

4. Gestaltung von Web-Seiten (CSS)

Prof. Dr. Jürgen Schneider

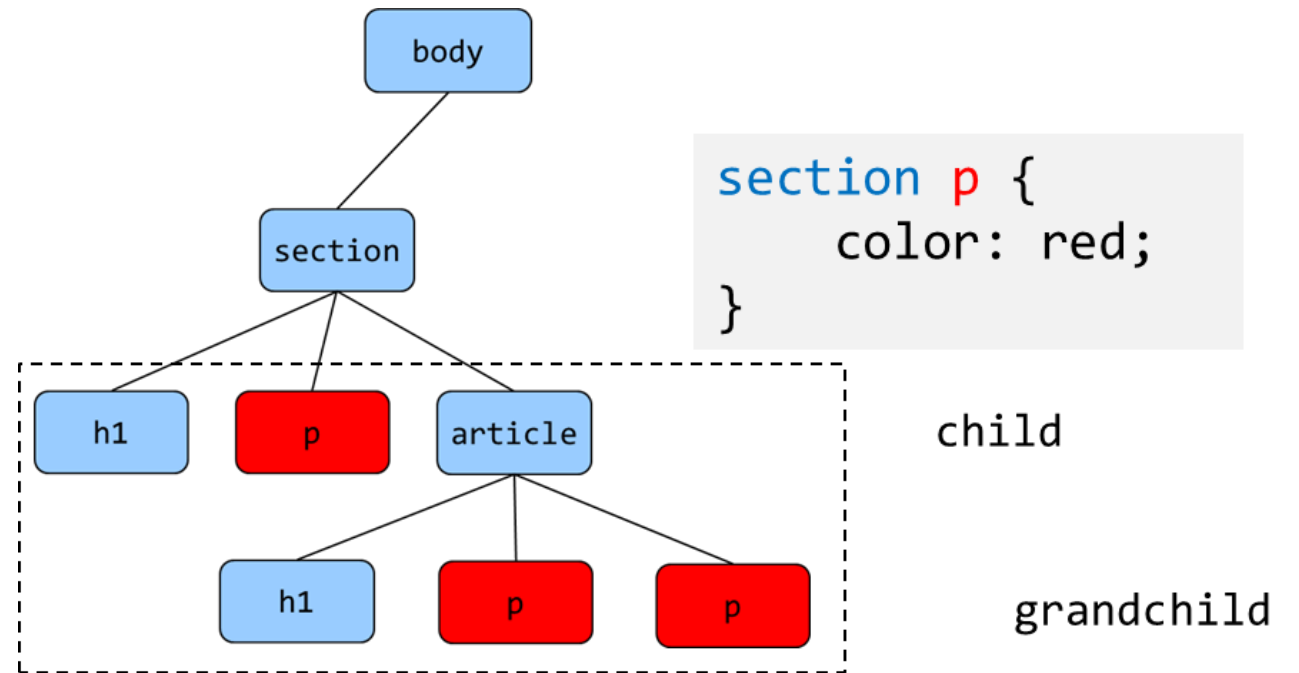
Lernziele:

- ❑ CSS-Selektoren und CSS-Eigenschaften zur gezielten Gestaltung von HTML-Elemente einsetzen können.
- ❑ Specifity and Cascading of CSS-Rules.
- ❑ Inheritance of CSS-Properties.
- ❑ Flexibles Webseitenlayouts mit Flexbox, Grid oder Float.
- ❑ Media Queries und Responsive Design

Überblick:

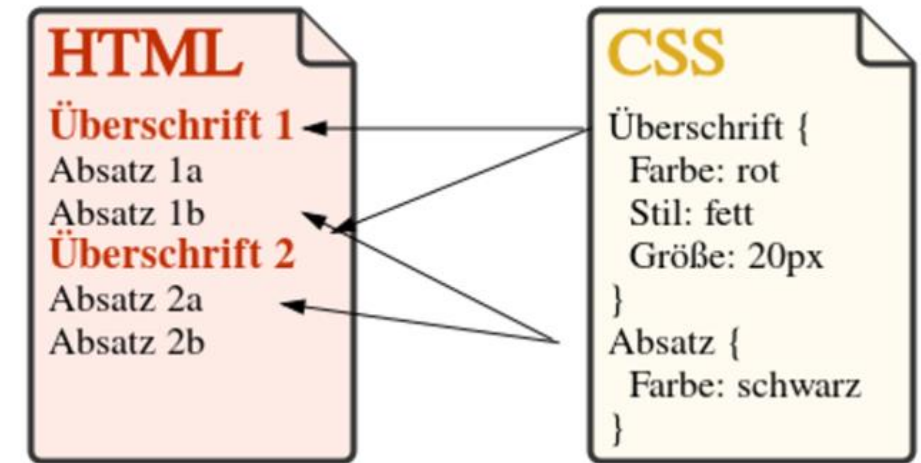
- 4.1 CSS – Selektoren
- 4.2 CSS – Eigenschaften
- 4.3 Gestaltung von Webseiten – Layouts

4.1 CSS-Selektoren



Trennung von Struktur und Gestaltung

- Gemäß dem Grundprinzip der "Trennung der Zuständigkeiten" (siehe Kapitel 1), trennt man die **inhaltliche Struktur** einer **Webseite** von ihrem optischen Aussehen (Design).
- **Prinzipielles Vorgehen:**
 - **Zuerst** entwickelt man die Webseite in HTML:
"Am Anfang ohne Gestalt"
 - HTML strukturiert die Webseiten und liefert eine auf den Inhalt fokussierte **Struktur** ihrer Webseite.
 - **Anschließend** können Sie mittels **Cascading Style Sheets (CSS)** dem Inhalt ein Aussehen verleihen, das den **Benutzer** in der **Bedienung** der Webseite **unterstützt**.
 - Das CSS-Regelwerkes definiert **Schriftarten**, **Farben** und die **Anordnung** der **Elemente** (Layout) auf der Webseite.



Beispiel: Dem erstellten HTML-Dokument wird mittels CSS die folgende Darstellung zugeordnet:

Überschrift <h1> - rote Farbe, Fettdruck, Größe 20 Pixel

Absatz <p> - Text in Absatz hat die Farbe schwarz

CSS-Regeln

- ❑ Eine **CSS-Regel** besteht aus einem
 - **CSS-Selektor**: Auswahl eines oder mehrerer HTML-Elemente z.B. ein h1-, ein p- oder div-Tag für das die Regel gelten soll.
 - **CSS-Deklarationen**: Besteht aus einer Eigenschaft wie z.B. der Schriftfarbe **color** und einem Wert wie z.B. **red**, die dem selektierten Element zugewiesen wird.
- ❑ Beispiel (siehe Bild):
 - CSS-Selektor: "**Element-Selektor**"
alle <p>-Elemente also alle Absätze im Dokument
 - CSS-Deklarationen:
 - Schriftart ist Helvetica
 - Schriftgröße ist 1.4 × Standardschriftgröße
 - Schriftfarbe ist rot
 - Hintergrund des Absatzes ist gelb

```
selector { Eigenschaft: Wert;  
          Eigenschaft: Wert;  
          Eigenschaft: Wert;  
        }
```

Beispiel:

```
p {  
  font-family: Helvetica;  
  font-size: 1.4em;  
  color: red;  
  background-color: yellow;  
}
```

1em : Standardschriftgröße des Browsers

CSS-Regeln einbinden im Dokument

- ❑ Mittels dem HTML `<style>`-Element können die CSS-Regeln direkt in ein HTML-Dokument eingebettet werden. Sie gelten dann **nur** innerhalb **dieses HTML-Dokumentes**.
 - Das `<style>`-Element muss im `<head>`-Element einer HTML-Datei hinzugefügt werden.
- ❑ Schließlich kann Style-Information auch **direkt pro HTML-Element** definiert werden.
 - Man verwendet dazu das **style-Attribut**.

`<h2 style="font-style:normal; color:red">`

Bilder in HTML

`</h2>`

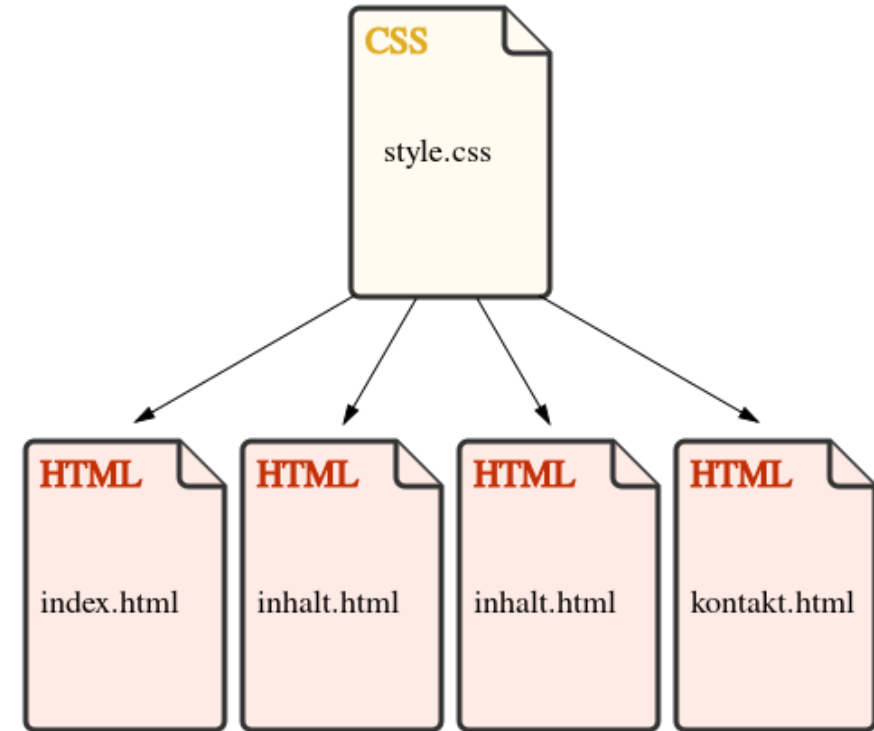
```
<head>
  <meta charset="utf-8"/>
  <title>Web-Tutorial</title>
  <style>
    h2 {
      font-size: 30px;
      font-style: oblique;
      color: coral;
    }
    body {
      background-color: beige;
    }
  </style>
</head>
```

CSS einbinden mittels externer Datei

- ❑ CSS-Regeln **sollten** in **externen Style-Dateien (*.css)** beschrieben werden. Neben der Trennung von Inhalt und Gestalt, ermöglicht dieser Ansatz **einheitliche Formatvorgaben** für eine Vielzahl von Webseiten zu spezifizieren.
 - Stichwort: **Corporate Design (CD)**
- ❑ Die Einbindung im HTML-Dokument erfolgt dann im **<head>**-Element mittels des **<link>**-Elementes

```
<head>
  ...
  <link rel="stylesheet" href="css/style.css">
</head>
```

- **rel-Attribut** definiert den logischen Beziehungstyp (hier: „stylesheet“) zwischen dem HTML-Dokument und der externen Datei.
- **href-Attribut** definiert den Pfad zur externen Datei (hier: Datei „style.css“ im Unterordner „css“)



- ❑ **Best Practise:**
 - ❑ Einsatz von **css-Dateien** zur Gestaltung eines HTML-Dokumentes.
 - ❑ Speichern der **CSS-Dateien** im **Unterordner css**.

ID-Selektor

- ❑ Neben dem schon diskutierten Element-Selektor (= Element Name) gibt es noch 2 weitere elementare Selektoren.
- ❑ **ID-Selector:**
 - Möchte man ein **einzelnes Element** (z.B.: einen einzelnen Absatz) in einem HTML-Dokument besonders gestalten, verwendet man das **id-Attribut** eines Elementes.
 - Das Selektieren des id-Attributs erfolgt dann mittels vorangestelltem **#-Zeichen** (siehe Beispiel).
 - **Konvention:** id - **Attributwerte** werden komplett **klein geschrieben**.
 - Im Beispiel wird die Schriftgröße auf **30px (CSS-Pixel)** und die Schriftfarbe als **blaue** Farbe festgelegt.

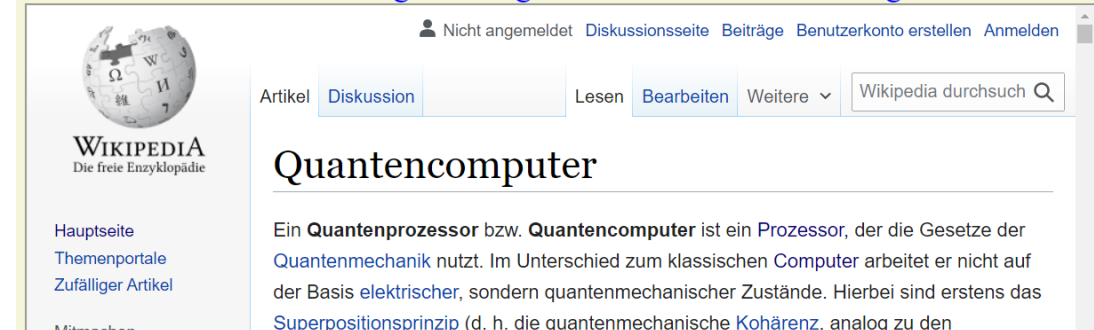
Style

```
#p_kap1{  
    font-size: 30px;  
    color: blue  
}
```

Struktur

```
<p id="p_kap1">  
    Zusätzliche Information zeigt der folgende  
    Artikel als iframe eingebunden!  
</p>
```

Zusätzliche Information zeigt der folgende Artikel als iframe eingebunden!.



Class-Selektor

- ❑ **Class-Selektor:** Um **mehrere Elemente** mit denselben Eigenschaften deklarieren zu können, verwendet man das **class-Attribut**.
 - Einem Element können **mehrere Klassen** zugewiesen werden, die mittels **Leerzeichen** getrennt werden
`<p class="wichtig geheim">`
 - Das Selektieren des class-Attributs erfolgt dann mittels vorangestelltem **Punkt**. (siehe Beispiel).
 - Im **Beispiel** wird ein Paragraph- und ein Figure-Element mit demselben Attribut "wichtig" gekennzeichnet.
 - Dem class-Attribut wichtig wird dann die Hintergrundfarbe „LightSteelBlue“ zugewiesen.
 - Auch **Klassen-Attributwerte** sollten immer **klein geschrieben** werden.

```
.wichtig{  
    background-color: LightSteelBlue;  
}
```

Style

```
<p class="wichtig geheim"> Bilder, Video und  
Audio,...  
</p>
```

```
<figure class="wichtig"> ... </figure>
```

Struktur

Bilder, Video und Audio, sogenannte **Multi-Medien Inhalte** werden über den Pfad zu ihrem Ordner und ihren Dateinamen in die Webseite gelinkt. Wenn das Bild nicht geladen werden kann wird alternativ der Wert des Attribut **alt** angezeigt. Mittels dem **loading** Attribut kann die Ladegeschwindigkeit beeinflusst werden. $3 \times 4 = 12$

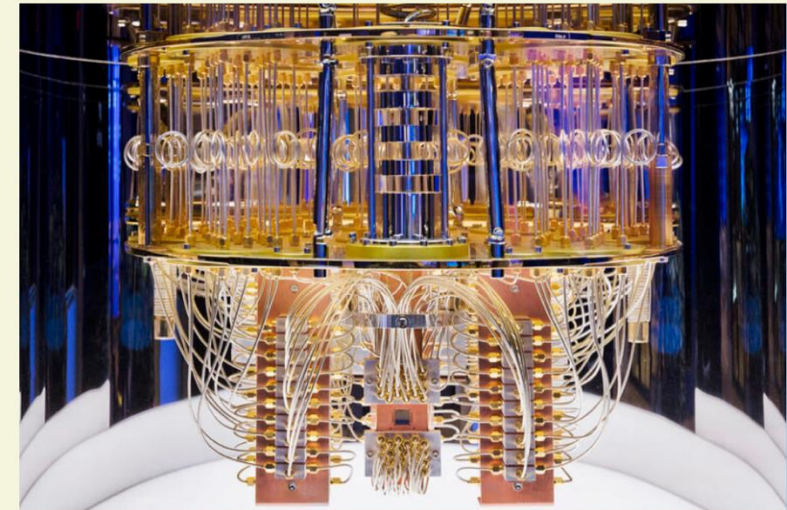


Abb. 1: Ein Quantencomputer der Firma IBM

Element- und Class-Selektor

- Durch die **Kombination** eines **Element-** mit einem **Class-Selektor**, können Eigenschaften im Stylesheet auf einzelne Elementtypen eingeschränkt werden.
- Dazu wird der **Element-Selektor ohne Leerzeichen** mit dem **class-Selektor** kombiniert

p.wichtig

Bilder, Video und Audio, sogenannte **Multi-Medien Inhalte** werden über den Pfad zu ihrem Ordner und ihren Dateinamen in die Webseite gelinkt. Wenn das Bild nicht geladen werden kann wird alternativ der Wert des Attribut **alt** angezeigt. Mittels dem **loading** Attribut kann die Ladegeschwindigkeit beeinflusst werden. $3 \times 4 = 12$

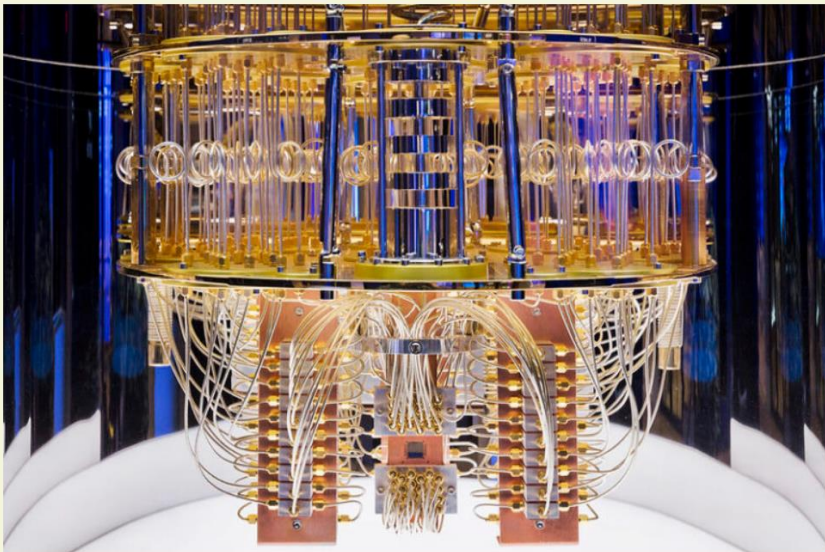


Abb. 1: Ein Quantencomputer der Firma IBM

- Im Beispiel wird **allen <p>-Elementen** mit **class-Attribut = "wichtig"** **zusätzlich** eine rote Schrift zugeordnet.

Style

```
.wichtig{
    background-color: LightSteelBlue;
}
p.wichtig{
    color:red
}
```

```
<p class="wichtig"> Bilder, Video und Audio,...
</p>
```

```
<figure class="wichtig"> ... </figure>
```

Struktur

Gruppierung von Selektoren

- Sollen mehrere Elemente dieselben Eigenschaften erhalten, kann man die **einzelnen Regeln** zu einer Regel **zusammenfassen**.

Man bezeichnet dies auch als **gruppieren**.

