```

Weiterführendes zu Flexbox

Auf folgender Seite lässt sich prima mit Flexbox experimentieren:
- https://web.archive.org/web/20210106151157/https://demos.scotch.io/visual-guide-to-css3-flexbox-flexbox-playground/demos/

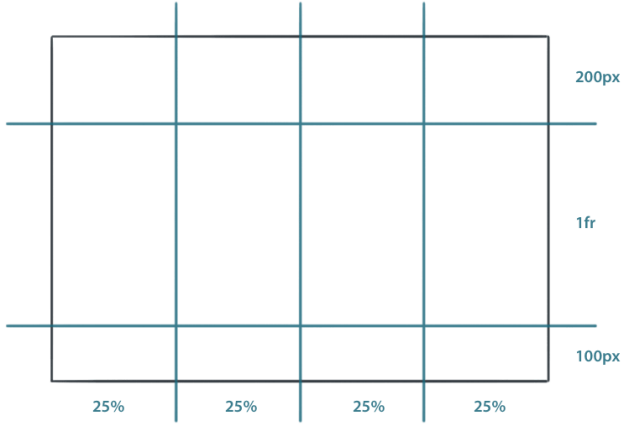
Gutes Tutorial zu Flexbox:
- https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Spiel um Flexbox Layouts zu üben (Flexbox Froggy):
- https://flexboxfroggy.com/

CSS-Grid

- CSS-Grid ermöglicht das Positionieren von Elementen innerhalb eines Rasters.

- CSS-Grids arbeiten mit einem Elternelement, in dem das Raster definiert wird und mit darin enthaltenen Kind-Elementen, die im Raster positioniert werden.



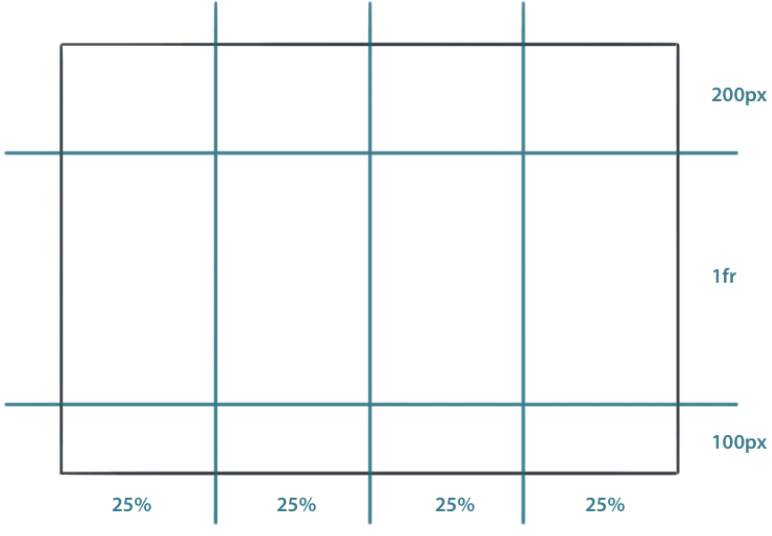
Erstellen eines CSS-Grids

- Auf dem Elternelement Eigenschaft `display` auf `grid` setzen
- Mit `grid-template-columns` und `grid-template-rows` Rasterlinien definieren

Beispiel: CSS-Grid mit 3 Zeilen und 4 Spalten

