

# HTML UND CSS – ERWEITERUNGEN

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Semantische Bereichselemente .....</b>                | <b>1</b>  |
| 1.1      | Bereiche und ARIA Landmarks .....                        | 2         |
| <b>2</b> | <b>Übersicht: Selektoren .....</b>                       | <b>3</b>  |
| <b>3</b> | <b>Boxmodell .....</b>                                   | <b>8</b>  |
| 3.1      | box-sizing .....   | 8         |
| 3.2      | Bildgröße flexibel halten.....                           | 8         |
| 3.3      | Collapsing Margins .....                                 | 9         |
| 3.4      | Abgerundete Ecken .....                                  | 10        |
| 3.5      | Schatteneffekte.....                                     | 10        |
| <b>4</b> | <b>Layouts .....</b>                                     | <b>11</b> |
| 4.1      | Floats.....  | 12        |
| 4.1.1    | Grundprinzip.....  | 12        |
| 4.1.2    | Clearing von Floats .....                                | 14        |
| 4.1.3    | Zusammenfassung: normaler Fluss und Positionierung ..... | 17        |
| 4.1.4    | Containing Floats .....                                  | 20        |
| 4.1.5    | Der Micro Clearfix Hack .....                            | 24        |
| 4.1.6    | Exkurs: Negative Margins .....                           | 25        |
| 4.2      | Übersicht CSS Tabellen .....                             | 26        |
| 4.3      | Übersicht: Flexbox.....                                  | 27        |
| <b>5</b> | <b>Mediaqueries .....</b>                                | <b>28</b> |
| 5.1      | Syntax .....   | 28        |
| 5.2      | Übersicht Features.....                                  | 29        |
| <b>6</b> | <b>Übersicht: CSS Einheiten .....</b>                    | <b>30</b> |
| 6.1      | CSS Längeneinheiten .....                                | 30        |
| 6.2      | CSS Winkel .....   | 30        |
| 6.3      | CSS Farben - RGB .....                                   | 31        |
| 6.4      | CSS Farben - HSL.....                                    | 32        |
| 6.5      | Transparenz mit RGB/HSL .....                            | 32        |



## 1 Semantische Bereichselemente

|       | Bereichselemente   |            |
|-------|--|------------|
|       | inhaltliche Struktur, auch für Layoutzwecke nutzbar            | nur Layout |
| HTML4 | div  | div        |
| HTML5 | header<br>main<br>footer<br>nav<br>section<br>article<br>aside | div        |

**Gliedernder Inhalt (*sectioning content*):** nav, section, article, aside.

Im Sinne einer Gliederung, deren "Kapitelüberschrift" in ein Inhaltsverzeichnis aufgenommen würde. Durch Schachtelung gliedernder Bereichselemente wird eine hierarchische Gliederung erzeugt (Outline Algorithmus).

Jedem Bereich mit gliederndem Inhalt sollte eine Überschrift zugeordnet sein. Einen *Outliner* zum Testen der Gliederungsstruktur finden Sie bei <https://gsnedders.html5.org/outliner/> bzw. im AddOn 'Web Developer' des Firefox Browsers unter *Informationen - Dokumentkontur anzeigen*.

Sonstige (gruppierende) Bereiche: header, main, footer.

## 1.1 Bereiche und ARIA Landmarks

| Element | Wichtige <i>Landmarks</i> . Zuordnung z.B. <code>&lt;header role="banner"&gt;...</code>          |
|---------|--|
| header  | <u><i>banner</i></u> , <i>presentation</i>   |
| main    | <u><i>main</i></u> , <i>presentation</i>   |
| footer  | <u><i>contentinfo</i></u> , <i>presentation</i>  |
| nav     | <u><i>navigation</i></u> , <i>presentation</i>   |
| section | <u><i>region</i></u> , <i>presentation</i> ,...  |
| article | <u><i>article</i></u> , <i>main</i> , <i>application</i> , <i>document</i> , <i>presentation</i> |
| aside   | <u><i>complementary</i></u> , <i>note</i> , <i>search</i> , <i>presentation</i>                  |

Die Standard *Landmark* ist unterstrichen und sollte nur zu Kompatibilitätszwecken explizit zugeordnet werden.

Bedeutung einzelner Landmarks:

### *banner*

Inhalt bezieht sich eher auf den gesamten Webauftritt (*site-oriented*), als auf eine spezifische Seite.

### *main*

Hauptinhalt

### *contentinfo*

Ein größerer Bereich mit Informationen über das umschließende Dokument.

### *navigation*

Liste von Links zur Navigation innerhalb dieses Dokuments oder zu ähnlichen Dokumenten.

### *region*

Ein größerer Bereich, wichtig genug, um einen Eintrag in ein Inhaltsverzeichnis zu erhalten.

### *article*

Eigenständiger Bereich innerhalb des Dokuments oder Webauftritts

### *complementary*

Unterstützend ergänzender Inhalt auf hierarchisch gleichartigem Level wie *main*, mit eigenständigem Inhalt.

### *presentation*

Nur zu Präsentationszwecken; wird nicht auf die Accessibility API abgebildet.

### *application*

Bereich, die eine Webanwendung (*application*) enthält; ohne Dokumentcharakter.

### *document*

Bereich, der Dokumentcharakter besitzt, keine Applikation.

### *note*

Eingeschobener, beiläufiger Inhalt; nebensächlich; unterstützt inhaltlich *main*.

### *search*

Bereich, der eine Suchfunktion enthält.

