CSS Fortgeschrittene Grundlagen

Inhalt

Setup

Für Übungen nutzen wir die folgende **CodeSandbox** als Startpunkt:

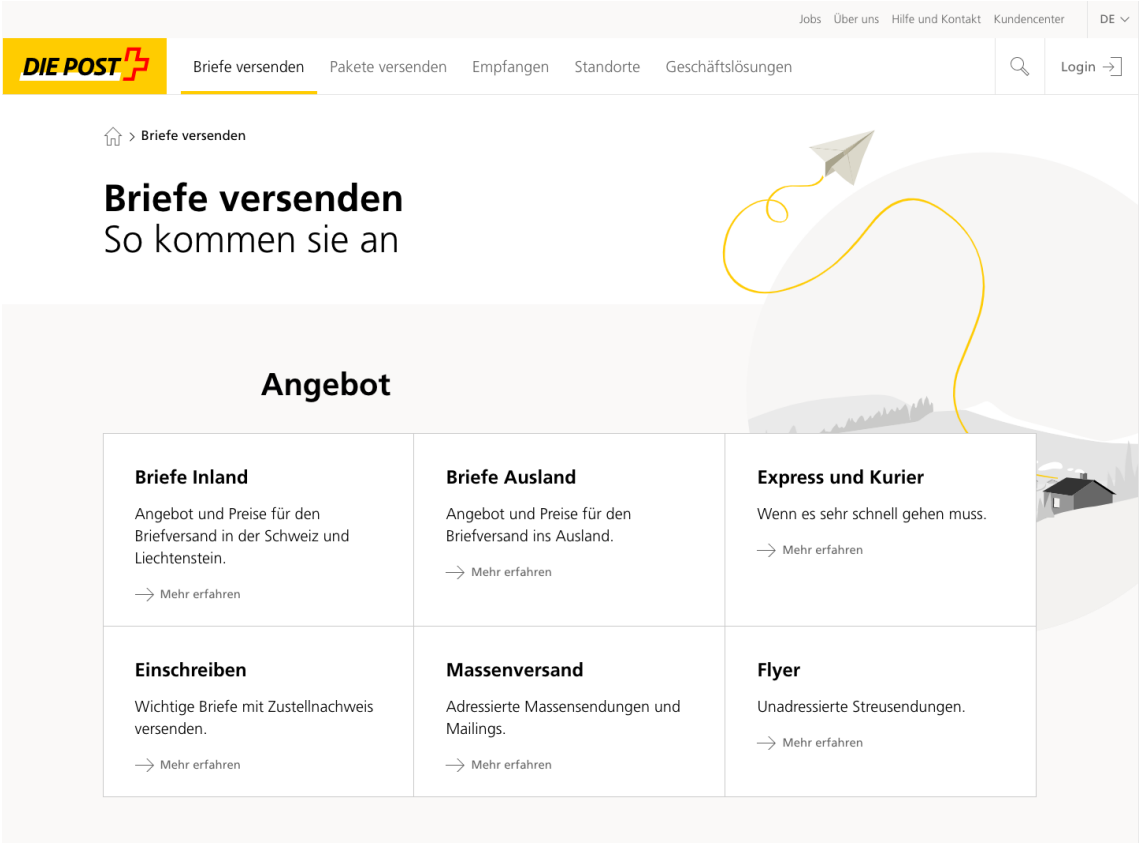
https://codesandbox.io/s/bnydy

Die Übungen bauen immer aufeinander auf. Aber keine Angst! Für den Fall, dass bei einer Übung etwas nicht klappt, gibts bei jeder Übung einen Link zur CodeSandbox mit dem aktuellen Stand.

Quick Recap

Flexbox & CSS Grid

Bei einem normalen Layout mit Block-Elementen sind alle Elemente untereinander, da Block-Elemente immer 100% der verfügbaren Breite einnehmen. Dies ist natürlich nicht immer gewünscht. Damit wir Block-Elemente nebeneinander darstellen können oder wenn wir das Verhalten von Inline-Elementen anpassen möchten, gibts Flexbox (`display: flex;`) und CSS Grid (`display: grid;`).



