HTML & CSS

typische Technologien einer Website

- HTML
  Struktur einer Website
- CSS
  Graphische Darstellung der Einzelnen Elemente
- JavaScript
  Animation, dynamische Änderungen

HTML

Hypertext Markup Language

Elemente & Attribute

Elemente

- beschreiben Inhaltselemente wie z.B. Überschriften
- bestehen aus einem Start- und End-Tag
  `<html>...</html>`
- dazwischen können andere Elemente oder Inhalt sein

```
<html>
  <head>...</head>
  <body>Beispiel</body>
</html>
```

Attribute

- Zusätzliche Informationen über HTML Elemente
  `<html lang="en">`
- bestehen aus einem Namen und einem Wert
  `name="wert"`
- ein HTML Element kann mehrere Attribute enthalten

Grundgerüst einer HTML-Datei

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ... Infos über das Dokument ...
  </head>
  <body>
    ... eigentlicher Inhalt ...
  </body>
</html>
```

- **Doctype**
  beschreibt die Art des Dokuments
- **Head**
  Enthält Informationen über das Dokument
- **Body**
  Eigentlicher Inhalt

Beispiel 1

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Beispiel 1</title>
  </head>
  <body>
    <h1>Beispiel 1</h1>
    <p>Erstes Beispiel</p>
  </body>
</html>
```

HTML Head

Enthält Information über das Dokument wie z.B.

- Zeichensatz
```
<meta charset="utf-8">
```
- Titel des HTML Dokuments
```
<title>Beispiel 1</title>
```
- Favicon
```
<link rel="icon" href="favicon.ico" type="image/x-icon">
```
- Keywords & Description für die Suchmaschinenoptimierung
```
<meta name="keywords" content="HTML, CSS">
<meta name="description" content="Kurzbeschreibung der Seite">
```
- CSS Stylesheets
```
<link rel="stylesheet" href="custom.css" type="text/css">
```
- JavaScript
```
<script src="jquery.min.js"></script>
```

HTML Body Elemente

Inhalts-Elemente

- Überschriften (h1-h6)
- Absätze/Zeilenumbrüche
- Aufzählungen/Listen
- Verlinkungen
- Bilder
- Tabellen
- Formulare
- Blöcke
- IFrames
- ...

HTML Body Überschriften

Für Überschriften (Headings) gibt es die Elemente h1-h6:

```
<h1>Überschrift 1</h1>
<h2>Überschrift 2</h2>
<h3>Überschrift 3</h3>
<h4>Überschrift 4</h4>
<h5>Überschrift 5</h5>
<h6>Überschrift 6</h6>
```

HTML Body Absätze

Absätze (Paragraphs) werden durch das Element <p> realisiert, Zeilenumbrüche (Line breaks) durch
:

```
<p>Absatz1<br>Zeilenumbruch1 mit weiterem Text...</p>
<p>Absatz2<br>Zeilenumbruch2<br>Zeilenumbruch3 mit weiterem Text...</p>
```

Absatz1
Zeilenumbruch1 mit weiterem Text...

Absatz2
Zeilenumbruch2
Zeilenumbruch3 mit weiterem Text...

HTML Body Listen

Es gibt zwei unterschiedliche Arten von Listen

- Aufzählungen (unordered lists) - Element
- Nummerierungen (ordered lists) - Element

Einzelne Elemente in den Listen werden mit Hilfe das Element <li> erstellt:

```
<ul>
  <li>Punkt 1</li>
  <li>Punkt 2</li>
  <li>Punkt 3</li>
</ul>
```

- Punkt 1
- Punkt 2
- Punkt 3

```
<ol>
  <li>Punkt 1</li>
  <li>Punkt 2</li>
  <li>Punkt 3</li>
</ol>
```

1. Punkt 1
2. Punkt 2
3. Punkt 3

HTML Body Verlinkungen

Mit Hilfe des <a> Elements (Anchor) können Verlinkungen realisiert werden.