```
.container {
  display: grid;
  grid-template-rows: 200px 1fr 100px;
  grid-template-columns: 25% 25% 25% 25%;
}
```

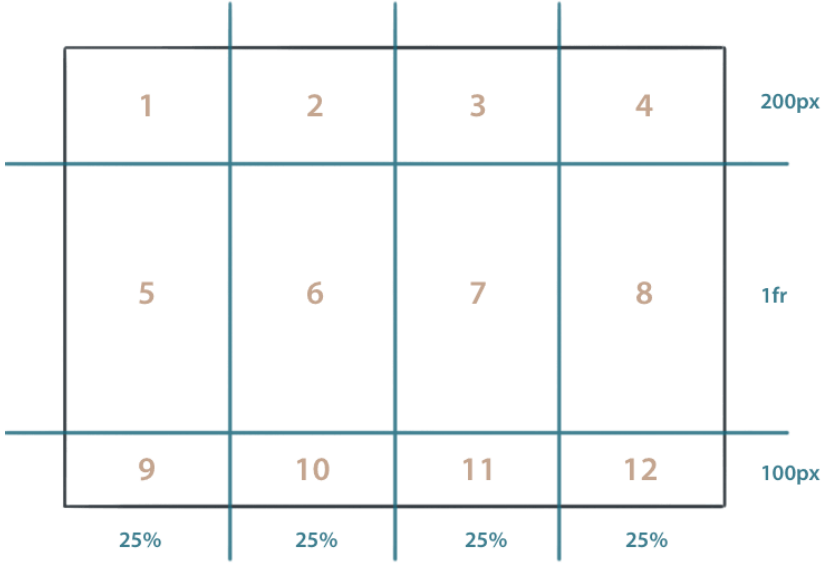
- Die mittlere Zeile hat die Angabe `1fr` (1 Fraction) erhalten – damit erstreckt Sie sich über den noch freien Platz.



Automatisches Positionieren innerhalb des CSS-Grids

- Wenn innerhalb des Elternelements nun Kind-Elemente liegen, fügen sich die Inhalte automatisch von oben links nach rechts unten in die Gridzellen ein.

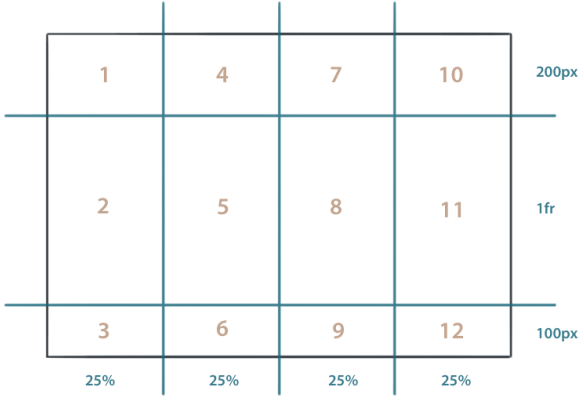
```
<div class="container">
  <div>Element 1</div>
  <div>Element 2</div>
  ...
  <div>Element 11</div>
  <div>Element 12</div>
</div>
```



Steuerung der automatischen Positionierung

- Mit der CSS-Eigenschaft `grid-auto-flow` kann beeinflusst werden, wie die Kind-Elemente im Grid einsortiert werden.
- Der Wert: `column` führt zu folgender Anordnung:

```
.container {
  display: grid;
  grid-template-rows: 200px 1fr 100px;
  grid-template-columns: 25% 25% 25% 25%;
  grid-auto-flow: column;
}
```

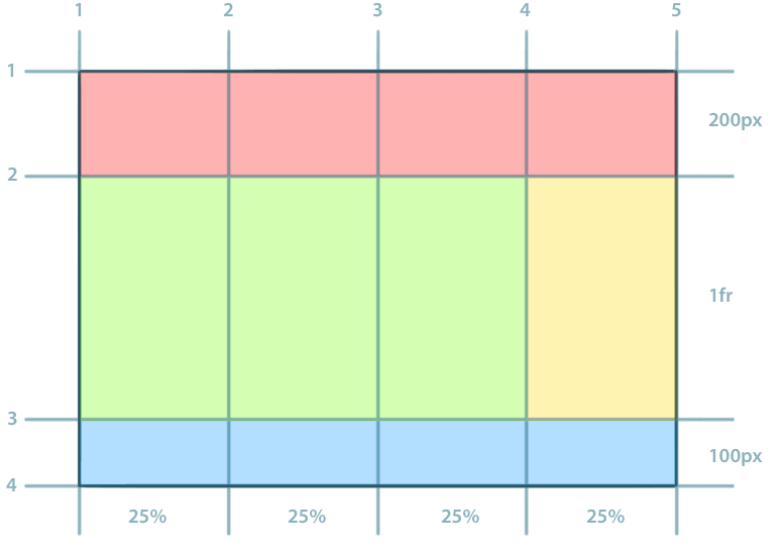


Manuelles Positionieren der Elemente im Grid

- Mit CSS Grids ist es möglich, Elemente völlig frei im Raster zu positionieren.

- Dazu wird den Kind-Elementen mit Hilfe der CSS-Eigenschaften ***grid-row-start*** und ***grid-row-end*** bzw. ***grid-column-start*** und ***grid-column-end*** mitgeteilt, wo sie sich im Grid befinden sollen.

- Wenn sich das erste Element über die volle Breite des Layouts gehen soll, dann streckt es sich von Gridlinie 1 (bei 0%) bis 5 (bei 100%).



Achtung: Positionierung erfolgt nicht über die Gridspalten sondern über die Gridlinien!

Beispiel: Positionierung

```html
<div class="container">
  <header>header</header>
  <article>article</article>
  <aside>aside</aside>
  <footer>footer</footer>
</div>
```