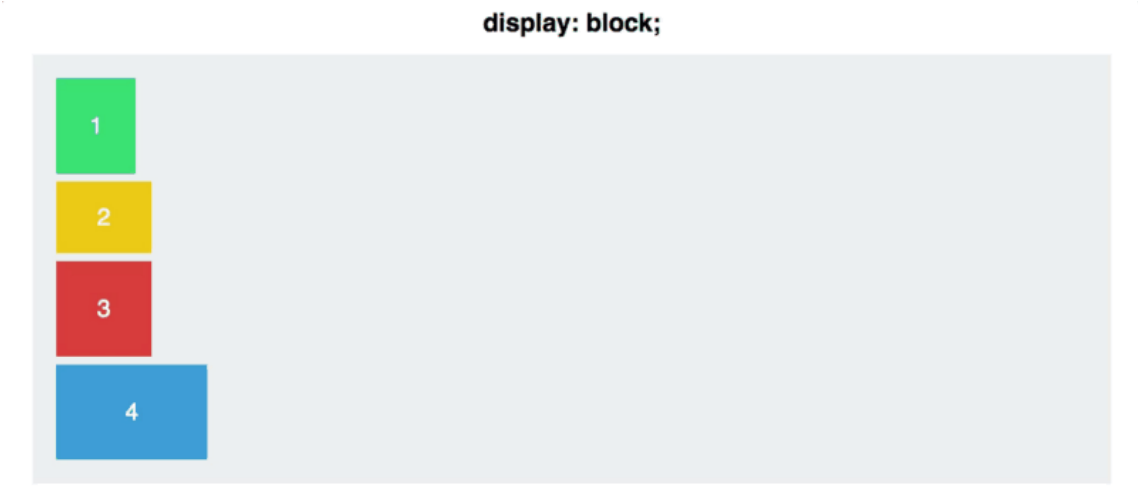
Flexbox

- Flexbox kann mit der CSS-Deklaration `display: flex;` genutzt werden
- `display: flex;` wird auf einem gemeinsamen Eltern-Element gesetzt
- Dieses Eltern-Element wird dann auch der **Flex-Container** genannt
- Kinder-Elemente erhalten dadurch einen **Flex-Context** und werden nun per default als Inline-Elemente nebeneinander dargestellt

Beispiele

```css
/* CSS */
.container {
  display: flex;
}
```

```html
<!-- HTML -->
<div class="container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

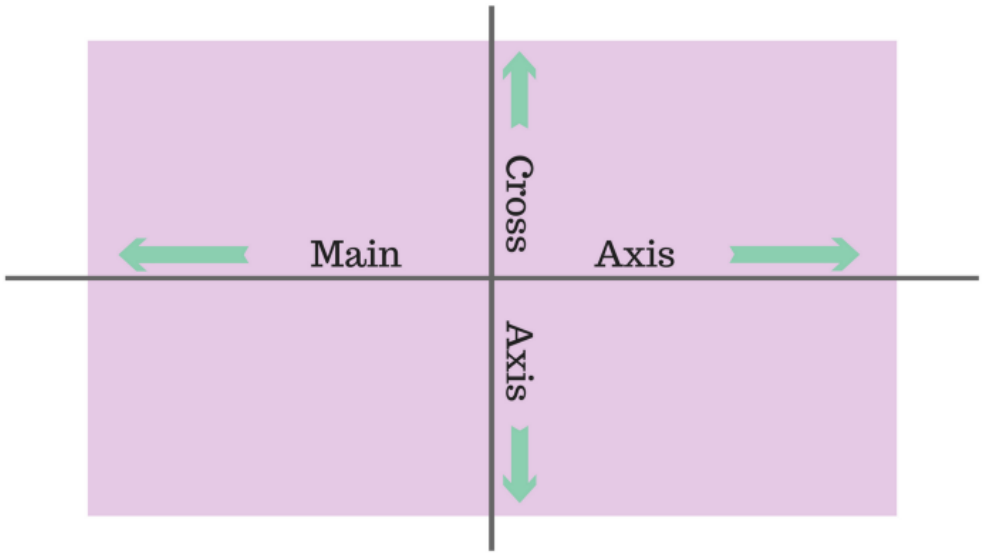


Demo 🤯

- Flexbox Basic

Flex Axis

- Ein Flex-Container hat eine **Main Axis** und eine **Cross Axis**
- Per Default verläuft die Richtung der Kinder-Elemente horizontal von links nach rechts (`flex-direction: row;`)



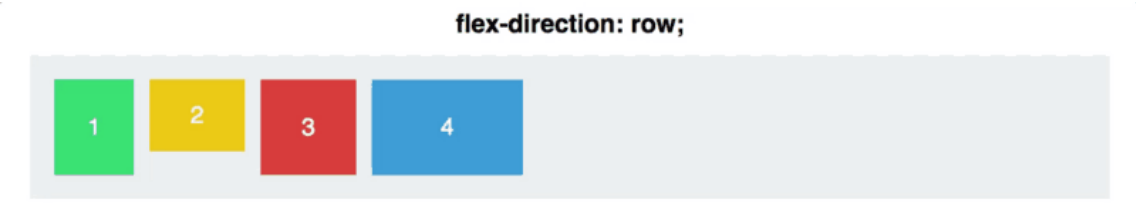
Flex Direction

- `flex-direction` ermöglicht den Richtungswechsel der **Main Axis**
- Kann nur auf einem **Flex-Container** gesetzt werden
- Mögliche Werte:
  - `row`

- `column`
- `row-reverse`
- `column-reverse`

Beispiele

```
/* CSS */
.container {
  display: flex;
  flex-direction: column;
}
```



Mit `row-reverse` und `column-reverse` kann Richtung/Flow umgekehrt werden.

Beispiele

```
/* CSS */
.container {
  display: flex;

  /* right to left */
  flex-direction: row-reverse;

  /* bottom to top */
  flex-direction: column-reverse;
}
```

Demo 🤯

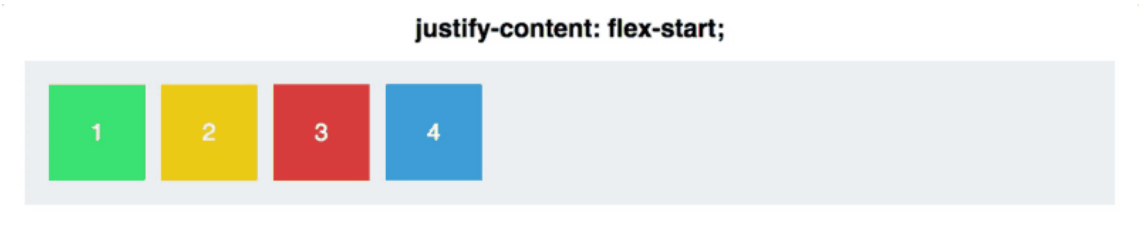
- Flexbox Direction

Justify Content

- `justify-content` deklariert die Position auf der **Main Axis**
- Mögliche Werte:
  - `flex-start`
  - `flex-end`
  - `center`
  - `space-between`
  - `space-around`

Beispiele