Zusätzliche Attribute

- **href** (Hyper Reference): gibt die zu verlinkende URL an
- **target**: gibt an, in welchem Fenster der Link geöffnet werden soll (_blank bedeutet neues Fenster)
- **title**: bietet eine zusätzliche Beschreibung der Verlinkung an

```
<a href="https://www.wifi-ooe.at" target="_blank" title="WIFI Oberösterreich">WIFI Oberösterreich</a>
```

WIFI Oberösterreich

HTML Body Bilder

Für die Einbindung von Bildern gibt es das <img> Element

Zusätzliche Attribute

- **src** (Source): Pfad zur Bilddatei
- **alt** (Alternate Text): Alternative Beschreibung
- **height**: Höhe des Bildes (in Pixel)
- **width**: Breite des Bildes (in Pixel)

```
<img src="images/wifi-logo.png" alt="WIFI Logo" height="91" width="72">
```

HTML Body Tabellen

Für die Erstellung von Tabellen gibt es das <table> Element

Struktur einer Tabelle:

- Kopfbereich: <thead> Element
- Datenbereich: <tbody> Element
- Fußbereich: <tfoot> Element

Jeder Bereich kann aus mehreren Zeilen (rows -> <tr> Element) und diese wiederum aus mehreren Spalten (columns -> <td> bzw. <th> Element) bestehen:

```
<table>
  <thead>
    <tr>
      <th>Artikel</th>
      <th>Preis</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>Summe</td>
      <td>€ 30</td>
    </tr>
  </tfoot>...
  ...
  <tbody>
    <tr>
      <td>HTML kurz und gut</td>
      <td>€ 15</td>
    </tr>
    <tr>
      <td>CSS kurz und gut</td>
      <td>€ 15</td>
    </tr>
  </tbody>
</table>
```

Artikel	Preis
HTML kurz und gut	€ 15
CSS kurz und gut	€ 15
Summe	€ 30

HTML Body Formulare

Formulare bestehen aus einer vielzahl an möglichen Elementen:

- dem Formular selbst
  ```
  <form action="send.php" method="post">...</form>
  ```
- einem oder mehreren Formularblöcken (Fieldsets), die wieder eine Überschrift (legend) haben:
  ```
  <fieldset>
    <legend>Persönliche Daten</legend>
  </fieldset>
  ```
- Labels
  ```
  <label for="vorname">Vorname</label>
  ```
- Eingabefelder
  - Text
    ```
    <input type="text" id="vorname" name="vorname" value="">
    ```
  - Passwort
    ```
    <input type="password" id="pwd" name="pwd" value="">
    ```
  - Checkoxen
    ```
    <input type="checkbox" id="cb1" name="cb1" value="Ja">
    ```
  - Radio-Buttons
    ```
    <input type="radio" id="rb1" name="rb" value="Ja">
    <input type="radio" id="rb2" name="rb" value="Nein">
    ```

HTML Body Formulare

- weitere Eingabefelder
  - Versteckte Felder
    ```
    <input type="hidden" name="kundennummer" value="1">
    ```
  - Email
    ```
    <input type="email" id="email" name="email" value="">
    ```
  - Textareas
    ```
    <textarea name="nachricht" rows="5"
    cols="80">Text</textarea>
    ```
  - Buttons
    ```
    <input type="submit" value="Absenden">
    ```
  - Auswahllisten
    ```
    <select name="anrede">
      <option value="m">Herr</option>
      <option value="f">Frau</option>
    </select>
    ```
  - Dateien / Files
    ```
    <input type="file" name="email" multiple>
    ```

HTML Body Struktur-elemente

Strukturelemente sollen dazu dienen HTML Dokumenten noch mehr Struktur bzw. semantische Bedeutung zu geben:

- Kopfzeile
  `<header>...</header>`
- Fußzeile
  `<footer>...</footer>`
- Navigation
  `<nav>...</nav>`
- Bereiche
  `<section>...</section>`
- Artikel
  `<article>...</article>`
- relevante Informationen zu einem Article bzw. zu einer Section
  `<aside>...</aside>`
- Container
  `<div>...</div>`
- Wort-Container
  `<span>...</span>`

Weitere Informationen zu Semantischen Elementen:
http://www.w3schools.com/html/html5_semantic_elements.asp