## 2 Übersicht: Selektoren

*selektorgruppe*

*selektor* [, *selektor*]\*

*selektor*

*einfache\_Selektorfolge* [ *Kombinator* *einfache\_Selektorfolge* ]\*

*einfache\_Selektorfolge*

{ *Typ* | *Universal* } [ { *ID* | *Klasse* | *Attribut* | *Pseudo* | *Negation* } ]\*

### 1. Einfache Selektorfolge

#### 1.1 Typselektor

- Ein Elementname

#### 1.2 Universalselektor

- Platzhalter '\*' für alle Elementnamen

#### 1.3 Attributselektoren

- stehen in eckigen Klammern []

|                      |  |
|----------------------|--|
| [ <i>att</i> ]       | Attribut gesetzt? Wert ist egal.             |
| [ <i>att=val</i> ]   | Attribut mit Wert <i>val</i> (genau).        |
| [ <i>att^=val</i> ]  | Attribut beginnt mit <i>val</i> .            |
| [ <i>att\$=val</i> ] | Attribut endet auf <i>val</i> .              |
| [ <i>att*=val</i> ]  | Attribut enthält <i>val</i> .                |
| [ <i>att~=val</i> ]  | Attribut enthält <i>val</i> als ganzes Wort. |

#### 1.4 Klassenselektor

- beginnt mit '.'
- Werte des HTML Attributes *class*
- *div.value* entspricht *div[class~=value]*

#### 1.5 ID Selektor

- beginnt mit '#'
- Werte des HTML Attributes *id*

#### 1.6 Pseudoklassen

- beginnen mit ':'
- basieren auf
  - ⇒ Informationen außerhalb des Dokumentbaumes

- ⇒ Informationen, die nicht durch andere einfache Selektoren ausgedrückt werden können

#### 1.6.1 Dynamische Pseudoklassen

- basieren nicht auf
  - ⇒ Name, Attribut, Inhalt eines Elements
- sondern auf
  - ⇒ Objekten, die nicht aus den Dokumentbaum zu erschließen sind (daher 'Pseudo')

##### 1.6.1.1 Link Pseudoklassen

- `:link` nicht-besuchter Hyperlink
- `:visited` besuchter Hyperlink
- schließen einander aus

##### 1.6.1.2 Benutzeraktion Pseudoklassen

- `:hover` Mauszeiger über der Elementbox
- `:active` Vom Benutzer aktiviertes Element (vor dem Loslassen des Mauszeigers)
- `:focus` Element wird ausgewählt, nicht aktiviert; Tastaturfokus
- schließen sich nicht gegenseitig aus

#### 1.6.2 Target Pseudoklasse

- `:target` Element wird durch Hyperlink angesprungen (lokaler Verweis, Sprungziel)

#### 1.6.3 Language Pseudoklasse

- `:lang(C)` Element mit Sprachumgebung C (==> HTML lang="C")

#### 1.6.4 Elementstatus im Benutzerinterface ('Browser')

##### 1.6.4.1 `:enabled`, `:disabled`

- Status des Elements aktiviert - deaktiviert ("gegraut")
- Ist nicht durch die CSS Eigenschaften display und visibility beeinflusst

##### 1.6.4.2 `:checked`

- ausgewählte Checkboxes und Radiobuttons

#### 1.6.5 Strukturelle Pseudoklassen

- `:root`
- `:nth-child( $a+n+b$ )` das  $a*n+b$ -te Kindelement,  $a$  und  $b$  sind ganze Zahlen, Minuszeichen ist auch möglich;  
Sonderfälle `odd(2n+1)`, `even(2n+0)`; `0n+3` (3), `2n+0` (2n) und `1n+0` (n+0,n), `-n+6` (die ersten 6)
- `:nth-last-child( $a+n+b$ )`
- `:nth-of-type( $a+n+b$ )` das  $a*n+b$ -te Element eines Typs
- `:nth-last-of-type( $a+n+b$ )`
- `:first-child` wie `:nth-child(1)`

- `:last-child` wie `:nth-last-child(1)`
- `:first-of-type` wie `:nth-of-type(1)`
- `:last-of-type` wie `:nth-last-of-type(1)`
- `:only-child` wie `:first-child:last-child`
- `:only-of-type` wie `:first-of-type:last-of-type`
- `:empty` leere Elemente, Text- und CDATA-Knoten

#### 1.6.6 Negation Pseudoklasse

- `:not(simple_Selector)`
- ist nicht schachtelbar ~~`:not(:not(Selektor))`~~ verboten
- Kombination möglich `*:not(:link):not(:visited)` erlaubt

#### 1.7 Pseudoelemente

- beginnen mit `::`, rückwärtskompatibel mit `'`
- erzeugen Elemente, welche über diejenigen der HTML Sprache hinausgehen, z.B. der erste Buchstabe oder die erste Zeile.
- erzeugen Inhalt (Content), der im Quellcode des Dokuments nicht vorhanden ist (`::before`, `::after`)
- pro Selektor darf nur ein Pseudoelement am Ende der einfachen Selektorfolge
- `::first-line` erste Zeile des Block-Container Elementes, Text vor dem ersten Umbruch im Fenster
- `::first-letter` erster Buchstabe (nicht: Sonderzeichen) des Block-Container Elementes ( `display:block|list-item|table-cell|table-caption|inline-block` )
- `::before`, `::after` generierter Inhalt vor bzw. nach einem Element

#### 2. Kombinatoren `'>'` `' '` `'~'` `'+'`

Kombinieren einfache Selektorfolgen. Die zugeordneten Styles beziehen sich auf die letzte einfache Selektorfolge des kombinierten Selektors; z.B. `header > p { color: red }` => Style bezieht sich auf `p`.