- Die **Selektoren** werden in der Gruppierungsregel per **Komma** ", " getrennt: **h1, h2, p**
- Die CSS-Eigenschaft **text-align** legt die **horizontale Ausrichtung** des Inhalts innerhalb eines Blockelements oder einer Tabellenzellenbox fest.
- **Beispiel:** Inhalt von <h1>, <h2> und <p>
 - Text wird zentriert
 - Farbe des Textes ist rot

```
h1 {  
  text-align: center;  
  color: red;  
}
```

```
h2 {  
  text-align: center;  
  color: red;  
}
```

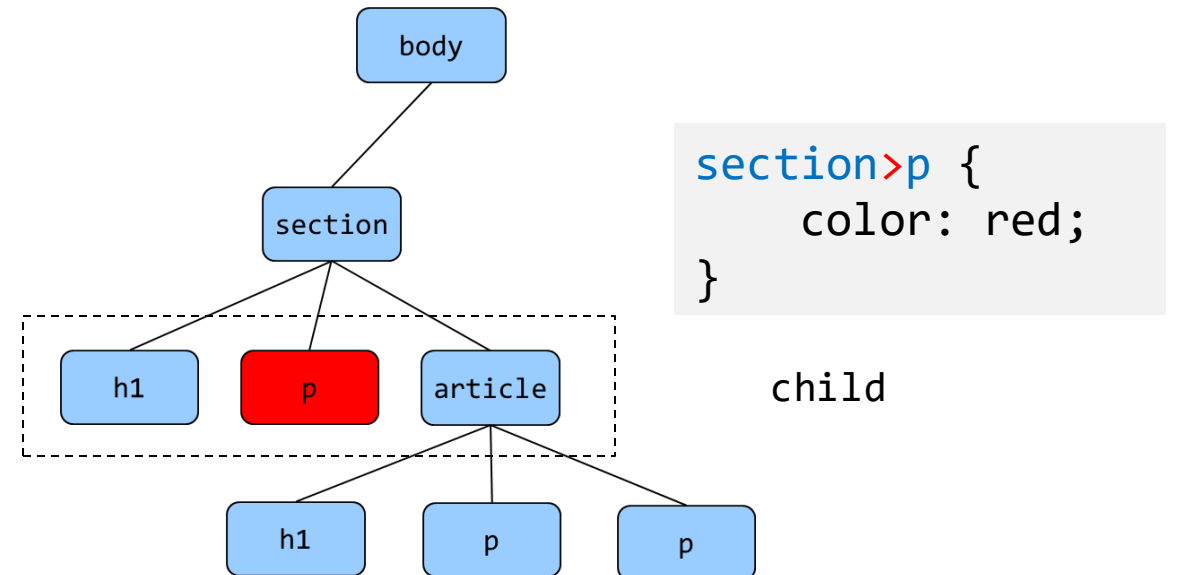
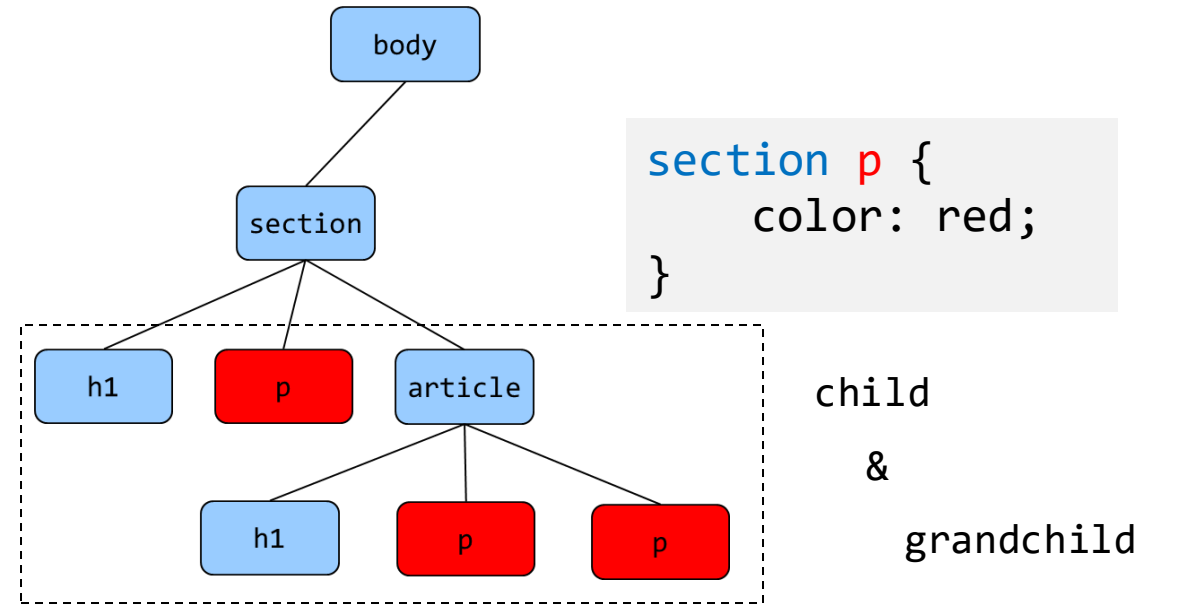
```
p {  
  text-align: center;  
  color: red;  
}
```

"gruppieren"

```
h1, h2, p {  
  text-align: center;  
  color: red;  
}
```

CSS Kontext-Selektoren

- ❑ Mittels der **Baumstruktur** eines **HTML-Dokumentes** lassen sich weitere Selektoren definieren, die die Beziehung der Elemente in der Baumstruktur beschreiben.
- ❑ **Nachfahren-Selektor "space"**
`section p { ... }`
trennen zweier Elemente mit einem **Leerzeichen**, bedeutet „alle Nachfahren von“. Im Beispiel werden alle `<p>`-Nachfahren von `<section>` selektiert.
- ❑ **Direkte Nachfahren-Selektor ">"**
`section>p { ... }`
wird auf alle p-Elemente angewendet, die direkte Nachfahren (=Child) von article sind.



CSS Kontext-Selektoren

- Beispiel "Alle Geschwister": Selektor „`~`“ :

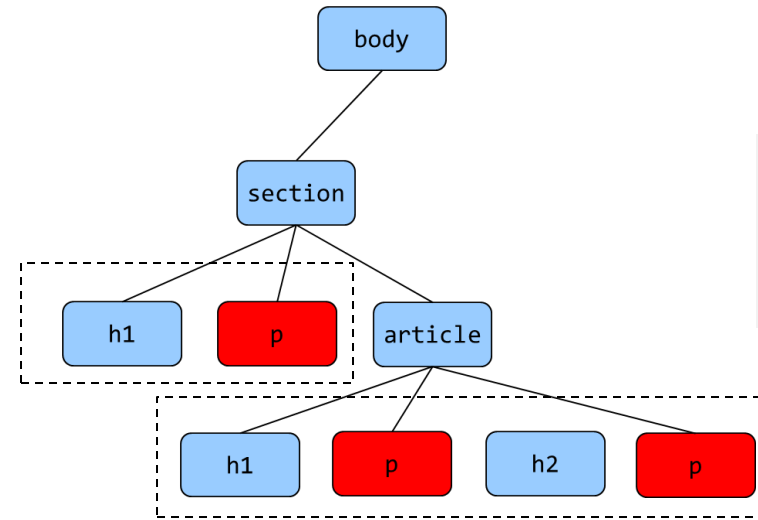
```
h1~p {...}
```

wird auf alle p-Elemente angewendet , die sich auf derselben Ebene (Geschwister) wie das h1-Element befinden.

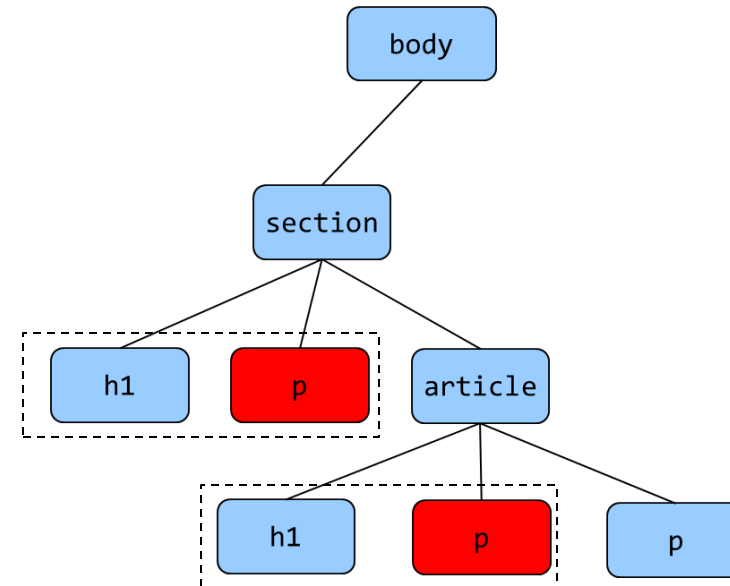
- Beispiel Benachbarte Geschwister: Selektor „`+`“ :

```
h1+p {...}
```

wird auf ein p-Element angewendet , das dem h1-Element direkt folgt. `h1` und `p` sind Nachbarn und haben dabei immer dasselbe Eltern-Element.



```
h1~p {  
  color: red;  
}
```



```
h1+p {  
  color: red;  
}
```

Attribut-Selektoren

- ❑ **CSS-Attribut-Selektoren** ermöglichen die Auswahl von Elementen anhand ihrer Attribute.
- ❑ Attribut-Selektoren werden in eckigen Klammern [attr] angegeben.
- ❑ Mögliche Vergleichsoperatoren: Alle Elemente deren
 - [attr] : attr gesetzt ist
 - [attr="wert"]: attr exakt den Wert wert besitzt
 - [attr~="wert"]: attr das Wort wert enthält (Leerzeichen davor und dahinter)
 - [attr^="wert"]: attr-Wert mit wert beginnt
 - [attr\$="wert"]: attr-Wert mit wert endet
 - [attr*="wert"]: attr-Wert die Zeichenfolge wert enthält
 - ...

❑ Beispiele:

- ❑ Alle p-Elemente deren lang Attribut gesetzt ist egal auf welchen Wert

```
p[lang] {  
    font-weight: bold;  
}
```

- ❑ Alle Eingabefelder vom Typ Password

```
input[type="password"] {  
    background-color: red;  
}
```

- ❑ Alle internen Links (=alle Links die mit #-beginnen)

```
a[href^="#"] {  
    background-color: gold;  
}
```

Pseudoklassen-Selektoren

- ❑ Bei **Pseudoklassen** handelt es sich um einfache **Selektoren**, die ein Element dann ansprechen, wenn es einen **bestimmten Zustand** besitzt.
- ❑ Ein gutes Beispiel für Pseudoklassen ist das Anchor-Element. Das Anchor-Element kann sich in folgenden Zuständen befinden:
 - **a:link** Hyperlink wurde noch nie angeklickt
 - **a:visited** Hyperlink auf eine bereits besuchte Seite
 - **a:hover** Hyperlink beim Hovern (Maus schwebt über dem Link)
 - **a:active** Hyperlink, der gerade anklickt wird
- ❑ Diese Selektoren können jetzt verwendet werden, um eine moderne Navigationsleiste zu gestalten

vorher:

DHBW Nachrichten

[Aktuelles](#) | [Studierende](#) | [Intern](#) | [Lehre](#)

```
a:link {
    text-decoration: none;      /* Unterstrich entfernen */
    border-style: solid;        /* Umrandung */
    background-color: #BDB76B; /* Füllung mit Farbe */
    /*          top | right | bottom | left          */
    border-color: yellow olive olive yellow;
    color: #8B008B;             /* Text Lila */
    text-align: center;         /* Menütext zentrieren */
    text-transform: uppercase; /* Großbuchstaben */
}
```

nachher:

DHBW Nachrichten

AKTUELLES

STUDIERENDE

INTERN

LEHRE

Pseudoklassen-Selektoren

- Für das `<input type="text" ...>`-Element kann man die Pseudoklasse `:focus` verwenden.
 - `:focus` - Klickt man in ein Eingabefeld fokussiert man dieses und kann so seine Darstellung für eine Eingabe durch den Benutzer optimieren.
 - Im Beispiel wird der **Rahmen** des Eingabefeldes verbreitert auf **2px** und **rot** gezeichnet. Das Eingabefeld erhält einen **schwarzen Hintergrund** bei **weißer Schriftfarbe**.

```
<style>
  /* :focus für das Input-Feld */
  input[type="text"]:focus {
    outline-color: red;
    outline-width: 2px;
    color: whitesmoke;
    background-color: black;
  }
</style>
```

```
<h2>Beispiel für :focus</h2>
<input type="text" placeholder="Geben Sie ihren
Namen ein ..." size="40" />
```

Beispiel für :focus

Geben Sie ihren Namen ein ...

Beispiel für :focus

Alice

Pseudoklassen-Selektoren

- ❑ Für das `<input type="checkbox" ...>`-Element kann man die Pseudoklasse `:checked` verwenden.
- `:checked` – Wählt man eine Checkbox aus, kann die Auswahl hervorgehoben werden, um so den Benutzer für seine Eingabe zu sensibilisieren.
- Im Beispiel wird nach der Auswahl der Checkbox, der Text im Label-Element grün eingefärbt und mit Fettdruck gezeichnet.

```
<style>
    input[type="checkbox"]:checked + label {
        color: green;
        font-weight: bold;
    }
</style>
-----
<h2>Beispiel für :checked</h2>
<input type="checkbox" id="check" name="checkbox"
value="Zitronen"/>
<label for="check">Täglicher Newsletter </label>
```

Beispiel für :checked

☐ Täglicher Newsletter ...

Beispiel für :checked

☒ Täglicher Newsletter ...

Pseudoelemente-Selektoren

- Ein **CSS-Pseudoelement** wird verwendet, um bestimmte **Teile** eines **Elements** zu gestalten.
- Beispielsweise kann der **erste Buchstabe** oder die **erste Zeile** eines Elements-Inhaltes in bestimmter Weise **formatiert** werden oder am **Anfang** oder **Ende** eines Elementes **Inhalt eingefügt werden**.

::first-line	erste Zeile des formatierten Textes
::first-letter	erstes Zeichen des formatierten Textes
::before	fügt Inhalte an den Anfang eines Elementes hinzu. Die Inhalte werden in der content-Property definiert.
::after	fügt Inhalte an das Ende eines Elementes hinzu. Die Inhalte werden in der content-Property definiert.
::selection	Der markierte Text eines Elementes wird entsprechend formatiert.
...	...

- Beispiel:

```
p.intro::first-letter {  
    color: #ff0000;  
    font-size: 200%;  
}
```

```
<p class="intro">Formatierung des 1. Buchstabens  
eines <p>-Elementes mit class-Attribut  
&quot;intro&quot;.</p>
```

Formatierung des 1. Buchstabens eines <p>-Elementes mit class-Attribut "intro".

Pseudoelemente-Selektoren

❑ Beispiel: `::before`

```
<style>
  ul>li, ul>li::before {
    content: "😊";
    list-style: none;
    margin-bottom: 1em;
  }
</style>
```

```
<ul>
  <li>Überschriften: &lt; h1 &gt;</li>
  <li>Paragraphen: &lt; p &gt; </li>
  <li>Tabellen: &lt; table &gt; </li>
</ul>
```

😊 Überschriften: `< h1 >`



😊 Paragraphen: `< p >`

😊 Tabellen: `< table >`

Pseudoelemente-Selektoren

- Nebstehendes Beispiel zeigt den Pseudoelement-Selektor **::selection**.
- Dieser kann dazu verwendet werden, mit der Maus markierten Text besonders hervorzuheben.
- Im Beispiel wird der Hintergrund von markiertem Text **gelb** eingefärbt.

Pseudoelement ::selection

 Markiere diesen Text, um den Effekt von ::selection zu sehen. 

```
<style>
  /* Fügt ein Symbol vor der Überschrift ein */
  p::before {
    content: "⚾ ";
    font-size: 1.2em;
  }
  /* Fügt ein Symbol nach dem Absatz ein */
  p::after {
    content: "🍔";
    font-size: 1.2em;
  }
  /* Markierter Text wird gelb */
  ::selection {
    background: yellow;
    color: black;
  }
</style>
```

```
<h3>Pseudoelement ::selection</h3>
```

```
<p>Markiere diesen Text, um den Effekt von ::selection zu sehen.</p>
```

CSS Specificity

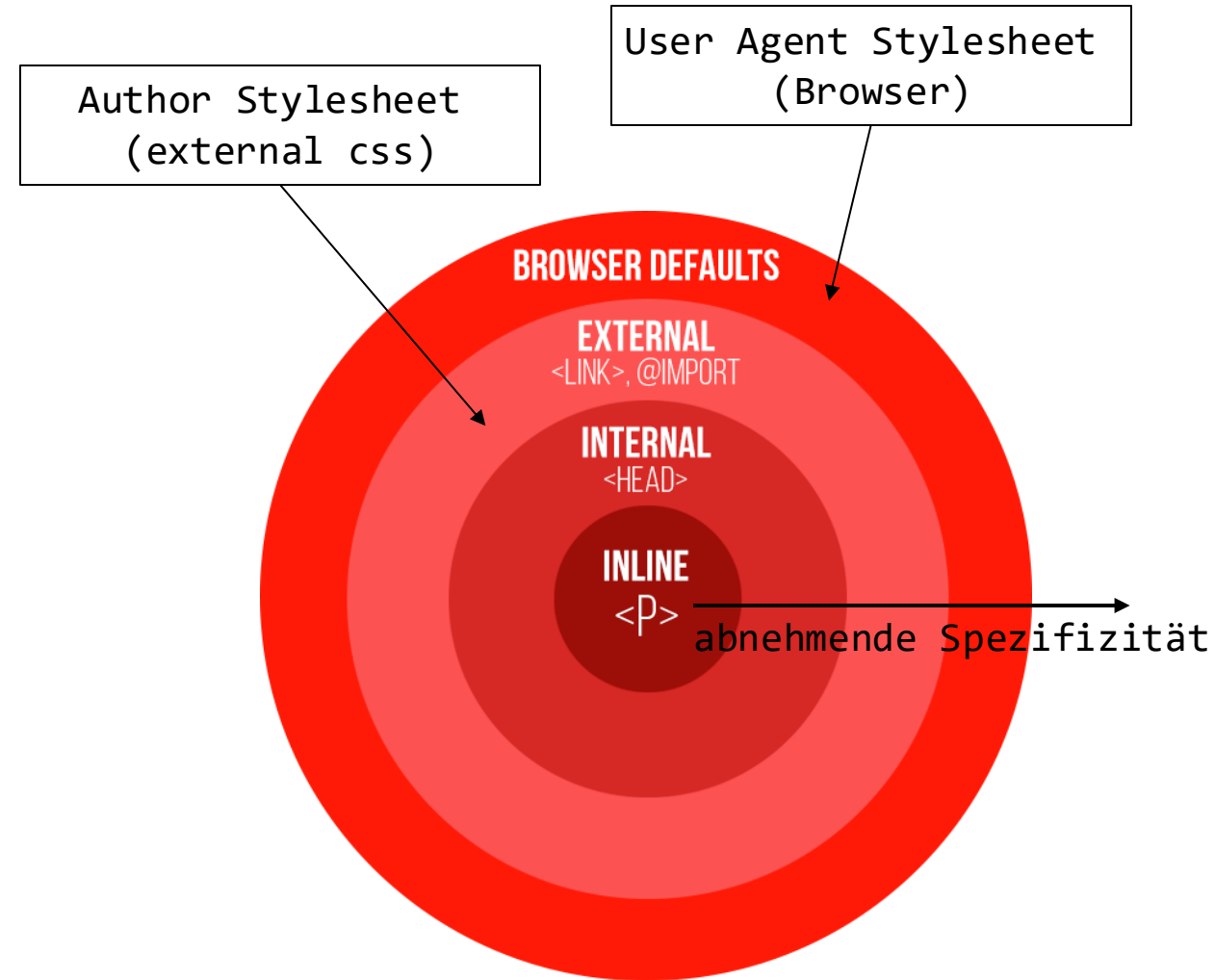
Specificity: Wenn es **zwei oder mehr** widersprüchliche **CSS-Regeln** gibt, die auf **dasselbe Element** verweisen, versucht der Browser zu bestimmen, welche Regel am **spezifischsten** ist. Die **spezifischste Regel** gewinnt.

- ❑ Bei **gleicher Spezifität** zählt die **zuletzt vom CSS-Parser interpretierte Regel**, also die **zuletzt hinzugefügte Regel** im **css-Sheet**.
- ❑ **Spezifitätshierarchie:** Je spezifischer ein Selektor, umso höher seine Spezifitätshierarchie.
 - Jeder Selektor hat einen Platz in der Spezifitätshierarchie.
 - Es gibt **vier Kategorien**, die das Spezifitätsniveau eines Selektors definieren:

- (1) **Inline-Stile** – Ein Inline-Stil wird direkt an das zu gestaltendes Element angehängt.
Beispiel: `<h1 style="color: #ffffff;">`
- (2) **IDs** – Eine ID ist eine eindeutige Kennung für die Seitenelemente, wie z. B. `#navbar`.
- (3) **Klassen, Pseudoklassen und Attribute** – Diese Kategorie umfasst **.classes**, **[Attribute]** und **Pseudoklassen** wie **:hover**, **:focus** usw.
- (4) **Elemente und Pseudoelemente** – Diese Kategorie umfasst **Elemente** und **Pseudoelemente** wie **h1**, **div**, **::first-line**, **::after**, **::before**
- (5) Der **universelle Selektor (*)** hat die geringste Spezifität

Cascading Order

- ❑ Die **CSS-Specifity** verwendet zusätzlich das Prinzip der **Cascading Order** ("kaskadierende Reihenfolge").
- ❑ Je „näher“ eine **Stileigenschaft** sich am **Element** befindet, desto höher ist deren Vorrang.
 - Inline-Stile überschreiben interne <style> - Anweisungen
 - Interne <style> Anweisungen im <head>-Bereich überschreiben Anweisungen in externen CSS-Dateien
 - Externe CSS-Dateien überschreiben Browser-Standard Einstellungen.
 - Browser Default-Einstellungen haben die niedrigste Spezifität
- ❑ **Best Practise:**
 - Nur Einsatz von **externen Style-Sheets**.
 - Style-Sheet mit **allgemeinen Designregeln** zuerst in HTML einbinden, dann Style-Sheets mit konkreten seitenspezifischen Design-Regeln.