CSS

Cascading Style Sheets

Einbindung in HTML

1. Direkt bei den HTML Elementen über das Style-Attribut

```
<h1 style="color: #444444;"></h1>
```

2. Über ein Style Tag im HTML-Head bzw. -Body

```
<style>
  h1 {
    color: #999999;
  }
</style>
```

3. Einbindung über eine externe Datei

```
<link rel="stylesheet" href="custom.css" type="text/css">
```

Die hier angeführte Reihenfolge entspricht auch der Wichtigkeit der Regeln. D.h. ein direkt beim Element angeführter Style hat eine höhere Priorität, als die Regel einer externen Datei.

Ausnahme: Verwendung einer !important Regel in einer externen Datei

```
h1 {
  color: #999999 !important;
}
```

Aufbau einer CSS Regel

```
selektor, selektor2 {
  eigenschaft: wert;
  eigenschaft2: wert;
}
```

CSS Selektoren

Selektoren, dienen wie der Name schon sagt, dazu HTML Elemente zu selektieren:

- Universalselektor
  *
- Element-/Typselektoren
  html, body, h1, p,...
- Klassenselektor
  .klassenname
- ID-Selektor
  #idname
- Attributselektoren
  input[id], input[type=”text”]
- Pseudoklassen/Pseudoselektoren
  - :hover, :focus
  - :last-child, :first-child, :nth-child()
  - :empty
- Hierarchie Selektoren
  - article h1 -> alle h1 Nachfolger von article
  - article > h1 -> alle direkten h1 Nachfolger von article

Elementselektor

Ein Elementselektor verwendet den Namen des Elements und ist somit für alle Elemente mit diesem Namen gültig.

HTML

```
<h1>...</h1>
```

CSS

```
h1 {
  color: #999999;
}
```

ID-Selektor

ID-Selektoren verwenden das Attribut “id” eines Elements zur Selektion. Da eine ID im Normalfall nur einmal im HTML Dokument enthalten ist, kann damit ein einzelnes Element identifiziert werden.

HTML

```
<h2 id="first-headline">...</h2>
<h2>...</h2>
```

CSS

```
#first-headline {
  color: #999999;
}
```

Hinweise:

- Jedes Element hat maximal eine eindeutige ID

Klassenselektor

Klassenselektoren verwenden das Attribut “class” für die Selektion der Elemente. Im Gegensatz zu Elementselektoren können mittels des Klassen-Attributes auch unterschiedliche Elemente angesprochen werden.

HTML

```
<h1 class="headline red">...</h1>
<p class="headline">...</p>
```

CSS

```
.headline {
  font-size: 25px;
}
```

Hinweis:

- Ein Element kann mehrere Klassen besitzen (diese werden durch ein Leerzeichen getrennt)

Attributselektor

Attributselektoren verwenden ein beliebiges Attribut bzw. dessen Wert für die Selektion der Elemente. Wie bei den Klassenselektoren können auch hier unterschiedliche Elemente selektiert werden.

HTML

```
<input type="text" name="firstname">
<input type="radio" name="age" value="20">
<input type="radio" name="age" value="30">
```

CSS

```
[value] {
  padding-left: 10px;
}
[type="text"] {
  padding-left: 10px;
}
```

Hinweis:

- Ein Element kann mehrere Klassen besitzen (diese werden durch ein Leerzeichen getrennt)

Pseudoselektor

Pseudoselektoren sind dazu da, um spezielle Zustände von Elementen gestalten zu können: Besuchte Elemente, gerade aktive Elemente, bestimmte Kinderelemente, etc.

HTML

```
<ul id="navigation">
  <li><a href="...">...</a></li>
  <li><a href="...">...</a></li>
  <li><a href="...">...</a></li>
</ul>
```

CSS

```
ul li a:hover {
  color: red;
}
ul li:first-child {
  margin-top: 0;
}
```

Beispiel Selektoren:
:hover, :visited, :active, :focus, :first-child, :last-child,...

Hierarchie-selektor

Hierarchie Selektoren nutzen die Verschachtelung bzw. die Beziehung zwischen Elementen um Elemente zu selektieren. Dazu müssen die zwei in Beziehung stehenden Elemente durch ein Leerzeichen im Selektor getrennt werden.