```
/* CSS */
.container {
  display: flex;
  justify-content: flex-start;
}
```



Demo 🤯

- Flexbox Justify Content

Align Items

- `align-items` bestimmt die Position auf der **Cross Axis**
- Mögliche Werte:
  - `flex-start`
  - `flex-end`
  - `center`
  - `stretch`
  - `baseline`

Beispiele

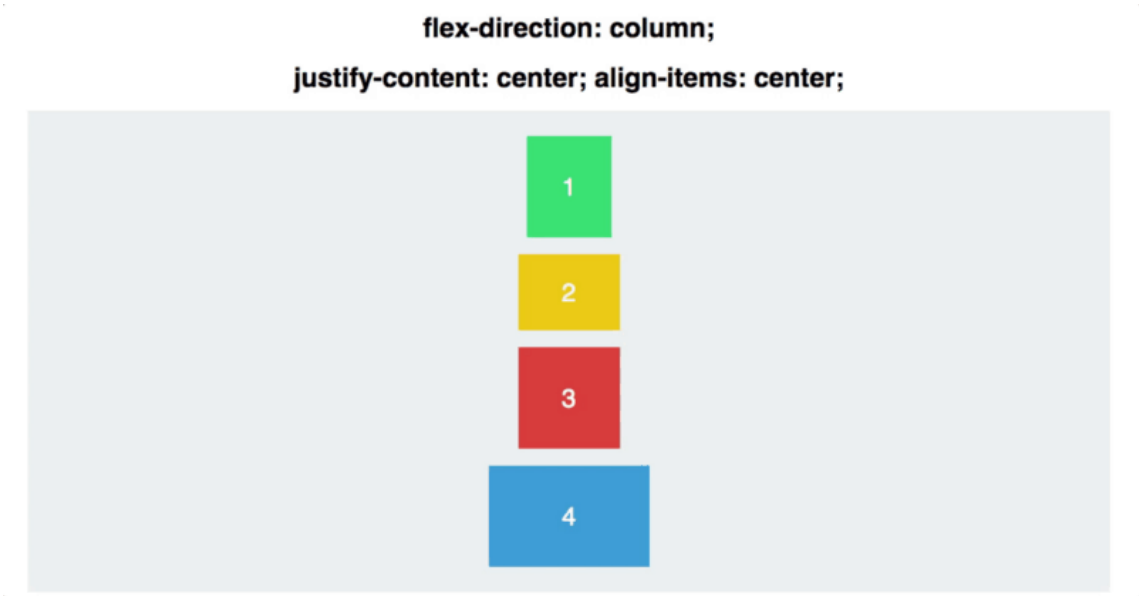
```css
/* CSS */
.container {
  display: flex;
  align-items: center;
}
```

Demo 🤯

- Flexbox Align Items

Demo

Hier eine kleine Demo wie die Properties zusammen funktionieren.



Flex

- Die `flex` -Property kann auf einem Element platziert werden, welche einen **Flex-Context** hat (Elternelement muss `display: flex;` haben)
- Die `flex` -Property ist ein Shorthand für `flex-grow` , `flex-shrink` und `flex-basis`
- `flex-basis` definiert, wie gross das Element per default sein kann (%-Wert der Main Axis)
- `flex-grow` definiert, ob und in welcher Relation das Element grösser sein kann als die flex-basis
- `flex-shrink` definiert, ob und in welcher Relation das Element kleiner sein kann als die flex-basis
- Alle diese Properties können **nur** auf **direkten** Kinder-Elementen eines Flex-Container gesetzt werden

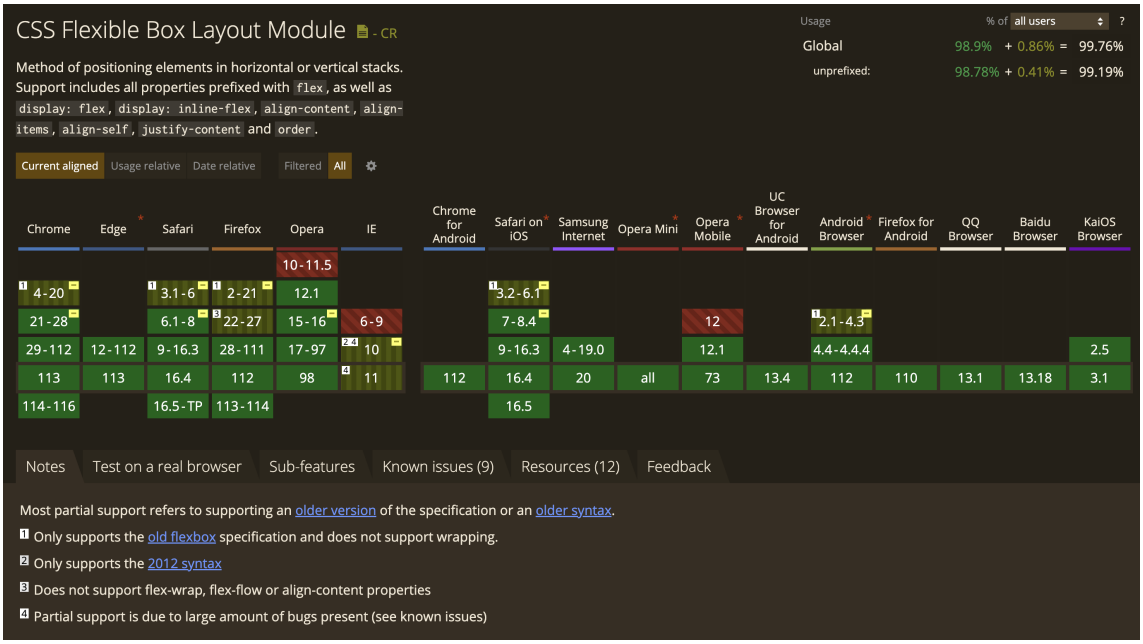
Beispiele