CSS Kommentare

- ❑ **Kommentare** in CSS haben die gleiche Syntax wie Blockkommentare in C oder Java mittels `/* ... */`

```
/* Überschrift h1 soll in Farbe chocolate  
dargestellt werden. */
```

```
h1 {  
    font-size: 40px;  
    font-style: normal;  
    color: chocolate;  
}
```

```
/*  
Dateiname: style.css  
Version: 1.0  
Author: JS  
Description: CSS-Stylesheet zu Kapitel4 der  
Vorlesung Web-Engineering  
*/  
h1 {  
    font-size: 40px;  
    font-style: normal;  
    color: chocolate;  
}  
h2 {  
    font-size: 30px;  
    font-style: oblique;  
    color: coral;  
}  
body {  
    font-family: Arial, "Times New Roman";  
    font-size: 1em;  
    background-color: beige;  
    line-height: 1.5;  
}
```

Integration von CSS in XML

- ❑ Auch XML-Dokumente können Sie mit CSS-Sheets gestalten.
- ❑ Der Browser hat im Gegensatz zu HTML keine XML-Default-Settings, da die verwendeten XML-Elemente individuell vom Entwickler angelegt werden.
- ❑ Das Einbinden von CSS erfolgt immer im **Prolog** der XML-Datei

```
<?xml version="1.0" encoding="UTF-8"
      standalone="no"?>
<?xml-stylesheet type="text/css"
      href="css/snowdon.css" ?>
<!DOCTYPE document SYSTEM "Snowdon.dtd"
[
<!ENTITY snow "Edward Snowdon">
]>
```

Snowdon.dtd

```
...
<!ELEMENT bold (#PCDATA)>
<!ELEMENT italic (#PCDATA)>
<!ELEMENT br EMPTY>
```

Snowdon.css

```
/* \A : newline character in CSS
   pre : preserve whitespace and newline as in HTML
*/
br::before {
    content: '\A';
    white-space: pre;
}
italic { font-style: italic; }
bold { font-weight: bold; }
```

@-Rules

- ❑ **At-Regeln** sind **CSS-Regeln**, die CSS anweisen, wie es sich verhalten soll. Sie beginnen mit einem **at-Zeichen** „@“, gefolgt von einem **Schlüsselwort** und umfassen alles bis zum **nächsten Semikolon** ";"

```
@[KEYWORD] RULE;
```

oder dem nächsten CSS Deklaration Block.

```
@[KEYWORD] RULE {Declaration Block};
```

- ❑ Die Rule kann aus einem **einfachen String**, einer **URL** oder einer sogenannten **Media-Query** bestehen.

- ❑ Beispiele:

- ❑ Die CSS-at-Regel **@charset** gibt die im Stylesheet verwendete Zeichenkodierung an.

- Es **muss das erste Element** im Stylesheet sein und es darf ihm keine Zeichen vorangestellt werden:

```
@charset "utf-8";
```


@-Rules

- ❑ Mit der **Import-Regel @import** können Sie ein **Stylesheet** in ein anderes Stylesheet **importieren**.
- ❑ Die **@import-Regel** muss am Anfang des Dokuments stehen (jedoch nach der @charset-Deklaration).
- ❑ Die **@import-Regel** unterstützt auch **Medienabfragen** (screen and max-width: 767px).
 - Im Beispiel wird eine CSS-Datei für große (min-width: 767px) und eine CSS für **kleine Bildschirme** (max-width: 767px) importiert

```
@charset "UTF-8";  
/* css für großen Bildschirm */  
@import url("screen_big.css") screen and (min-  
width: 768px);  
/* css für großen Bildschirm */  
@import url("screen_small.css") screen and (max-  
width: 767px);
```

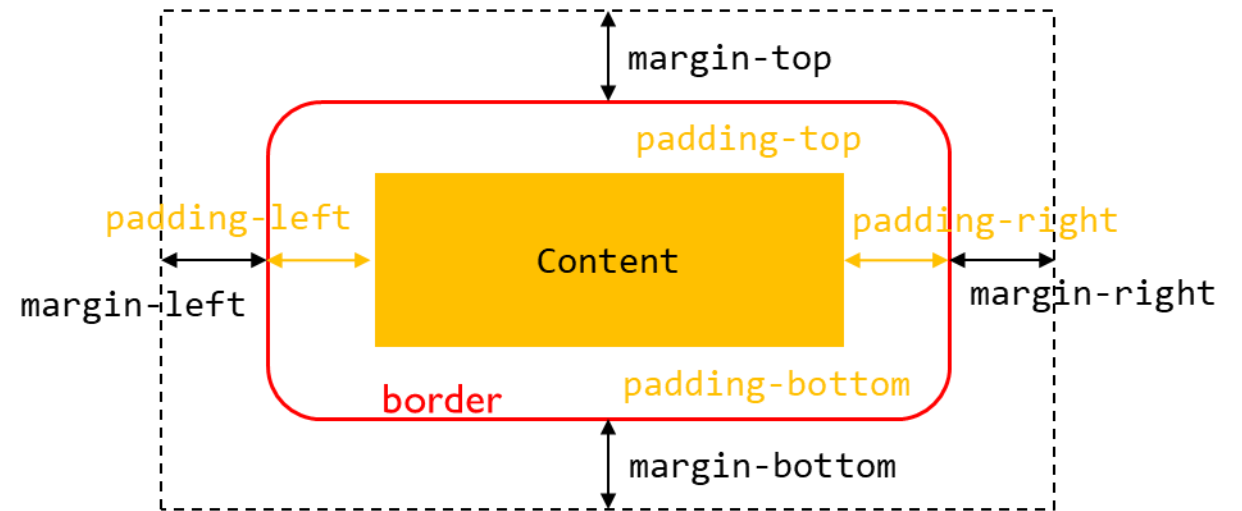
CSS Universalselektor und Normalisierung

- ❑ Browser besitzen für die einzelnen **HTML-Elemente** vordefinierte CSS-Einstellungen
https://www.w3schools.com/cssref/css_default_values.asp
- ❑ Möchte man die Stileigenschaft aller Elementen einer Webseite **browserunabhängig** setzen, erfolgt dies über den **Universalselektor** `*`.
- ❑ Beispielsweise besitzen die Abstände **margin** (Rand) und **padding** (Füllung) im Box-Modell unterschiedliche Default-Werte für verschiedene Browser.
- ❑ Um für alle Browser dasselbe **Look-and-Feel** der eigenen Webseite zu erreichen, ist es **Best-Practise** die Abstände initial auf **null** zu setzen.
- ❑ Dies wird als **Normalisieren** bezeichnet.

```
h1 {  
    display: block;  
    font-size: 2em;  
    margin-top: 0.67em;  
    margin-bottom: 0.67em;  
    margin-left: 0;  
    margin-right: 0;  
    font-weight: bold;  
}
```

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

4.2 CSS – Eigenschaften



Vererbung von CSS-Eigenschaften

- ❑ Unter **Vererbung** (en: inheritance) versteht man die Weitergabe von CSS-Eigenschaften eines **Elternelements an seine Kindelemente**.
- ❑ Vererbt werden **nur Eigenschaften**, für die im Kindelement **kein expliziter Wert** angegeben wird.
- ❑ CSS-Eigenschaften können in zwei Typen eingeteilt werden:
 - **Vererbbare Eigenschaften**, die standardmäßig auf den **Wert** des Elternelementes gesetzt werden.
 - **Nicht vererbbare Eigenschaften**, die auf die CSS-**Standardwerte** eines Elementes gesetzt werden.

- ❑ Beispiele:
- ❑ Vererbbare Eigenschaften:
 - Die meisten Eigenschaften von **Textinhalten** werden vererbt.
 - Beispiele: color, font-size, font-family, line-height, ...
- ❑ Nichtvererbbare Eigenschaften
 - **Box-, Layout- und Grundeigenschaften** werden nicht vererbt.
 - Beispiele: border, padding, background-color, overflow, ...

Farben

- ❑ CSS kennt die folgenden Basis-Farbeigenschaften:

`color`: Schriftfarbe
`background-color`: Hintergrundfarbe für ein Element.

- ❑ Der Farbwert kann auf verschiedene Arten angegeben werden.
- ❑ Vordefinierte Farbnamen wie zum Beispiel `blue`, `green`, `red`, `tomato`,...
(https://www.w3schools.com/css/css_colors.asp)
- ❑ Farben können über ihre **RGB-Werte** [0,255], also die Farbintensität von Red, Green und Blue angegeben werden:
Rot: `rgb(255,0,0)` ; Gelb `rgb(255,255,0)`
- ❑ Mit den **RGBA-Werten** kann über einen vierten Wert, den **Alphawert**, die **Transparenz** angegeben können: **Alpha** kann Werte zwischen 0 (`transparent`) und 1 (`opaque`) annehmen.

```
p {  
  color: cyan;  
  background-color: LightSteelBlue;  
}
```

mit Farbnamen

```
p {  
  color: rgb(0,255,255);  
  background-color: rgb(176, 196, 222);  
}
```

in rgb

```
p {  
  color: rgba(0,255,255,0.5);  
  background-color: rgb(176, 196, 222);  
}
```

in rgba, halbdurchsichtig

Formatierung von Text

- ❑ Die **Schriftart** wird mittels **font-family** definiert, beispielsweise **Arial**.
- ❑ Besitzt die Schriftart im Namen ein Leerzeichen muss Sie in Hochkomma angegeben werden **"Times New Roman"**.
- ❑ Da nicht jeder Browser alle Schriftarten installiert hat, kann man über eine **Fallback-Option** mehrere Schriftarten, durch ein **Komma** getrennt angeben (siehe Beispiel).
- ❑ Der Browser versucht von **links nach rechts** die Schriftarten auszuwählen.
- ❑ W3schools hat eine Liste von **websicheren** Schriftarten gelistet: **Arial, Verdana, Tahoma, Times New Roman, ...** (https://www.w3schools.com/cssref/css_websafe_fonts.asp)

```
body {  
    font-family: Arial, "Times New Roman";  
    font-size: 1.2em;  
    background-color: beige;  
}
```

- ❑ Die Schriftgröße legt man mit **font-size** fest.
- ❑ Die Größenangabe kann wie folgt angegeben werden:
 - **px** : CSS-Pixel, wird vom Browser an die jeweilige Bildschirm Auflösung angepasst, ausgehend von Full-HD (1920 x 1080).
 - **em** : relativ zur Browser-Standardschriftgröße (default: 1em = 16px), bzw. zur geerbten Schriftgröße
 - **%** : relativ zur geerbten Schriftgröße
 - **vw/vh** : prozentuale Teil der Viewportbreite oder -höhe

Web Fonts von einem Web Repository laden

- ❑ Mit der `@font-face` AT-Regel können Webdesigner eigene Schriftarten verwenden.
- ❑ In der `@font-face` AT-Regel geben Sie zunächst einen Namen für die Schriftart an z. B. `myFont` und verweisen dann per `url`-Methode auf die Schriftartdatei, die Sie verwenden wollen.

```
@font-face {  
    font-family: myFont;  
    font-style: normal;          /* Default Font Style: nicht italic oder oblique */  
    font-weight: bold;          /* Stärke der Schriftzeichen bold = Fettdruck */  
    src: url("https://example.io/web-fonts/myFont.woff") format("woff");  
}
```

woff: Web Open Font Format

```
body {  
    font-family: myFont, Sans-Serif;  
}
```

Formatierung von Text

- ❑ Achtung: Größenangaben wie z.B. **em** können sich zusammensetzen ("compounding"). **em** bezieht sich immer auf die **Font-Size des Elternelementes**.
- ❑ Möchten Sie das vermeiden können Sie **rem (root em)** verwenden.

- ❑ **Beispiel:**

style.css

```
span {  
    font-size: 1.6em;  
}
```

index.html

```
<div>  
    <span>Outer <span>inner</span>  
    outer</span>  
</div>
```

- ❑ **Erklärung:** Unter der Annahme, dass die
 - **Standardschriftgröße** des Browsers **16 Pixel** beträgt, würden die Wörter "outer" mit **25,6px** ($= 16\text{px} * 1.6$) gerendert, das Wort "inner" jedoch mit **40,96px** ($= 16\text{px} * 1.6 * 1.6$).



Formatierung von Text

- ❑ Die **Schriftstärke** wird mittels **font-weight** (normal, bold, lighter, bolder) definiert.
- ❑ Der **Schriftstil** wird mittels **font-style** (normal, italic, oblique) definiert.

```
p.normal {  
  font-style: normal;  
}
```

```
p.italic {  
  font-style: italic;  
}
```

```
p.oblique {  
  font-style: oblique;  
}
```

```
<p class="normal">This is a paragraph in normal style.</p>
```

```
<p class="italic">This is a paragraph in italic style.</p>
```

```
<p class="oblique">This is a paragraph in oblique style.</p>
```

This is a paragraph in normal style.

This is a paragraph in italic style.

This is a paragraph in oblique style.

Formatierung von Text

- Mit `text-align` können Sie Ihren Text linksbündig (`left`), rechtsbündig (`right`) oder zentriert (`center`) anzeigen lassen.



- Mit `text-align: justify;` können Sie einen **Blocksatz** festlegen, bei dem der Text links- und rechtsbündig abschließt.



- Mittels `line-height` kann die **Zeilenhöhe** verändert werden. Der **Text** steht dabei **immer in der Mitte** der Zeilenhöhe.



```
p {  
    font-weight: bold;  
    font-style: italic;  
}
```

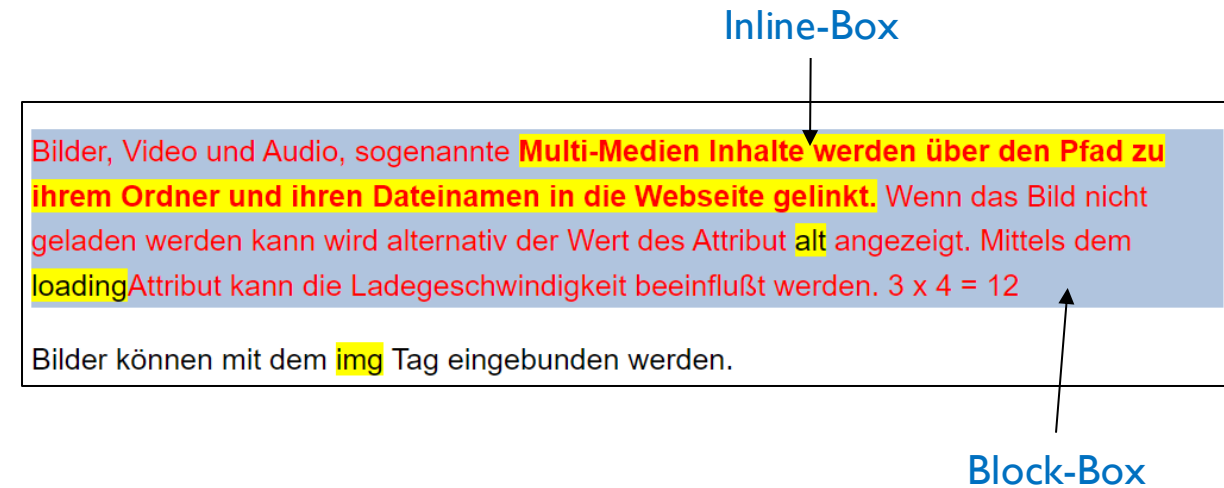
```
h1 {  
    text-align: center;  
}
```

```
p {  
    text-align: justify;  
    line-height: 1.5  
}
```



Box-Modell

- ❑ Jedes HTML-Element – gleich ob Block- oder Inline-Element – wird im **Box-Modell** als **rechteckige Box** betrachtet.
- ❑ Bei einem **Blockelement** (div, p, table, ...) nimmt die **Box** die **ganze Breite** der **übergeordneten Box** an, wenn die **Breite (width)** für das Element nicht angegeben ist.
 - Die **Höhe passt sich dem Inhalt der Box an**, wenn eine Angabe der Höhe des Elementes (**height**) fehlt.
 - Am Anfang und Ende eines Blocks erfolgt dann ein Zeilenumbruch (Bild: **blaue Box**)
- ❑ Bei **Inlineelementen** (a, span, em, ...) wird die **Höhe** und die **Breite** der Box durch den Inhalt bestimmt.
 - Die zugehörige Box erstreckt sich über einen Zeilenumbruch hinweg (Bild: **gelbe Box**)

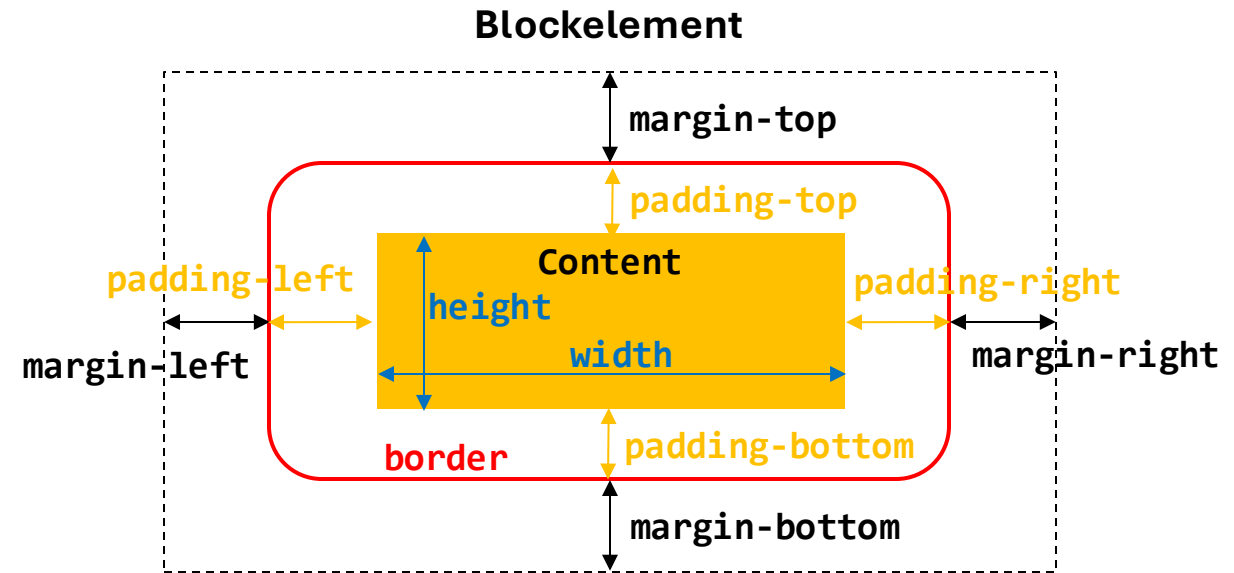


Box-Modell im Überblick

- **width, height**: Breite und Höhe des Elementinhaltes
- **padding, margin, border**: können für links, rechts, oben, unten einzeln festgelegt werden
- Gesamtelementbreite **B** ist die Summe aus
$$B = \text{margin-left} + \text{border-left} + \text{padding-left} + \text{width} + \text{padding-right} + \text{border-right} + \text{margin-right}$$
- **Kurzschreibweise** für margin, padding und border möglich. Beispiele

```
padding: 25px 50px 75px 100px
```

```
/* Start Top padding is 25px  
   Dann Right padding is 50px  
   Dann Bottom padding is 75px  
   Dann Left padding is 100px */
```



```
/* Top Right Bottom Left 2pt */  
border: 2pt; /* Ein Wert */  
  
/* Top & Bottom 0 | Right & Left auto */  
margin: 0 auto; /* Zwei Werte */  
  
/* Top 10px | Right & Left 5px | Bottom 1em */  
padding: 10px 5px 1em; /* Drei Werte */
```

auto: Box wird horizontal zentriert

<div>-Element

- ❑ Das **<div>-Tag** ist eine **Block-Box** und kann dazu verwendet werden ihr HTML-Dokument zu **gestalten**.
- ❑ Das **<div>-Tag** dient als **Container** für Inhalte – die mit CSS gestylt oder mit JavaScript manipuliert werden sollen.
- ❑ Das **<div>-Tag** lässt sich leicht mit dem **class-** oder **id-Attribut** gestalten.
- ❑ Jede Art von Inhalt kann in das **<div>-Tag** eingefügt werden.
- ❑ **Tip:** **<div>-Tags** sollten nur verwendet werden, wenn kein geeignetes HTML-Element für denselben Zweck vorhanden ist.

Einbinden von Bildern

Bilder können mit dem **img** Tag eingebunden werden.