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...</li>
</ul>
```

CSS

```
ul li {
  border: 2px solid black;
}
```

Beschreibung:

- Selektiere alle li-Elemente innerhalb eines ul-Elements

Hierarchie-selektor

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

CSS

```
ul li {
  border: 2px solid black;
}
```

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

Hierarchie-selektor

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

CSS

```
ul ul li {
  border: 2px solid black;
}
```

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

Hierarchie-selektor

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

CSS

```
.level1 li {
  border: 2px solid black;
}
```

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

Hierarchie-selektor

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

CSS

```
.level2 li {
  border: 2px solid black;
}
```

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

Hierarchie-selektor

HTML

```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

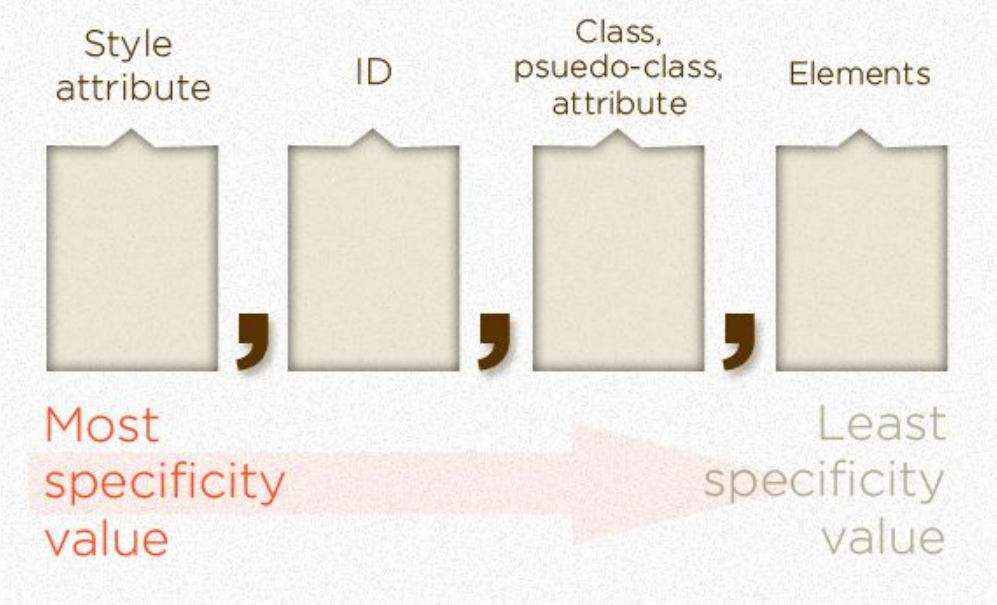
CSS

```
.level1 > li {
  border: 2px solid black;
}
```

