CSS

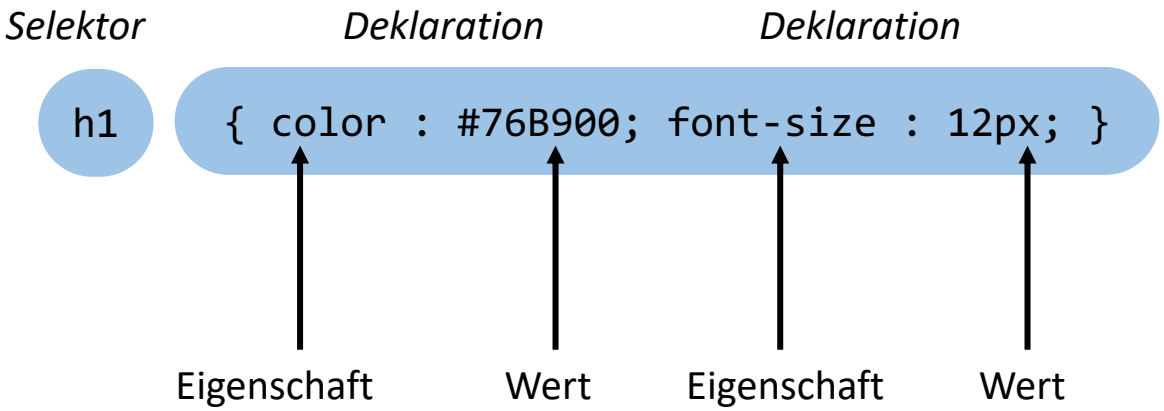
Einführung

CSS

- **C**ascading **S**tyle**s**heets

- HTML ist für den Inhalt
- CSS ist für die Form
- CSS beschreibt, wie HTML-Elemente dargestellt werden sollen (Aussehen)

- CSS-Syntax:

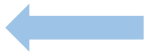


CSS

Allgemeiner Aufbau

```
Selektor {
        Deklaration1;
        Deklaration2;
        Deklaration3;
}

Selektor {
        Eigenschaft1: Wert;
        Eigenschaft2: Wert1 Wert2 Wert3;
}
```

```
p {
    margin: 2px;
    padding: 0.5em 0;
    display: block;
}
```



Beispiel

CSS

Eigenschaften werden kaskadierend weitergegeben

CSS:

```
<style>

   body {
      font-family:Verdana;
      color: #ba0e25;     /* red */
      font-style:italic;
   }

   div {
      font-weight : bold;
   }

   p {
      color : blue;
   }

</style>
```

HTML:

```
<body>
<h1>Überschrift</h1>
<p>Absatz außerhalb des &lt;div&gt;-
Tags</p>
<div>
   jetzt das &lt;div&gt;
   <p>Absatz innerhalb des &lt;div&gt;-
Tags</p>
</div>
</body>
```

Einbinden von CSS

1. Variante: innerhalb des **<style>**-Elementes im **<head>**-Bereich

```
<style type="text/css">

        body {
                font-family:Verdana;
                color:#0082D1;
        }

        h1 {
                font-weight:normal;
                color: #FF5F00;
        }

        p {
                color: #AFAFAF;
        }

</style>
```

Einbinden von CSS

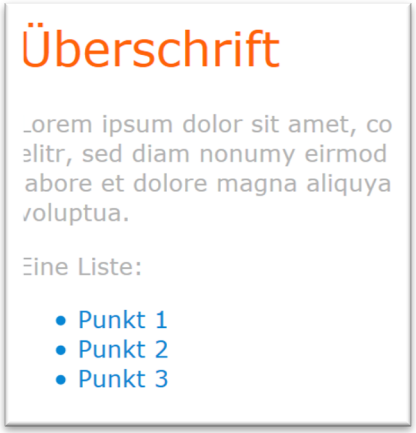
2. Variante: Einbinden einer externen **.css**-Datei im **<head>**-Bereich

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

format.css

```
body {
        font-family:Verdana;
        color:#0082D1;
}

h1 {
        font-weight:normal;
        color: #FF5F00;
}

p {
        color: #AFAFAF;
}
```

Einbinden von CSS

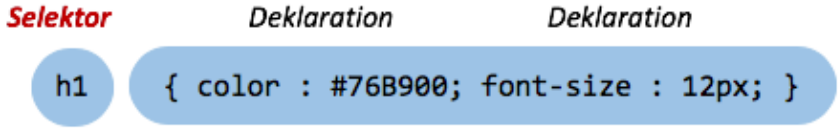


3. Variante: Inline-Style in HTML-Elementen

```
<body style="font-family:Verdana;color:#0082D1;">
        <h1 style="font-weight:normal;color: #FF5F00;">Überschrift</h1>
        <p style="color: #AFAFAF;">Lorem ipsum dolor sit amet, consetetur
        sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore
        et dolore magna aliquyam erat, sed diam voluptua.
        </p>
        <p> Eine Liste:
                <ul>
                        <li>Punkt 1</li>
                        <li>Punkt 2</li>
                        <li>Punkt 3</li>
                </ul>
        </p>
</body>
```

Selektoren

HTML-Elemente




```
* {
        margin: 0;
        padding: 0;
}
```

Universalselektor
(für alle HTML-Elemente)

```
HTML-Tag {
        Eigenschaft1: Wert;
        Eigenschaft2: Wert1 Wert2 Wert3;
}
```

allgemein

```
h1 {
        padding: 0.5em 0 0.5em 1em;
        font-family: 'Optimus Princeps', Georgia, serif;
        font-size: 1.5em;
}
```

Beispiel

Selektoren

Operatoren

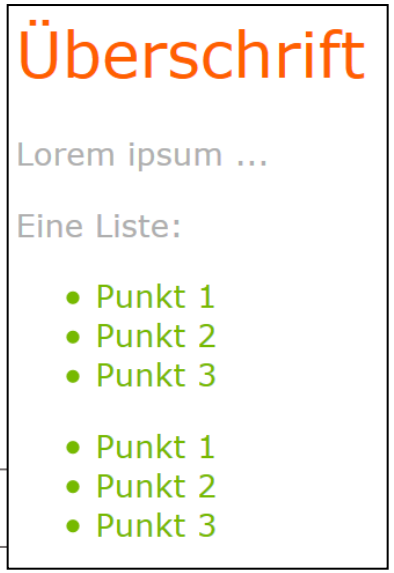
Beispiel:

html:

```
<body>
    <h1>Überschrift</h1>
    <div>Lorem ipsum ... </div>
    <div> Eine Liste:
        <ul>
            <li>Punkt 1</li>
            <li>Punkt 2</li>
            <li>Punkt 3</li>
        </ul>
    </div>
    <ul>
        <li>Punkt 1</li>
        <li>Punkt 2</li>
        <li>Punkt 3</li>
    </ul>
</body>
```

css:

```
body {
    font-family:Verdana;
    color:#0082D1;
}
h1 {
    font-weight:normal;
    color: #FF5F00;
}
div {
    color: #AFAFAF;
}
ul {
    color:#76B900;
}
```



Selektoren

Operatoren (Leerzeichen)

el1 el2 → alle el2 IN el1

html:

```
<body>
    <h1>Überschrift</h1>
    <div>Lorem ipsum ... </div>
    <div> Eine Liste:
        <ul>
            <li>Punkt 1</li>
            <li>Punkt 2</li>
            <li>Punkt 3</li>
        </ul>
    </div>
    <ul>
        <li>Punkt 1</li>
        <li>Punkt 2</li>
        <li>Punkt 3</li>
    </ul>
</body>
```

css:

```
body {
    font-family:Verdana;
    color:#0082D1;
}
h1 {
    font-weight:normal;
    color: #FF5F00;
}
div ul {
    color: #AFAFAF;
}
```

Überschrift

Lorem ipsum ...
Eine Liste:

- Punkt 1
- Punkt 2
- Punkt 3

- Punkt 1
- Punkt 2
- Punkt 3

Selektoren

Operatoren (Komma)

`el1, el2` → alle el1 UND alle el2

html:

```
<body>
    <h1>Überschrift</h1>
    <div>Lorem ipsum ... </div>
    <div> Eine Liste:
        <ul>
            <li>Punkt 1</li>
            <li>Punkt 2</li>
            <li>Punkt 3</li>
        </ul>
    </div>
    <ul>
        <li>Punkt 1</li>
        <li>Punkt 2</li>
        <li>Punkt 3</li>
    </ul>
</body>
```

css:

```
body {
    font-family:Verdana;
    color:#0082D1;
}
h1 {
    font-weight:normal;
    color: #FF5F00;
}
div, ul {
    color: #AFAFAF;
}
```

Überschrift

Lorem ipsum ...
Eine Liste:

- Punkt 1
- Punkt 2
- Punkt 3

- Punkt 1
- Punkt 2
- Punkt 3

Selektoren

Operatoren (>)

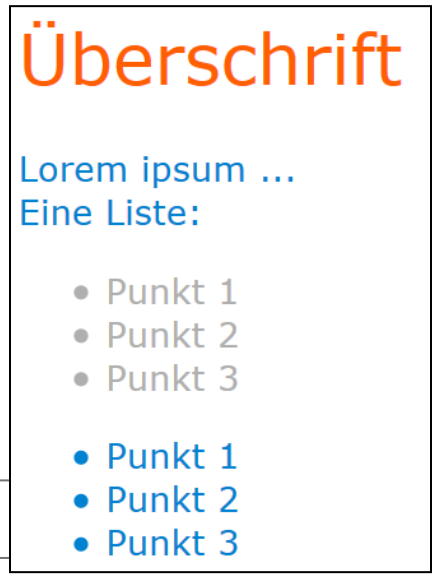
`el1 > el2` → alle el2 direktes KIND von el1

html:

```
<body>
    <h1>Überschrift</h1>
    <div>Lorem ipsum ... </div>
    <div> Eine Liste:
        <ul>
            <li>Punkt 1</li>
            <li>Punkt 2</li>
            <li>Punkt 3</li>
        </ul>
    </div>
    <ul>
        <li>Punkt 1</li>
        <li>Punkt 2</li>
        <li>Punkt 3</li>
    </ul>
</body>
```

css:

```
body {
    font-family:Verdana;
    color:#0082D1;
}
h1 {
    font-weight:normal;
    color: #FF5F00;
}
div > ul {
    color: #AFAFAF;
}
```



Selektoren

Operatoren (+)