##### Childkombinator `S1 > S2`

Das durch `S2` ausgewählte Element muss das durch `S1` ausgewählte Element als Elternelement haben.

##### Descendantkombinator (**Leerzeichen**) `S1 S2`

Das durch `S2` ausgewählte Element muss das durch `S1` ausgewählte Element als Vorfahrenelement haben.

##### (Following)sibling Kombinator `S1 ~ S2`

Das durch `S2` ausgewählte Element befindet sich hinter dem durch `S1` ausgewählte Element als Geschwisterelement. Anders ausgedrückt: alle durch `S2` ausgewählten Geschwisterelemente, die auf das durch `S1` ausgewählte Element folgen.

##### Adjacent (Following)sibling Kombinator `S1 + S2`

Das durch `S2` ausgewählte Element befindet sich *unmittelbar* hinter dem durch `S1` ausgewählte Element als Geschwisterelement.

## Die Kaskade

Auf ein Element können gleichzeitig mehrere CSS-Anweisungen aus zum Teil unterschiedlichen *Quellen* wirken, die sich manchmal auch auf die gleiche CSS-Eigenschaft beziehen. Die Kaskade errechnet für Regeln und Eigenschaften eine Gewichtung, anhand derer die tatsächlich für ein Element geltende Formate bestimmt werden.

### *Browser-Stylesheet*

Browser-Regeln sind im Browser "eingebaute" Stilvorlagen und wirken mit niedrigster Priorität.

### *User-Stylesheet*

Benutzer-Regeln können durch den Besucher der Webseite in den Browser eingebunden werden. Sie haben Vorrang vor Browser-Regeln.

### *Author-Stylesheet*

Autor-Regeln werden durch den Autor der Webseite eingebunden. Sie haben Vorrang vor Browser-Regeln und Benutzer-Regeln.

Ausnahme: Wenn ein Benutzer Style mit der !important Regel (siehe unten) verknüpft ist, setzt er sich auch gegen Autor-Regeln durch.

Bei gleicher Quelle der Stilvorlage bestimmt die Treffgenauigkeit des Selektors (die *Spezifität*), welcher Stil angewandt wird (z.B. bei zwei widersprüchlichen Autor-Regeln).

## Die Spezifität des Selektors

Die *Spezifität* des Selektors kann wie folgt ermittelt werden:

Jeder Selektor hat die Zähler A, B und C. Jedes Vorkommen eines Selektortyps erhöht einen der genannten Zähler.

| Typ               | erhöht Zähler           |
|-------------------|-------------------------|
| ID Selektor       | A                       |
| Attributselektor  | B                       |
| Klassenselektor   | B                       |
| Typselektor       | C                       |
| Pseudoelement     | C                       |
| Universalselektor | keine Wertung           |
| :not()            | nur Klammerinhalt zählt |

### Beispiele:

Gegeben sind folgende CSS-Styles:

|                   |                    |   |
|-------------------|--------------------|---|
| *                 | { color: aqua }    | /* A=0 + B=0 + C=0 -> Spezifität = 0 0 0 */ |
| li                | { color: blue }    | /* A=0 + B=0 + C=1 -> Spezifität = 0 0 1 */ |
| ul li             | { color: magenta } | /* A=0 + B=0 + C=2 -> Spezifität = 0 0 2 */ |
| ul ol li          | { color: maroon }  | /* A=0 + B=0 + C=3 -> Spezifität = 0 0 3 */ |
| #x34y             | { color: yellow }  | /* A=1 + B=0 + C=0 -> Spezifität = 1 0 0 */ |
| li+li[class~=red] | { color: red }     | /* A=0 + B=1 + C=2 -> Spezifität = 0 1 2 */ |
| li.red.level      | { color: green }   | /* A=0 + B=2 + C=1 -> Spezifität = 0 2 1 */ |
| ul ol li.red      | { color: lime }    | /* A=0 + B=1 + C=3 -> Spezifität = 0 1 3 */ |



und folgendes HTML-Fragment:

```
<ul>
  <li>
    <p>Nachtisch</p>
    <ol>
      <li>Eis</li>
      <li class="red level" id="x34y">Schokolade</li>
    </ol>
  </li>
</ul>
```

Die Zähler der Spezifität sind voneinander unabhängig, d.h. o 11 hat weniger Gewicht als o 1 o.

Alle Selektoren treffen auf das Element `li` mit Inhalt "Schokolade" zu! In welcher Schriftfarbe wird denn nun der Inhalt dargestellt?

*Zum Nachtisch gibt es gelbe Schokolade! - Aber, wie sehen das Eis und der Nachtisch aus?*

Hinweis:

Das HTML-Attribut `style` wird noch stärker als ein `ID`-Selektor gewichtet.

## Reihenfolge und !important

Bei gleicher Quelle und gleicher Spezifität des Selektors entscheidet die Reihenfolge, in der die Regeln im Dokument deklariert sind: der Letzte gewinnt. Eine Ausnahme bildet die **Wichtigkeit** oder *!important* Regel (das Schlüsselwort *!important* muss unmittelbar hinter dem deklarierten Style stehen).