HTML

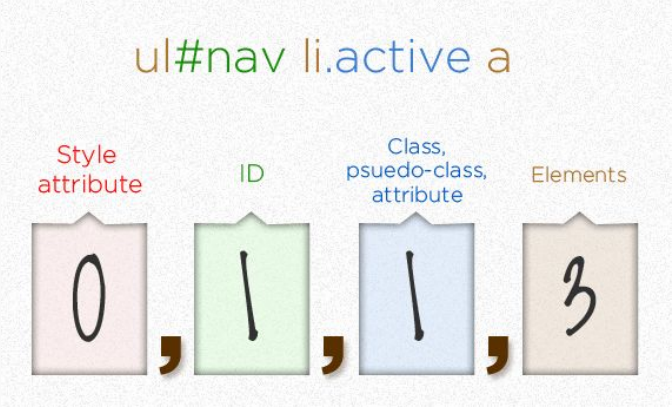
```
<ul class="level1">
  <li>...</li>
  <li>...
    <ul class="level2">
      <li>...</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

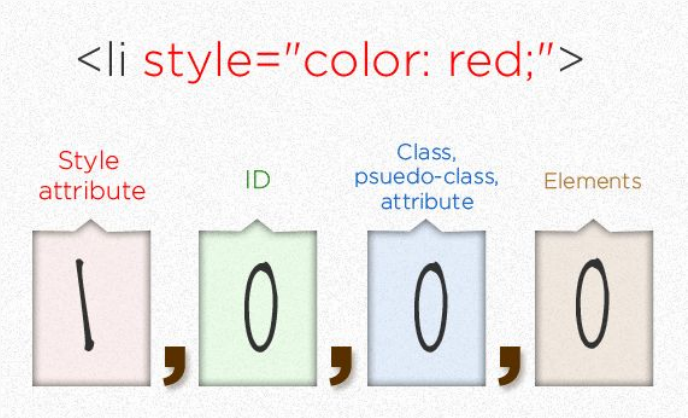
Selektor - Reihenfolge



Bildquelle: CSS Tricks

Selektor - Reihenfolge Beispiele





Bildquelle: CSS Tricks

Block-/Inline-Elemente

Inline-Elemente können im Gegensatz zu Block-Elementen auch innerhalb einer Textzeile angezeigt werden. Sie nehmen also nur so viel Platz ein, den sie auch brauchen.

Inline-Elemente
a, img, input, span, b, i, em, abbr, br, button, select, textarea, sub, super,...

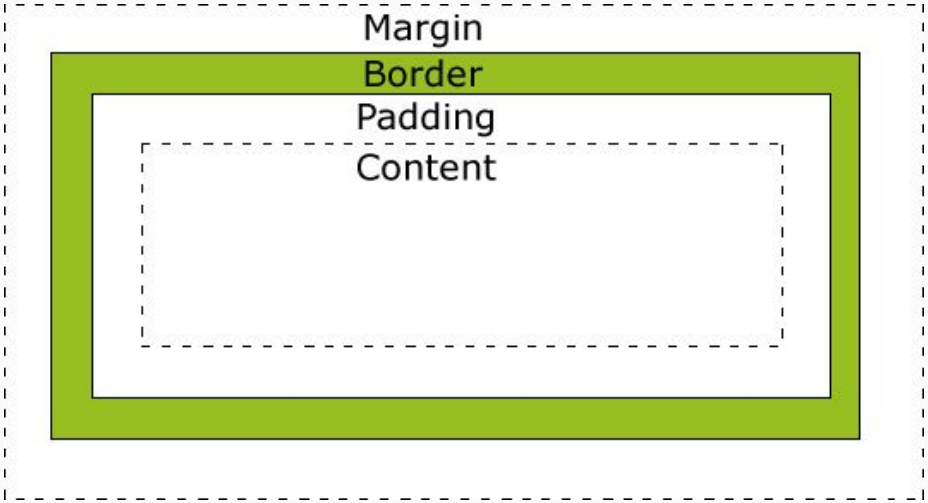
Block-Elemente verwenden standardmäßig die gesamte Breite des Elternelements und erzeugen automatisch einen Zeilenumbruch. Zudem kann auch ein Außenabstand (margin) zu anderen Block-Elementen definiert werden.

Block-Elemente
h1-h6, p, div, article, section, header, footer, ul, ol, li, table, form, fieldset,...

Mit CSS kann die Art jedoch geändert werden:

```
display: block;      /* Element wird als Block-Element dargestellt */
display: inline;     /* Element wird als Inline-Element dargestellt */
```

CSS Box Model



Bildquelle: W3 Schools

CSS Positionierung

Spalten & schwebende Elemente durch die CSS Eigenschaft “float”

```
.column-left {
  float: left;
}
```

Mögliche Werte: left, right, none, inherit

Andere Positionierungsmöglichkeiten