`el1 + el2` → alle el2, die direkt NACH el1

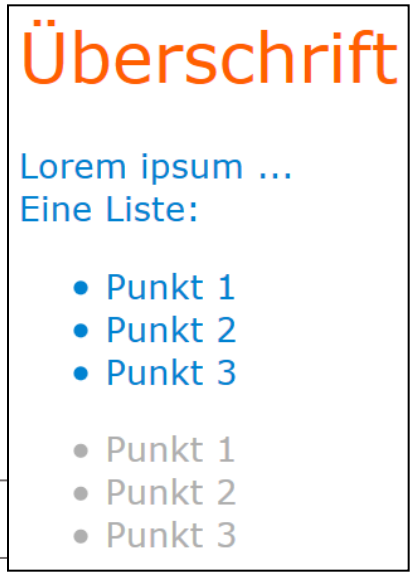
html:

```
<body>
    <h1>Überschrift</h1>
    <div>Lorem ipsum ... </div>
    <div> Eine Liste:
        <ul>
            <li>Punkt 1</li>
            <li>Punkt 2</li>
            <li>Punkt 3</li>
        </ul>
    </div>
    <ul>
        <li>Punkt 1</li>
        <li>Punkt 2</li>
        <li>Punkt 3</li>
    </ul>
</body>
```

css:

```
body {
    font-family:Verdana;
    color:#0082D1;
}
h1 {
    font-weight:normal;
    color: #FF5F00;
}
div + ul {
    color: #AFAFAF;
}
```

Häufig: der Absatz, der direkt nach einer Überschrift kommt



Selektoren

Operatoren (~)

`el1 ~ el2` → alle el2, die GESCHWISTER von el1 sind (und NACH e1 kommen)

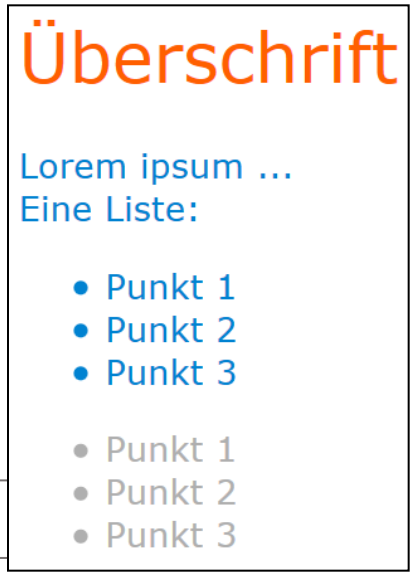
html:

```
<body>
    <h1>Überschrift</h1>
    <div>Lorem ipsum ... </div>
    <div> Eine Liste:
        <ul>
            <li>Punkt 1</li>
            <li>Punkt 2</li>
            <li>Punkt 3</li>
        </ul>
    </div>
    <ul>
        <li>Punkt 1</li>
        <li>Punkt 2</li>
        <li>Punkt 3</li>
    </ul>
</body>
```

css:

```
body {
    font-family:Verdana;
    color:#0082D1;
}
h1 {
    font-weight:normal;
    color: #FF5F00;
}
div ~ ul {
    color: #AFAFAF;
}
```

Alle jüngeren Geschwister



CSS
Pseudo-Klassen

- Mit Pseudo-Klassen können spezielle Zustände eines Elementes definiert werden
- Beispiele:
  - Ändern des Aussehens von Elementen, wenn man mit der Maus darüber fährt (mouse over)
  - Ändern des Aussehens von Hyperlinks, je nachdem, ob sie bereits angeklickt wurden oder noch nicht
  - Ändern des Aussehens von Eingabeelementen, wenn sie Fokus haben
- allg. Syntax:

```
selector:pseudo-class {
    property:value;
}
```

CSS

Pseudo-Klassen für Hyperlinks

- 4 Pseudo-Klassen für Hyperlinks: **`link, visited, hover, active`**

```
<body>
<a href="#">Neu laden</a><br/>
</body>
```

```
a {          /* all links */
   text-decoration: none;
}

a:link {     /* unvisited link */
   color: red;
}

a:visited {  /* visited link */
   color: darkgray;
}

a:hover {    /* mouse over link */
   color: limegreen;
}

a:active {   /* selected link */
   color: orange;
}
```

CSS

weitere Pseudo-Klassen (Auswahl)

Pseudo-Klassen	Beispiel	Erläuterung des Beispiels
`:empty`	`p:empty`	alle <p>-Elemente, die keine Kinder haben
`:first-child`	`p:first-child`	alle <p>-Elemente, die das jeweils erste Kind ihres Elternelementes sind
`:first-of-type`	`p:first-of-type`	alle <p>-Elemente, die das jeweils erste <p>-Element ihres Elternelementes sind
`:nth-child(n)`	`p:nth-child(2)`	alle <p>-Elemente, die das jeweils zweite Kind ihres Elternelementes sind
`:nth-of-type(n)`	`p:nth-of-type(2)`	alle <p>-Elemente, die das jeweils zweite <p>-Element ihres Elternelementes sind
`:only-child`	`p:only-child`	alle <p>-Elemente, die das jeweils einzige Kind ihres Elternelementes sind
`:only-of-type`	`p:only-of-type`	alle <p>-Elemente, die das jeweils einzige <p>-Element-Kind ihres Elternelementes sind

auch `:last-child`, `:last-of-type`, `:nth-last-child(n)`, `:nth-last-of-type(n)`

CSS

Pseudo-Klassen für input-Elemente (Auswahl)

Pseudo-Klassen	Beispiel	Erläuterung des Beispiels
`:checked`	`input:checked`	alle checked <input>-Elemente
`:disabled`	`input:disabled`	alle disabled <input>-Elemente
`:enabled`	`input:enabled`	alle enabled <input>-Elemente
`:focus`	`input:focus`	das <input>-Element, das Fokus hat
`:invalid`	`input:invalid`	alle <input>-Elemente mit einem ungültigen Wert
`:optional`	`input:optional`	alle <input>-Elemente, die kein „required“-Attribut haben
`:read-only`	`input:read-only`	alle <input>-Elemente mit einem „readonly“-Attribut
`:read-write`	`input:read-write`	alle <input>-Elemente, die kein „readonly“-Attribut haben
`:required`	`input:required`	alle <input>-Elemente mit einem „required“-Attribut
`:valid`	`input:valid`	alle <input>-Elemente mit einem gültigen Wert

CSS
Pseudo-Elemente

- Mit Pseudo-Elementen können spezielle Teile eines Elementes gestylt werden
- Beispiele:
  - erster Buchstabe oder erste Zeile eines Elementes besonderer Style
  - Füge Inhalt vor (oder nach) dem Inhalt eines Elementes ein
- allg. Syntax:

```
selector::pseudo-element {
    property:value;
}
```

- **Vergleich**:
  - Pseudoklassen für spezielle Zustände eines Elementes; Syntax: :
  - Pseudoelemente für spezielle Teile eines Elementes; Syntax: ::

CSS

Pseudo-Elemente

Pseudo-Elemente	Beispiel	Erläuterung des Beispiels
`::after`	`p::after`	füge etwas hinter den Inhalt aller <p>-Elemente ein
`::before`	`p::before`	füge etwas vor den Inhalt aller <p>-Elemente ein
`::first-letter`	`p::first-letter`	alle ersten Buchstaben aller <p>-Elemente
`::first-line`	`p::first-line`	alle ersten Zeilen aller <p>-Elemente
`::selection`	`p::selection`	alle vom Nutzer markierten (selektierten) Teile eines <p>-Elementes

HTML + CSS

`class` und `id` - Attribute

- Allen HTML-Elementen kann das **`id-`** und das **`class`**-Attribut zugeordnet werden
- Das **`class`**-Attribut wird verwendet, um für verschiedene Elemente (mit dem gleichen Klassennamen) den gleichen Style zu definieren.
- Das **id**-Attribut wird verwendet, um für **ein** Elemente (mit der entsprechenden Id) einen Style zu definieren – meistens Layout. Wird später auch in javaScript verwendet.
- **`class`**:
  - Mehreren Elementen kann die gleiche Klasse zugeordnet werden.
  - Einem Element können mehrere Klassen zugeordnet werden.
- **`id`**:
  - Jedes Element kann höchstens eine Id haben.
  - Eine Id kann pro HTML-Seite nur einmal vergeben werden.

HTML + CSS

`class` und `id` – Attribute (Syntax)

- Normale Attribut-Syntax:

```
<h1 id="firsth1" class="blackbackground yellowcolor">Überschrift 1</h1>
<h1 id="secondh1" class="bluebackground yellowcolor">Überschrift 2</h1>
<h1 id="thirdh1" class="blackbackground whitecolor">Überschrift 3</h1>
```

- Namen der Klassen und Ids sind selbst vergeben
  - müssen mindestens einen Buchstaben enthalten (mit Buchstaben beginnen)
  - sind case-sensitiv
  - dürfen keine Leerzeichen, Tabs etc. enthalten
- Ids nur einmal!
- Mehrere Klassen durch Leerzeichen trennen

HTML + CSS

`class` und `id` – Attribute (als Selektoren verwenden)

- Klassen und Ids können als Selektoren in CSS verwendet werden
- Klassen über Punkt:

```
.blackbackground {
        background-color: black;
}
```

- Ids über Doppelkreuz:

```
#firsth1 {
   font-family: "Palatino Linotype", Palatino, serif;
   font-variant: all-small-caps;
}
```

HTML + CSS

`class` und `id` – Attribute (Beispiel)

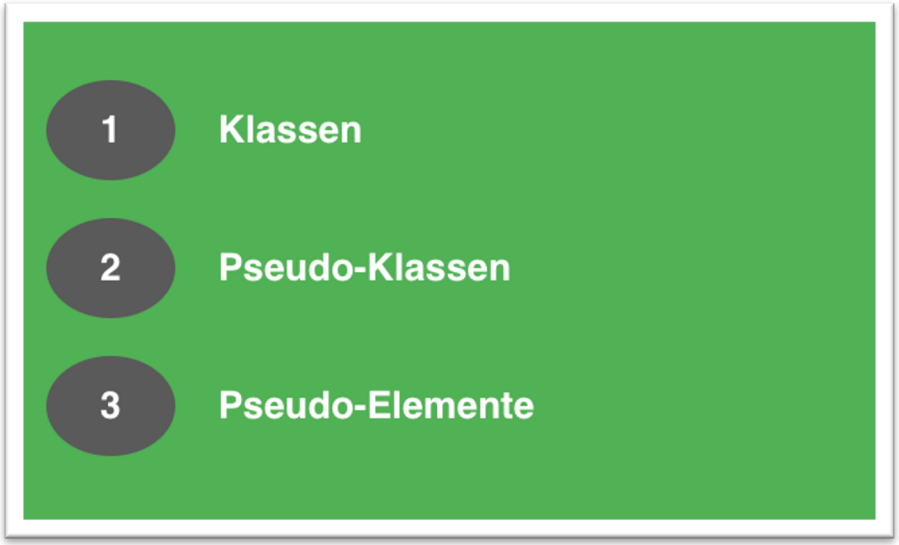
```
.blackbackground {
    background-color: black;
}
.bluebackground {
    background-color: blue;
}
.yellowcolor {
    color:yellow;
}
.whitecolor {
    color:white;
}
```

```
#firsth1 {
    font-family: "Palatino Linotype", Palatino, serif;
    font-variant: all-small-caps;
}
#secondh1 {
    font-family: "Verdana", Geneva, sans-serif;
    font-style: italic;
}
#thirdh1 {
    font-family: "Courier New", Courier, monospace;
    font-size: 110%;
    padding: 40px;
    text-align: center;
```