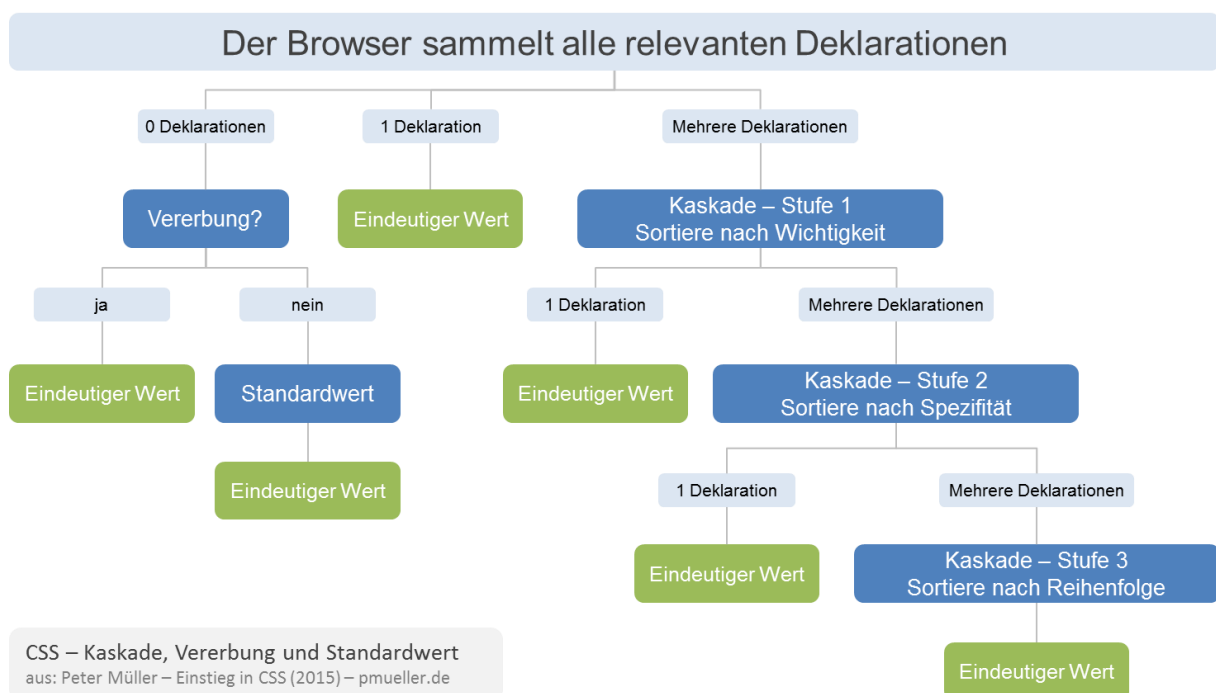
```
p { color:black!important }
p { color:yellow}
p.wichtig { color:red }
```

...

```
<p>Hello</p>
<p class="wichtig">Hello again!</p>
```

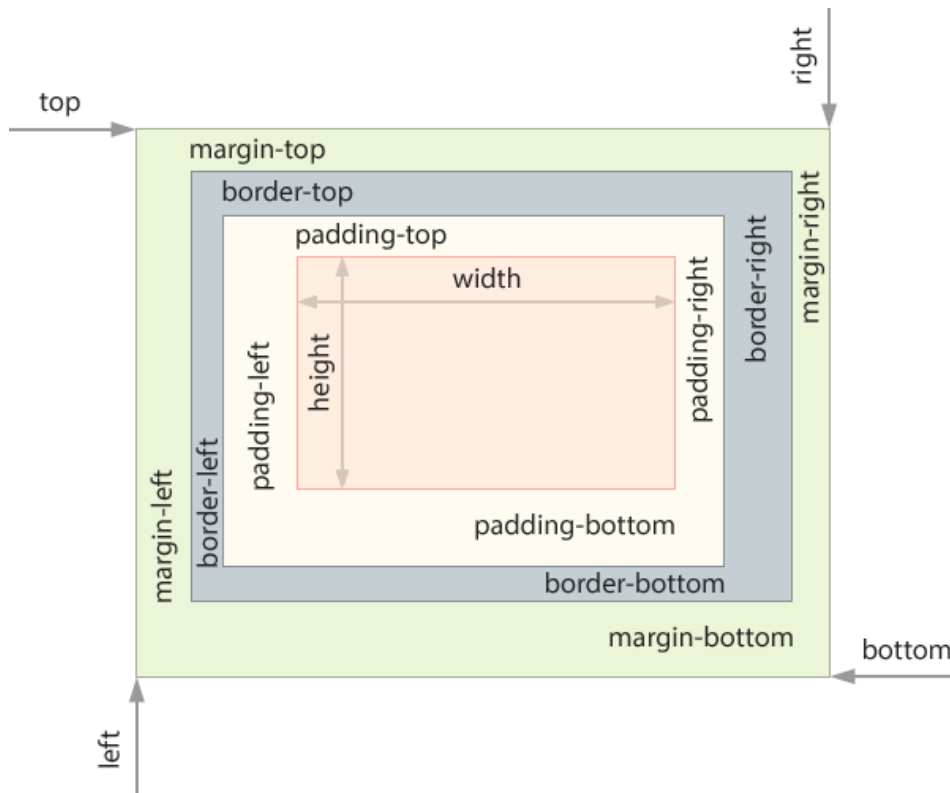
=> Beide Absätze sind in schwarzer Schriftfarbe

Unter Vernachlässigung eventuell vorhandener Benutzer-Regeln gilt folgende Kaskade:



### 3 Boxmodell

Klassisches Boxmodell (Skizze aus <https://wiki.selfhtml.org/wiki/CSS/Box-Modell>)



#### 3.1 box-sizing

Zuordnung von `width` und `height` entspricht dem Modell für die Eigenschaft `box-sizing: content-box`

Für `box-sizing: border-box` enthält `width` und `height`:

1. Inhaltsbereich (content area) plus
2. Paddingbereich (padding area) plus
3. Rahmenbereich (border area)

Die Eigenschaft `box-sizing` wird nicht vererbt und lässt sich wie folgt für *alle* Elemente des Dokuments setzen:

```
*, *:before, *:after { box-sizing: border-box }
```

#### 3.2 Bildgröße flexibel halten

Für flexible Layouts wird die Bildgröße im `<img>` Tag weggelassen. Stattdessen kann die Bildgröße automatisch auf die Breite des Elternelements reduziert werden mit:

```
img {  
    max-width: 100%;  
    height: auto;  
}
```

### 3.3 Collapsing Margins

1. Benachbarte Container haben einen gemeinsamen vertikalen Außenrand, dessen Betrag dem größeren der beiden Werte für `margin` entspricht.

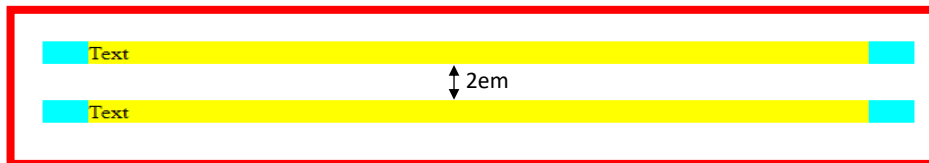
Dazu ein *Beispiel*: Einer Überschrift `h1` folgt ein Absatz `p`. `h1` hat die Eigenschaft `margin-bottom: 2em`, `p` hat die Eigenschaft `margin-top: 1em`. Der Abstand zwischen Beiden ist nun nicht etwa  $2em + 1em = 3em$ , sondern  $2em$ .

2. Ineinander geschachtelte Container haben ebenfalls einen gemeinsamen vertikalen Außenrand, dessen Betrag dem größeren der beiden Werte für `margin` entspricht. Dieser Außenrand zählt dann als Außenrand des umgebenden Containers.

Dazu ein weiteres *Beispiel*: Ein Absatz `p` ist in ein Element `section` eingeschachtelt. `p` hat die Eigenschaft `margin: 2em`, `section` hat die Eigenschaft `margin: 1em`. Der Abstand zum nächsten Element nach `section` ist nun nicht etwa  $2em + 1em = 3em$ , sondern  $2em$ . Hinzu kommt, dass dieser gemeinsame Vertikalabstand dem Element `section` zugesprochen wird, so dass `p` nun keinen Vertikalabstand mehr hat, `section` jedoch  $2em$ .

```
<section style="margin: 1em; background-color: aqua;">
  <p style="margin: 2em; background-color: yellow;">Text</p>
</section>
<section .... (Es folgt ein weiteres, gleichartig aufgebautes Element section)>
```

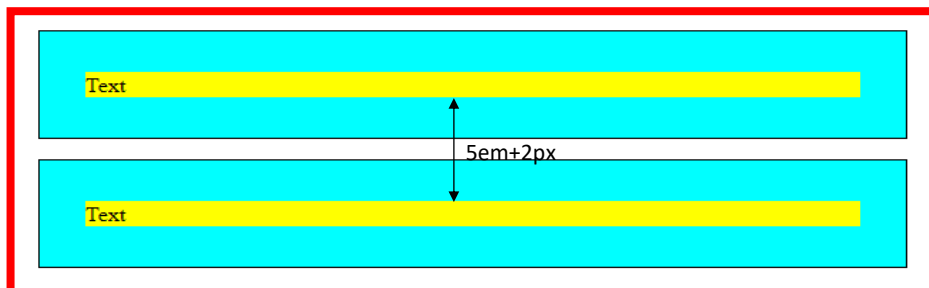
*Im Browser*: Die vertikalen Außenränder kollabieren, die horizontalen Außenränder addieren sich auf, zu sehen an den Elementen `section` (blau) und `p` (gelb). Der Vertikalabstand zwischen beiden Texten beträgt nicht  $6em$ , sondern nur  $2em$ .



3. **Ausnahmen**: Befindet sich zwischen dem Containerinhalt und seinem Außenabstand (`margin`) ein Rahmen (`border`) oder ein Innenabstand (`padding`), so überlappen sich die vertikalen Außenabstände nicht mehr mit denjenigen der eingeschachtelten Blöcke.

```
<section
  style="margin: 1em; background-color: aqua; border: 1px solid black">
  <p style="margin: 2em; background-color: yellow;">Text</p>
</section>
<section ...
```

*Im Browser*: Die vertikalen Außenränder kollabieren nur teilweise, Abstand insgesamt:  $2 * 2em(padding) + 2 * 1px(border-width) + 1 * 1em(margin, kollabiert) = 5em + 2px$ .



Eine weitere Ausnahme bildet der erwähnte Block Formatting Context (BFC). Margins eines BFC kollabieren nicht mit den Margins des Elternelements (siehe unten, BFC und Floats).

## 3.4 Abgerundete Ecken

Eigenschaft `border-radius` rundet die Ecken der Box.