```
.column-left {
  position: absolute;
}
```

Mögliche Werte: static, absolute, fixed, relative, initial, inherit

Flexible Layouts

Die bisher kennengelernten Möglichkeiten um Elemente vor allem nebeneinander darzustellen (Float, Inline-Block) sind zum einen weder komfortabel noch können damit bestimmte Anforderungen umgesetzt werden:

- Gleiche Höhe für alle Elemente
- Definierbare Reihenfolge von Kind-Elementen
- Flexible Breite
- Einfache Ausrichtung der Elemente

Darum wurde mit CSS 3 das **Flexbox-Layout** eingeführt.

Beim Eltern-Element spricht man vom Flex-Container, bei den Kind-Elementen von Flex-Items.

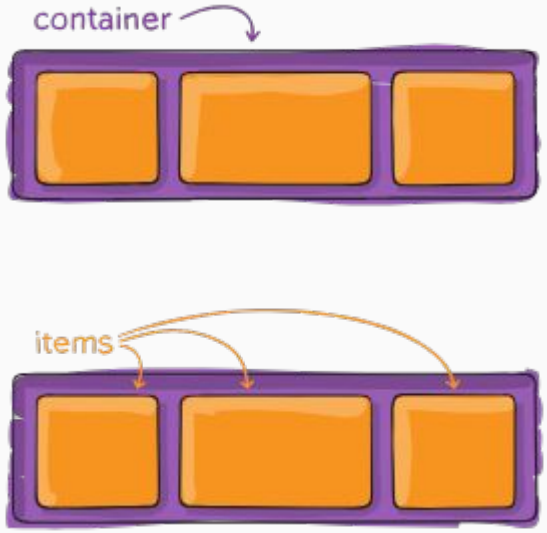
Flexbox-Layout

HTML

```
<div class="container">
  <div class="item">...</div>
  <div class="item">...</div>
  <div class="item">...</div>
</div>
```

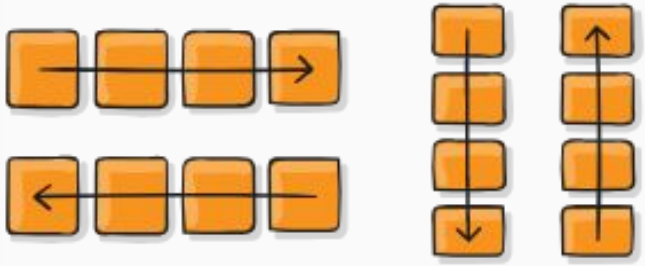
CSS

```
.container {
  display: flex;
}
```



Bildquelle: CSS Tricks

Flexbox-Layout flex-direction (container)

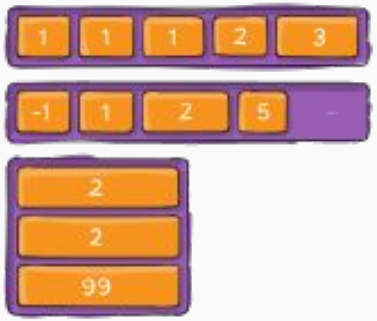


Bildquelle: CSS Tricks

```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
}
```

row (standard): Kind-Elemente von links nach rechts in einer Zeile
row-reverse: Kind-Elemente von rechts nach links in einer Zeile
column: Kind-Elemente von oben nach unten in einer Spalte
column-reverse: Kind-Elemente von unten nach oben in einer Spalte

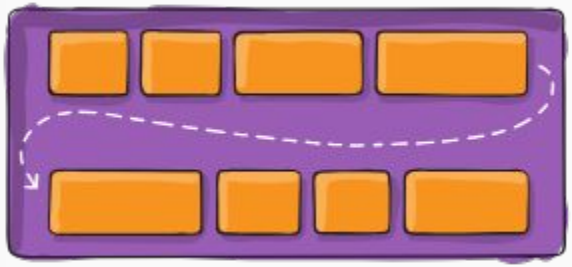
Flexbox-Layout order (items)



Bildquelle: CSS Tricks

```
.item {
  order: <integer>; /* default is 0 */
}
```

Flexbox-Layout flex-wrap (container)



Bildquelle: CSS Tricks

```
.container {
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

nowrap (standard): kein Umbruch, Breite der Kinder wird automatisch entsprechend angepasst, sodass diese in einer Zeile Platz finden.
wrap: Umbruch, wenn Breite der Kind-Elemente zu breit für Zeile
wrap-reverse: Wie wrap, jedoch haben Kind-Elemente umgekehrte Reihenfolge.