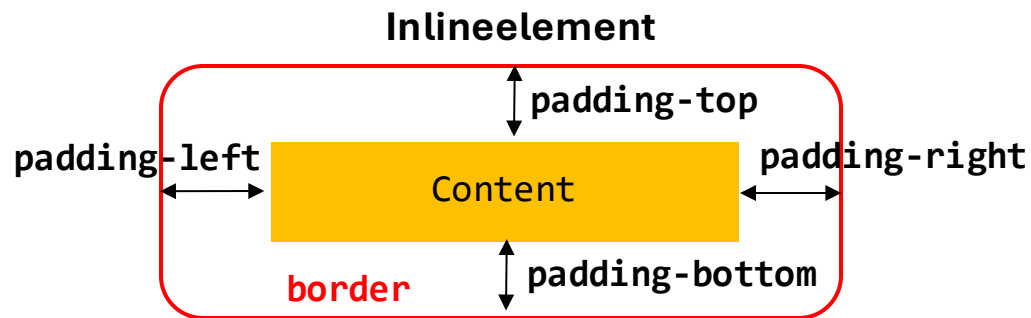
Das **src-Attribut** spezifiziert den Ablageort der zugehörigen Datei. Ein Beispiel für ein img-tag ist gegeben durch
``

`<div class="images">`

```
div.images {  
    width: 500px;           /* Content-Breite */  
    height: 400px;          /* Content-Höhe */  
    background-color: orange; /* Hintergrund orange */  
    border: 3px solid red;   /* Roter Rahmen 3px Dicke */  
    border-radius: 7px;     /* Runde Ecken */  
}
```

-Element

- ❑ Das **-Tag** ist ein **generisches Inline-Element** und kann dazu verwendet Teile des Dokumententextes zu markieren und gestalten.
- ❑ Das **-Tag** ist eine **Inline-Box**.
- ❑ Für **Inline-Boxen** kann keine Breite (**width**) und Höhe (**height**) definiert werden. Die Breite und Höhe wird durch den Content bestimmt.
- ❑ Inline-Boxen besitzen keine **margin** (Rand).



```
<span id="attribut">src-Attribut</span>
```

```
#attribut {  
    font-family: "Times New Roman";  
    font-size: 60px;  
    font-style: normal;  
    color: red;  
    padding-left: 6px;  
    padding-right: 6px;  
    border: 2px dotted black;  
}
```

Einbinden von Bilder

Bilder können mit dem **img** Tag eingebunden werden.

Über das **src-Attribut** spezifiziert

den Ablageort der zugehörigen Datei. Ein Beispiel für ein img-tag ist gegeben durch

```

```

Box-Eigenschaften: Border

□ **Block-Elemente** und **Inline-Elemente** besitzen einen **Grenze** (**border**). Der Grenze können die folgenden Eigenschaften zugeordnet werden

- Linienstärke: `border-width`
- Linienfarbe: `border-color`
- Linienmuster: `border-style`
- Runde Ecke: `border-radius`

```
div {  
  border-style: dashed;  
  border-color: red;  
  border-width: 2px;  
  border-radius: 7px;  
}
```

Rand

Einbinden von Bilder

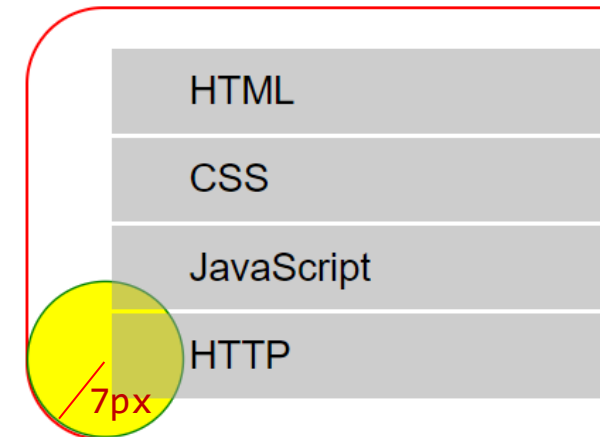
Bilder können mit dem **img** Tag eingebunden werden.

Das **src-Attribut** spezifiziert den Ablageort der zugehörigen Datei. Ein Beispiel für ein img-tag ist gegeben durch

```

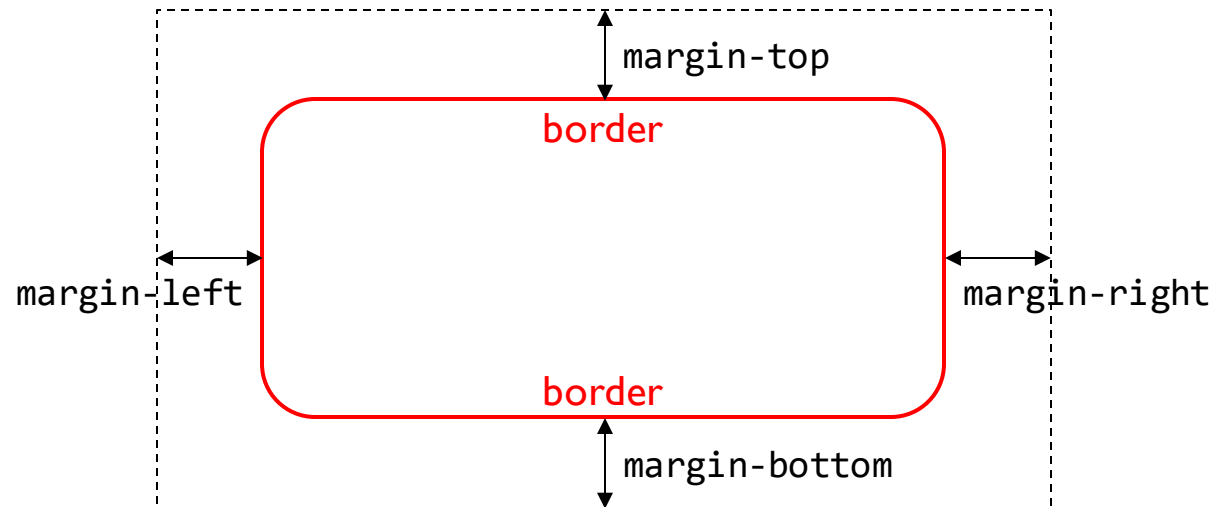
```

border-radius



Box-Eigenschaften: Margin

- ❑ Für **Block-Elemente** (nicht für Inline-Elemente) lässt sich für die **Box** ein **Abstand (margin)** zum **nächsten Box-Element** definieren.
- ❑ **Bildlich**: Die Boxen innerhalb einer HTML-Seite erhalten einen horizontalen und vertikalen Zwischenraum.



```
div.images {  
  margin-top: 40px;  
  margin-left: 20px;  
  margin-bottom: 40px;  
  background-color:  
  white;  
  border: 3px solid red;  
  border-radius: 7px;  
}
```

Einbinden von Bilder

Bilder können mit dem **img** Tag eingebunden werden.

Das **src-Attribut** spezifiziert den Ablageort der zugehörigen Datei. Ein Beispiel für ein img-tag ist gegeben durch

```

```

geladen werden kann alternativ der Wert des Attribut **alt** angezeigt. Mittels dem **loading** Attribut kann die Ladegeschwindigkeit beeinflusst werden. $3 \times 4 = 12$



Margin-Collapse

- **Margin Collapse:**
- **vertikal:** Die oberen (margin-top) und unteren (margin-bottom) Ränder von zwei **aneinander grenzenden** Elementen werden zu einem **einigen Rand** zusammengelegt, der dem **größten** der **beiden Ränder** entspricht.
- **horizontal:** Die seitlichen Ränder von benachbarten Elementen **addieren** sich.

```
<style>
  p {
    margin-top: 24px;
    margin-bottom: 24px;
  }
</style>
```

<p>Paragraph One</p>
<p>Paragraph Two</p>

<p>Paragraph One</p>

<p>Paragraph Two</p>

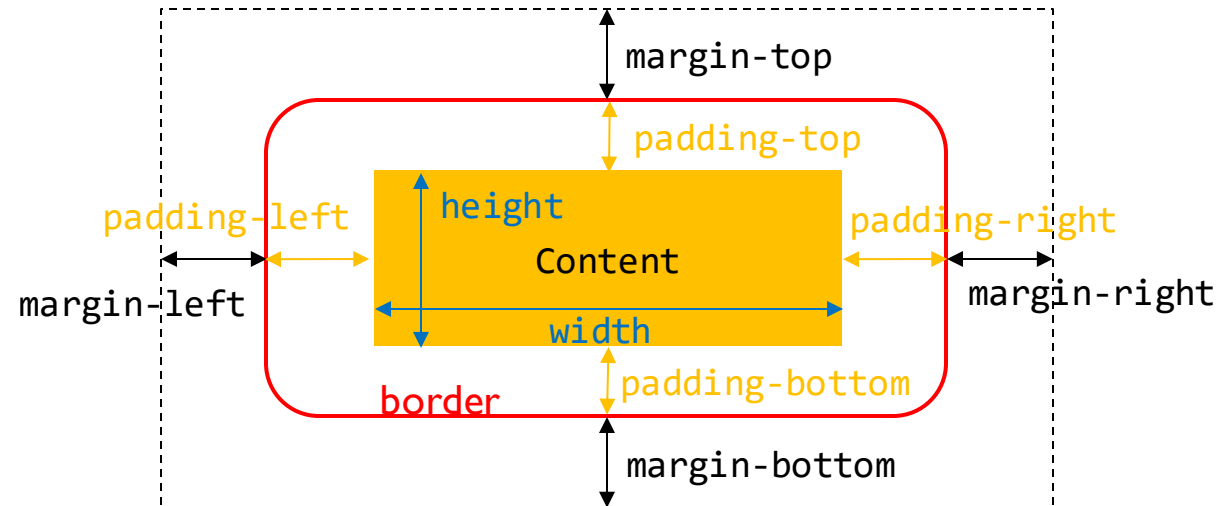
24px

24px

24px

Box-Eigenschaften: Padding

- Der Innenabstand (**padding, Polsterung**) definiert, den Abstand des Element-Inhaltes (**content**) vom Rand der Box.
- Die Innenabstände **addieren** sich zur Höhe (**height**) und Breite (**width**) des **Contents**.
- Innenabstände** können für **Block-** und **Inline-Elemente** angewendet werden.



```
div.images {  
  width: 500px;  
  height: 400px;  
  ...  
  padding-top: 10px;  
  padding-left: 30px;  
  padding-right: 30px;  
  padding-bottom: 10px;  
  border: 3px solid red;  
  border-radius: 7px;  
}
```

Einbinden von Bildern

Bilder können mit dem **img** Tag eingebunden werden.

Über das **src-Attribut**

spezifiziert den Ablageort der zugehörigen Datei. Ein Beispiel für ein img-tag ist gegeben durch

```

```

Hintergrundbilder

- Die Eigenschaft `background-image` stapelt ein oder mehrere Hintergrundbilder für ein Element.
- Standardmäßig wird ein Hintergrundbild in der oberen linken Ecke eines Elements platziert und sowohl vertikal als auch horizontal wiederholt.
- Der Hintergrund eines Elements ist die Gesamtgröße des Contents (Width & Height), einschließlich Innenabstand (Padding) und border.
- Das Bild `url("img/Jeep.png")` liegt über dem Bild `url("img/desert.jpg")`. Die border (rote Linie) wird zum Schluss über den Stapel gezeichnet.

padding-left width padding-right



```
div.image {  
    border: 2px solid red;                                /* Rote Grenzlinie*/  
    background-image: url("img/jeep.png"), url("img/desert.jpg");    /* Stapel 2 Bilder */  
    background-repeat: repeat-x;                          /* soll in x-Richtung wiederholt werden */  
}
```

Hintergrundbilder

- ❑ Hintergrundbilder können **positioniert** und in der **Größe** angepasst werden.
- ❑ **Tipp:** Legen Sie immer eine Hintergrundfarbe fest, die verwendet werden soll, falls das Bild nicht verfügbar ist.
 - Die Hintergrundfarbe wurde im Beispiel in Form einer **HEX-Darstellung** der rgb-Farbwerte angegeben:
#xyyzzz ; x,y,z ∈ [0,1,2,3 ..., 9, a, b, ..., f]

```
div.image {  
  width: 400px;  
  border: 2px solid red;  
  background-image: url("img/Jeep.png"), url("img/desert.jpg");  
  background-color: #cccccc;  
  background-repeat: no-repeat;  
  background-position: top center, bottom right;  
  background-size: 30% auto, cover;  
}
```



```
/* helles grau rgb(204,204,204) */  
/* jedes Bild nur 1-mal anzeigen */  
/* Position der Bilder */  
/* Breite und Höhe eines Bildes */
```

Benutzerdefinierte Eigenschaften

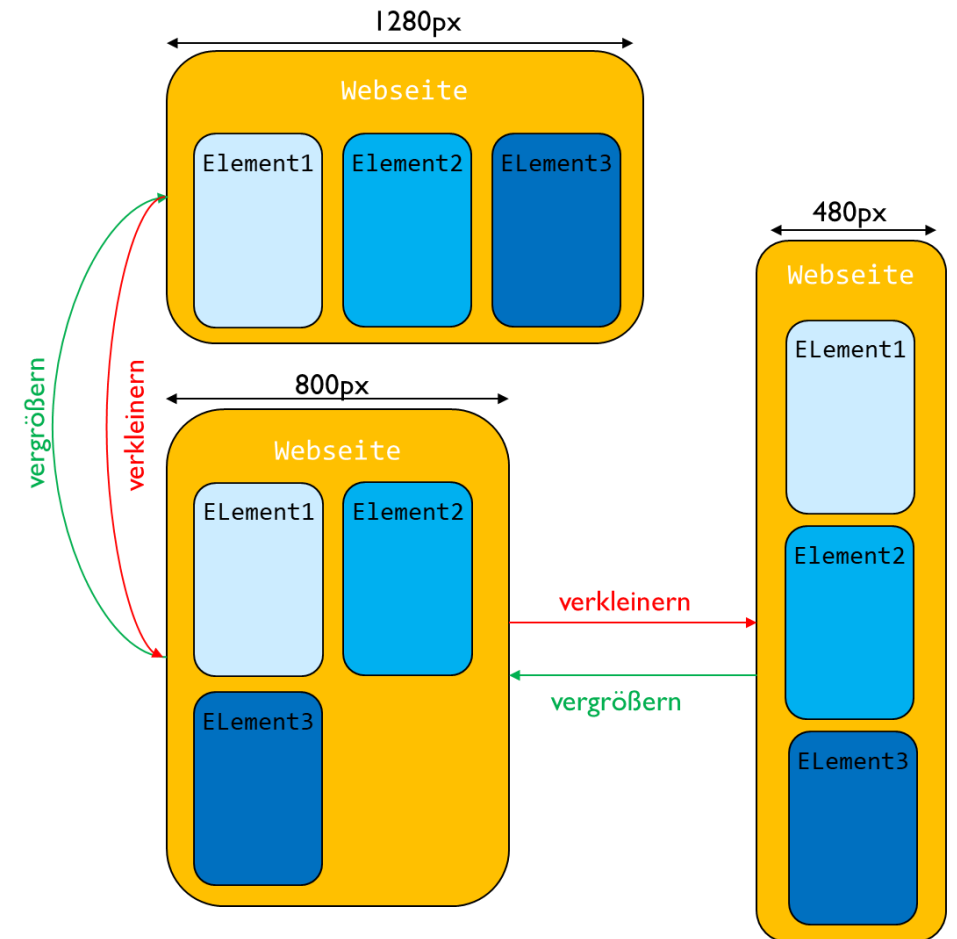
- ❑ CSS ermöglicht es **benutzerdefinierte Eigenschaften** (**custom properties**) in einem **Stylesheet** festzulegen.
 - Der **Name** einer custom property muss mit **zwei Minuszeichen** beginnen und ist **case-sensitive**.
- ❑ **Globale** custom properties können über die **strukturelle Pseudo-Klasse¹ :root** für das **ganze HTML-Dokument** definiert werden.
 - **:root** repräsentiert das oberste Element eines HTML-Dokumentes <html>
- ❑ Der Wert einer CSS custom property wird mittels der CSS-Funktion **var()** einer normalen Property zugewiesen.

¹**Strukturelle Pseudoklassen** identifizieren Elemente aufgrund ihrer Position im **DOM**-Tree.

```
:root {  
  --main-color: white;  
  --main-bg-color: red;  
}  
  
span {  
  background-color: var(--main-bg-color);  
  color: var(--main-color);  
}
```

prata iam serena **Lorem ipsum dolor sit amet...** Cytharizat

4.3 Gestaltung von Webseiten-Layouts



Gestalten eines Webseiten-Layouts

Bei der Erstellung eines Webseiten-Layouts können Sie wie folgt vorgehen:

1. **Seitenaufbau planen:** Zunächst fertigen Sie eine Skizze des gewünschten Seitenlayouts auf Papier an (**sketching**) und übertragen den **finalen** Entwurf in eine Software.
2. **HTML erstellen:** Erstellen Sie anschließend das HTML-Dokument für jede Webseite (siehe Kapitel3).
3. **Elemente formatieren:** Weisen Sie den Elementen im HTML-Dokument per Cascading Stylesheets (CSS) die gewünschten Formatierungsmerkmale wie Hintergrundfarbe, Schriftmerkmale, Rahmen, **Höhe** und **Breite**.
4. **Blockelemente positionieren:** Positionieren Sie die einzelnen Elemente an der gewünschten Stelle auf der Webseite mittels CSS.

- ❑ Die **Punkte (1) – (3)** haben wir in den zurückliegenden Kapiteln besprochen, jetzt betrachten wir die **Positionierung** der Blockelemente auf der Webseite.
- ❑ Damit Sie ein Blockelement auf einer Web-Seite frei positionieren können, müssen Sie per CSS seine **Breite definieren**, **ansonsten** füllt es die gesamte Breite des Anzeigefensters bzw. des übergeordneten Eltern-Elementes aus.