CSS

Beispiel für `::before` : `counter`

```
<div class="container" style="max-
width:300px">
   <h3>Klassen</h3>
   <h3>Pseudo-Klassen</h3>
   <h3>Pseudo-Elemente</h3>
</div>
```



Variable

```
.container {
      counter-reset: section;
      background:#4CAF50;
      padding: 10px;
      font: 400 12px/1.8 "Lato", sans-serif;
      color: white;
}
.container h3::before {
      counter-increment: section;
      content: counter(section);
      display:inline-block;
      color:white;
      background: #555;
      border: 5px solid #555;
      padding:1px 15px;
      margin-right:16px;
      border-radius: 50%;
}
```

fügt inhalt ein

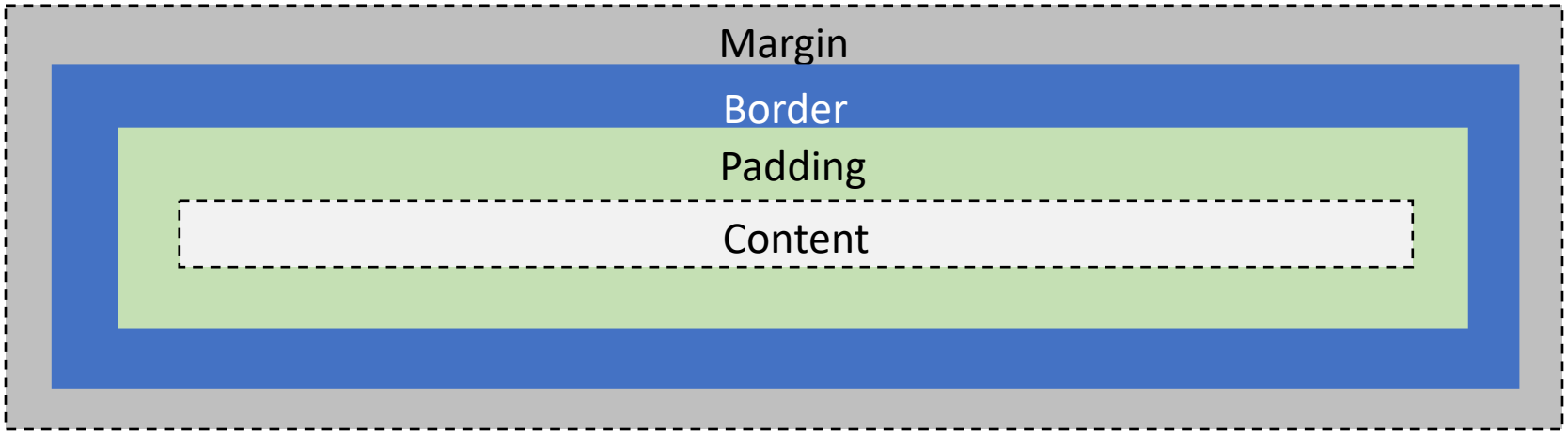
CSS
Attribut-Selektoren

- Mit Attribut-Selektoren können alle HTML-Elemente angesprochen werden, für welche spezielle Attribute oder spezielle Attributwerte definiert sind, z.B.:
- **`a[target]`** alle <a>-Elemente mit einem target-Attribut
- **`a[target="_blank"]`** alle <a>-Elemente, deren target-Attribut den Wert ="_blank" hat

- **`[attribute~="value"]`** enthält einen Wert "value"
- **`[attribute|="value"]`** erster Wert "value" (oder "value-")
- **`[attribute^="value"]`** Wert beginnt mit "value"
- **`[attribute$="value"]`** Wert endet mit "value"
- **`[attribute*="value"]`** Wert enthält "value"

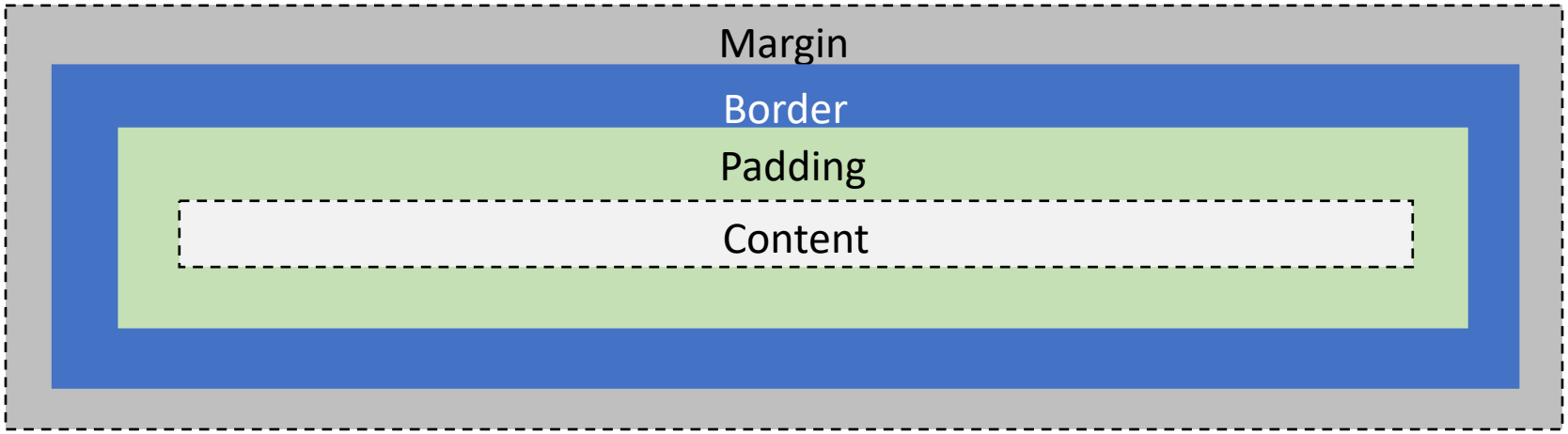
CSS
Box Model

- Alle HTML-Elemente können als eine rechteckige „Box“ verstanden werden
- Diese „Box“ besteht aus:
  - **Inhalt** (Text, Bilder)
  - **Padding** (Abstand des Inhalts zur Border)
  - **Border** (dem Rahmen)
  - **Margin** (Abstand der Border zu den Nachbarelementen)



CSS
Box Model

- 2 Varianten:
  - box-sizing: content-box; // Standard; width und height enthalten nicht // padding, margin oder border
  - box-sizing: border-box; // width und height enthalten padding und // border, aber nicht margin ; aber Probleme // mit Drittanbietern, z.B. Google Maps



CSS

Box Model - Beispiel

```
<style>
   div {
      width: 320px;
      padding: 10px;
      border: 5px solid gray;
      margin: 0;
   }
</style>
```


```
<body>
<h3>Box Model</h3>
<img src="../img/fiw.jpg" alt="fiw logo" style="width:350px"/>
<div>Das FIW-Logo hat eine Breite von 350px (width:350px). Der Inhalt dieser
Box hat eine Breite von 320px. Dazu kommt padding von 10px (auf beiden Seiten)
und ein Rahmen mit der Breite von 5px. Macht zusammen 350px.</div>
</body>
```

CSS
`display`

- Mit der **`display`**-Eigenschaft können wesentliche Eigenschaften für das Layout bestimmt werden
- Zur Erinnerung: HTML-Elemente sind entweder
  - Block-Elemente (starten in einer neuen Zeile und nutzen die gesamte Breite) oder
  - Inline-Elemente (in der gleichen Zeile, nutzen soviel Breite, wie nötig)
- Mit display kann diese Eigenschaft umgeschaltet werden, d.h. aus Block-Elementen können Inline-Elemente werden und umgedreht
- **`display : block`** setzt Elemente auf die Block-Eigenschaft
- **`display : inline`** setzt Elemente auf die Inline-Eigenschaft
- **`display : none`** zeigt Elemente nicht an (benötigen auch keinen Platz)

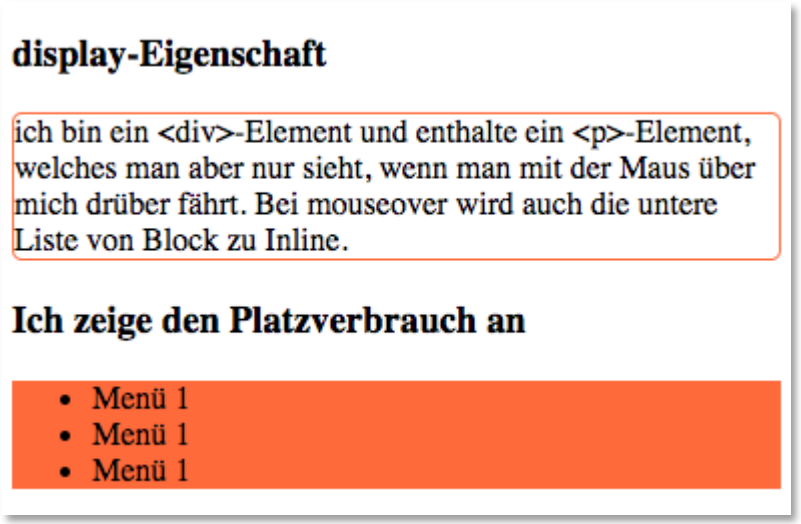
CSS

`display` - Beispiel

```
<body>
<h3>display-Eigenschaft</h3>
<div>
   ich bin ein &lt;div&gt;-Element und enthalte ein &lt;p&gt;-Element, welches ... Inline.
   <p>mich sieht man nur, wenn man mit der Maus über das div fährt. Ich nehme keinen Platz weg, wenn ich nicht sichtbar bin. Das sieht man an der Unterschrift unter mir.</p>
</div>
<h3>Ich zeige den Platzverbrauch an</h3>
<ul>
   <li>Menü 1</li>
   <li>Menü 1</li>
   <li>Menü 1</li>
</ul>
</body>
```

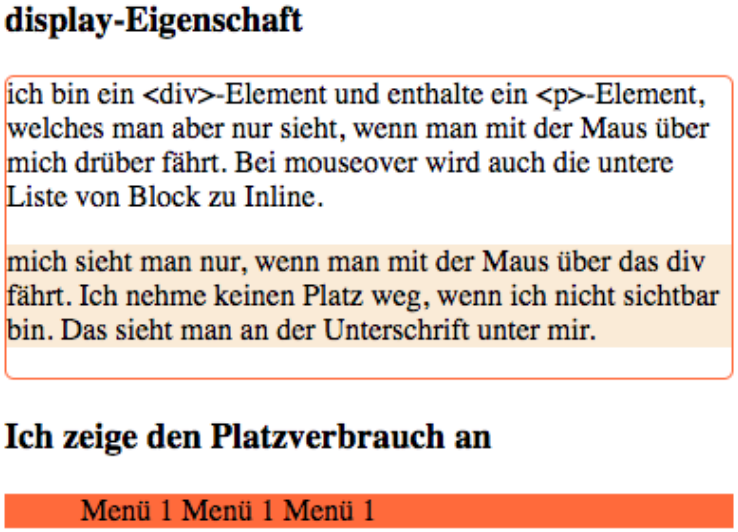
```
<style>
   div {
      border-style: solid;
      border-width: 1px;
      border-color : #ff6a3b;
      border-radius : 5px;
   }
   ul,li {
      background-color : #ff6a3b;
   }
   div:hover ~ ul li{
      display : inline;
   }
   div p {
      display:none;
   }

   div:hover p {
      display:block;
      background-color : antiquewhite;
   }
</style>
```