```
/* CSS */
.container {
  display: flex;
}

.children {
  flex-grow: 0;
  flex-shrink: 1;
  flex-basis: 100%;

  /* Shorthand */
  flex: 0 1 100%;
}
```

Demo 🤯

- Flexbox Flex

Browser Support



Caniuse - Flexbox

Hilfreiche Links

- CSS Tricks Artikel
- Freecodecamp Artikel
- CSS Flexible Box Layout - MDN
- 🔥 Spielerisch lernen

CSS Grid

Mit `display: grid;` kann CSS Grid eingesetzt werden. Ein Grid ist wie eine *Tabelle*, wobei die einzelnen Kinder-Elemente frei in dieser *Tabelle* plaziert werden können.

Das Element, welches auf dem `display: grid;` deklariert wird, wird dadurch zum **Grid-Container** und die direkten Kinder-Elemente erhalten einen Grid-Context (wie auch bei Flexbox).

Vorteile:

- Mit CSS Grid kann praktisch jedes erdenkliche Layout erstellt werden, da Elemente *frei* vom content Flow platziert werden können.

Nachteile:

- Komplexe Syntax

Da dieses Thema sehr gross ist, sind in diesem Script **nur** die Basics zu finden.
Für mehr Informationen, bitte die nachfolgenden hilfreichen Links beachten.

Template Columns

Mit `grid-template-columns` können wir deklarieren, wieviele Spalten unser Grid hat und wie breit diese sein sollen. Mögliche Angaben sind nicht nur alle normalen Weitenangaben wie `px`, `%`, `auto`, etc., sondern auch eine neue Breitenangabe `fr` → Fraction. Es gibt aber auch spezielle Funktionen wie `minmax()` und `repeat()`.

```css
/* CSS */
.container {
  display: grid;
  grid-template-columns: 1fr 1fr auto 20%;
  grid-column-gap: 4px;
}
```

Demo 🤯

- Template Columns

Template Rows

Mit `grid-template-rows` können wir die Höhen der Rows steuern. Dies funktioniert grundsätzlich gleich wie `grid-template-columns`, einfach für die vertikale Achse.

```css
/* CSS */
.container {
  display: grid;
  grid-template-columns: 1fr 1fr auto 20%;
  grid-column-gap: 4px;
  height: 200px;
  grid-template-rows: 30% 70%;
  grid-row-gap: 4px;
}
```

Demo 🤯

- Template Rows

Row / Column Start und Ende

- Bestimmt, wo ein Grid-Child beginnt und endet (column & row)
- Die Position des Grid-Child ist unabhängig von der Reihenfolge im HTML

```css
/* CSS */
.container {
  /* ... */
}

.first {
  grid-column-start: 2;
  grid-column-end: 4;
  /* Shorthand */
  grid-column: 2 / span 2;

  grid-row-start: 1;
  grid-row-end: 3;
  /* Shorthand */
  grid-row: 1 / span 2;
}
```

Demo 🤯

- Row / Column Start und Ende

Justify / Align

- Positionierungs-Attribute sind wie bei Flexbox anwendbar
- Positioniert die Grid-Child-Elemente innerhalb des Grid, nicht den Grid-Container selbst
- Die Positionierung ist innerhalb der zugeteilten Column & Row

```
/* CSS */
.container {
  /* Grid-Child-Elemente werden zentriert */
  justify-items: center;
  align-items: center;

  /* Grid-Child-Elemente werden links unten angezeigt */
  justify-items: start;
  align-items: end;
}
```

Demo 🤯

- Justify / Align

Grid Areas

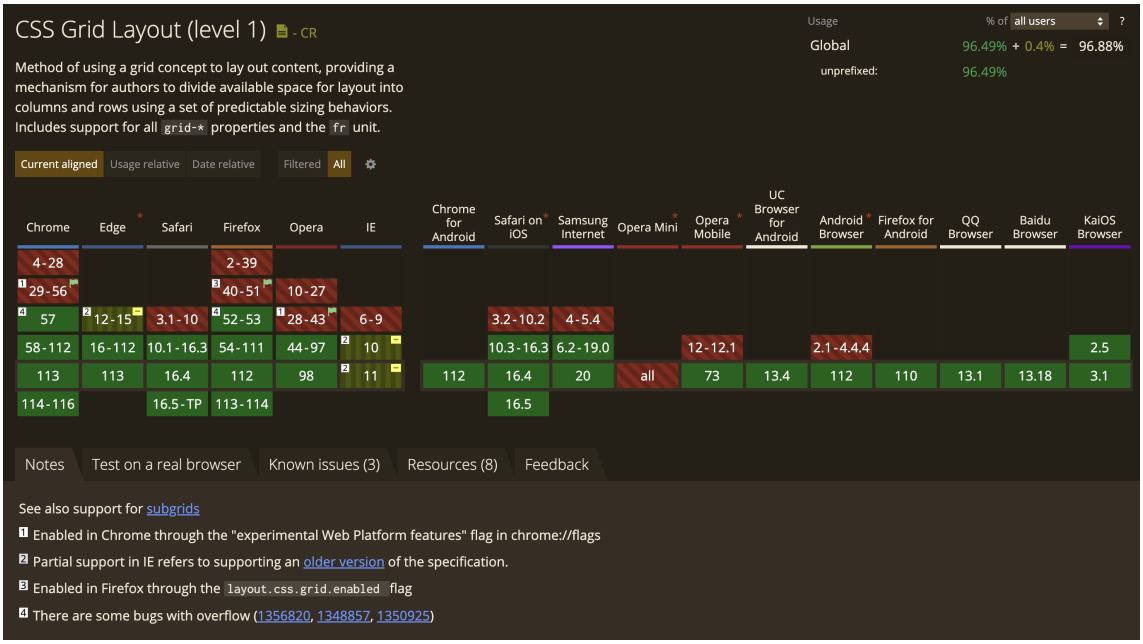
- Eine übersichtliche Alternative zu einzelnen `grid-column` / `grid-row` Deklarationen
- Grid Areas werden auf dem Grid-Container deklariert
- Grid-Child-Elemente können anschliessend die `grid-area` -Property nutzen