... ist die Kurzschreibweise für

- `border-top-left-radius`
- `border-top-right-radius`
- `border-bottom-right-radius`
- `border-bottom-left-radius`

Beispiele:

```
border-radius: 0.5em;      /* Beispiel-1 */
border-radius: 0 0 1em 1em; /* Beispiel-2 */
```

- Beispiel-1: rundet alle vier Ecken mit einem Radius von 0.5em
- Beispiel-2: entspricht
  - `border-top-left-radius: 0`
  - `border-top-right-radius: 0`
  - `border-bottom-right-radius: 1em`
  - `border-bottom-left-radius: 1em`

## 3.5 Schatteneffekte

Eigenschaft `box-shadow` erzeugt Schatten in der Ebene unter der Box.

- hat die Ausdehnung der Box
- Mindestens drei Eigenschaftswerte pro Schatten:
  - `offset-x` (horizontale Verschiebung; neg. Wert: nach links)
  - `offset-y` (vertikale Verschiebung; neg. Wert: nach oben), Schatten kann aus der Box ragen
  - `color` (Farbe des Schattens, ggf. mit Transparenz)

Beispiel:

```
box-shadow: offset-x offset-y color;
box-shadow: 4px 4px #888;
```

- Optionale Angaben *vor* der Farbangabe:
  - `blur` (optional, verringert die Konturschärfe; Default: 0)
  - `spread` (optional, Vergrößert den Schatten; Default: 0)
- Mehrere Schatten als Liste möglich

Beispiel:

```
box-shadow: 5px 7px 2px #333,
            5px 7px 2px 9px #F00;
```

## 4 Layouts

Welche Techniken kennen Sie, um Bereiche nebeneinander zu stellen, z.B. so:



## 4.1 Floats

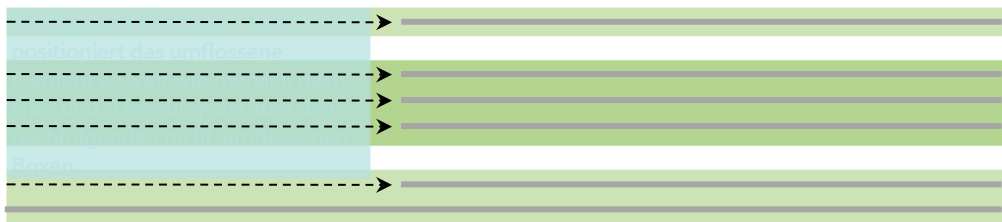
### 4.1.1 Grundprinzip

Beispiel:

```
/** Umflossenes Element */  
p.floatleft {  
    float: left;  
    width: 15em;  
}
```

Die Eigenschaft `float: left` bzw. `float: right` positioniert das umflossene Element so weit wie möglich nach links bzw. nach rechts und schwebt quasi über den nachfolgenden nicht-umflossenen Block-Boxen. Die *Textzeile* in der Nachfolge-Box wird vom Browser verkürzt, während der Hintergrund bzw. der Rahmen der Nachfolgebox unter der Float-Box sichtbar ist, falls diese transparent ist. Der Außenrand `margin` des umflossenen Elements wird berücksichtigt.

Skizze:



`float:`

... legt fest, ob das betreffende Element umflossen sein soll. Ein umflossenes Element wird automatisch zu einem Block-Element, das aus dem normalen Fluss entfernt wird. Die umflossene Box ist standardmäßig so breit wie ihr Inhalt.

Genau wie nicht gefloatete Block-Elemente werden Floats direkt nach dem letzten vorausgehenden Block-Element angeordnet.

`none`

(Standardwert) Das Element ist nicht umflossen.

`left`

Das Element wird so weit wie möglich links über dem normalen Fluss positioniert. Nachfolgende Zeilenboxen werden auf der linken Seite verkürzt, so dass deren Inhalt vollständig sichtbar ist. Dabei wird der rechte Außenrand des umflossenen Elements berücksichtigt.

`right`

Das Element wird rechts positioniert und nachfolgende Zeilenboxen werden auf der rechten Seite verkürzt.

Beispiel: Prinzip von Floats ([test\\_floats/Float\\_Prinzip.html](#))

Der HTML Code (Auszug):

```
<div class="static_box">Eine Block-Box im normalen Flow ...</div>
<div class="leftfloatbox">Die Eigenschaft float:&nbsp;left
positioniert ...</div>
<div class="static_box">Der in der Nachfolge-Box enthaltene ...</div>
<div class="static_box">Hintergrund bzw. der Rahmen ...</div>
```

Das zugehörige CSS (Auszug):

```
.leftfloatbox {
    float: left;
    width: 15em;
    margin-left: 0.7em; /* Folgeboxen im Hintergrund sichtbar */
    margin-right: 0.7em; /* Schafft Abstand zum umfließenden Text */
    background: #FAA;
}
.static_box {
    background: #AFA;
    margin-bottom: 0.5em;
}
```

Im Browser:

## Floats

Eine Block-Box im normalen Flow (grüner Hintergrund).

Die Eigenschaft `float: left` positioniert das umflossene Element so weit wie möglich nach links und schwebt quasi über den nachfolgenden nicht-umflossenen Boxen.

Der in der Nachfolge-Box enthaltene Zeilenbox wird links verkürzt, wobei der Außenrand `margin` des Float berücksichtigt wird.

Hintergrund bzw. der Rahmen der Nachfolgeboxen unter der Float-Box sind sichtbar, falls diese transparent ist. Die Textzeilen werden nicht mehr verkürzt, so bald alle Floats abgeschlossen sind.