CSS
`display` - Beispiel



*ohne Mouseover über das *

*mit Mouseover über das *



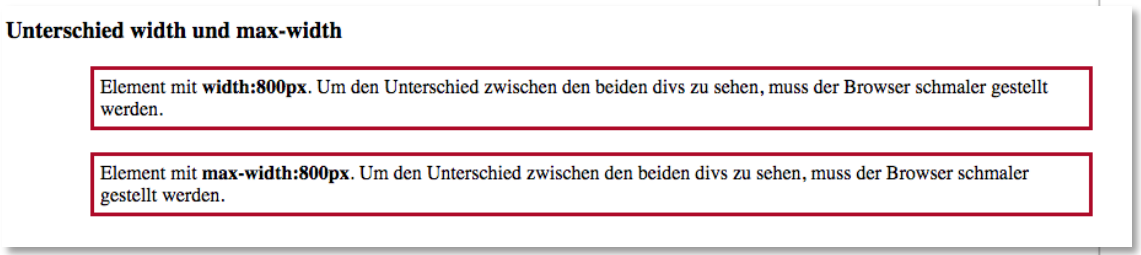
CSS

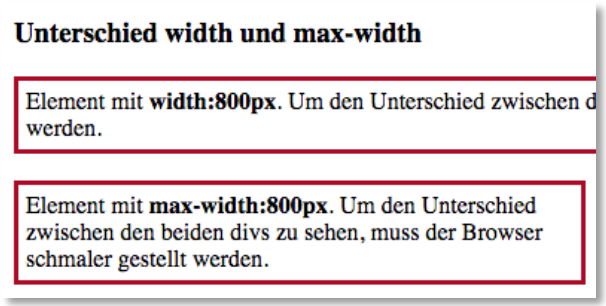
`max-width`

- Block-Elemente nehmen die gesamte verfügbare Breite ein
- Um das zu verhindern, kann man deren Breite setzen.
  - **`width`** legt die Breite fest; ist der Viewport allerdings schmaler, wird das Element nicht mehr vollständig angezeigt
  - **`max-width`** legt die maximale Breite fest. Ist der Viewport schmaler, wird auch das Element angepasst

CSS

max-width - Beispiel






```
<style>
   div {
      margin: auto;
      padding: 5px;
      border: 3px solid #ad0a28;
   }
   div:first-of-type {
      width:800px;
   }

   div:last-of-type {
      max-width:800px;
   }
</style>
```

```
<body>
<h3>Unterschied width und max-width</h3>
<div>
   Element mit <strong>width:800px</strong>. Um ...</div>
<br/>
<div>
   Element mit <strong>max-width:800px</strong>. Um ...</div>
</body>
```

CSS

Gewichtung der Selektoren

```
<ul id="navigation">
   <li><a href="startseite.html" class="link">Startseite</a></li>
   <li><a href="unterseite.html" class="link">Unterseite</a></li>
</ul>
```

```
a:link {                  color:blue      }
.link {                   color:green     }
#navigation a.link {      color:red       }
li a {                    color:magenta   }
#navigation li a {        color:black     }
```

Welche Farbe haben die Hyperlinks?

CSS
Gewichtung der Selektoren

- um die Gewichtung der Selektoren zu ermitteln, werden diese kategorisiert

- **Kategorie A**: erhält den Wert 1, wenn CSS-Format-Definitionen direkt im style-Attribut eines HTML-Elementes notiert sind
- **Kategorie B**: erhält den Wert 1 bei Selektoren für Elemente mit id-Attributen
- **Kategorie C**: Anzahl der von einem Selektor betroffenen Klassen und Pseudoklassen
- **Kategorie D**: Anzahl der von einem Selektor betroffenen Elementnamen und Pseudo-Elemente

- Reihenfolge der Sortierung: A -> B -> C -> D
  - also 1 0 0 0 vor 0 1 2 2

- Universalselektor * wird nicht berücksichtigt

CSS

Gewichtung der Selektoren

```
a:link {                color:blue      }
.link {                 color:green     }
#navigation a.link {    color:red       }
li a {                  color:magenta   }
#navigation li a {      color:black     }
```

a:link 0 0 1 1 (eine Pseudoklasse (link) und ein Element (a))

.link 0 0 1 0 (eine Klasse)

#navigation a.link 0 1 1 1 (Element mit id-Attribut, eine Klasse und eine Element)

li a 0 0 0 2 (zwei Elemente)

#navigation li a 0 1 0 2 (Element mit id-Attribut, zwei Elemente)

CSS

Layout

Container für Elemente

- jedes HTML-Element (egal ob Block oder Inline) existiert in einem übergeordnetem Element (Container)
- das oberste Element in dieser Hierarchie ist das <html>..</html>-Element (wird als ***Viewport*** bezeichnet) → Anzeigebereich, der dem HTML-Dokument im Browser zur Verfügung steht
- Bsp:

```
<html lang="en">

<body>
   <header>
      <h1>Überschrift mit <em>Hervorhebung</em></h1>
   </header>
</body>
</html>
```

Wiederholung Block- und Inline-Elemente

- Blockelemente gehen über die ganze Breite (so breit wie möglich); Höhe so hoch wie nötig
- Inline-Elemente so breit und hoch wie nötig

```
<h1>Block-Element</h1>
<p>Mitten im Absatz ein <em>Inline-Element.</em> Inline-Elemente sind z.B. auch &lt;span&gt; und &lt;img&gt;.</p>
```

```
h1 { background-color: silver; color:black;}
em { background-color: silver; color:black;}
```

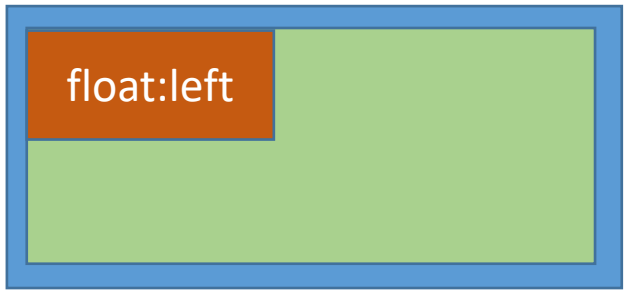
Block-Element

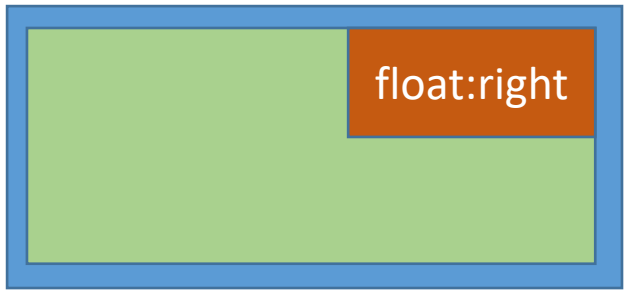
Mitten im Absatz ein *Inline-Element*. Inline-Elemente sind z.B. auch und <img>.

- um Blockelemente nebeneinander anzuordnen,
  - kann man sie entweder mit `display : inline;` als Inline-Elemente definieren oder
  - es ist die `float`-Eigenschaft erforderlich

CSS Layout
`float`

- Mit der **`float`**-Eigenschaft kann man festlegen, wie sich zwei Elemente umfließen
- **`float`** kann folgende 4 Werte haben:
  - **`left`** – das Element ist links in seinem Elternelement (Container) → die anderen Elemente im Conteiner fließen rechts darum herum
  - **`right`** – das Element ist rechts in seinem Elternelement (Container) → die anderen Elemente im Conteiner fließen links darum herum
  - **`none`** – kein Umfließen (default)
  - **`inherit`** – das Element erbt den **`float`**-Wert des Containers



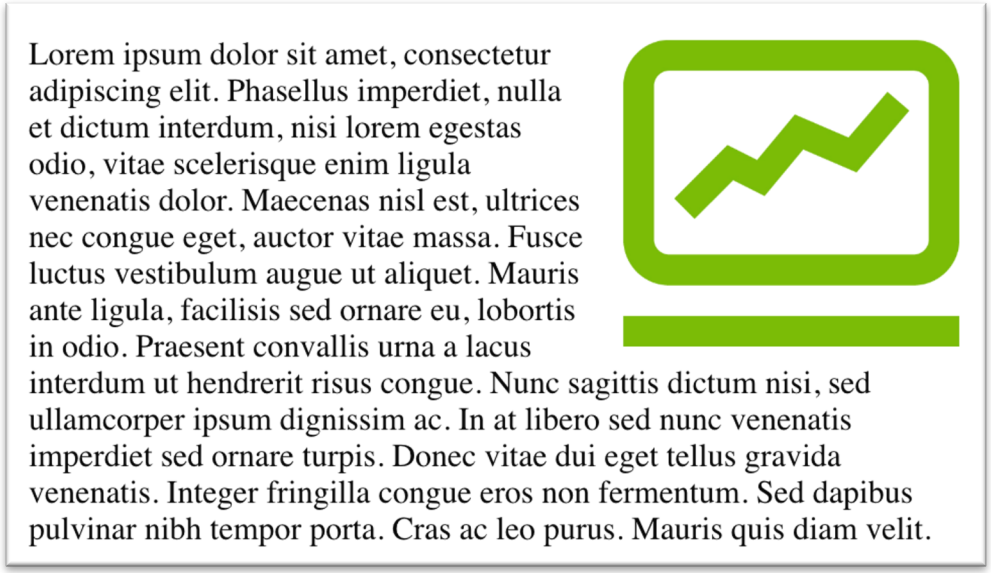


CSS Layout

`float` Beispiel

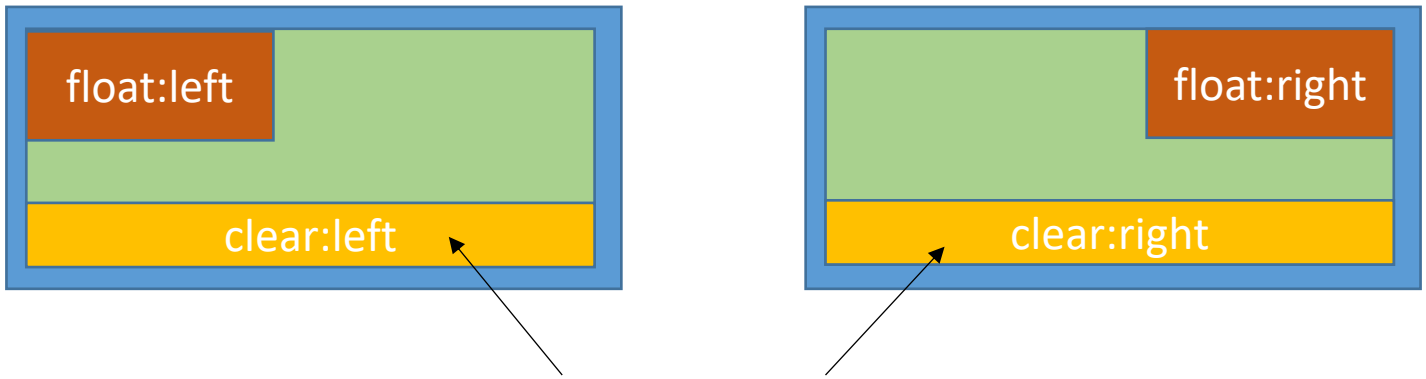
```
<body>
   <img src="../img/fiw.jpg" alt="fiw-logo" />
   <p>Lorem ipsum dolor sit amet, consectetur
            adipiscing elit. Phasellus imperdiet, nulla et
            dictum interdum, ... velit.</p>
</body>
```

```
<style>
   img {
      float:right;
      width:100px;
      margin-left:15px;
   }
</style>
```