```
/* CSS */
.container {
  height: 250px;
  display: grid;
  grid-template-columns: 1fr auto;
  grid-template-rows: auto 1fr auto;
  grid-template-areas:
    "header header"
    "content sidebar"
    "footer footer";
}

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

Demo 🤯

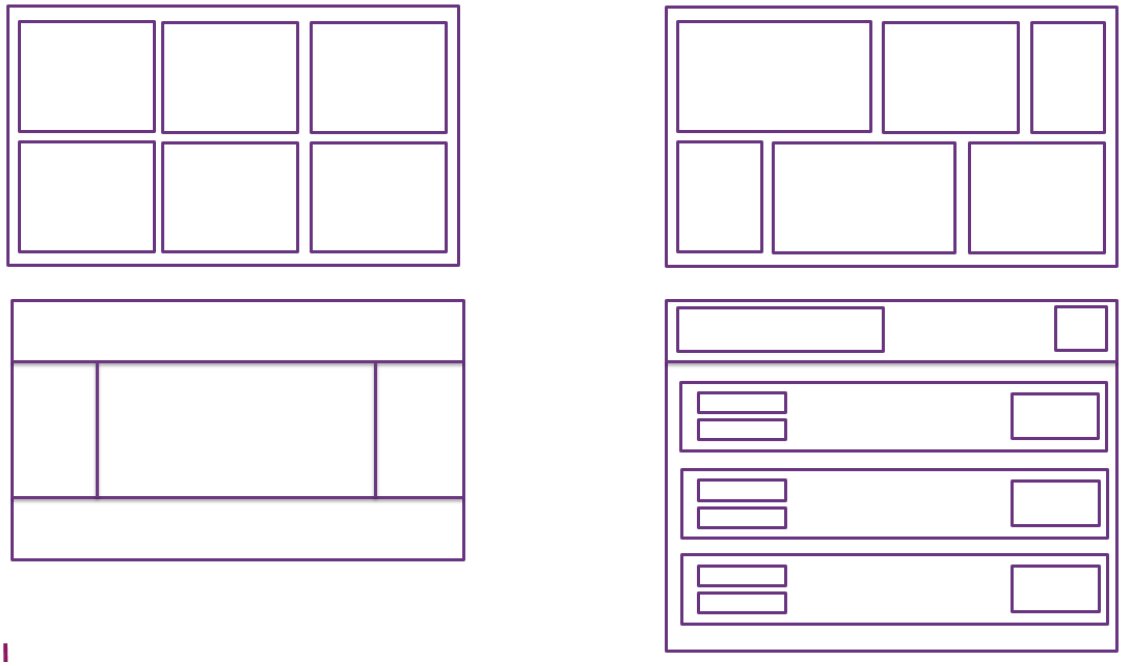
- Grid Areas

Browser Support



[Caniuse - CSS Grid]

Flexbox vs CSS Grid Quiz



Hilfreiche Links

- [CSS Tricks Artikel]
- [Cheatsheet]
- [CSS Grid Layout - MDN]
- 🔥 [Spielerisch lernen]

Practice 🔥

Öffne diese **CodeSandbox** als Startpunkt.

- [ ] Verwende **Flexbox** oder **CSS Grid** um deine vorhandene ToDo-App zu strukturieren
  - [ ] Füge zur rechten Seite vom `<main>` -Element noch eine Sidebar (`<aside>`) ein, um einen kurzen Text darzustellen
  - [ ] Strukturiere das Formular so, dass es für die User besser bedienbar ist.

Zeit: ~ 15 min

Solution Flexbox: https://codesandbox.io/s/2eylr
Solution CSS Grid: https://codesandbox.io/s/pyxnf

Webfonts

Heutzutage, kann jede beliebige **Custom-Font** im Web genutzt werden.
Dazu gibt es verschiedene Implementierungsmöglichkeiten:

- CSS
  - `@font-face` : Schriftdateien werden im eigenen CSS referenziert
  - `@import` : Externe CSS-Datei wird im eigenen CSS referenziert (nicht empfohlen)
  - `<link>` : Externe CSS-Datei wird im `<head>` referenziert (am einfachsten zu nutzen)
- JavaScript
  - `FontFace` Web API

Custom Fonts werden unter anderem von den folgenden Anbietern geliefert:

- https://fonts.google.com/
- https://fonts.adobe.com/typekit
- https://www.fonts.com/
- ...