### 4.1.2 Clearing von Floats

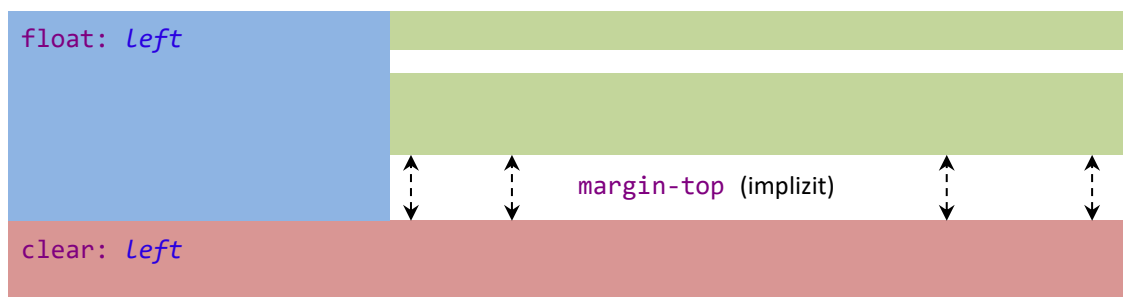
Soll der Textfluss wieder normal fortgesetzt werden, so muss die betreffende Nachfolge-Box die CSS-Eigenschaft `clear` erhalten, mit dem Werten `left`, `right` oder `both` (wenn bei zuvor stehenden Elementen sowohl `float: left` als auch `float: right` gesetzt war). In der deutschsprachigen Fachliteratur heißt es dann so schön 'Das Element wurde *gecleart*'.

Dies hat zur Folge, dass der obere Rand `margin-top` des *geclearten* Elements automatisch so weit vergrößert wird, dass das Element unterhalb der *Floats* platziert wird.

Beispiel:

```
/**/ Umflossenes Element ***/
p.floatleft {
    float: left;
    width: 15em;
}
/**/ Gecleartes Element ***/
p.themenwechsel {
    clear: left;
}
```

Skizze:



**clear:** ... beendet den Umfluss vorheriger Elemente. Dem Element wird implizit so viel oberer Außenrand hinzugefügt, dass sein oberer Rahmen `border-top` an der unteren Außenkante `margin-bottom` des *Floats* liegt.

`none` (Standardwert) Der Umfluss von vorherigen Elementen wird *nicht* beendet.

`left` Der Umfluss von vorherigen Elementen wird auf der *linken* Seite beendet. .

`right` Der Umfluss von vorherigen Elementen wird auf der *rechten* Seite beendet.

`both` Der Umfluss von vorherigen Elementen wird *auf beiden Seiten* beendet.



## Beispiel: Clearing von Floats ([test\\_floats/Float\\_Clear.html](#))

Der HTML Code (Auszug):

```
<div class="static_box">Eine Block-Box im normalen Flow ...</div>
<div class="leftfloatbox">Die Eigenschaft <code>float:&nbsp;left</code>
positioniert ...</div>
<div class="static_box">Der in der Nachfolge-Box enthaltene ...</div>
<div class="static_box clearing">Diese Box hat die Eigenschaft <code>clear:
left</code>. Der Browser ...</div>
```

Das zugehörige CSS (Auszug):

```
.leftfloatbox {
    float: left;
    width: 15em;
    margin-right: 0.7em; /* Schafft Abstand zum umfließenden Text */
    background: #CCF;
}
.static_box {
    background: #AFA;
    margin-bottom: 0.5em;
}
.clearing {
    clear: left;
    background: #FAA;
}
```

Im Browser:

### Clearing von Floats

Eine Block-Box im normalen Flow (grüner Hintergrund).

Die Eigenschaft `float: left` positioniert das umflossene Element so weit wie möglich nach links und schwebt quasi über den nachfolgenden nicht-umflossenen Boxen.

Der in der Nachfolge-Box enthaltene Zeilenbox wird links verkürzt, wobei der Außenrand `margin` des Float berücksichtigt wird.

Diese Box hat die Eigenschaft `clear: left`. Der Browser verleiht ihr automatisch einen oberen Außenrand (`margin-top`), so groß, dass das Element unterhalb des Floats positioniert wird.

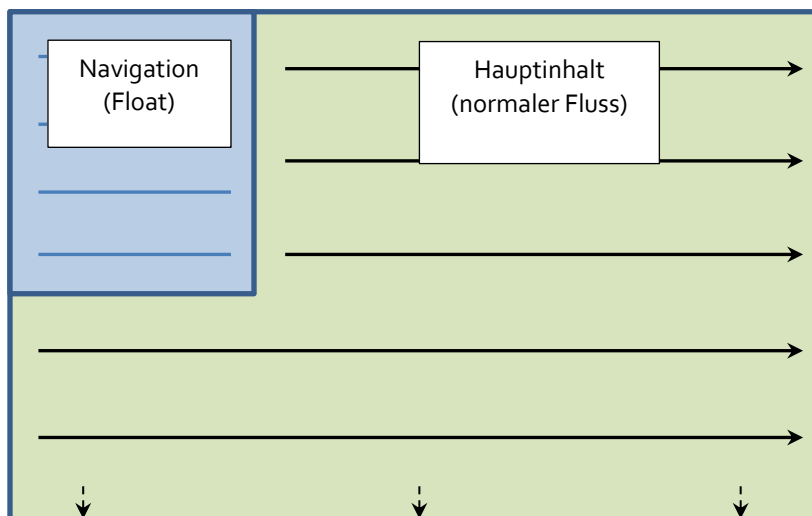
Beispiel: Einfaches, zweispaltiges Layout mit *Floats*

Ein Seitenlayout kann erstellt werden, indem größere Bereiche zu Floats deklariert werden. Exemplarisch soll hier ein einfaches Beispiel vorgestellt werden.