```css
.container {
  height: 100vh;
  display: grid;
  grid-template-rows: 200px 1fr 100px;
  grid-template-columns: 25% 25% 25% 25%;
}
header {
  background: tomato;
  grid-column: 1/5;
  grid-row: 1/2;
}
article {
  background: gold;
  grid-column: 1/4;
  grid-row: 2/3;
}
aside {
  background: lightgreen;
  grid-column: 4/5;
  grid-row: 2/3;
}
footer {
  background: lightblue;
  grid-column: 1/5;
  grid-row: 3/4;
}
```

Kurzform:
grid-column-start:1;
grid-column-end:5;

Gridzellen benennen (1)

- Die einzelnen Bereiche des CSS-Grids können mit Hilfe des Befehls ***grid-template-areas*** bei der Definition des Grids im Elternelement benannt werden.
- Dazu werden die einzelnen Zellen innerhalb einer Zeile mit Namen versehen. Die einzelnen Namen werden mit Leerzeichen von einander getrennt.

```
.container {
  height:100vh;
  display: grid;
  grid-template-rows: 200px 1fr 100px;
  grid-template-columns: 25% 25% 25% 25%;
  grid-template-areas: "header header header header"
                       "content content content sidebar"
                       "footer footer footer footer" ;
}
```

Gridzellen benennen (2)

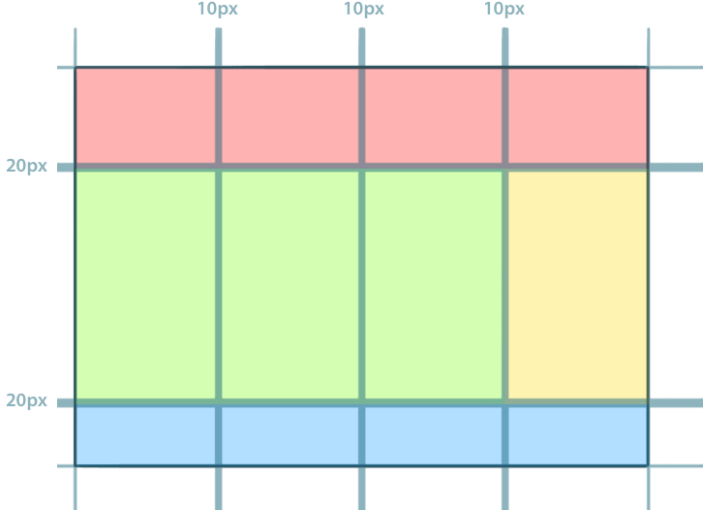
- Um Kind-Elemente einem benannten Bereich im Grid zu zuweisen, wird der Befehl grid-area verwendet.

```
header {
  grid-area: header;
}
article {
  grid-area: content;
}
aside {
  grid-area: sidebar;
}
footer {
  grid-area: footer;
}
```

Abstände zwischen Gridzellen

- Wenn Abstände zwischen den Spalten bzw. Zeilen des Grids entstehen sollen, kann mit Hilfe der Befehle ***grid-column-gap*** bzw. ***grid-row-gap*** im Elternelement eine Breite für die Gridlinien festgelegt werden.
- Achtung: die Abstände können nur zwischen Spalten erzeugt werden. Die erste und letzte Gridlinie bleibt also unverändert.