Flexbox-Layout flex-grow (items)

Bildquelle: CSS Tricks

```
.item {
  flex-grow: <number>; /* default is 0 */
}
```

Die Nummer gibt an, wie der freie Platz innerhalb eines Flexbox-Containers auf die einzelnen Elemente aufgeteilt werden soll. Haben die Elemente alle die gleiche Nummer (z.B. 1), wird der Platz gleich aufgeteilt. Hat eines einen höheren Wert, so (z.B. 2), so bekommt dieses doppelt so viel Platz wie die restlichen Elemente.

Flexbox-Layout flex-shrink (items)

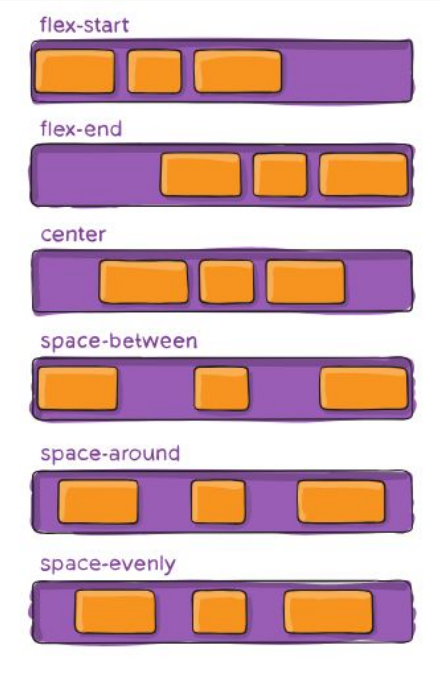
```
.item {
  flex-shrink: <number>; /* default is 1 */
}
```

Gegenteil zu flex-grow:
Wenn die Elemente in einer Zeile/Spalte nicht mehr Platz haben, gibt dieser Wert an, in welchem Verhältnis die Breite/Höhe der Elemente verringert werden soll.

Flexbox-Layout justify-content (container)

```
.container {
  justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;
}
```

Horizontale Ausrichtung der Kind-Elemente innerhalb des Containers:
flex-start (standard): linksbündige Anzeige
flex-end: rechtsbündige Anzeige
center: Zentrierte Anzeige
space-between: Blocksatz Anzeige (freier Platz zwischen den Elementen)
space-around: Um jedes Element herum wird der freie Platz gleich aufgeteilt -> *Hinweis*: treffen zwei Elemente aufeinander so ist zwischen ihnen natürlich der doppelte Abstand
space-evenly: Wie space-around nur, dass Abstand vor bzw. zwischen den Elementen immer gleich.

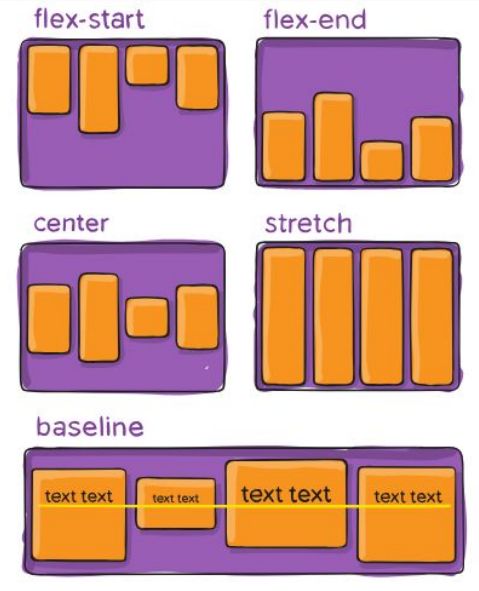


Bildquelle: CSS Tricks

Flexbox-Layout align-items (container)

```
.container {
  align-items: stretch | flex-start | flex-end | center | baseline;
}
```

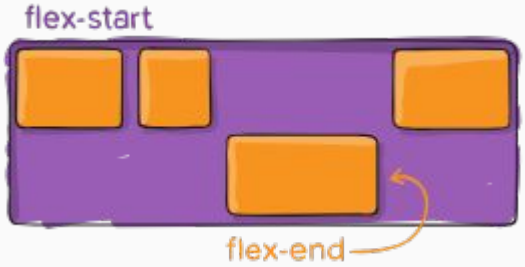
Vertikale Ausrichtung der Kind-Elemente innerhalb des Containers:
stretch (standard): die Höhe alle Elemente wird an die Zeilenhöhe angepasst
flex-start: Ausrichtung an der Oberkante
flex-end: Ausrichtung an der Unterkante
center: Zentrierte Ausrichtung
baseline: Elemente werden anhand ihrer Baseline ausgerichtet.