Sobald die Schriften referenziert sind, können diese per `font-family: <font-name>;` verwendet werden.

CSS

Lokale Font

Per `@font-face` können eigene Schriftdateien als Custom Fonts deklariert werden. Sobald der Browser den ersten Text rendert, wo diese Font genutzt wird, lädt er diese herunter und wendet sie an.
Achtung, die Schriften die angewendet werden, verursachen einen Repaint/Reflow der Website. Dies heisst, dass die Website neu gerendert werden muss, sobald die Schriften geladen sind.

```
/* CSS */
@font-face {
  font-family: "MyWebFont";
  src: url("http://fonts.gstatic.com/path/to/my/font.woff2") format('woff2');
}

body {
```

```
  font-family: "MyWebFont", serif;
}
```

Im Web werden meist auch verschiedene Schrift-Schnitte der gleichen Schrift verwendet (z.B. verschiedene Schriftdicken oder kursiv). Diese kann man auch unter der gleichen `font-family` registrieren.

```
/* CSS */
@font-face {
  font-family: 'MyWebFont';
  font-weight: 400;
  src: url('./path/normal.woff2') format('woff2');
}

@font-face {
  font-family: 'MyWebFont';
  font-weight: 700;
  src: url('./path/bold.woff2') format('woff2');
}

@font-face {
  font-family: 'MyWebFont';
  font-weight: 700;
  font-style: italic;
  src: url('./path/bold-italic.woff2') format('woff2');
}

body {
  font-family: "MyWebFont", serif;
  font-weight: 400;
}

/* Bold */
h1 {
  /* font-family is inherited from body */
  font-weight: 700;
}

/* Bold & Italic */
h2 {
  /* font-family is inherited from body */
  font-weight: 700;
  font-style: italic;
}
```

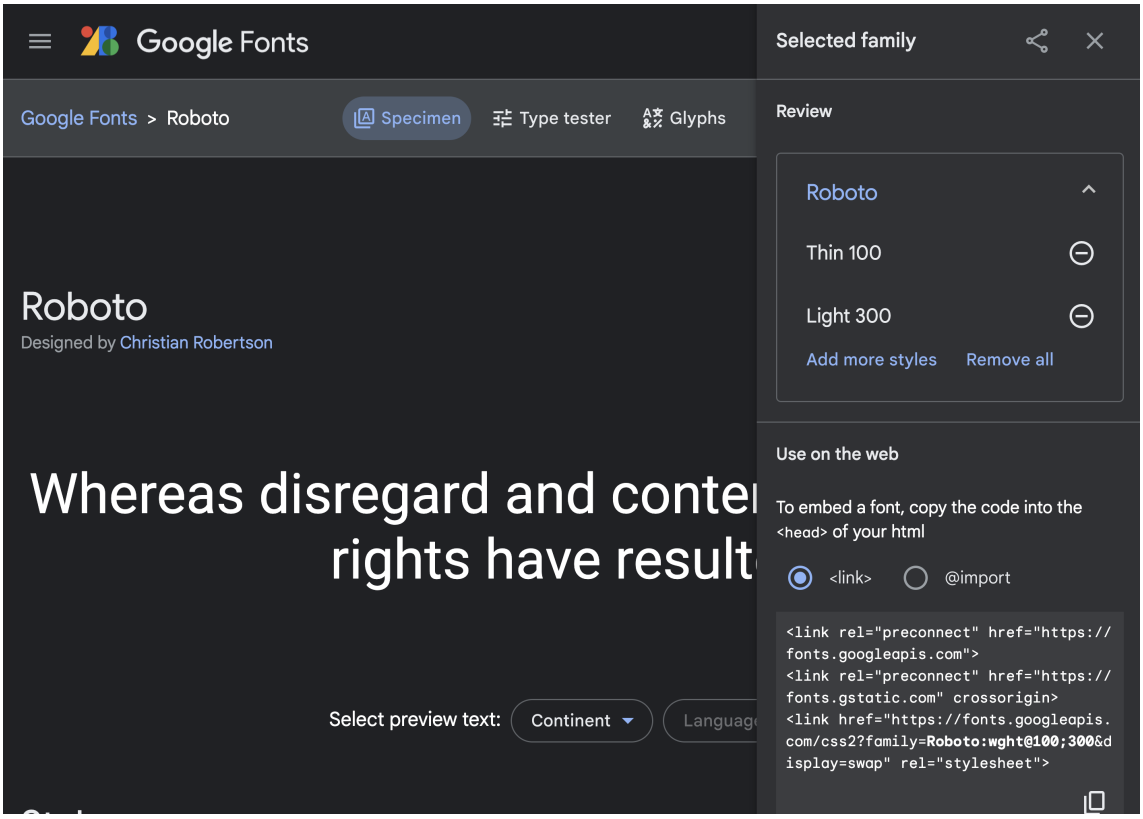
Demo 🤯

- Lokale Custom Fonts
- Lokale Custom Fonts mit mehreren Schrift-Schnitten

Externe Font (Beispiel fonts.google.com)

Auf fonts.google.com können beliebige Schriften ausgewählt und importiert werden.
Einfach die Schriften auswählen und anschliessend einbetten. Die Schriften können aber auch heruntergeladen werden und dann als lokale Schriften eingebunden werden.

fonts.google.com



Einbetten/Embedding

Die Schriften können entweder per `<link>` oder per `@import` eingebettet werden.