CSS Layout
`clear`

- Mit der **`clear`**-Eigenschaft wird das Umfließen beendet
- Nach einer **`float`**-Eigenschaft sollte irgendwann eine (die passende) **`clear`**-Eigenschaft kommen (z.B. **`float:left`** wird durch **`clear:left`** beendet)
- **`clear`** kann folgende 5 Werte haben:
  - **`left`**, **`right`**, **`none`**, **`inherit`** – wie **`float`**
  - **`both`** – keine **`float`**-Elemente mehr (weder links noch rechts)



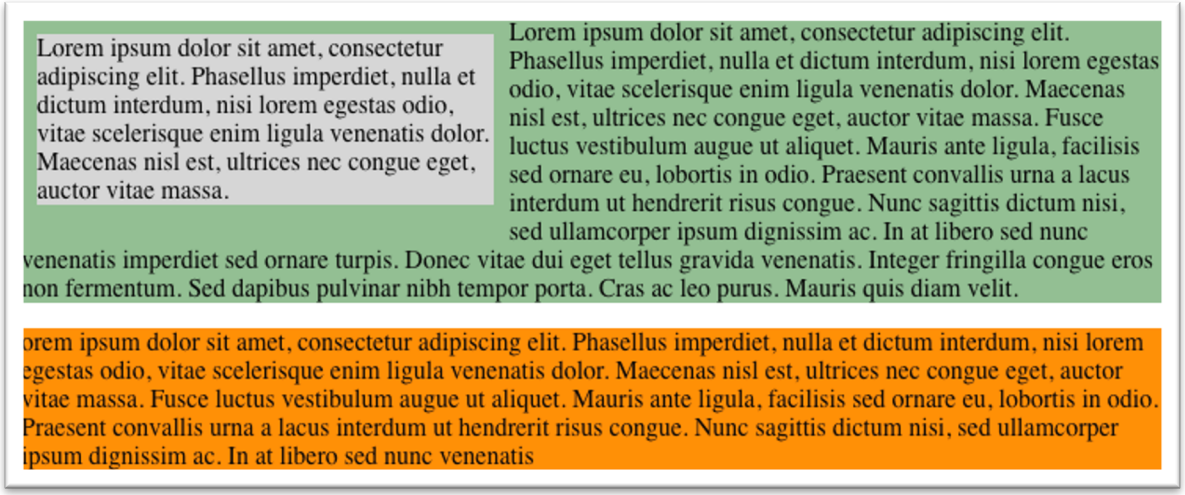
sehr häufig sieht man auch **`clear:both`** hier

CSS Layout

`clear` Beispiel

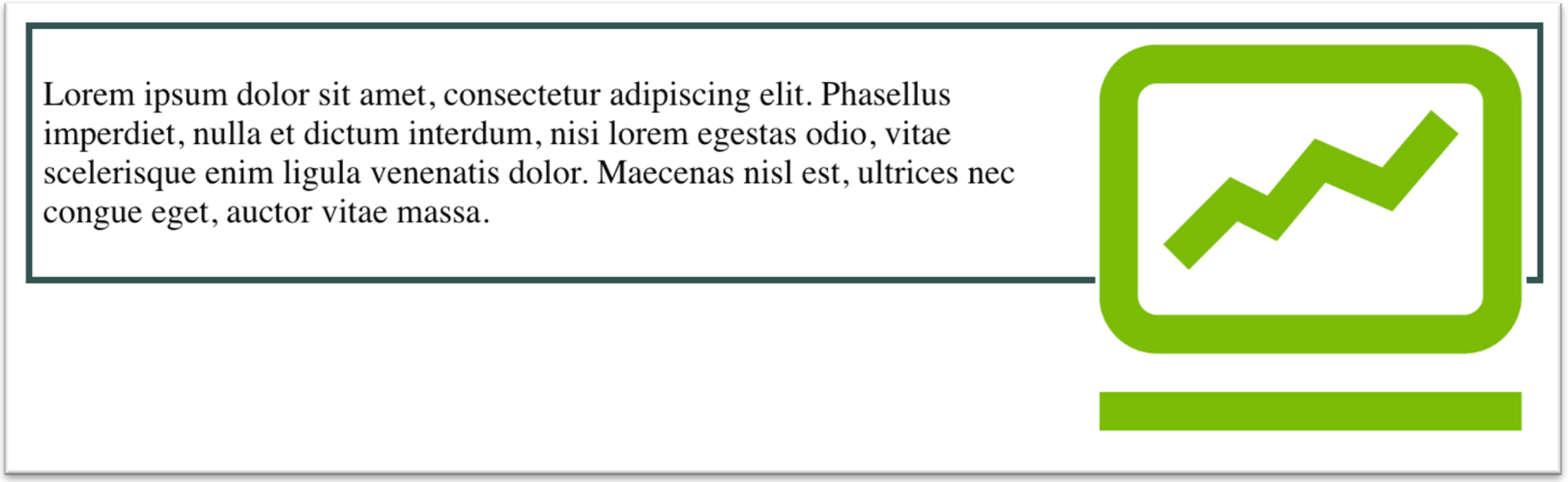
```
<div>
   <p style="background-color:lightgray;">
      Lorem ipsum dolor sit amet, ...</p>
   <p style="background-color:darkseagreen;">
      Lorem ipsum dolor sit amet, ...</p>
   <p style="background-color:darkorange;">
      orem ipsum dolor sit amet, ...</p>
</div>
```

```
div p:first-of-type {
   float:left;
   width : 40%;
   margin:10px;
}
div p:last-of-type {
   clear:left;
}
```



CSS Layout
`clearfix-Hack`

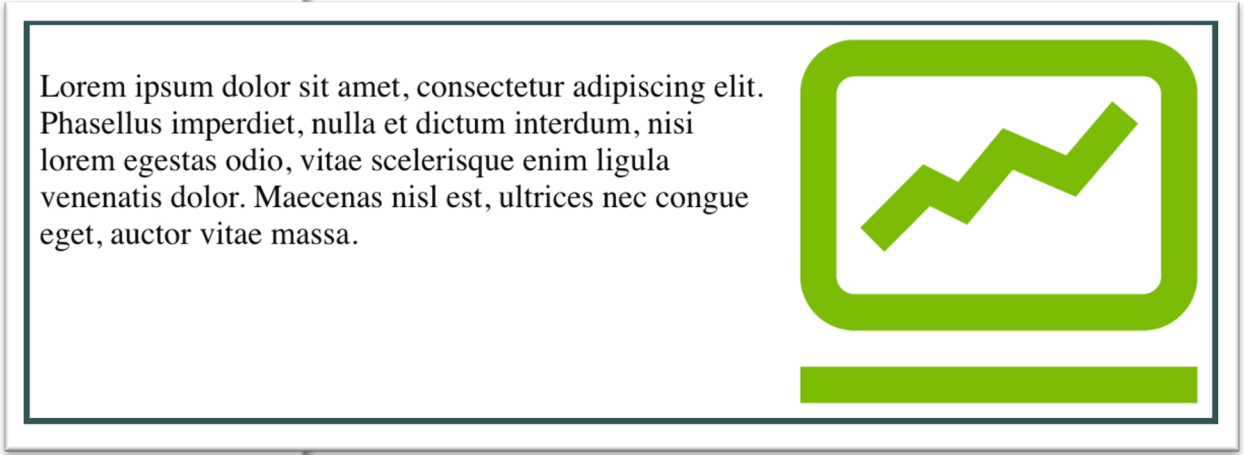
- Wenn das Element, das umflossen wird, größer ist, als das umfließende Element, entstehen unschöne Ansichten:



- Das Bild (`float:right;`) soll eigentlich vom Text umflossen werden, ist aber höher als der Text → sieht unschön aus
- Lösung: clearfix-Hack(nächste Folie)

CSS Layout
clearfix-Hack

```css
div {
    border: 3px solid darkslategray;
    padding: 5px;
}
.clearfix::after {
    content: "";
    clear: both;
    display: table;
}
img {
    float:right;
}
```




```html
<div class="clearfix">
<img src="../img/fiw.jpg" alt="fiw-logo" />
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum,  ... congue eget, auctor vitae massa. </p>
</div>
```

CSS Layout

`float` „verschachteln“ (mehrere nebeneinander)

- Das Umfließen kann „verschachtelt“ werden → Grundidee aller Grid-Konzepte (später)


```
.box {
   float: left;
   width: 25%;
   padding: 50px;
}
```

```
<div class="clearfix">
   <div class="box" style="background-color:#bbb">
      <p>Box 1</p>
   </div>
   <div class="box" style="background-color:#ccc">
      <p>Box 2</p>
   </div>
   <div class="box" style="background-color:#ddd">
      <p>Box 3</p>
   </div>
   <div class="box" style="background-color:#eee">
      <p>Box 4</p>
   </div>
</div>
```

CSS

Layout - Flexbox

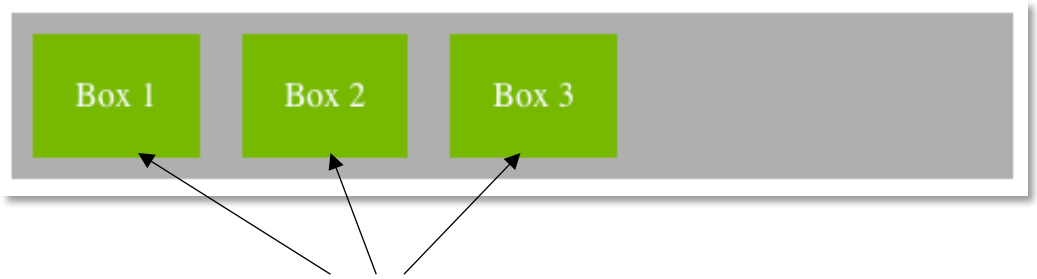
CSS Layout

Flexbox (`display:flex;`)

- Flexbox ist ein Layout-Modell, einfacher als der Umgang mit float
- Ein Container enthält „Boxen", die nebeneinander angeordnet werden → der Container bekommt die Eigenschaft `display:flex;`

```
.flex-container {
   display: flex;
   background-color: #AFAFAF;
}

.flex-container > div {
   background-color: #76B900;
   margin: 10px;
   padding: 20px;
   color:white;
}
```

```
<div class="flex-container">
   <div>Box 1</div>
   <div>Box 2</div>
   <div>Box 3</div>
</div>
```