Bildquelle: CSS Tricks

Flexbox-Layout align-self (item)

```
.item {
  align-self: auto | flex-start | flex-end
| center | baseline | stretch;
}
```



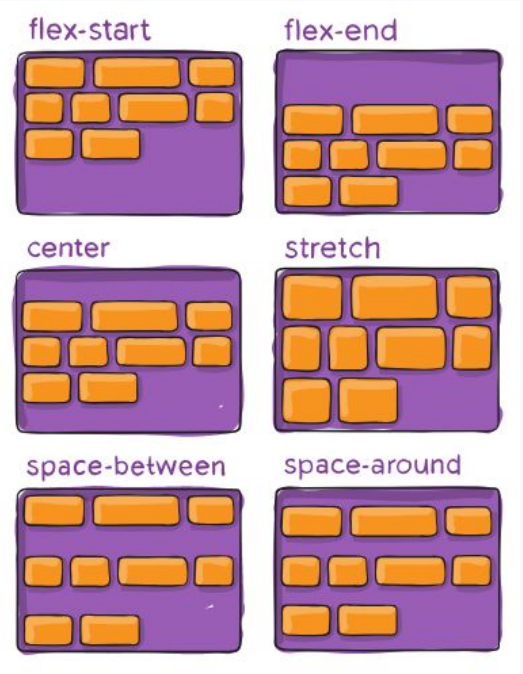
Bildquelle: CSS Tricks

Mit dieser Eigenschaft kann die Standard-Ausrichtung (vom Container vorgegeben) eines Kind-Elements überschrieben werden.

Flexbox-Layout align-content (container)

```
.container {
  align-content: flex-start | flex-end |
center | space-between | space-around |
stretch;
}
```

Wenn der Container eine Mindesthöhe hat und die Zeilen den Platz nicht ausfüllen:
flex-start (standard): Ausrichtung der Zeilen an der Oberkante des Containers
flex-end: Ausrichtung der Zeilen an der Unterkante des Containers
center: Zentrierte Ausrichtung der Zeilen
stretch: Aufblasen der Zeilenhöhen, sodass diese den vollen Platz einnehmen
space-between: freien Platz zwischen den Zeilen aufteilen
space-around: freien Platz zwischen bzw. vor und nach den Zeilen gleich aufteilen



Bildquelle: CSS Tricks

Viewport/Media Queries

Um die Darstellung von Webseiten-Inhalten bzw. des Layouts an bestimmte Ausgabegeräte anzupassen, kann man Media Queries verwenden.

Media Queries erlauben es, abhängig von einer bestimmten Eigenschaft (z.B. Bildschirmbreite) eigene CSS-Regeln zu definieren.

Voraussetzung dafür ist, dass im HTML-Head ein entsprechender Viewport gesetzt ist:

HTML

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

CSS

```
div { width: 33%; }

@media(max-width: 767px){
  div { width: 100%; }
}
```

nützliche Quellen

- SELFHTML - HTML Referenz
  https://selfhtml.org/
- W3Schools
  http://www.w3schools.com/
- W3Schools - HTML Referenz
  http://www.w3schools.com/tags/default.asp
- Semantische Elemente
  http://www.w3schools.com/html/html5_semantic_elements.asp
  http://html5doctor.com/
- W3Schools - CSS Referenz
  http://www.w3schools.com/cssref/
- CSS - Selektor Reihenfolge berechnen
  http://specificity.keegan.st/
- Pseudo-Klassen
  https://css-tricks.com/pseudo-class-selectors/
- CSS-Tricks (Flexbox-Layout)
  https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- W3Schools - Viewport
  https://www.w3schools.com/css/css_rwd_viewport.asp