Use on the web

To embed a font, copy the code into the <head> of your html

(•) <link> ( ) @import

```
<link rel="preconnect" href="https://
fonts.googleapis.com">
<link rel="preconnect" href="https://
fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.
com/css2?family=Roboto:wght@100;300&d
isplay=swap" rel="stylesheet">
```

CSS rules to specify families

```
font-family: 'Roboto', sans-serif;
```

Demo 🤯

- Externe Custom-Fonts

JavaScript

FontFace Web API

Mit der FontFace API gibt es auch die Möglichkeit, Custom Fonts per JavaScipt zu laden.
Wie die Schrift angewendet werden kann, bleibt genau gleich (per `font-family` CSS-Deklaration)

```
const fontFace = new FontFace('MyWebFont', 'url("xy.woff2") format("woff2")');

fontFace.load().then(function(loadedFontFace) {
  document.fonts.add(loadedFontFace); // Font wird hinzugefügt
  document.documentElement.classList.add('font-loaded');
});
```

```
body {
  font-family: Arial, sans-serif;
}

.font-loaded body {
  font-family: 'MyWebFont', sans-serif;
}
```

Demo 🤯

- Custom Fonts per JavaScript

Hilfreiche Links

- Smashing Magazine
- Custom Fonts - Loading Strategies
- Google Fonts herunterladen

> ***Das Wichtigste in Kürze***
>
> - *Custom Fonts werden in allen aktuellen Browsern unterstützt*
> - *Es gibt verschiedene Möglichkeiten diese hinzuzufügen, lokale Schriften sind immer zu bevorzugen*
> - *ACHTUNG: Wenn der Browser die Schriften anwendet, verursacht dies ein Repaint/Reflow der Website*

Practice 🔥

Öffne diese **CodeSandbox** als Startpunkt.

- [ ] Füge deiner ToDo-App eine Custom Font hinzu, nutze dafür Google Fonts

Zeit: ~ 5 min

Solution: https://codesandbox.io/s/i3nb9

CSS Variablen

Mit CSS Variablen kann eine Variable definiert und anschliessend mehrmals genutzt werden.
Damit können Redundanzen im CSS verhindert werden.

CSS Variables werden auch **Custom Properties** genannt.

Syntax

Variablen definieren

```
/* CSS */
:root {
   --variable-name: <value>;
}
```

Variablen nutzen

```
/* CSS */
.my-awesome-class {
  property: var(--variable-name);
}
```

Scope

Variablen können im Root definiert werden, somit sind diese global verfügbar.
Sie können aber auch innerhalb eines Selektors definiert werden, dadurch erhält die Variable auch nur innerhalb des Selektors den vergebenen Wert.

Beispiele

```
/* CSS */
:root {
   --default-padding: 10px;
}

.card {
   --default-padding: 20px;
}

p {
  padding: var(--default-padding);
}
```

```
<!-- HTML -->
<p>This paragraph is padded by <b>10px</b> on each side.</p>

<div class="card">
  <p>This paragraph is padded by <b>20px</b> on each side.</p>
</div>
```

```
<p>This paragraph is padded by <b>10px</b> on each side.</p>
```

Demo 🤯

- CSS Variablen

Limitationen

Vererbung hat auch Einschränkungen. Wenn eine Variable in einem Selektor überschrieben wird, wird diese nicht mehr vererbt.

```
/* CSS */
:root {
  --default-padding: 10px;
  --used-value: var(--default-padding);
}

.card {
  /* --used-value wird nicht auf 20px überschrieben */
  --default-padding: 20px;
}

p {
  padding: var(--used-value);
  background: rgba(0, 0, 0, 0.1);
}
```

```
<!-- HTML -->
<p>This paragraph is padded by <b>10px</b> on each side.</p>

<div class="card">
  <p>This paragraph is still padded by <b>10px</b> on each side.</p>
</div>

<p>This paragraph is padded by <b>10px</b> on each side.</p>
```

Demo 🤯

- CSS Variablen - Limitationen

Dafault Value

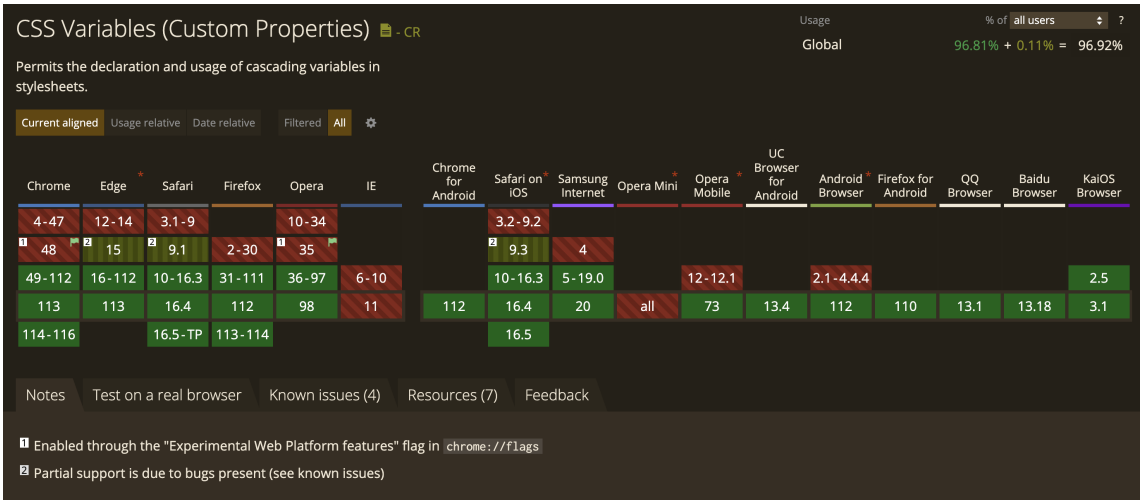
Bei der Nutzung von Variablen kann auch ein default angegeben werden, falls die Variable nicht definiert ist.

```
p {
  /* Nutzt den Wert #007bff, solang --color nicht definiert ist */
  background-color: var(--color, #007bff);
}
```