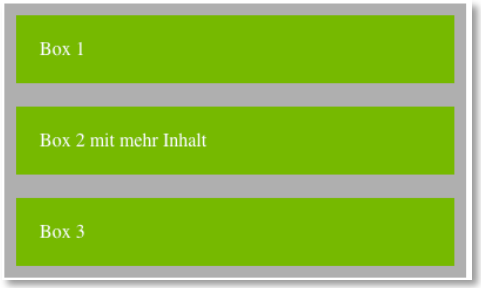


Breite der Boxen soviel wie nötig,
Boxen "hängen" links

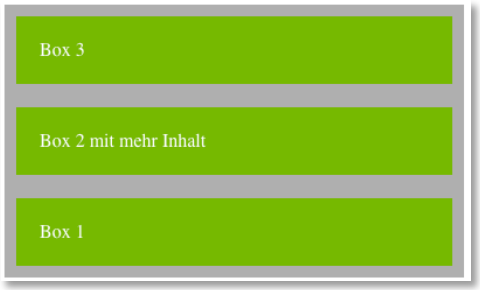
CSS Layout
Flexbox (**`flex-direction`**)

- Dem Flex-Container kann auch die Eigenschaft **`flex-direction`** zugeordnet werden, um die Richtung der Anordnungen der inneren Boxen zu definieren
- Standardwert ist **`row`** (siehe Folie zuvor)

flex-direction:**column**;



flex-direction:**column-reverse**;

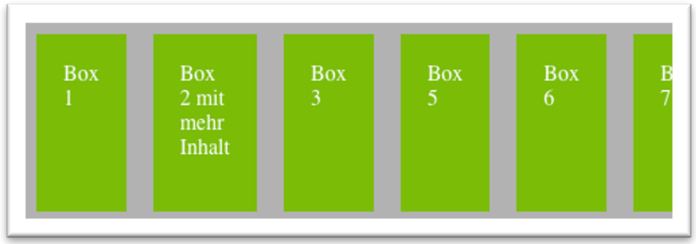


flex-direction:**row-reverse**;

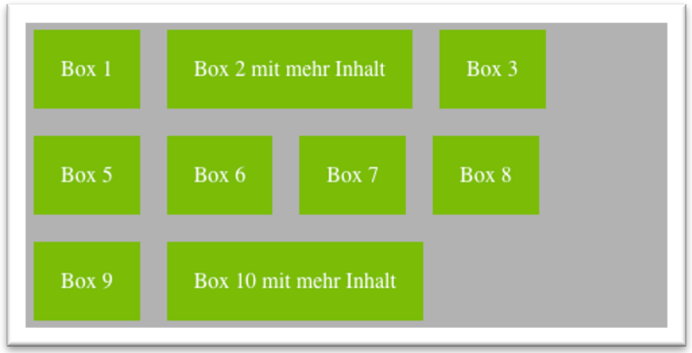
CSS Layout
Flexbox (**`flex-wrap`**)

- **`flex-wrap`** (Definition für den flex-container) gibt an, ob für die inneren Boxen ein Zeilenumbruch erlaubt ist oder nicht (**`nowrap`** ist Standard)

```
flex-wrap:nowrap;
```




```
flex-wrap:wrap;
```

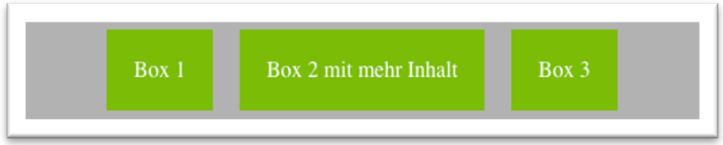


`flex-flow` kombiniert `flex-direction` und `flex-wrap`, z.B. `flex-flow: row wrap;`

CSS Layout
Flexbox (`justify-content`)

- **`justify-content`** ordnet die inneren Container an (**`flex-start`** ist Standard)

justify-content:**center**;



zentriert

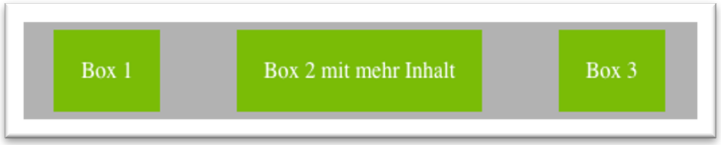
justify-content:**flex-start**;

linksbündig

justify-content:**flex-end**;

rechtsbündig

justify-content:**space-around**;



verteilt
Platz zwischen, davor und danach

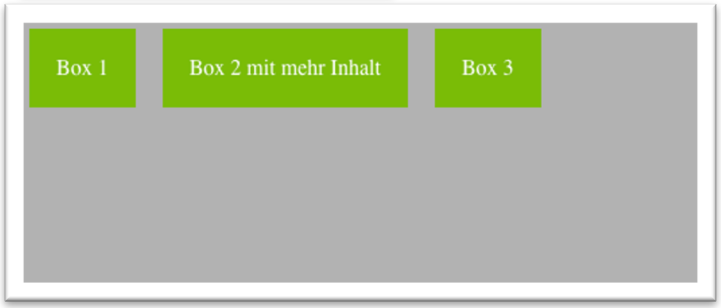
justify-content:**space-between**;

verteilt
Platz dazwischen

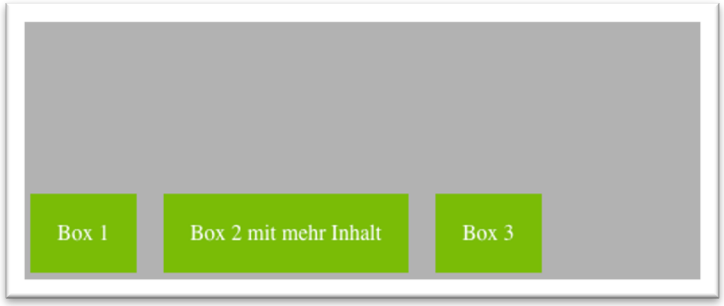
CSS Layout
Flexbox (**`align-items`**)

- **`align-items`** ordnet die inneren Container vertikal an (**`stretch`** ist Standard)

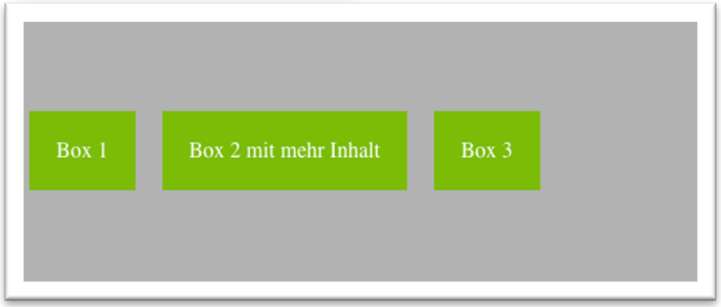
align-items:**flex-start**;



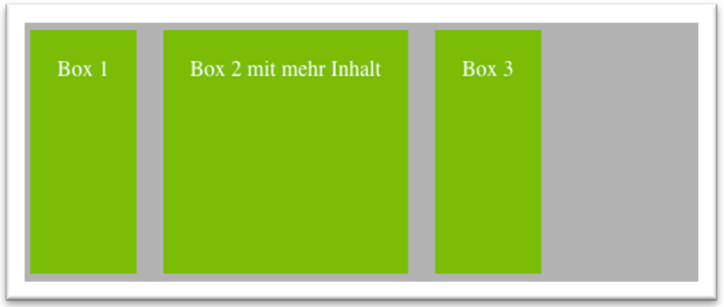
align-items:**flex-end**;



align-items:**center**;



align-items:**stretch**;

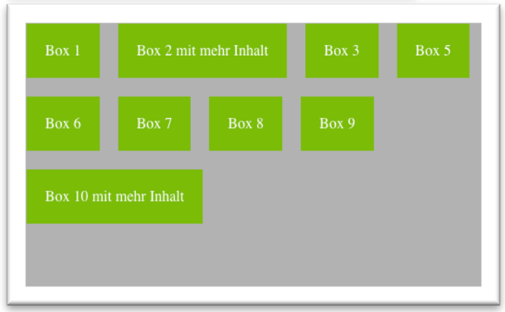


- Auch noch **`baseline:`** ordnet die Boxen an einer horizontalen Linie an → untere Linie aller Texte (Inhalte) in den Boxen

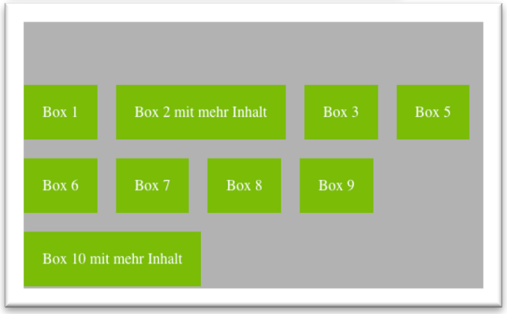
CSS Layout
Flexbox (`align-content`)

- **`align-content`** ordnet die inneren Container verteilt an (**`stretch`** ist Standard)

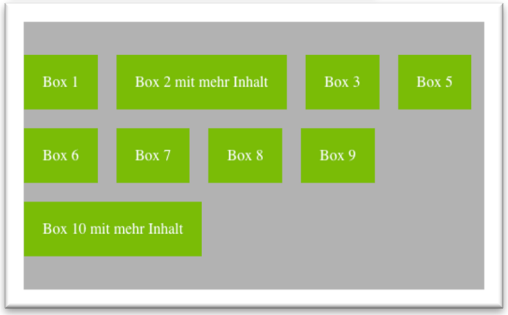
align-content:**flex-start**;



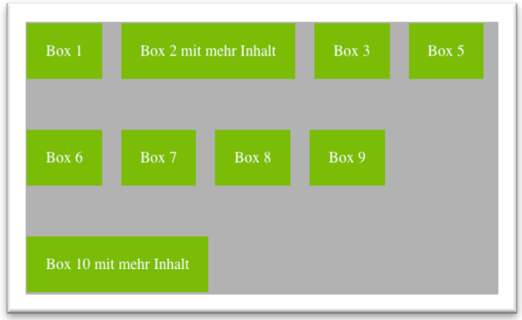
align-content:**flex-end**;



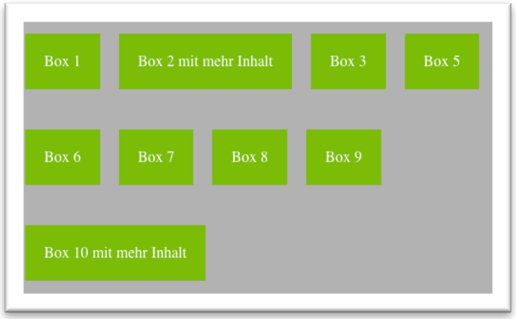
align-content:**center**;