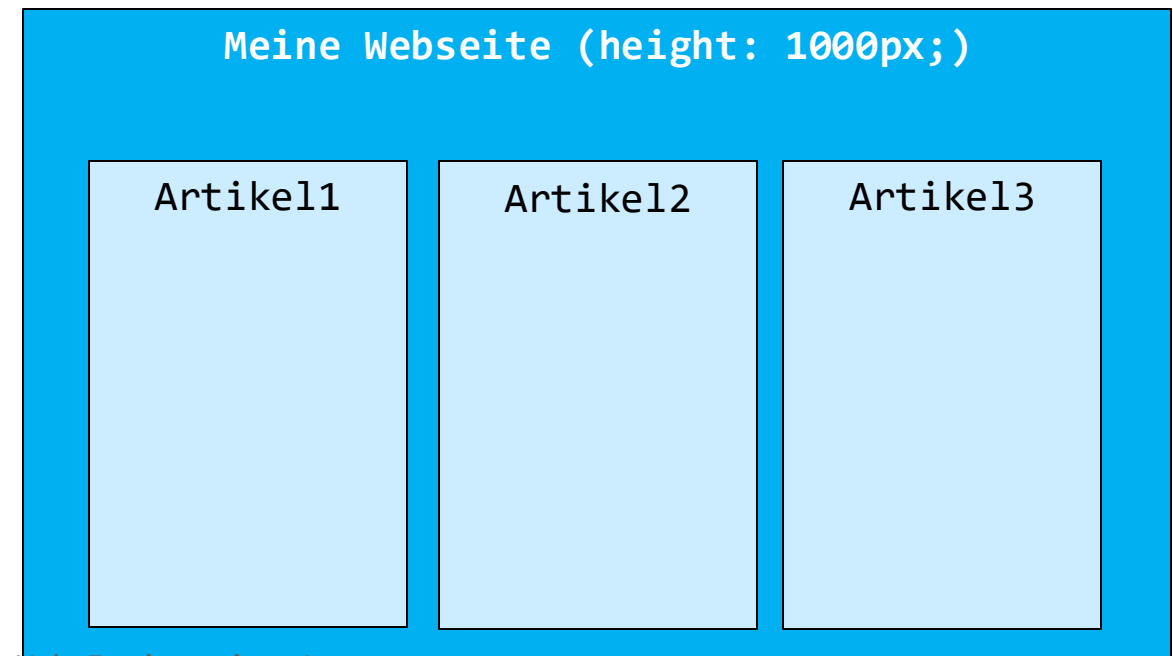
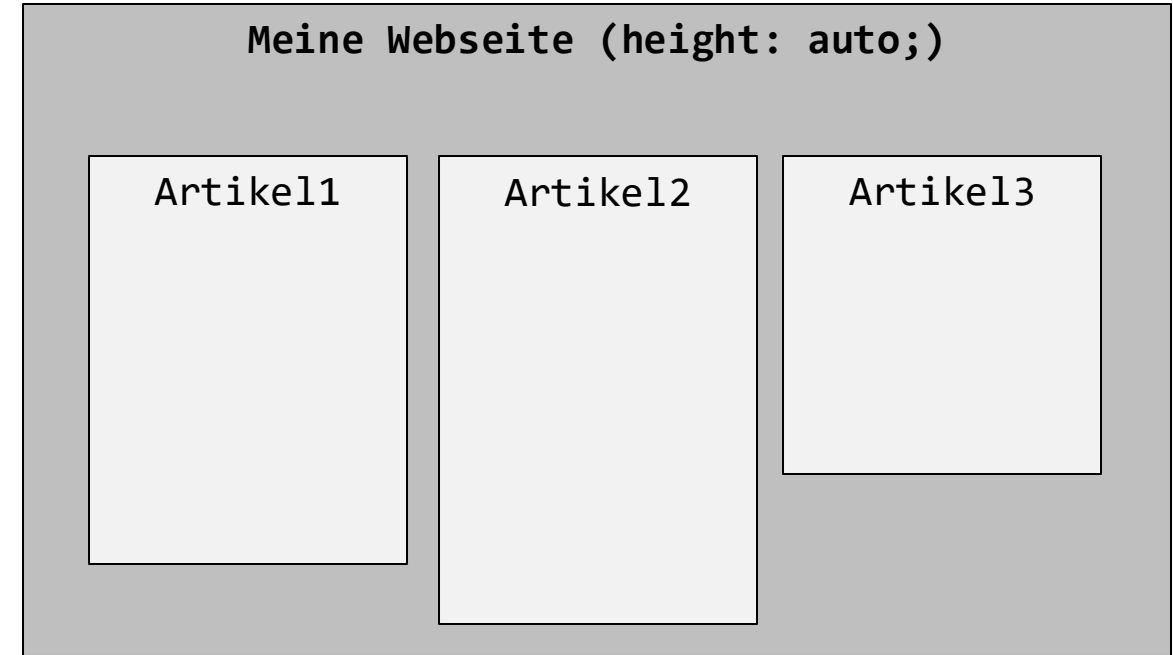
Breite von Blockelementen

- ❑ Mit der CSS-Eigenschaft `width` und `height` können Sie die Breite und Höhe eines **Elements-Inhaltes** festlegen.
- ❑ Für die **Größenangabe** gibt es die folgenden Möglichkeiten:
 - **Exakte Zahlenangaben** mit entsprechender Maßeinheit (px oder pt : 1/72-inch).
 - **Relative Angaben** (%, vw, vh) relativ zur Breite des Eltern-Elements oder zum Viewport.
 - **`width:auto`** ist der **Default-Wert**, der dazu führt, dass der Container die **komplette Breite** des **übergeordneten Elementes** ausfüllt, mit dem Effekt das in Abhängigkeit von der **Positionierung**
 - die Boxen **untereinander angeordnet** werden,
 - oder **sich überlagern** und damit nur teilweise sichtbar sind.

```
section.a {  
    width:  auto;           /* Komplette Breite */  
    height: auto;          /* Komplette Höhe */  
    border: 1px solid black;  
}  
  
section.b {  
    width: 150px;           /* 150 Pixel */  
    height: 50vh;          /* 50% der Viewport-Höhe */  
    border: 1px solid black;  
}  
  
section.c {  
    width:  30%;           /* 30 Prozent Eltern-Element */  
    height: 100%;          /* 100 Prozent Eltern-Element */  
    border: 1px solid black;  
}
```

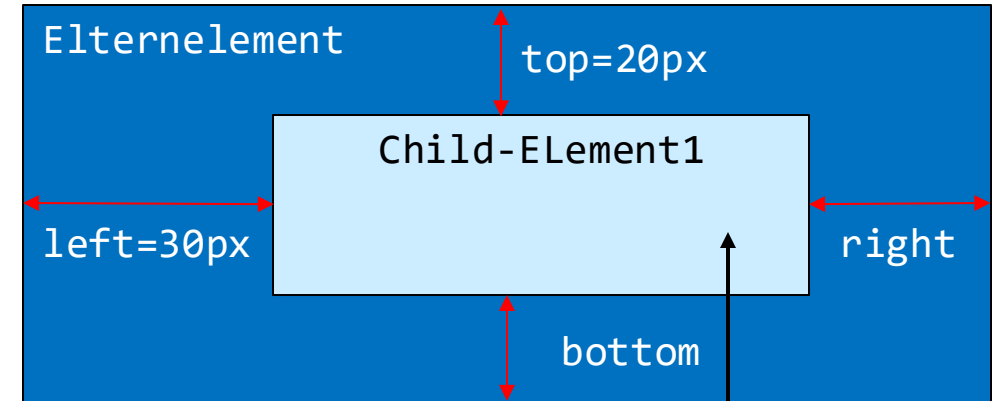

Höhe von Blockelementen

- **height:auto** ist das Default-Verhalten, Element passt seine **Höhe an den Inhalt** und an die **Höhe des übergeordneten Elementes** an.
- Um ein gleichmäßiges Seitenlayout zu erreichen ist es sinnvoll gleichartigen Elementen eine **gleiche Höhe height** zuzuordnen.
 - **height:50em;** Vorgabe einer Zahl mit Maßeinheit (z.B. in Abhängigkeit von der Standardschriftgröße)
 - **height:100%;** Angabe als Prozentwert der Höhe der umgebenden Eltern-Box.
 - **height:50vh;** Angabe als Prozentwert der Viewport-Höhe



CSS-Position

- Eine Möglichkeit ein Webseiten-Layout zu gestalten ist, die **Blockelemente explizit** auf einer Webseite zu platzieren.
- Über die CSS-Eigenschaft **position** können unterschiedliche **Bezugspunkte** festgelegt werden:
 - `position:static`; folgt dem normalen Fluß im Dokument
 - `position:relative`; in Bezug zum Textfluss
 - `position:absolute`; in Bezug zum Eltern-Container
 - `position:sticky`; in Bezug zum Eltern-Container
 - `position:fixed`; in Bezug zum Viewport
- Über die vier Positionsparameter **top**, **right**, **bottom**, **left** kann die Element-Box entsprechend verschoben werden.
- Als **Zahlenwerte** sind **Maßeinheiten (px)** oder **prozentuale Angaben (%)** in Bezug zur Höhe und Breite des Bezugs-Containers möglich.



```
.absolute-box {  
    position: absolute;  
    top: 20px;  
    left: 30px;  
    width: 100px;  
    height: 40px;  
}
```

CSS-Position: static und relative

- Mögliche Positionierungen
 - `position:static (default)`; keine spezielle Formatierung; Element positioniert sich im Textfluss gemäß Reihenfolge im Quelltext.
`top`, `left`, `right`, `bottom` haben keinen Einfluss auf die Position des Elementes.
 - `position:relative`; Abweichung von der Position, die das Element im Textfluss eingenommen hätte.
 - Für `static` und `relative` werden als Bezugsrahmen für die Positionierung die **Kanten** der **Content-Box** des **Elternelementes** verwendet.

Position: Static

```
Element2 {  
  position: static;  
}
```

Parent Element

Child-Element1 not shifted.

Child-Element2 has a static position according to default flow.

Position: Relative

```
Element2 {  
  position: relative;  
  left: 100px;  
  bottom: 20px;  
}
```

Parent Element

Child-Element1 not shifted

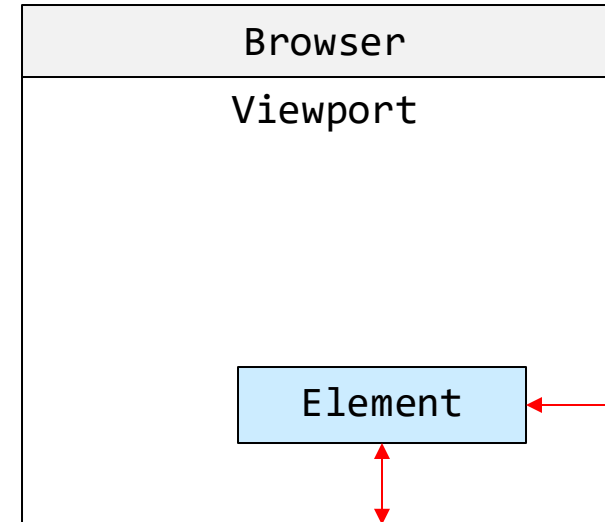
Child-Element2 shifted to the (left: 100px;) and shifted up (bottom: 20px;).

← left →

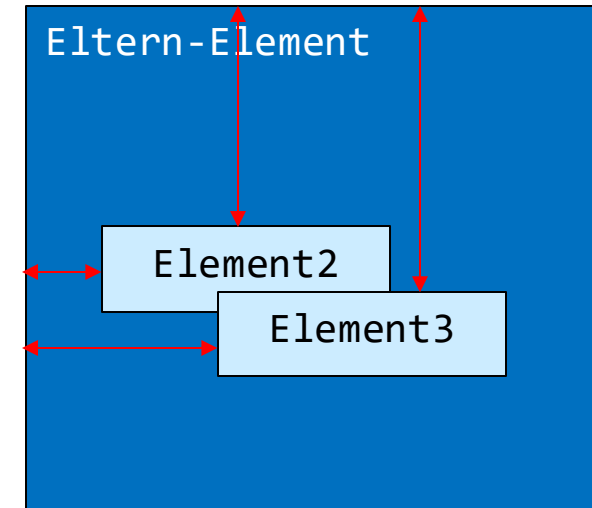
↑ bottom ↓

CSS Position

- Weitere Möglichkeiten sind
 - `position:fixed;` Feste Positionierung des Elements in Bezug zum **Viewport** (zum Fenster), bleibt beim Scrollen fix an seinem Platz.
 - `position:absolute;` Positionierung des Elements mit Bezug auf das **übergeordnete Elternelement**.
Als Bezugsrahmen werden die **Kanten** der **Padding-Box** des **Elternelementes** verwendet.
Element wird aus dem Textfluss entfernt.



```
element {  
  position: fixed;  
  bottom: 20px;  
  right: 20px;  
  width: 300px;  
}
```



```
element2 {  
  position: absolute;  
  top: 250px;  
  left: 80;  
  width: 200px;  
}  
element3 {  
  position: absolute;  
  top: 270px;  
  left: 180px;  
  width: 200px;  
}
```

z-Index

- Positionierte Elemente (*absolut, fixed, relativ, sticky*) liegen per Voreinstellung in der **Reihenfolge** übereinander, in der sie im Quelltext aufgeführt werden.
 - Das **letzte Element** liegt oben.
- In der **3D-ComputerGrafik** verwendet man den **Wert** der **z-Koordinate**, um zu bestimmen welches Objekt sichtbar ist. Objekte die näher zum Betrachter liegen werden angezeigt.
- Diese Idee verwendet man auch bei CSS um **positionierte Elemente** gezielt **sichtbar** zu machen, **unabhängig** von ihrer **Reihenfolge** im Quelltext.
- Dabei weist die z-Achse in **Richtung des Betrachters**.
- Standardmäßig liegen alle Elemente in einer Ebene mit **z-index = 0**.

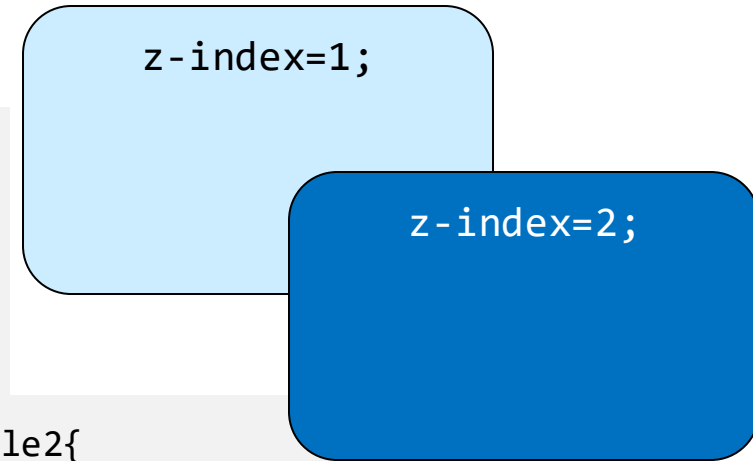
- **Syntax:**

```
z-index: auto | Ganzzahl ;
```

- Elemente mit höherer Ganzzahl liegen über den Elementen mit niedriger Ganzzahl.
- **auto**: Element erhält den z-Index seines Parent-Elementes.

```
#article1{  
  position: absolute;  
  top: 40px;  
  left: 10px;  
  z-index: 1;  
}
```

```
#article2{  
  position: absolute;  
  left: 50px;  
  top: 50px;  
  z-index: 2; }  
}
```

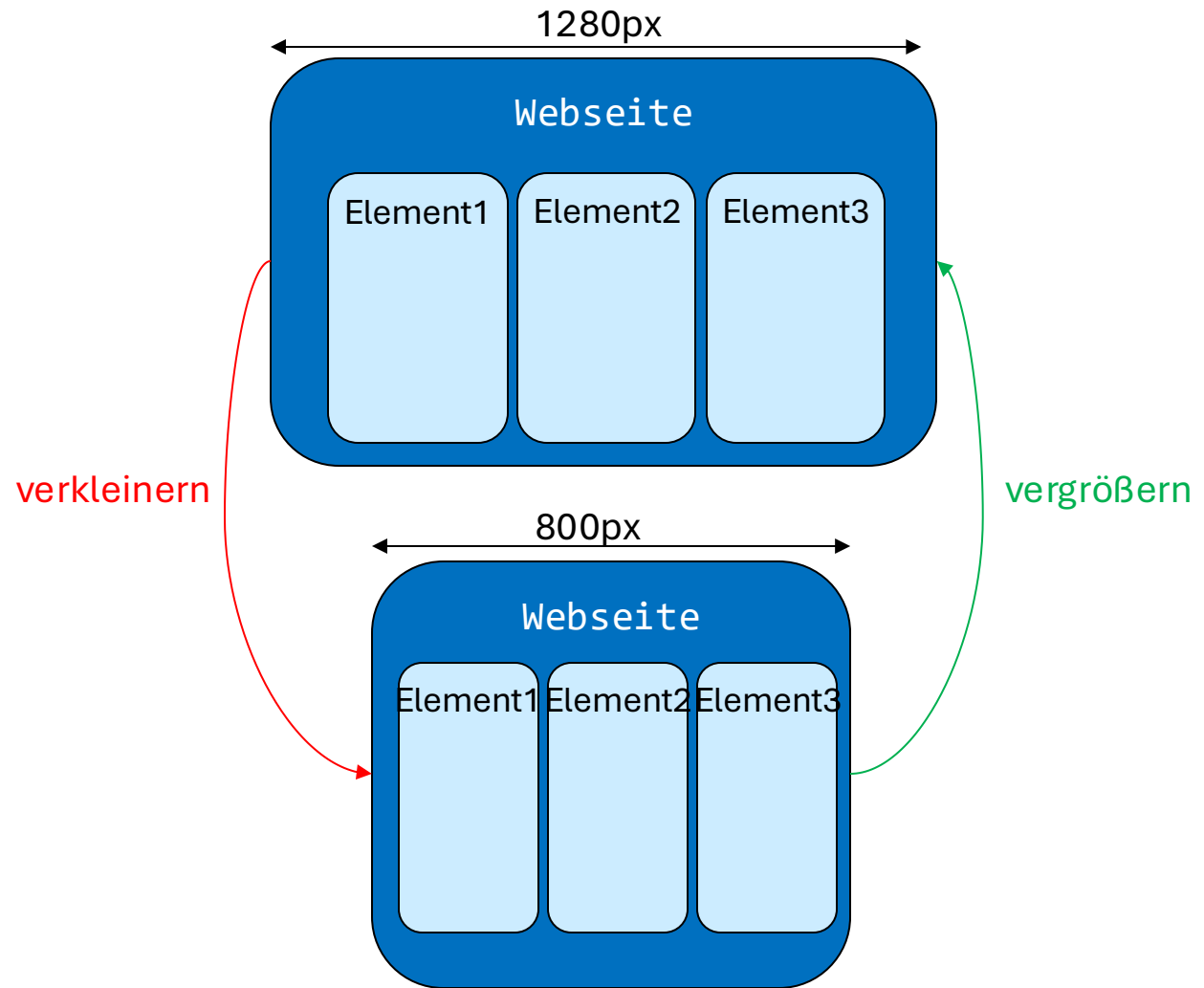


Fluides Layout

Fluides Layout: Das Fensterlayout (z.B.: 3 Spalten) bleibt bei der Veränderung der Fensterbreite erhalten. Elementgröße verändert sich kontinuierlich mit der Fensterbreite.

Inhalte wie Texte oder Bilder bleiben in ihrer Größe erhalten.

- Ein fluides Layout läßt sich mit den Eigenschaften `position`, `width`, und `left` herstellen (siehe vorherige Seite).
 - `position: absolute`
 - `width` und `left` oder `right` wird in %-Werten des Elternelementes angegeben
 - Elemente werden links nach rechts positioniert



Beispiel: Fluides Layout mit 3 Spalten

- Beispiel: 3-spaltiges Design mit Eltern-Element `<section>`
 - 3 Artikel sollen innerhalb des `<section>`-Elementes nebeneinander angeordnet werden, und ihre Größe dynamisch an die Größe des Viewports anpassen.

```
<section>
  <article id="article1">
    ...
  </article>

  <article id="article2">
    ...
  </article>

  <article id="article3">
    ...
  </article>
</section>
```

```
#article1{
  position: absolute;
  left: 1%;
  width: 32%;
}
```

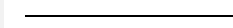
```
#article2{
  position: absolute;
  left: 34%;
  width: 32%;
}
```

```
#article3{
  position: absolute;
  left: 67%;
  width: 32%;
}
```

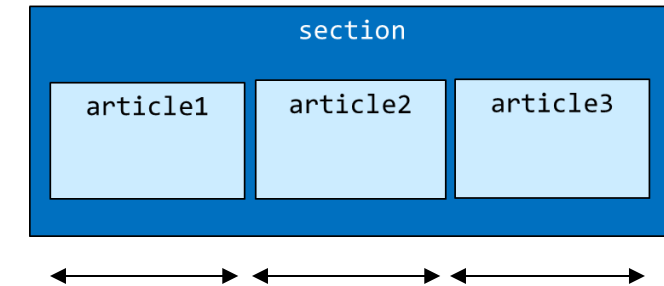
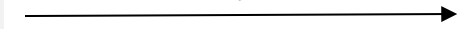
left: 1%;



left: 34%;



left: 67%;



width: 32%;

width: 32%;

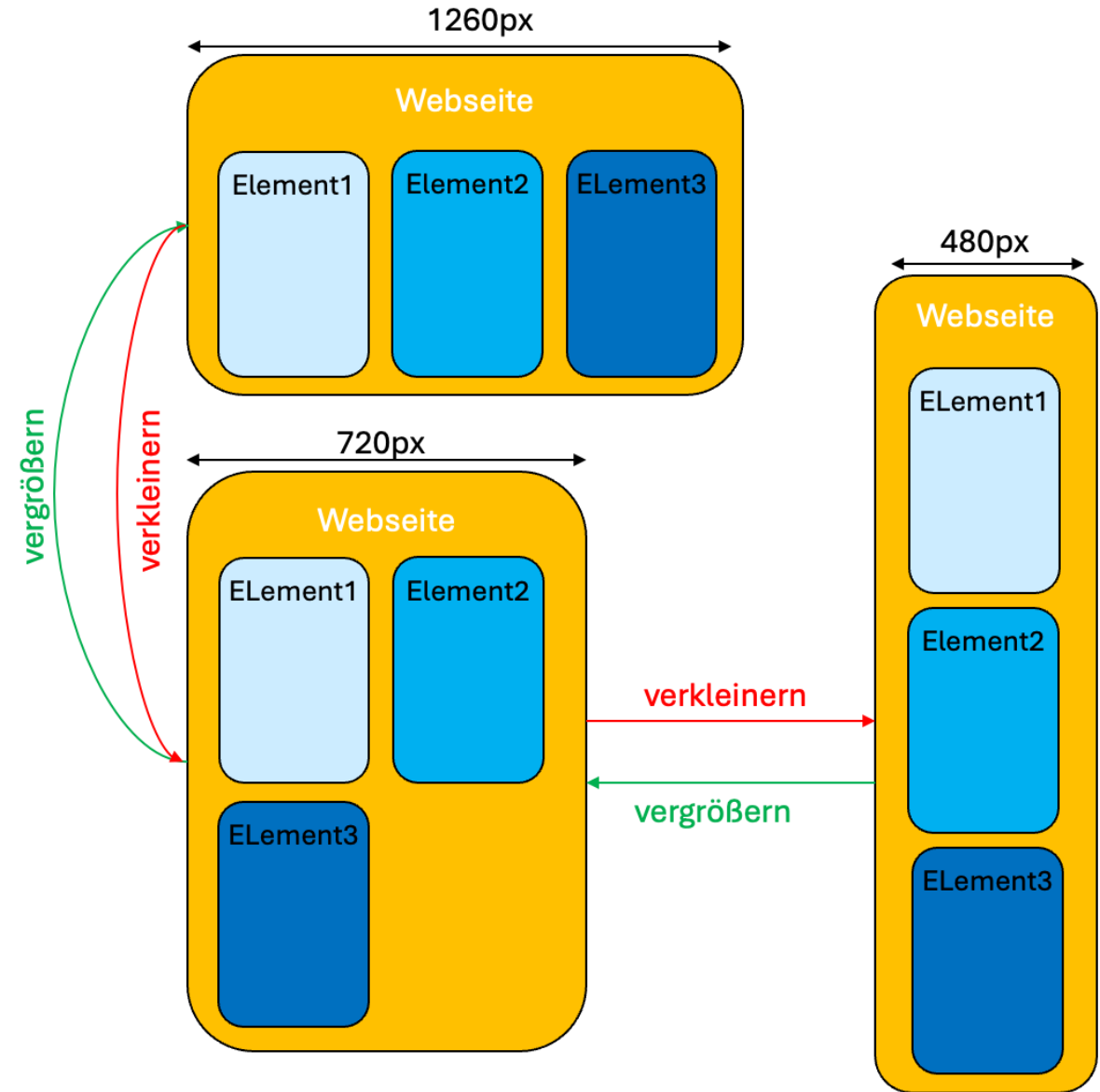
width: 32%;

Adaptives Layout mit float-Elementen

Adaptives Layout: Bei der Vergrößerung oder Verkleinerung des Viewports ordnen sich die **Blockelemente neu an**, sobald ihre **Mindestbreite (min-width)** unter- oder überschritten wird.