Demo 🤯

- CSS Variablen - Default Value

Browser support



Caniuse - CSS Variables (Custom Properties)

Practice 🔥

Öffne diese **CodeSandbox** als Startpunkt.

- [ ] Erstelle zwei CSS Variablen um eine Farbe für einen Background und eine Color zu speichern
- [ ] Verwende die Variablen im `body { ... }` damit deine App eine custom `background-color` und eine custom `color` erhält
- [ ] Erstelle einen Button, welcher dem `<body>` beim Klick eine neue Klasse `dark-theme` hinzufügt, bzw. beim nächsten Klick wieder entfernt
- [ ] Passe dein CSS nun so an, dass wenn diese Klasse auf dem Body ist, sich `background-color` und `color` deiner App anpassen, indem Du die CSS Variablen überschreibst

Als Hilfestellung kann für die Funktionalität des Buttons dieses JavaScript eingefügt werden:

```
const themeButton = document.getElementById('theme-button');
themeButton.addEventListener('click', () => {
  document.body.classList.toggle('dark-theme');
})
```

Damit das JavaScript funktioniert, muss der Button die ID `theme-button` haben.

Zeit: ~ 15 min

Solution: https://codesandbox.io/s/scy0o

Motion, Transitions und Animationen

Mit Animationen können mehrere Sachen erzielt werden:

- **Informativ**: Informationen können besser wiedergegeben werden

- **Fokus**: Der Fokus des Users kann auf ein gezieltes Element gezogen werden
- **Expressiv**: Aktionen des Users können unterstrichen werden, damit dieser ein Feedback zu seinen Aktionen erhält

Im Web gibt es zwei Möglichkeiten wie Elemente animiert werden können:

- **Transitions** um eine oder mehrere Properties von *x* zu *y* zu animieren
- **Keyframe-Animations** um komplexe Animationsabläufe zu erstellen

Transitions

- Transitions sind/sollten state-abhängig sein
- Haben immer einen bestimmten 'Start' und ein bestimmtes 'Ende'
- Können auch durch JavaScript getriggert werden (z.B. in dem im DOM eine Klasse hinzugefügt wird)

Beispiele

```
a {
  color: blue;

  /* transition: <property> <duration> [<timing-function>]; */
  transition: all .3s ease;
}

a:hover {
  color: red
}
```

Mehrere transition properties

```
a {
  color: blue;
  opacity: 1;

  /* transition: <property> <duration> [<timing-function>]; */
  transition: color .3s ease, opacity 1s;
}

a:hover {
  color: red;
  opacity: .6;
}
```

Demo 🤯

- Transition

Animations

- Kann komplexe Animationen mit mehreren Schritten abbilden
- Kann sich wiederholen oder auch pausiert werden
- Sind mächtiger und flexibler als Transitions, aber auch komplizierter in der Nutzung
- Eignen sich, um mit JavaScript getriggert zu werden (z.B. das hinzufügen einer Klasse im DOM)

Beispiele

```
/* Deklaration der Animation */
@keyframes animation-name {
  0% {
    opacity: 0;
  }
  30% {
    opacity: .6;
  }
  100% {
    opacity: 1;
  }
}

/* Nutzung der Animation */
.spinner {
  animation: animation-name 1s;
}
```

```
/* Name of the animation */
animation-name: none;

/* Xs, or Xms */
animation-duration: 0s;

/* Number, or "infinite" */
animation-iteration-count: 1;

/* normal, reverse, alternate, alternate-reverse */
animation-direction: normal;

/* or: ease-in, ease-out, ease-in-out, linear, cubic-bezier(x1, y1, x2, y2) */
animation-timing-function: ease;

/* or: forwards, backwards, both */
animation-fill-mode: none;

/* Xs, or Xms */
animation-delay: 2s;
```

Demo 🤯

- Keyframe-Animation

Hilfreiche Links

- Material Design - Motion
- CSS Tricks - Transitions
- CSS Tricks - Animations

Practice 🔥

Öffne diese **CodeSandbox** als Startpunkt.

- [ ] Die Navigationslinks sollten eine Transition erhalten, damit der Wechsel der `color` beim Hover animiert wird
    - [ ] Verwende bei der Transition eine Timingfunktion, sodass die Transition schnell beginnt und langsam ausklingt. Nutze dafür ein Hilfstool wie https://cubic-bezier.com/
- [ ] Passe den Send-Button des Formulars an, sodass der Hintergrund des Buttons abwechslungsweise in verschiedenen Regenbogenfarben gezeigt wird

Zeit: ~ 10 min

Solution: https://codesandbox.io/s/3i6kc

Hilfreiche Links

- New CSS Features In 2022