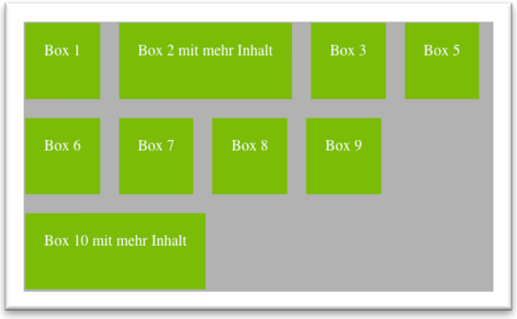
align-content:**space-between**;



align-content:**space-around**;



align-content:**stretch**;

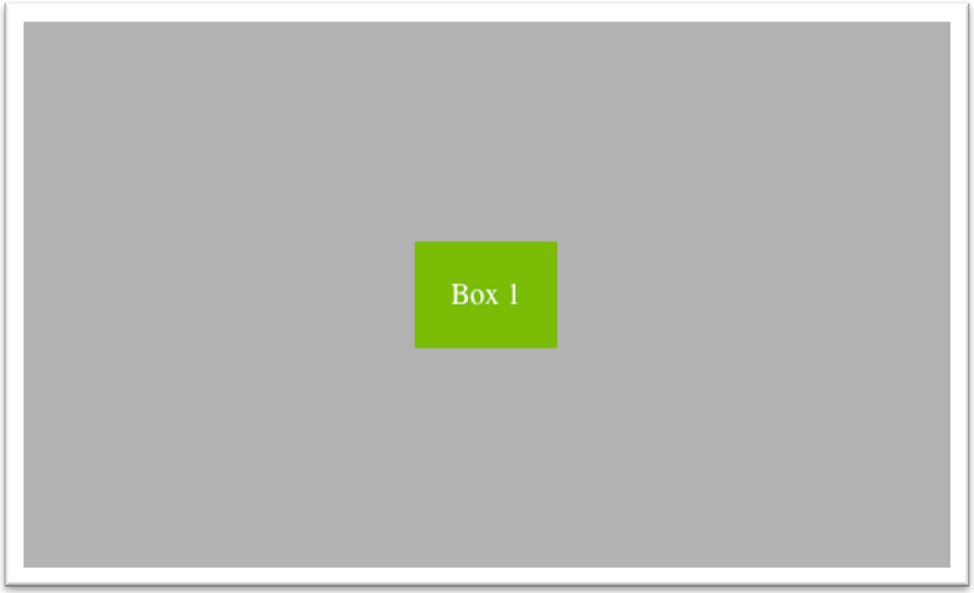


CSS Layout
Flexbox (perfektes Zentrieren)

- Häufig möchte man einen Inhalt (z.B. ein Bild) perfekt in ein Elternelement zentrieren (horizontal und vertikal)
- Das geht mit Flexbox sehr gut: einfach **`justify-content`** und **`align-items`** für das Elternelement (den flex-container) jeweils auf **`center`** setzen

```
.flex-container {
   display: flex;
   height:300px;
   background-color: #AFAFAF;
   justify-content:center;
   align-items:center;
}
```

```
<div class="flex-container">
   <div>Box 1</div>
</div>
```



CSS Layout
Flexbox (Eigenschaften für die inneren Boxen)

- Die bisherigen Eigenschaften wurden alle für das Elternelement (den flex-container) definiert
- Für die inneren Boxen gibt es ebenfalls Eigenschaften für das Layout:
  - **`order`** definiert eine Reihenfolge des Erscheinens der inneren Boxen im Elternelement (anders als die Reihenfolge im HTML-Code)
  - **`flex-grow`** gibt für jede Box eine Breite in Bezug zu den anderen Boxen an, z.B. 1:2:4
  - **`flex-shrink`** gibt an, wie stark die Breite einer Box zusammengestaucht werden darf
  - **`flex-basis`** definiert die Breite einer Box (falls releativ, dann zum flex-container)
  - **`flex`** oberen drei zusammen (flex-grow, flex-shrink, flex-basis)
  - **`align-self`** überschreibt für eine Box die align-items-Definition des flex-containers

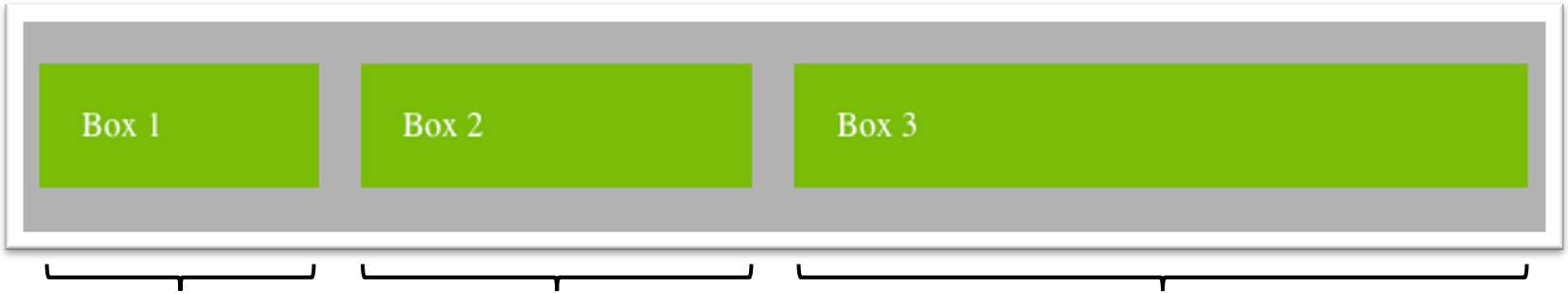
- Am wichtigsten/nützlichsten: **`flex-grow`**

CSS Layout

Flexbox (Eigenschaften für die inneren Boxen: **`flex-grow`**)

```
<div class="flex-container">
  <div style="flex-grow:1;">Box 1</div>
  <div style="flex-grow:2;">Box 2</div>
  <div style="flex-grow:5;">Box 3</div>
</div>
```

Anders als später bei den Grids, bei denen die Spaltensumme 12 sein muss



- *relative Breite 1*
- *relative Breite 2*
- *also doppelt so breit wie Box 1*
- *relative Breite 5*
- *also 5x so breit wie Box 1 und 2,5x so breit wie Box 2*

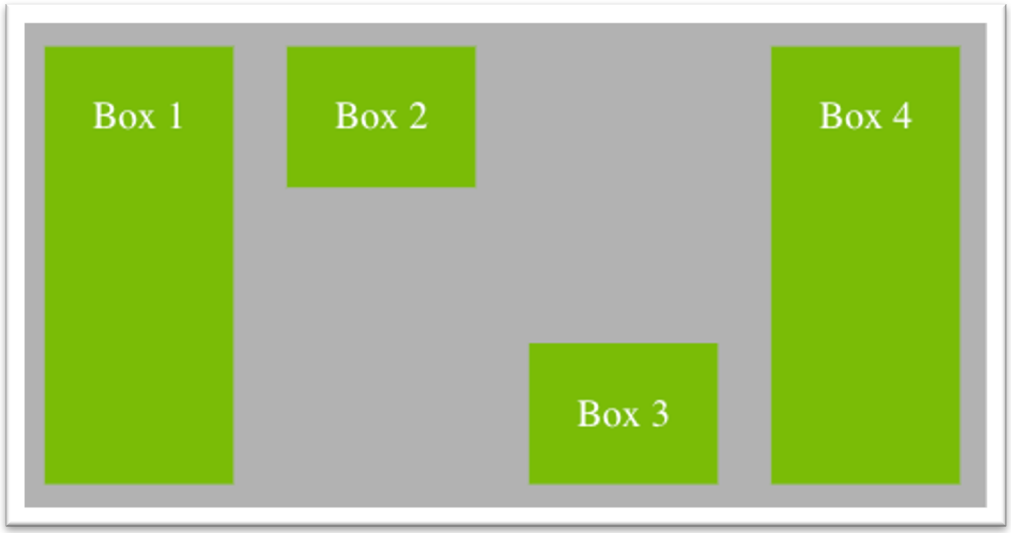
- *innere Boxen werden „eingepasst“ →* ***`justify-content`*** *des Elternelementes egal (wie* ***`spacce-around`****)*

CSS Layout

Flexbox (Eigenschaften für die inneren Boxen: **`align-self`**)

```
<div class="flex-container">
<div>Box 1</div>
<div style="align-self:flex-start;">Box 2</div>
<div style="align-self:flex-end;">Box 3</div>
<div>Box 4</div>
</div>
```

Box 1 und Box 4 haben Standard:
`align-items:stretch;`



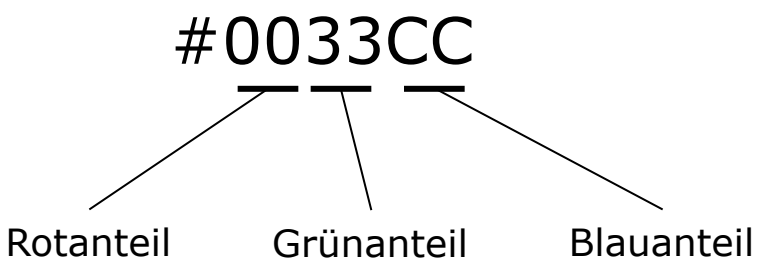
CSS

Sonstiges

Wertangaben in CSS

- CSS-Formatdefinitionen bestehen aus CSS-Eigenschaften, denen Sie Werte zuweisen
- unterschiedliche Arten von Werten:
  - feste Werte: z.B. left, right, center, justify (z.B. bei text-align)
  - Zeichenketten: z.B. Schriftartennamen (bei font-family)
  - numerische Werte: z.B. Angabe von Höhe von Breite, Abständen, Schriftgrößen
  - Farbangaben: hexadezimal oder Zeichenkette
  - Angaben in einem bestimmten Format: z.B. URL

- z.B. Farbwert:



Wertangaben in CSS
numerische Werte

Abkürzung	Angabetyp	Bedeutung
%	relativ	relativ entweder zur elementeigenen Größe, zur Größe des Elternelements oder zum allegemeinen Kontext
cm	absolut	Zentimenter
em	relativ	Schriftgröße des Elements (bei font-size aber Schriftgröße des Elternelements)
ex	relativ	Höhe des Kleinbuchstabens x im Element (bei font-size aber die Schriftgröße von x im Elternelement)
in	abolut	inch (1 inch = 2,54 cm)
mm	absolut	Milimeter
pc	absolut	pica (1 pc = 12 Punkt)
pt	absolut	Punkt (1 Punkt = 1/72 inch)
px	absolut relativ	absolut auf ein und dasselbe Ausgabegerät bezogen relativ von Ausgabegerät zu Ausgabegerät (Pixeldichte)

CSS

custom properties

- Eigene Variablen für die Verwendung definieren *(custom properties)*:
  - Variablen: müssen mit -- beginnen, z.B. **`--variablenname`**
  - In einem Scope, z.B. überall: **`*, body, :root`**
  - Zugriff mittels **`var(--variablenname)`** → alles ohne Leerzeichen

```
* {
   --htw-gruen: #76B900;
   --htw-grau:#AFAFAF;
   --htw-orange:#FF5F00;
   --htw-blau:#0082D1;
   --htw-font:Verdana, Geneva, sans-serif;
}
body {
   font-family:var(--htw-font);
}
```

```
li a:hover {
   background-color: var(--htw-orange);
}

.active {
   background-color: var(--htw-gruen);
}
```