- ❑ Die fixe Positionierung von Block-Elementen zur Gestaltung einer Webseite, ist sehr **unflexibel** angesichts der Vielzahl an **möglichen Monitorgrößen** (Handy, Tablett, Flat-Screen, ...).
- ❑ Ziel ist es ein Webseitenlayout zu gestalten, das sich **flexibel** an die **Größe** des **Viewports** anpasst.
- ❑ Mit der Eigenschaft **float** werden Elemente - z.B. eine Spalte aus dem regulären Layout **herausgelöst werden** und können nach links (**left**) oder nach rechts (**right**) **positioniert** werden.

float: left | right | none



Beispiel: Bild als float-Element

- Ein **Bild** soll sich am **linken Rand befinden** und von dem beschreibenden Text rechts umflossen werden. Die **Lage** im Text bestimmt die **obere Kante** des **float-Elements**.

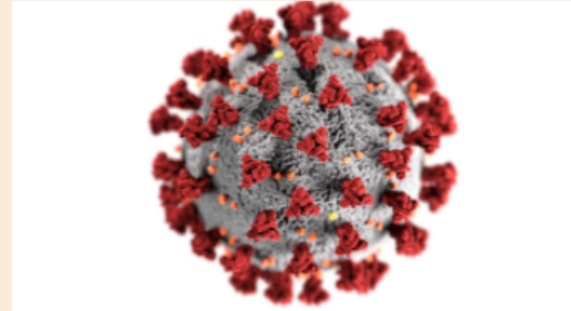
```
<p>Fortune plango vulnera stilantibus ocellis  
...•
```

```

```

```
Cytharizat cantico dulcis Philomena, flore  
rident ...  
</p>
```

auscultare vult. Nunc qua ad enarrandum hoc argumentums comit, si
ad auscultandum vostra erit benignitas. Cytharizat cantico dulcis



Philomena, flore rident vario prata
iam serena Omnia Sol temperat
purus et subtilis, novo mundo
reserat faciem Aprilis; ad Amorem
prosperat animus herilis et
iocundis imperat deus puerilis.

Rerum tanta novitas in solemni

vere et veris auctoritas iubet nos gaudere, vias prebet solitas, et in tue
vere fides est et probitas tuum retinere. Ama me fiedeliter! Fidem

```
img {  
    float: left ; /* Position linker Rand */  
    width: 15em;  
    margin: 10px 15px 10px 10px;  
}
```

Adaptives Layout mit float-Elementen

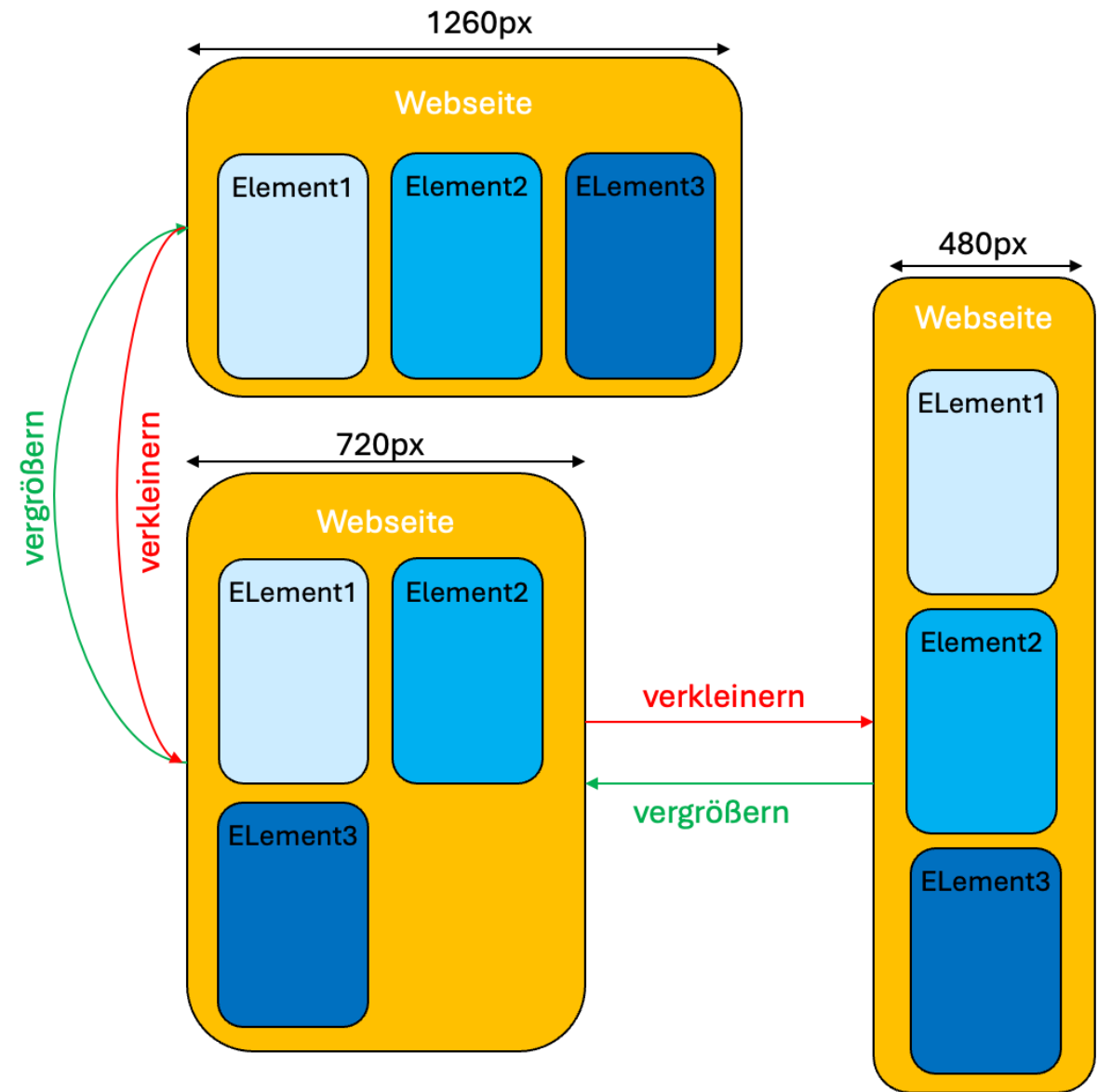
- ❑ Die ein **float-Element** umgebenden Elemente **umfließen** das aus dem Layout befreite Element (im Beispiel das Bild) .
- ❑ Die folgenden Werte sind für float erlaubt:
 - **left**: float-Element wird links ausgerichtet, die umgebenden Elemente umfließen das Element rechts.
 - **right**: float-Elemente wird rechts ausgerichtet, die umgebenden Elemente umfließen das Element links.
 - **none**: Element darf nicht fließen.
- ❑ Die Eigenschaft **float** lässt sich **nicht** auf **absolut positionierte** Elemente anwenden.
- ❑ Mittels float kann man sehr einfach **adaptive Web-Layouts** erzeugen, indem man **mehrere Elemente floaten** lässt.
- ❑ Diese Elemente überdecken sich nicht, sondern **schweben** nur so weit nach links oder rechts, bis sie an ein **anderes Float-Element stoßen**.

```
#article1 {  
    /* adaptiv: nach links fließen */  
    float: left;  
    /* fluid: Breite in % */  
    width: 32%;  
    /* adaptiv: Minimale Breite */  
    min-width: 380px;  
}
```

Adaptives und fluides Layout mit float für 3 Elemente

- Mittels der Kombination aus `float`, `width` und `min-width` lässt sich dann ein **adaptives** und **fluides** Design erstellen.
- Mittels `float` lassen sich z.B.: die 3 Artikel **links nebeneinander ausrichten**. Wird die **Mindestbreite unterschritten**, **umfließt** das **rechte äußere Element** die beiden linken Elemente. Wird die Mindestbreite erneut unterschritten fließt Element2 zwischen Element1 und Element3.

```
#article1, #article2, #article3 {  
    float: left;  
    width: 32%;  
    min-width: 460px;  
    height: 50em;  
    overflow: scroll;  
    border: 2px solid black;  
    background-color: AntiqueWhite;  
}
```

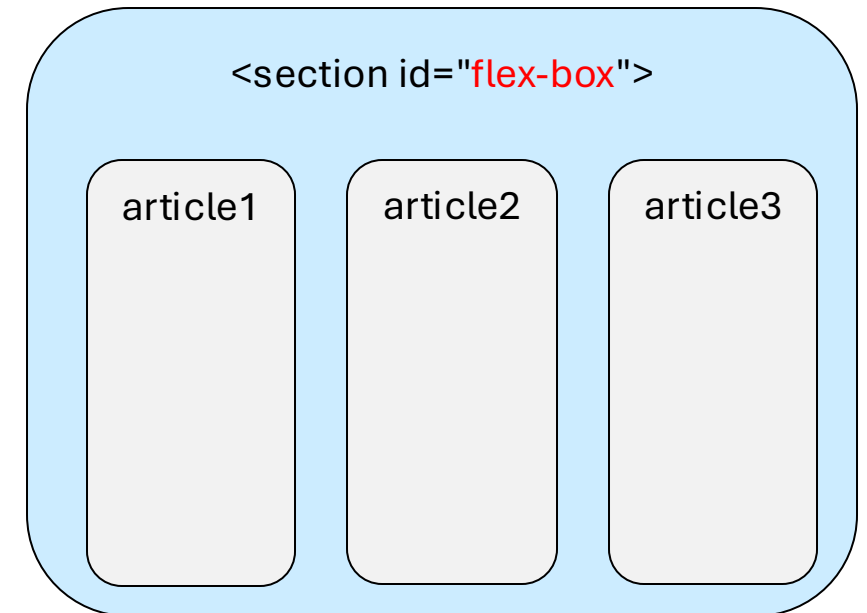
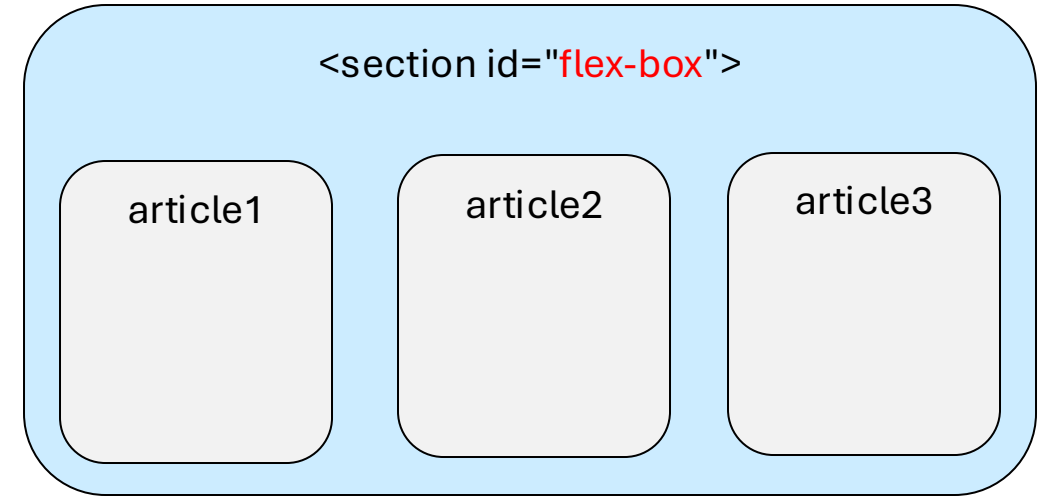


Flexbox

- ❑ Flexboxen stellt eine sehr flexible Layout-Methode für das Anordnen und Ausrichten von Elementen auf einer Webseite dar.
- ❑ Die Flexbox stellt ein Blockelement dar, das wiederum andere Block-Elemente enthält.

```
#flex-box {  
  display: flex;  
}
```

- ❑ Eigenschaften:
 - Flexbox verteilt den Platz in der Breite auf die enthaltenen Child-Elemente möglichst gleichmäßig.
 - Elemente werden auch in der Höhe angeglichen.



Flexbox-Typen

□ Zwei Möglichkeiten zur Aktivierung der Flexbox-Darstellung

○ `display: flex`

Die Flexbox verhält sich zu den umgebenden Elementen wie ein Blockelement.

oder

○ `display: inline-flex`

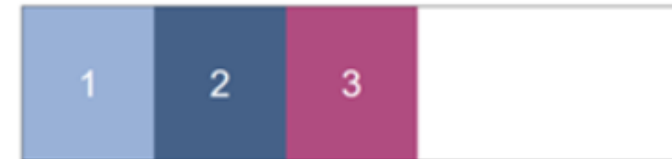
Die Flexbox verhält sich zu den umgebenden Elementen wie ein Inline-Element.

Hier einmal `display: flex`:



Und hier folgt Text

Und hier `display: inline-flex`:



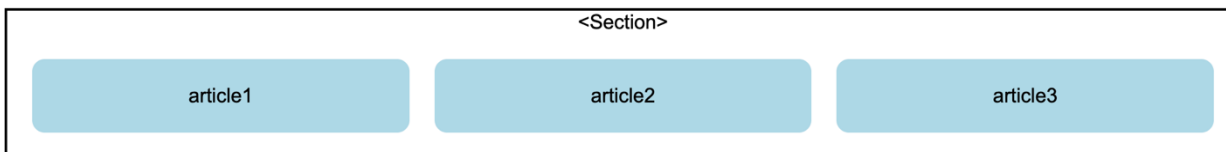
Und hier folgt Text

Flexbox: flex-wrap

- Mittels der Eigenschaft **flex-wrap** können Sie festlegen ob ein **Zeilenumbruch durchgeführt** werden soll, wenn die Elemente nicht mehr in eine Zeile passen.

```
#flex-box {  
    display: flex;  
    flex-wrap: wrap;  
    gap: 20px;  
}
```

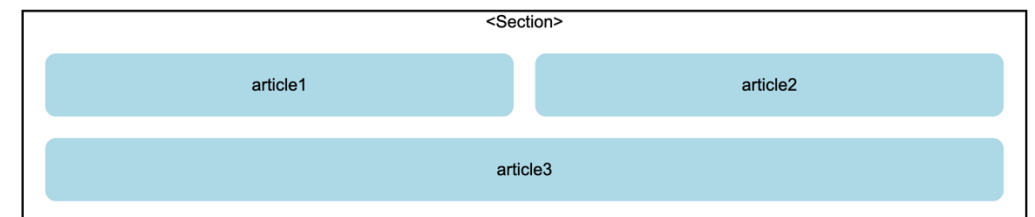
- flex-wrap: **wrap**; mit Zeilenumbruch
- flex-wrap: **nowrap**; ohne Zeilenumbruch (Default-Wert), Elemente **verschwinden** beim **Verkleinern** der Box.
- **gap: 20px**; Spalte zwischen den Child-Elementen, horizontal und vertikal



- Um ein **adaptives** und **fluides** Design zu erhalten, weist man den Child-Elementen mittels **min-width** eine **minimale** Breite zu.
- **flex: 1**; stellt sicher, dass alle Elemente einer Reihe gleich breit sind und die volle Breite des Elternelementes verwenden.

```
flex: <flex-grow> <flex-shrink> <flex-basis>;
```

```
#article1, #article2, #article3 {  
    min-width: 15em;    /* minimale Breite */  
    flex: 1 1 0%;      /* = flex: 1; */  
}
```



Flexbox: flexdirection

- ❑ **flex-direction** ermöglicht die **Richtung der Anordnung** der Child-Elemente vorzugeben
 - `flex-direction: row;` von links nach rechts (**default**)
 - `flex-direction: column;` von oben nach unten
 - `flex-direction: row-reverse;` von rechts nach links
 - `flex-direction: column-reverse;` von unten nach oben

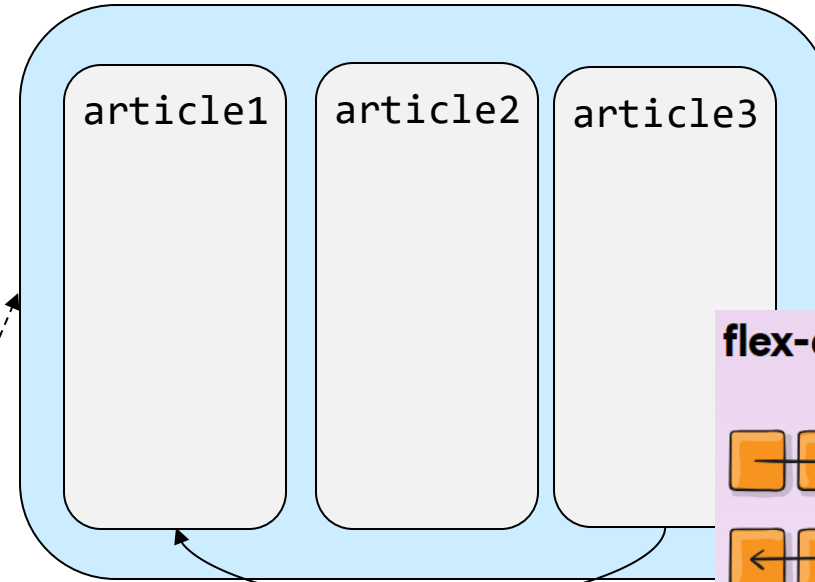
```
<section id="flex-box">  
  <article id="article1">...</article>  
  <article id="article2">...</article>  
  <article id="article3">...</article>  
</section>
```

```
#flex-box{  
  display: flex;  
  flex-wrap: wrap;  
  flex-direction: column;  
  gap: 20px;  
}
```

flex-direction: row;

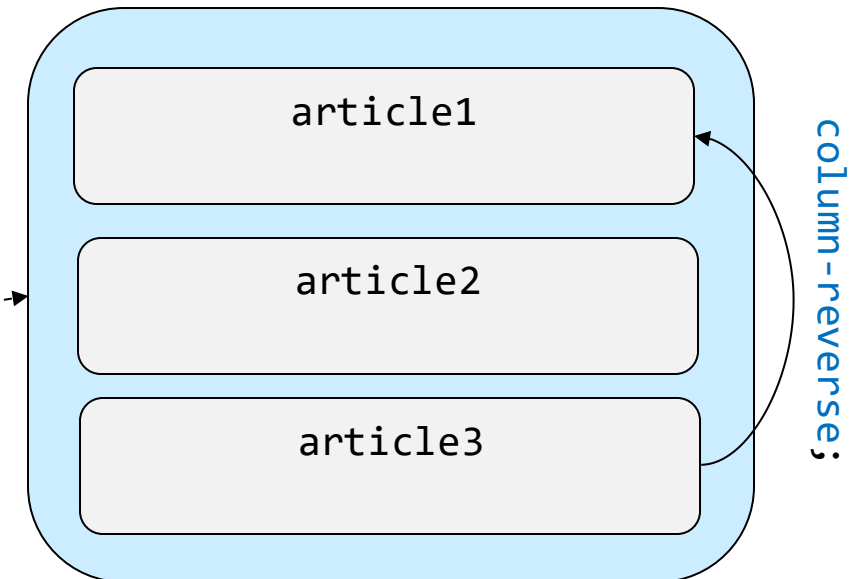
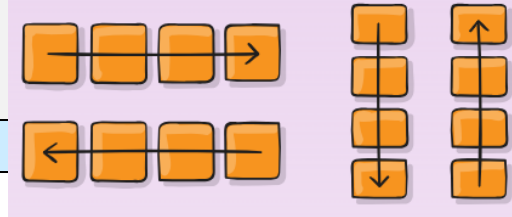
flex-direction: column;