```
.container {
  grid-row-gap: 20px;
  grid-column-gap: 10px;
}
```

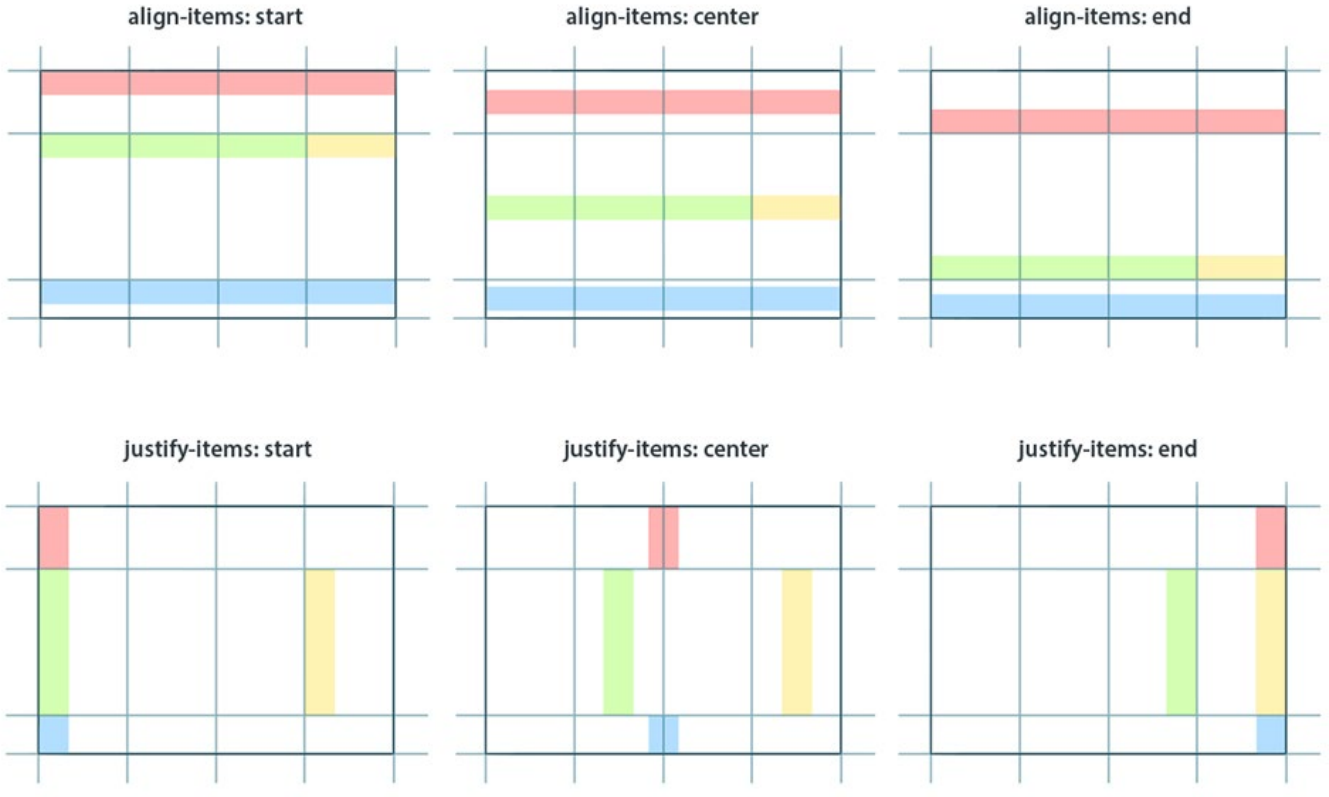


Ausrichtung innerhalb des Grids (1)

Globale Ausrichtung im Eltern-Element festlegen

- Die Befehle für das Eltern-Element lauten ***justify-items*** für das horizontale Verhalten, und ***align-items*** für das vertikale Verhalten.
- Es stehen jeweils die Werte ***start, end, center*** und ***stretch*** zur Wahl.
- ***stretch*** ist der Standardwert und selbstverständlich können ***justify-items*** und ***align-items*** kombiniert werden.

Ausrichtung innerhalb des Grids (2)



Weiterführendes zu Grid

Spiel um Grid Layouts zu üben (CSS Grid Garden):
- https://cssgridgarden.com/

Vergleich Flexbox und Grid

- Das Flexbox-Layoutmodell erfreut sich grosser Beliebtheit, weshalb häufig gefragt wird, worin der Vorteil von CSS-Grids besteht.

- Flexbox bietet sich besser für lineare Strukturen an, CSS-Grids für komplexe verschachtelte Konstruktionen.

- Es könnte gut sein, dass CSS-Grids in Zukunft für das globale Layout und Flexbox für einzelne Komponenten eingesetzt werden.

- Gute Demonstration Flexbox vs. Grid: https://tutorialzine.com/2017/03/css-grid-vs-flexbox

Zusammenfassung

- Layouts sind ein zentrales Thema in der Entwicklung von Webapplikationen.
- Wir haben vier Layouttechniken (CSS kennt weitere) kennengelernt:
  - Absolute und relative Positionierung zum Fine-Tuning und für spezielle Layouts.
  - Fliessende Positionierung um Bilder oder andere Elemente innerhalb eines Textes zu positionieren.
  - Flexbox-Layouts, welche flexibel auf Veränderungen der Breite des Fensters des Webbrowsers sowie des Containerinhalts reagieren sollen.
  - Grid-Layouts um exakte Layouts zu kreieren (z.B. für Single-Page-Webapplikationen)
- Jede Technik hat Ihre Vor- und Nachteile.
- Kombination von Techniken erlaubt komplexe Layouts mit wenigen CSS-Regeln.