CSS Layout

Beispiel custom properties (Variablen)

```
{
   --htw-gruen: #76B900;
   --htw-grau:#AFAFAF;
   --htw-orange:#FF5F00;
   --htw-blau:#0082D1;
   --htw-font:Verdana, Geneva, sans-serif;
}
body {
   font-family:var(--htw-font);
}
ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: var(--htw-grau);
}
li {
   float: left;
}
```

```
li a {
   display: inline-block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}

li a:hover {
   background-color: var(--htw-orange);
}

.active {
   background-color: var(--htw-gruen);
}
```

```
<ul>
  <li><a href="#A" class="active">Alle</a></li>
  <li><a href="#A">A-G</a></li>
  <li><a href="#H">H-L</a></li>
  <li><a href="#M">M-R</a></li>
  <li><a href="#M">S-Z</a></li>
</ul>
```

Positionieren mit CSS

`position`

- Früher (vor Flexbox- und Grid-Zeiten) spielte das Positionieren von Elementen noch eine größere Rolle → CSS-Eigenschaft **`position`**
- Mögliche Werte:
  - **`static`** Element wird in den normalen Ablauf positioniert (Standard)
    die Eigenschaften left, right, bottom, top haben keine Wirkung
  - **`relative`** wie static, aber mit left, right, bottom, top kann das Element relativ zu seiner Position im Eleternelement verschoben werden
  - **`fixed`** das Element kann fest im Viewport positioniert werden; bleibt immer dort; mit left, right, bottom, top positioniert
  - **`absolute`** ähnlich zu fixed, aber positioniert relativ zum Vorgängerelement (nicht zum Viewport wie bei fixed) und scrollt auch mit (fixed nicht)
  - **`sticky`** Mischung aus relative und fixed; scrollt normal mit, verlässt aber den Viewport nicht

- wir nutzen **`fixed`** und **`sticky`**, um Navigationsleisten oder Tabellenüberschriften zu fixieren und **`absolute`** und **`relative`** für Dropdown.Menüs

Positionieren mit CSS

`position:fixed`

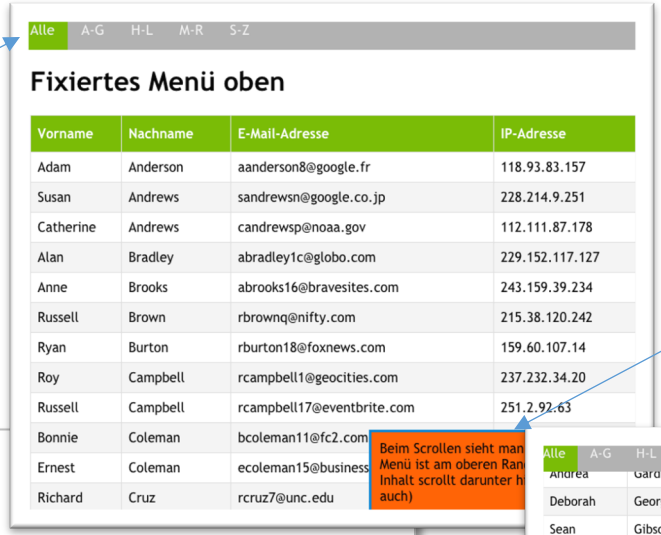
siehe ../Layout/fixed.html

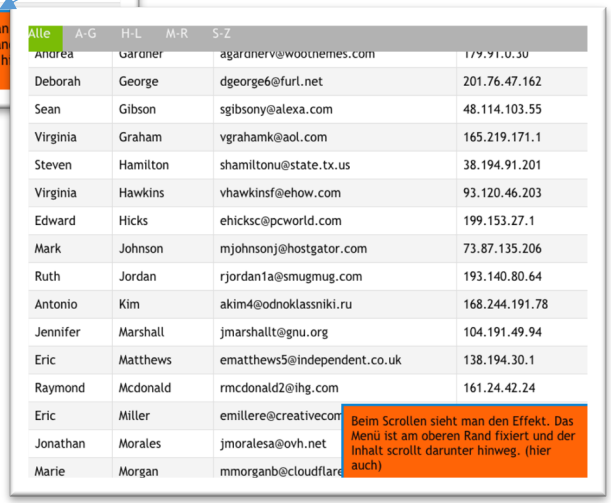
- Sehr häufig verwendet, um das Navigationsmenü im <header> permanent im Viewport zu lassen

```
header {
   position: fixed;
   top: 0;
   width: 100%;
}
```

```
main p {
   position:fixed;
   bottom:0px;
   right:0px;
}
```

```
<header>
   <ul>
      <li><a href="#A" class="active">Alle</a></li>
      <li><a href="#A">A-G</a></li>
      <li><a href="#H">H-L</a></li>
      <li><a href="#M">M-R</a></li>
      <li><a href="#M">S-Z</a></li>
   </ul>
</header>
```





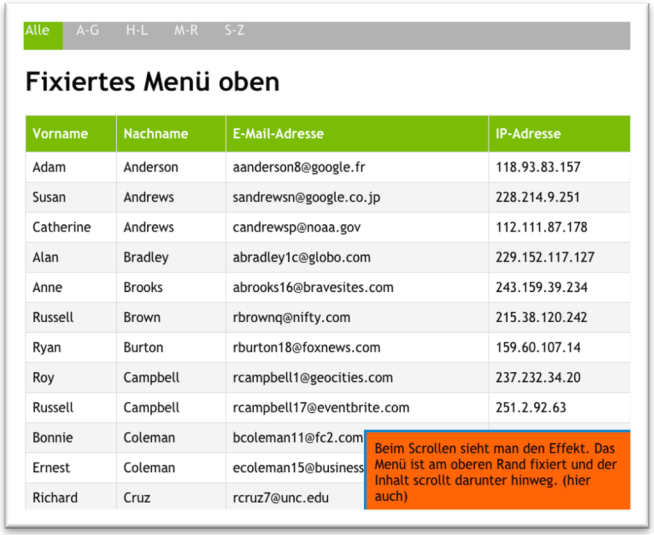
Positionieren mit CSS
`position:sticky`

siehe ../Layout/fixed.html

- Sehr häufig verwendet, um den Tabellenkopf einer Tabelle im Viewport zu lassen

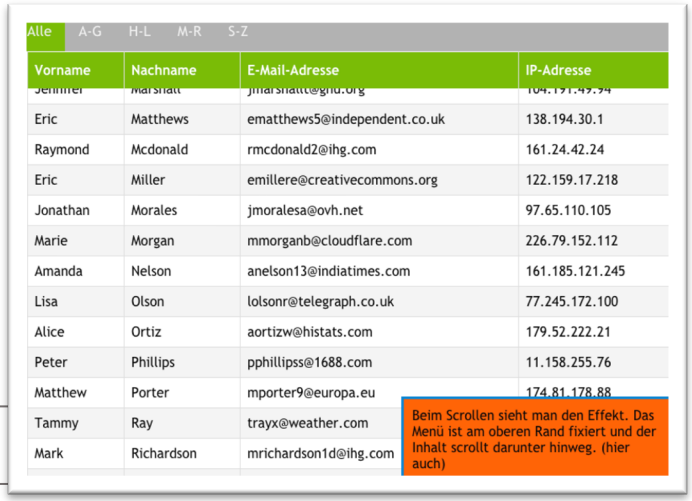
```
table th {
   position : sticky;
   top:48px;
}
```

wenn `top:0px;`, dann würde Navigationsmenü überdeckt




```
<table>
 <thead>
  <tr>
     <th>Vorname</th>
     <th>Nachname</th>
     <th>E-Mail-Adresse</th>
     <th>IP-Adresse</th>
  </tr>
 </thead>
 <tbody>
  <tr>
     ...
  </tr>
 </tbody>
</table>
```

<thead> und <tbody> HTML-Elemente in <table>; erleichtern das Ansprechen der Elemente



Positionieren mit CSS

siehe ../Layout/dropdown.html

`position:absolute` und `position:relative`

- Für Dropdown-Menüs

```css
.dropdown {
    position: relative;
    display: inline-block;
}
```

Inline-block ist eine Inline-Element, aber man kann `width` und `height` definieren

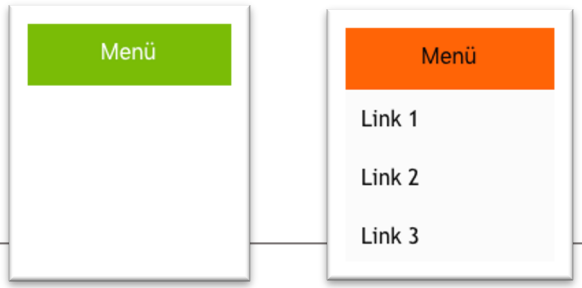
```css
.dropdown-content {
    display: none;
    position: absolute;
    z-index: 1;
}
```

`z-index` definiert das Über- oder Unterlappen von Anzeigen

```css
.dropdown:hover .dropdown-content {
    display: block;
}
```

Links erscheinen bei `:hover`

```html
<div class="dropdown">
  <button class="dropbtn">Menü</button>
  <div class="dropdown-content">
    <a href="#">Link 1</a>
    <a href="#">Link 2</a>
    <a href="#">Link 3</a>
  </div>
</div>
```



Inhalt hat nicht genug Platz
`overflow`

- Mit der **`overflow`**-Eigenschaft wird definiert, ob Inhalt abgeschnitten, trotzdem dargestellt oder mit Scrollbars verfügbar wird, falls er nicht in „sein Element passt“
- 4 mögliche Werte:
  - **`visible`** Inhalt wird auch über das Element hinaus dargestellt (Standard)
  - **`hidden`** Inhalt wird „abgeschnitten“ und ist nicht sichtbar
  - **`scroll`** Inhalt wird abgeschnitten, aber ein Scrollbar erscheint → Inhalt wird durch scrollen sichtbar
  - **`auto`** falls der Inhalt nicht passt, wird ein Scrollbar hinzugefügt, ansonsten nicht

- Achtung!
  - die **`overflow`**-Eigenschaft wirkt nur auf Blockelemente mit definierter Höhe
  - beim Mac erscheint der Scrollbar immer nur bei Bedarf (auch bei **`overflow:scroll`**;)

Inhalt hat nicht genug Platz

siehe ../Layout/overflow.html

`overflow`

```
<style>
   div {
      width: 200px;
      height: 50px;
      background-color: lightgray;
      border: 1px dotted black;
   }
   .contentvisible {
      overflow: visible;
   }
   .contenthidden {
      overflow: hidden;
   }
   .contentscroll {
      overflow: scroll;
   }
   .contentauto {
      overflow: auto;
   }
</style>
```

```
<div class="contentvisible">
   The overflow ...
</div>

<div class="contenthidden">
   The overflow ...
</div>

<div class="contentscroll">
   The overflow ...
</div>

<div class="contentauto">
   The overflow ...
</div>
```