`flex-direction: row;`



`row-reverse;`

flex-direction



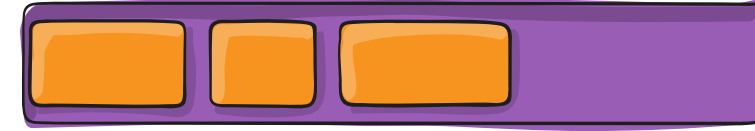
`column-reverse;`

Flexbox: Verteilung der Elemente

- ❑ Mit `justify-content` bestimmen Sie die Verteilung der Elemente entlang der `flex-direction`.
- ❑ Mögliche Werte für die `flex-direction` sind
 - `justify-content: flex-start`
linksbündig in der Flexbox (Standardeinstellung)
 - `justify-content: flex-end`
rechtsbündig in der Flexbox
 - `justify-content: center`
zentriert in der Flexbox
 - `justify-content: space-between`
gleichverteilt, Randelemente schließen ab
 - `justify-content: space-around`
gleichverteilt, Randabstände halbiert
 - `justify-content: space-evenly`
gleichverteilt, alle Abstände gleich

```
#flex-box{  
  display: flex;  
  flex-wrap: wrap;  
  flex-direction: row;  
  justify-content: space-evenly;  
  gap: 20px;  
}
```

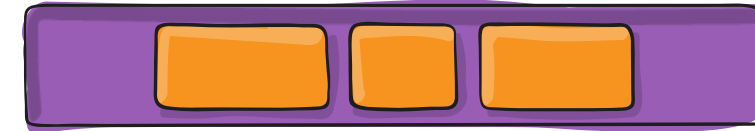
flex-start



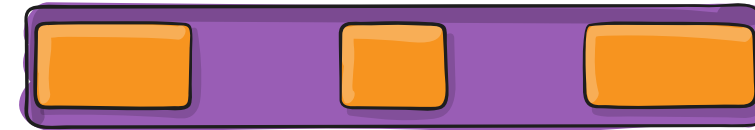
flex-end



center



space-between



space-around



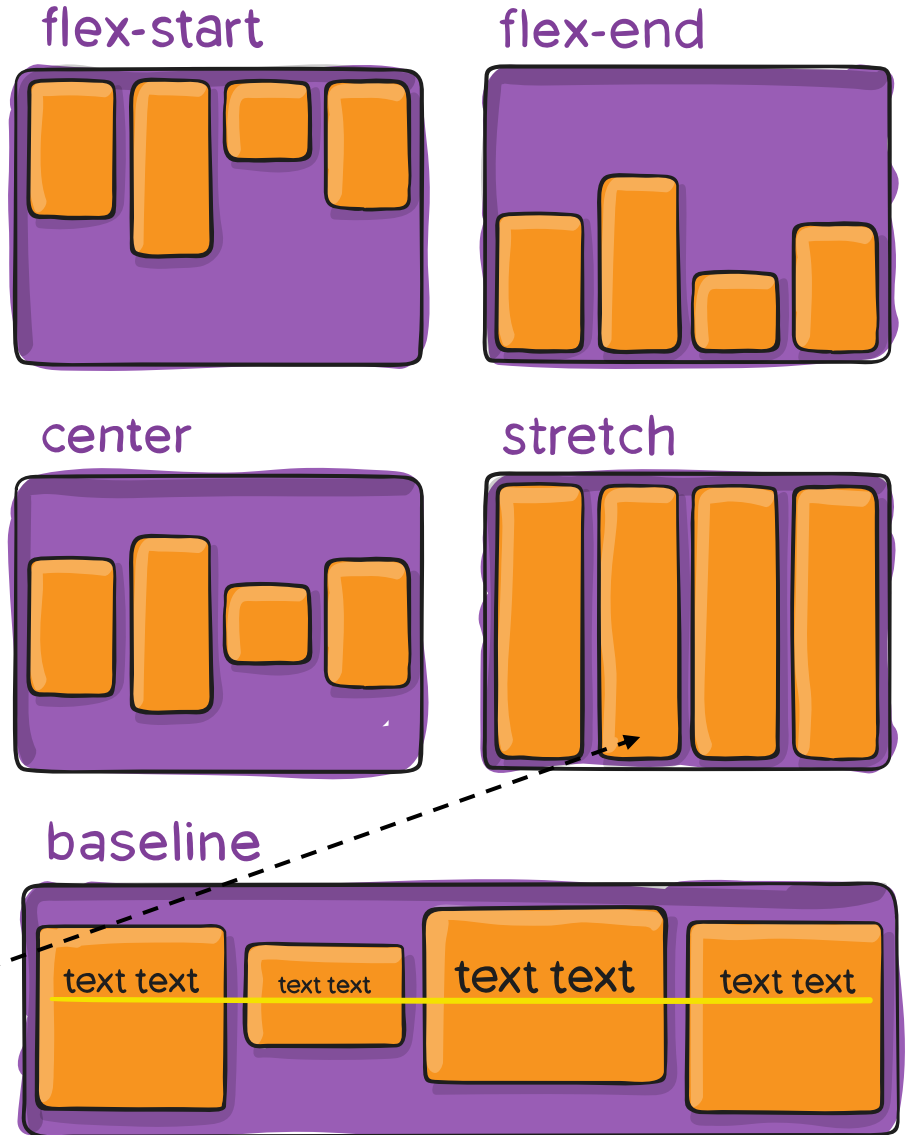
space-evenly



Flexbox: Ausrichten der Items

- ❑ Mit `align-items` können Sie die Alignierung in Bezug auf der zur `flex-direction` senkrechten Achse bestimmen.
- ❑ Mögliche Werte für die `align-items` sind
 - `align-items: stretch;` Items werden gestreckt (Standardeinstellung)
 - `align-items: flex-start;` bündig mit Start-Achse
 - `align-items: flex-end;` bündig mit End-Achse
 - `align-items: center;` zentriert zwischen beiden Achsen
 - `align-items: baseline;` Items haben alle gleiche Baseline

```
#flex-box{  
  display: flex;  
  flex-wrap: wrap;  
  flex-direction: row;  
  justify-content: space-evenly;  
  align-items: stretch;  
}
```



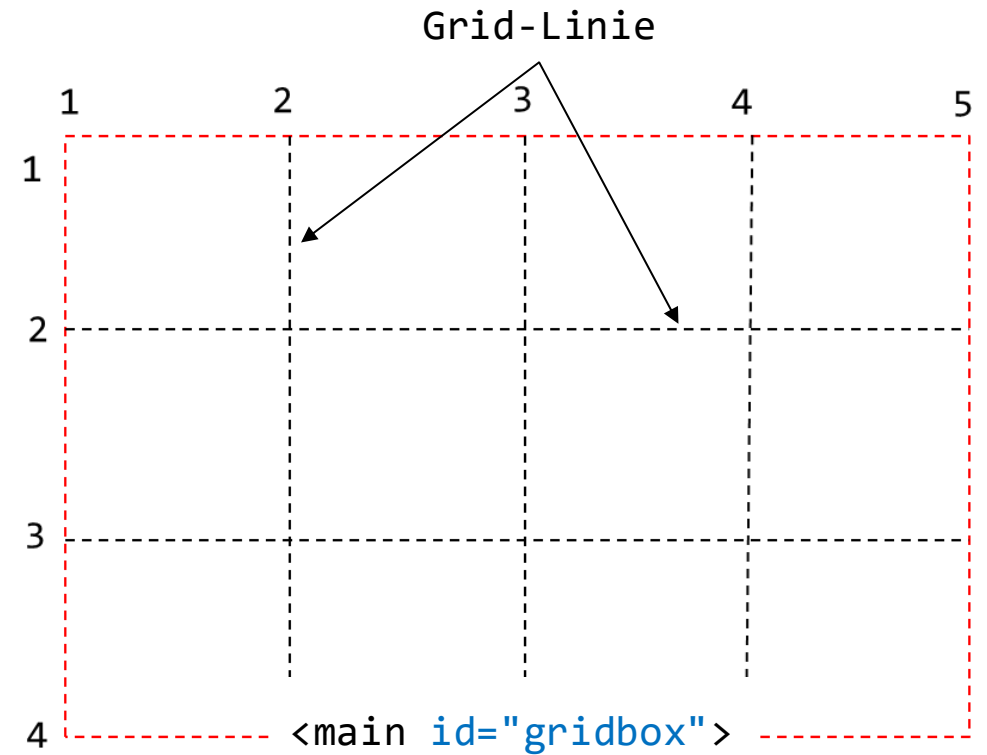
Gridbox

- Eine **Gridbox** wird mittels analog zur Flexbox mit der `display`-Eigenschaft definiert.

```
#gridbox{  
    display: grid; (oder inline-grid)  
}
```

- Das CSS Grid-Layout erstellt innerhalb eines **Elementes** (z.B: `<div>`) ein Raster (**Grid**) aus **gedachten Linien** in **vertikaler** und **horizontaler** Richtung
- Mit `grid-template-columns` und `grid-template-rows` unterteilen Sie das Grid in **Spalten** und **Zeilen**.
- **Einheiten**: relative Größen (%) und `fr` – fraction) oder absolute Größen (px, ...)

```
grid-template-columns: 25% 25% 25% 25%;  
grid-template-rows: 1fr 1fr 1fr;
```



```
#gridbox{  
    display: grid;  
    grid-template-columns: 25% 25% 25% 25%;  
    grid-template-rows: 1fr 1fr 1fr;  
}
```

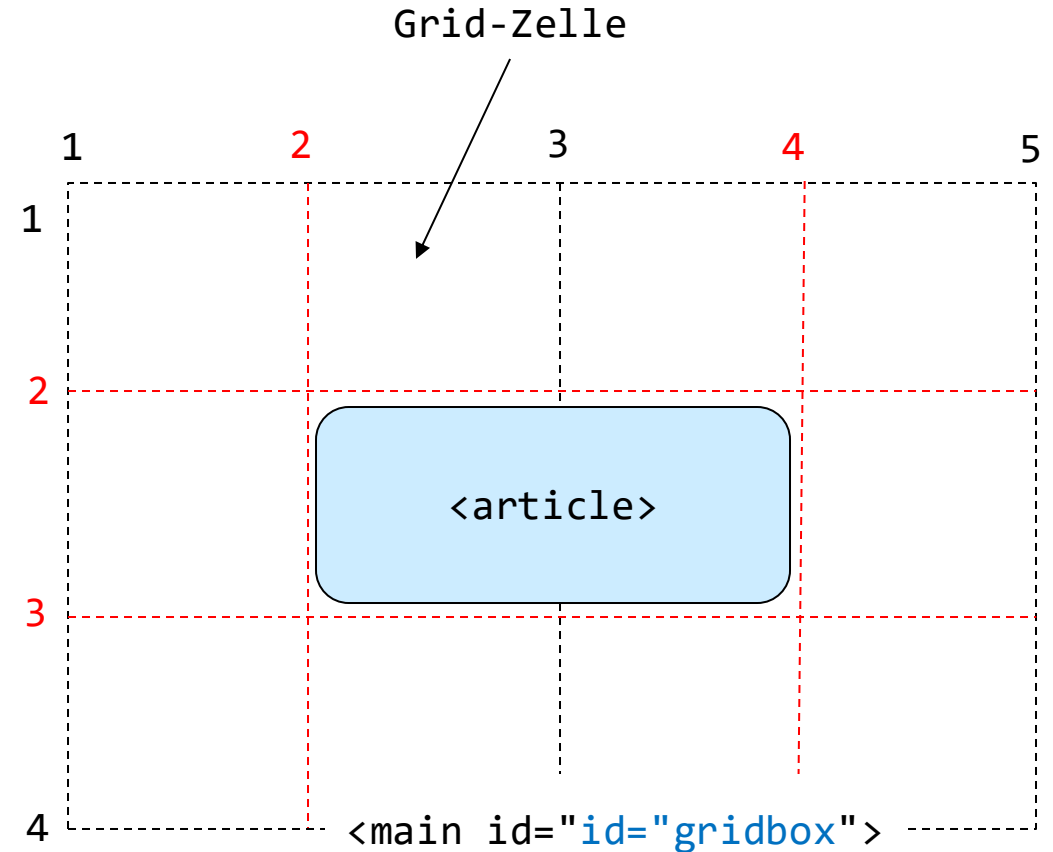
Grid-Layout: Platzierung der Elemente

- Elemente im Grid werden anschliessend über die Nummern der horizontalen und vertikalen Linien platziert.

```
#article1{  
    grid-column-start: 2;  
    grid-column-end: 4;  
    grid-row-start: 2;  
    grid-row-end: 3;  
}
```

- Kurzschreibweise:

```
grid-column: 2 / 4; /* von column 2 bis 4 */  
grid-row: 2;      /* nur row 2          */
```



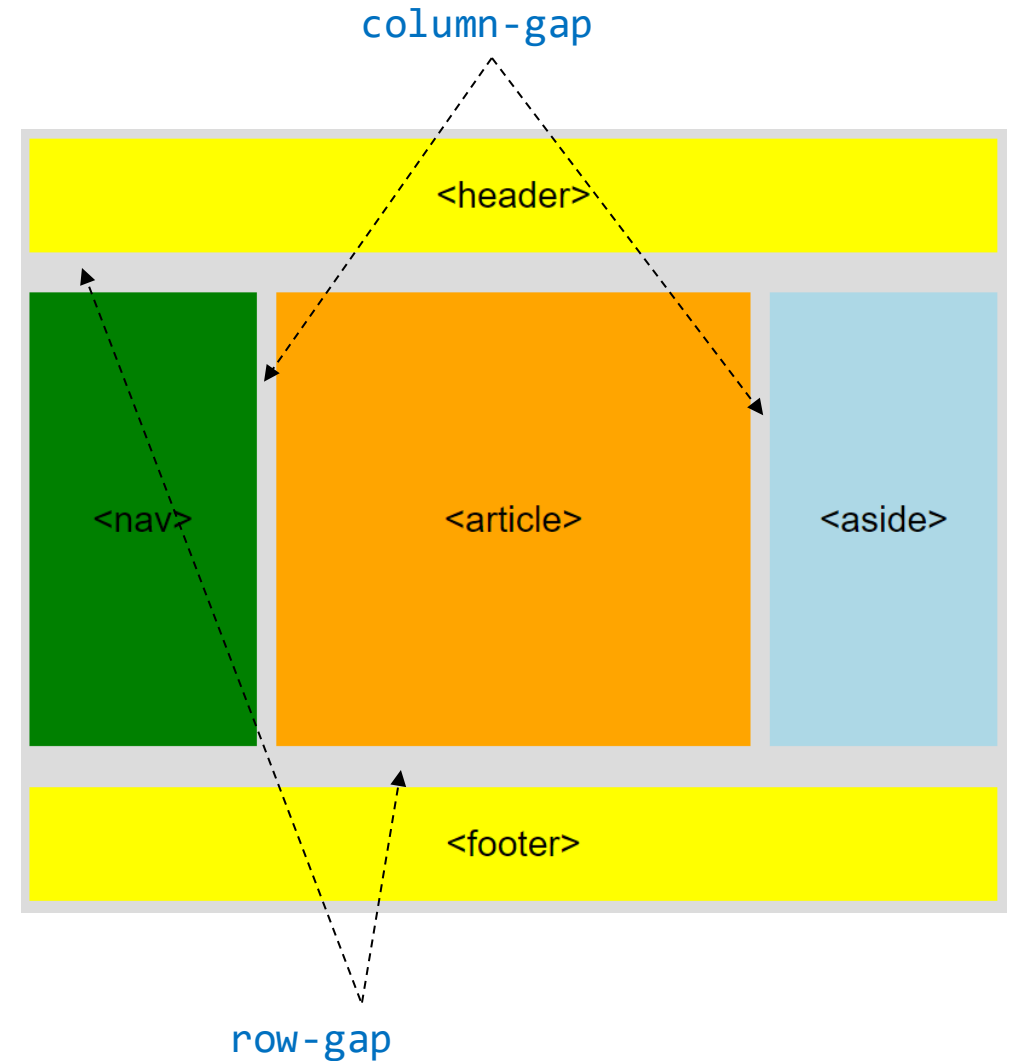
Grid-Gap: Breite der Linien

- Die Breite der Linien und damit der Abstand zwischen den Elementen eines Grids kann horizontal mittels `column-gap` und vertikal mittels `row-gap` bestimmt werden.

```
#gridbox{  
  display: grid;  
  grid-template-columns: 25% 25% 25% 25%;  
  grid-template-rows: 1fr 1fr 1fr;  
  column-gap: 10pt;  
  row-gap: 20pt;  
}
```

- Kurzschreibweise (siehe Flexbox)

```
gap: <grid-row-gap> <grid-column-gap>;
```



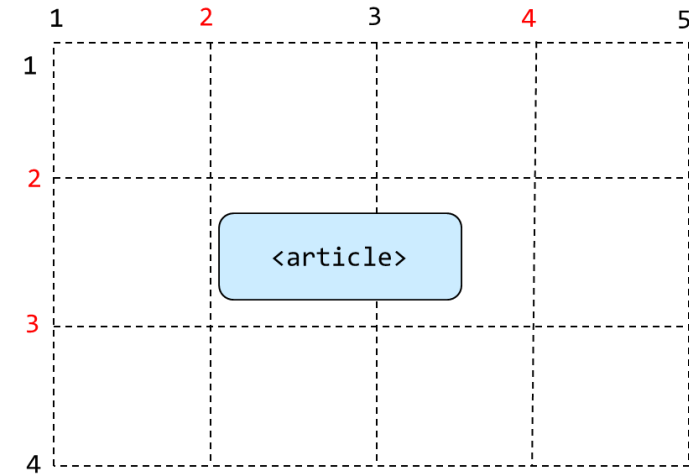
Ausrichtung der Grid-Elemente

- Grid-Elemente lassen sich **innerhalb** ihrer **Grid-Area** entlang der **Zeilen-Achse** mittels **justify-self** und entlang der **Spalten-Achse** mittels **align-self** ausrichten.

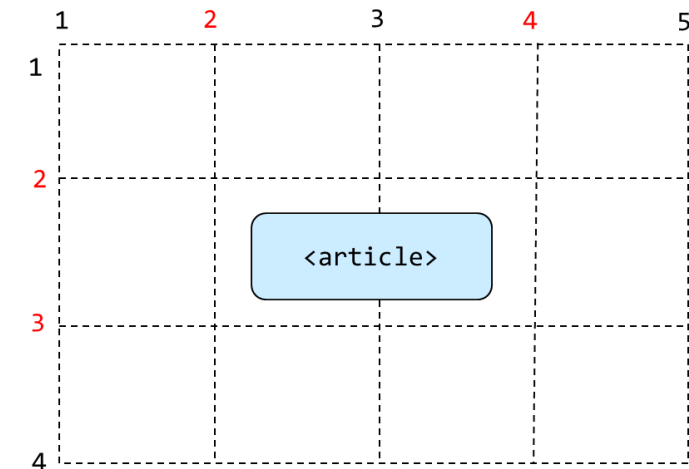
```
justify-self: start | end | center | stretch;  
align-self:  start | end | center | stretch;
```

- start** – richtet das Rasterelement bündig mit der Startkante der Zelle aus
- end** – richtet das Rasterelement bündig mit der Endkante der Zelle aus
- center** – richtet das Rasterelement in der Mitte der Zelle aus
- stretch** – füllt die gesamte Breite der Zelle aus (dies ist die **Standardeinstellung**)

- #article1 { **justify-self: start;** }



- #article1 { **align-self: center;** }



Grid-Area

- ❑ Die Gridbox ermöglicht, das sich ein Grid-Element über **mehrere Grid-Zellen** erstrecken kann.
- ❑ Mehrere als Rechteck zusammengefasste **Grid-Zellen** werden auch als **Grid-Area** bezeichnet.
- ❑ Eine **Grid-Area** kann über die Angabe der Reihen- und Zeilen der Gridbox oder nur durch die **Angabe eines Namens** definiert werden:

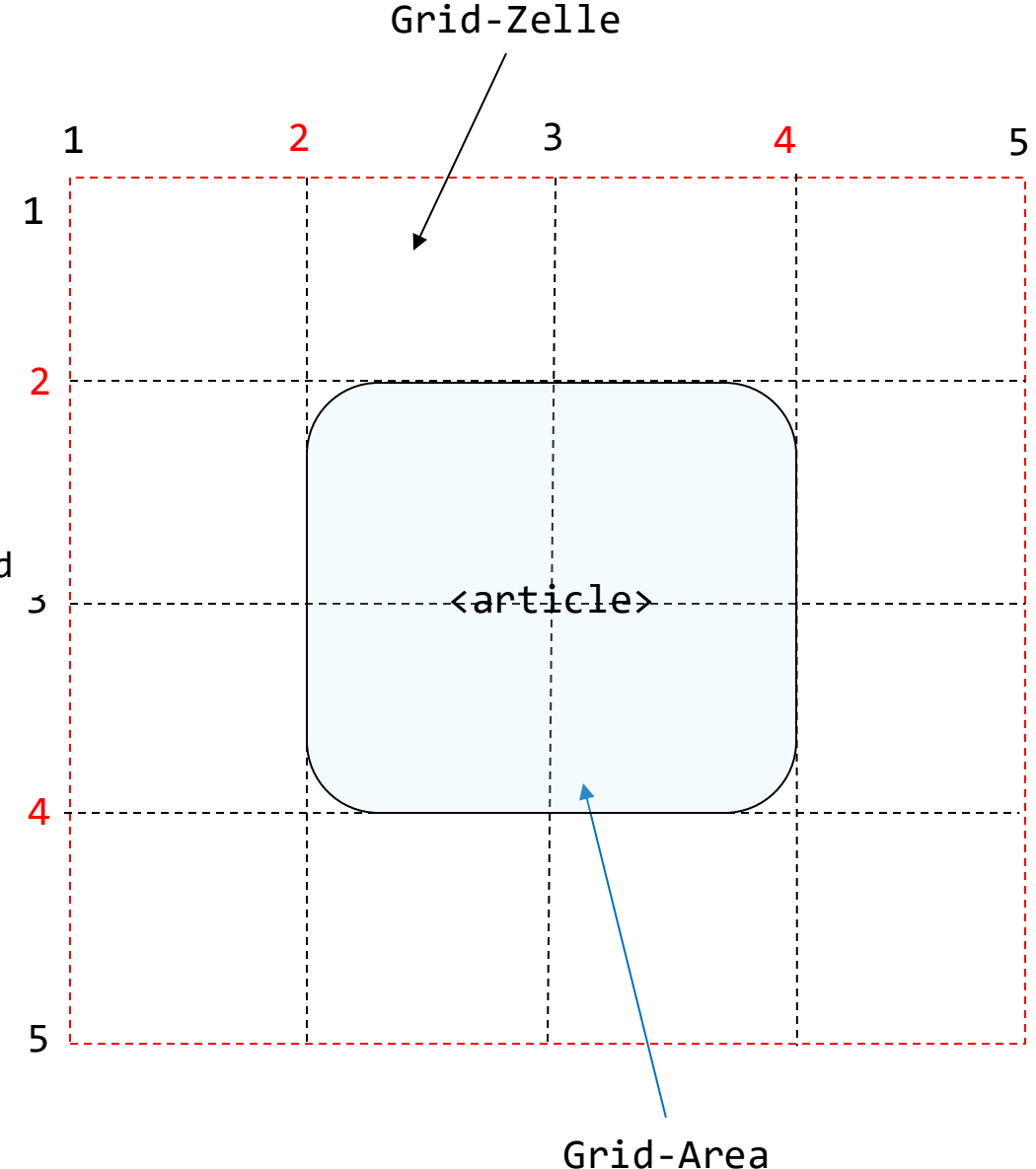
grid-row-start / grid-column-start / grid-row-end / grid-column-end

```
#article1 { grid-area: 2 / 2 / 4 / 4; }
```

oder

```
#article1 { grid-area: ga__a; }
```

- ❑ Bei der Verwendung von Namen werden diese mittels einer der Eigenschaft **grid-template-areas** bestimmten Zellen in der Grid-Box zugeordnet.

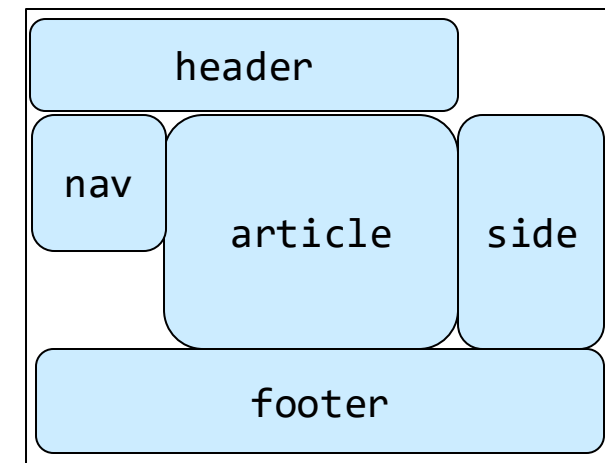


Grid-Template-Area

- ❑ Über die Eigenschaft `grid-template-areas` lässt sich in Verbindung mit dem Namen einer `grid-area` ein sehr flexibles Grid aufbauen.
- ❑ Ein Punkt "." bedeutet eine leere Zelle.

```
#article1 {  
  grid-area: ga__a;  
}  
#nav1 {  
  grid-area: ga__n;  
}  
#aside1 {  
  grid-area: ga__s;  
}  
#footer1 {  
  grid-area: ga__f;  
}  
#header1 {  
  grid-area: ga__h;  
}
```

```
#gridbox {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr 1fr;  
  grid-template-rows: 1fr 1fr 1fr 1fr;  
  grid-template-areas:  
    "ga__h  ga__h  ga__h  ."   
    "ga__n  ga__a  ga__a  ga__s"  
    ".      ga__a  ga__a  ga__s "  
    "ga__f  ga__f  ga__f  ga__f";  
}
```



Media Queries

- ❑ Mittels **CSS Media Queries** und einer **Gridbox** ist es möglich einer Webseite **unterschiedliche Layout** zuzuordnen.
- ❑ So kann in Abhängigkeit von der **Viewport-Breite** auf einer Webseite
 - eine unterschiedliche Spaltenanzahl,
 - Text unterschiedlicher Größe,
 - Bilder unterschiedlicher Auflösung, verwendet werden.
- ❑ Ein **Media Query** beginnt mit einer **verschachtelten (en: nested) AT-Rule @media**

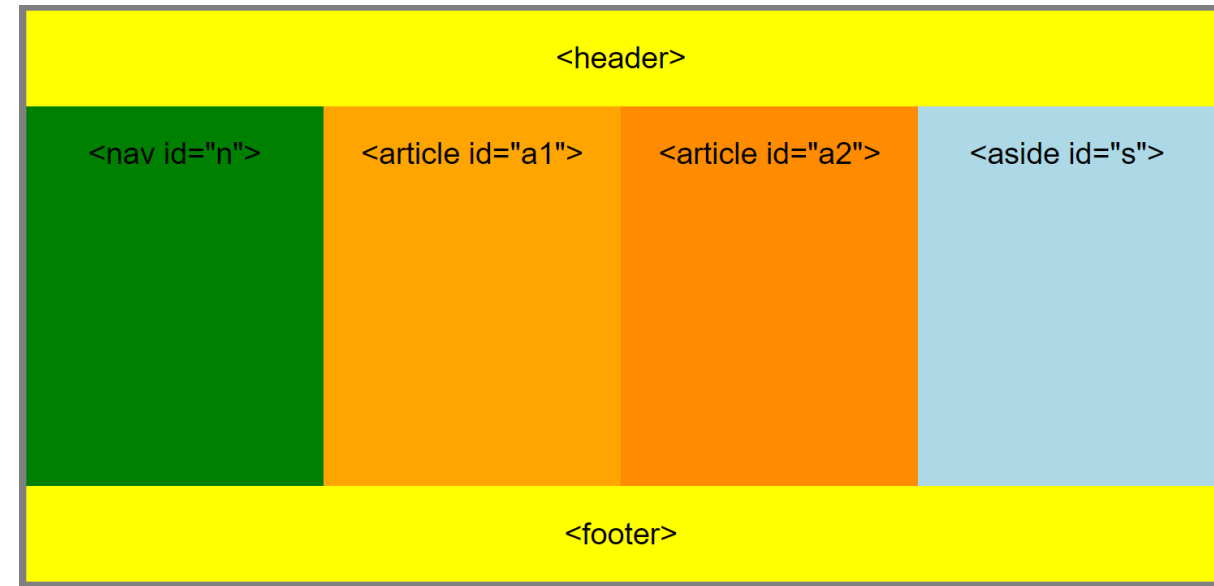
```
@media type1, type2 and (expr1) and (expr2) {  
    /* CSS - Eigenschaften */  
}
```

- ❑ Anschliessend kommen ein oder mehrere **Medientypen type1, type2, ...** durch Komma getrennt, auf die die Abfrage angewandt wird
screen, print, projection, tv, ...
- ❑ Abschließend kommt dann die **Medieneigenschaft**, die mit der **Query** geprüft wird, beispielsweise
 - **min-/max-width**: minimale, maximale Breite des Viewports
 - **min-/max-height**: minimale, maximale Höhe des Viewports
 - **orientation**: portrait oder landscape
- ❑ Der Query folgt dann ein von geschweiften Klammern **{...}** umschlossener Block an CSS-Anweisungen.

Beispiel: Media-Query & Flexbox

```
@media screen and (min-width: 800px) {  
  #gridbox {  
    display: grid;  
    grid-template-columns: 1fr 1fr 1fr 1fr;  
    grid-template-rows: auto;  
    grid-template-areas:  
      "header header header header"  
      "ga__n ga__a1 ga__a2 ga__s"  
      "footer footer footer footer";  
  }  
}  
  
header {  
  grid-area: header;  
}  
...
```

Für große Viewports (> 800px) wird ein 4-spaltiges Layout definiert:



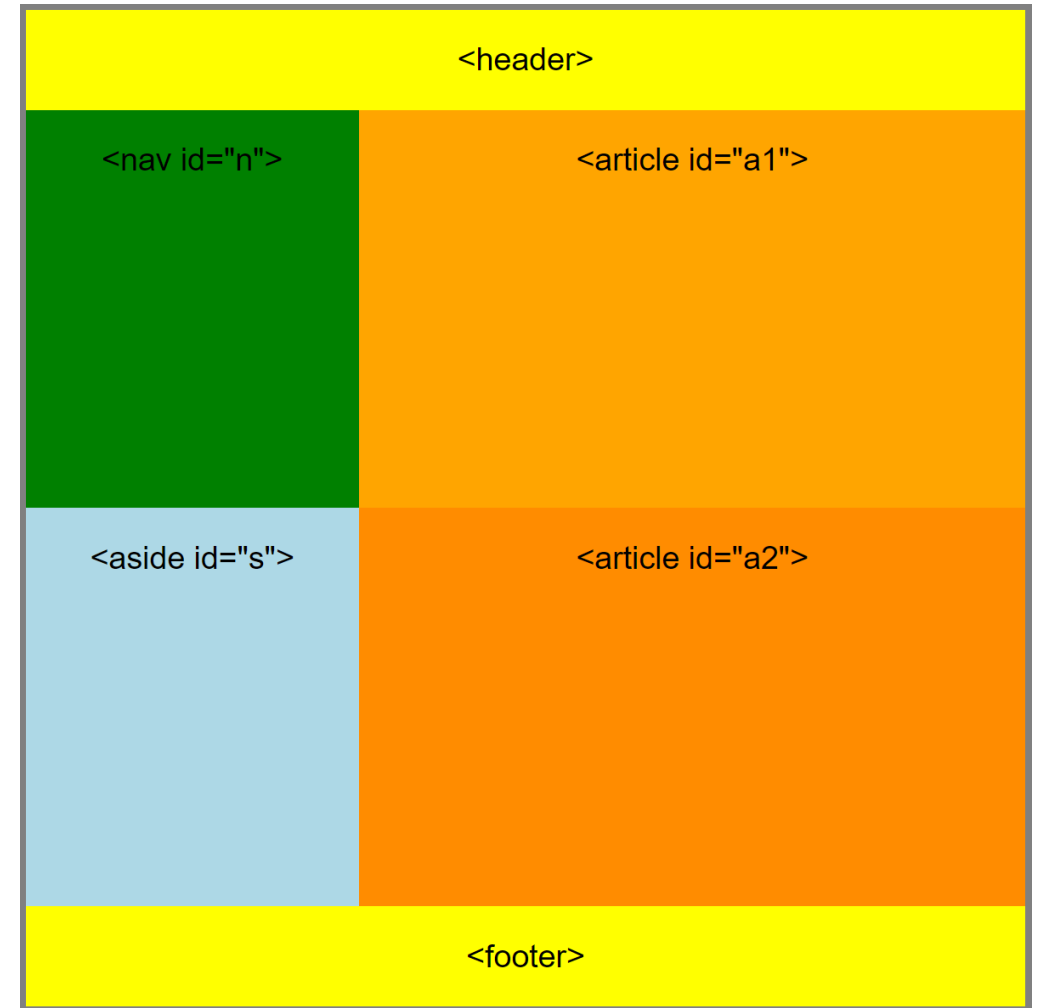
Beispiel: Media-Query & Flexbox

```
@media screen and (min-width: 481px) and (max-width: 799px) {
```

```
  #gridbox {  
    display: grid;  
    grid-template-columns: 1fr 1fr;  
    grid-template-rows: auto;  
    grid-template-areas:  
      "header header"  
      " ga__n ga__a1"  
      " ga__s ga__a2"  
      "footer footer";  
  }
```

```
}
```

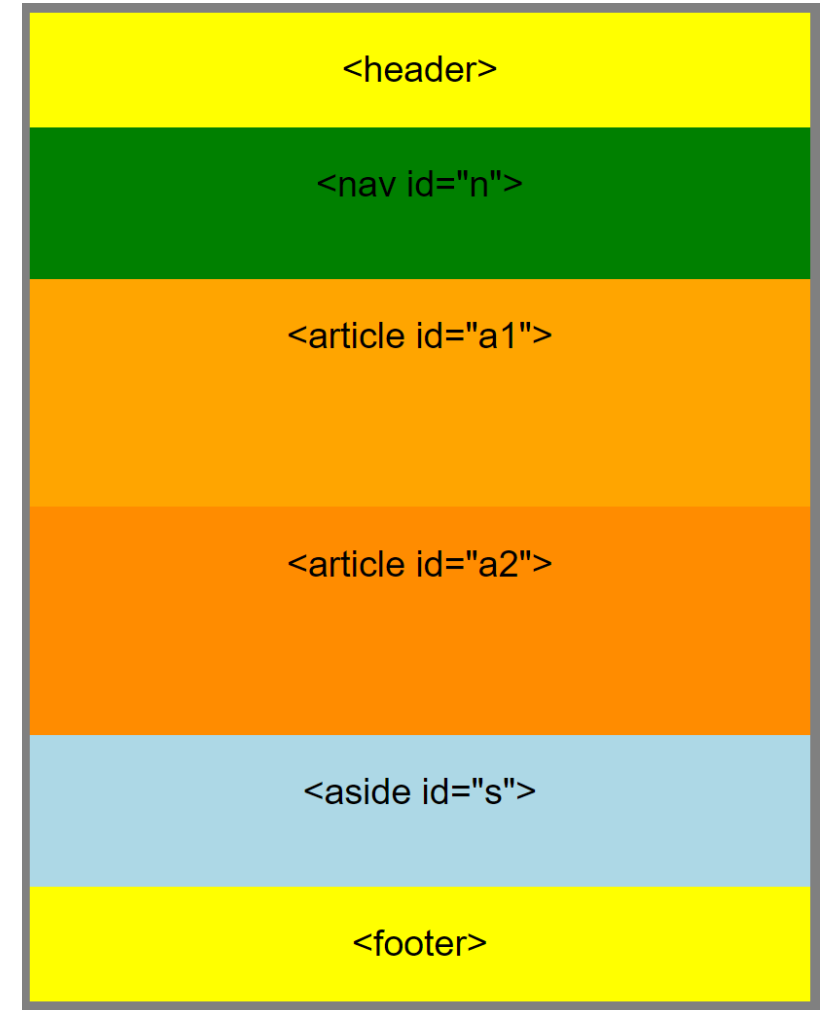
Für mittelgroße Viewports (481px - 799px) wird ein 2-spaltiges Layout definiert:



Beispiel: Media-Query & Flexbox

```
@media screen and (max-width: 480px){  
  #gridbox {  
    display: grid;  
    grid-template-columns: 1fr;  
    grid-template-rows: auto;  
    grid-template-areas:  
      "header"  
      "ga__n"  
      "ga__a1"  
      "ga__a2"  
      "ga__s"  
      "footer";  
  }  
}
```

Für kleine Viewports (<480px) wird ein 1-spaltiges Layout definiert:



Media Queries und Responsive Design

Ein **responsives Design** bezeichnet das Design einer Website, deren **Layout** (fluid, adaptiv) und **Inhalt** (Bilder, Text), sich automatisch an die Bildschirmgröße und Bildschirmauflösung des Geräts anpasst.

- Das **Layout** läßt sich flexibel über die **media-Queries** in Verbindung mit **Grid-Boxen** innerhalb einer CSS-Datei steuern.
- Um die **Textgröße** anzupassen kann die die clamp()-Funktion in CSS verwendet werden.

```
clamp(min, preferred, max)
```

min → Der kleinstmögliche Wert.

preferred → Der ideale Wert (normalerweise eine auf dem Ansichtsfenster basierende Einheit wie vw).

max → Der größtmögliche Wert.

- Das folgende Beispiel zeigt die responsive Definition der Schriftgröße des <h1>-Elementes:
 - Die Schriftgröße wird **nicht kleiner** als **1,5 rem**.
 - Die Schriftgröße wird **dynamisch** mit **5vw** (5 % der Ansichtsfensterbreite) **skaliert**.
 - Die Schriftgröße wird **nicht größer** als **3 rem**.

```
h1 {  
    font-size: clamp(1.5rem, 5vw, 3rem);  
}
```

Media Queries und Responsive Design:Bilder

- ❑ Bilder für unterschiedliche Bildschirmgrößen lassen sich über das `<picture>`-Element im [HTML-Dokument](#) angeben.

Bildaauflösung in Pixel als Breite (w), als Hinweis für den Browser, der die Beste Auflösung für die Darstellung wählt.

Bildschirmgröße

```
<picture>
  <source media="(min-width: 800px)" srcset="large.jpg 800w, largemedium.jpg 1200w largehigh.jpg 1600w">
  <source media="(max-width: 799px)" srcset="medium.jpg 600w mediumhigh.jpg 700w ">
  <source media="(max-width: 480px)" srcset="small.jpg 340w, smallhigh.jpg 480w">
  
</picture>
```

Media Queries und Responsive Design: Bilder

- Alternativ können Sie in ihrer StyleSheet-Datei mittels media-Queries und der Eigenschaft `background-image`, Bilder unterschiedlicher Auflösung einbinden.

```
/* Medium screens (up to 799px) */
@media screen and (max-width: 799px) {
    .responsive-image {
        background-image: url('medium.jpg');
    }
}

/* Small screens (up to 480px) */
@media screen and (max-width: 480px) {
    .responsive-image {
        background-image: url('small.jpg');
    }
}
```

Aufgabe 7: Testing von Webseiten

- (1) Bitte schauen Sie sich das nebenstehende YouTube Video “User Testing: Why & How von Jakob Nielsen”.
- (2) Warum sollten Sie nach Nielson ihre Webseiten von ihren Kunden (ihren Benutzern) testen lassen?
- (3) Wieviel Benutzer benötigen Sie für einen aussagekräftigen Test?
- (4) Wie sollten Sie das User Testing durchführen?



<https://youtu.be/v8JJrDvQDF4>

Aufgabe 8: CSS

- (1) Was versteht man unter der **BEM**-Konvention? Erstellen Sie die folgenden Beispiele mit BEM:
 - a. Section-Element in Main-Element soll rote Schrift erhalten.
 - b. Button mit ``-Element und großer Schrift
- (2) Erklären Sie die folgenden CSS-Selektoren
`h1[title]`, `span[hello="Cleveland"][goodbye="Columbus"]`, `a[href^="http"]`
- (3) Erstellen Sie ein Hintergrundimage für eine einfache Webseite mit einem **linearen Farbverlauf** von hellem Grau (`rgb(190,190,190)`) zu hellem Blau (`rgb(195,215,250)`). Der **Farbgradient** soll von links nach verlaufen.
- (4) Erstellen Sie einen Button der beim **Hovern** ("schweben über") mit ihrem Mauszeiger einen **Tooltip** (Kurzinformationen über Funktion des Buttons) anzeigt. Der Tooltip verschwindet nach 4s.
- (5) Erstellen Sie für eine Webseite ein sogenanntes **Holy-Gray-Layout ("heiliger Gral")** mit Hilfe einer Grid-Box. Die Webseite besitze einen `<header>`, einen `<footer>`, eine `<nav>`-, eine `<aside>`-Bar sowie einen `<article>`.