Die Seitennavigation soll auf der linken Seite in fester Breite dargestellt werden; der Hauptinhalt soll den Rest des Browserfensters einnehmen.

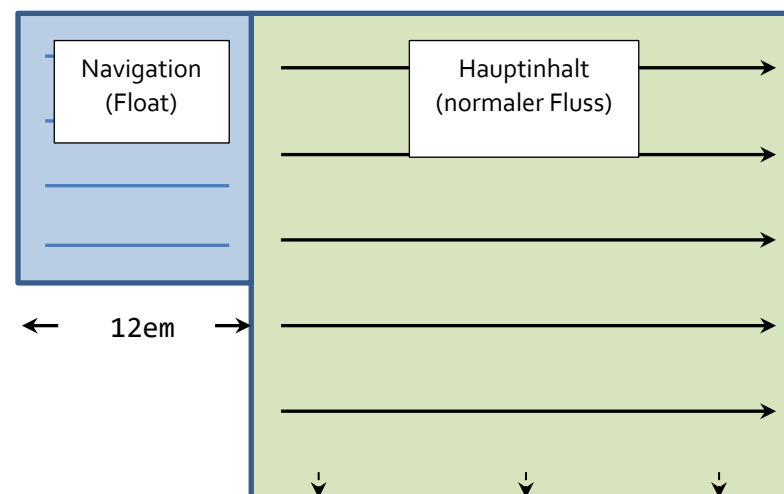
```
nav {  
  width: 12em;  
  float: left;  
}
```

Das Ergebnis sähe etwa wie folgt aus:



Damit der Navigationsbereich eine eigene "Spalte" erhält, muss der Inhaltsbereich einen *linken Rand* bekommen, welcher der Breite der Navigation entspricht.

```
article {  
  margin-left: 12em;  
}
```



### 4.1.3 Zusammenfassung: normaler Fluss und Positionierung

| Eigenschaft                             | <i>positioniert?</i> | <i>absolut positioniert?</i> | <i>im normalen Fluss?</i> |
|---|----------------------|------------------------------|---------------------------|
| <code>float: none</code>                | nein                 | nein                         | ja                        |
| <code>float: left   right</code>        | nein                 | nein                         | nein                      |
| <code>position: static</code>           | nein                 | nein                         | ja                        |
| <code>position: relative</code>         | ja                   | nein                         | ja                        |
| <code>position: fixed   absolute</code> | ja                   | ja                           | nein                      |

### Zusammenhang zwischen display, float und position

Die Eigenschaften `display`, `float` und `position` sind voneinander abhängig:

Es gelten folgende Regeln:

1. Ist `display: none` eingestellt, werden die Eigenschaften `position` und `float` ignoriert. `display: none` hat absoluten Vorrang.
2. Hat `position` ansonsten den Wert `absolute` oder `fixed`, wird `display` auf `block` und `float` auf `none` gesetzt. Ist ein Element *absolut positioniert*, wird es automatisch als Block-Box dargestellt. `position: absolute` bzw. `position: fixed` hat Vorrang vor `float`.
3. Hat `float` ansonsten einen anderen Wert als `none`, wird `display` auf `block` gesetzt. Ist ein Element umflossen, wird es automatisch als Block-Box dargestellt.
4. Andernfalls werden die restlichen `display`-Eigenschaften verwendet wie beschrieben.

#### Beispiel:

Weisen Sie einem *inline*-Element die Eigenschaft `float: left` zu, so wird es implizit zu einer Block-Box (auch ohne zugewiesene Eigenschaft `display: block`).

Weisen Sie diesem Element anschließend die Eigenschaft `position: absolute` zu, so verliert es dadurch seine *float*-Eigenschaft.

### Komplexere Layouts mit Floats

Will man komplexere Layouts mit *Floats* gestalten, kommt man oft nicht umhin *Floats* zu schachteln und/oder mehrere *Floats* nebeneinander zu stellen. Versucht man dies, kommt es mitunter zu unerwarteten Ergebnissen, die mit dem so genannten Blockformatierungs-Kontext (engl. *block formatting context*, im Folgenden kurz *BFC* genannt) zusammen hängen.

Was ist ein Blockformatierungs-Kontext?

Beim *BFC* handelt es sich um einen Bereich, in dem das Layout von Boxen stattfindet *und in welchem Floats miteinander interagieren*. Das Stammelement `html` erzeugt beispielsweise einen *BFC*.

Für *Floats* gilt in diesem Zusammenhang:

- Jeder *Float* (jedes Element mit Eigenschaft `float: Left` oder `float: right`) erzeugt einen neuen *BFC*.
- Clearing erfolgt nur für *Floats* aus dem gleichen *BFC*. Bei geschachtelten *Floats* ist dies zu beachten.
- Ein *BFC* schiebt sich *nicht* unter vorhandene *Floats*. Stattdessen erhält der *BFC* implizit einen Außenrand, wenn er neben einem *Float* steht.

Beispiel: Floats und *block formatting context* ([test\\_floats/Float\\_und\\_BFC.html](#))

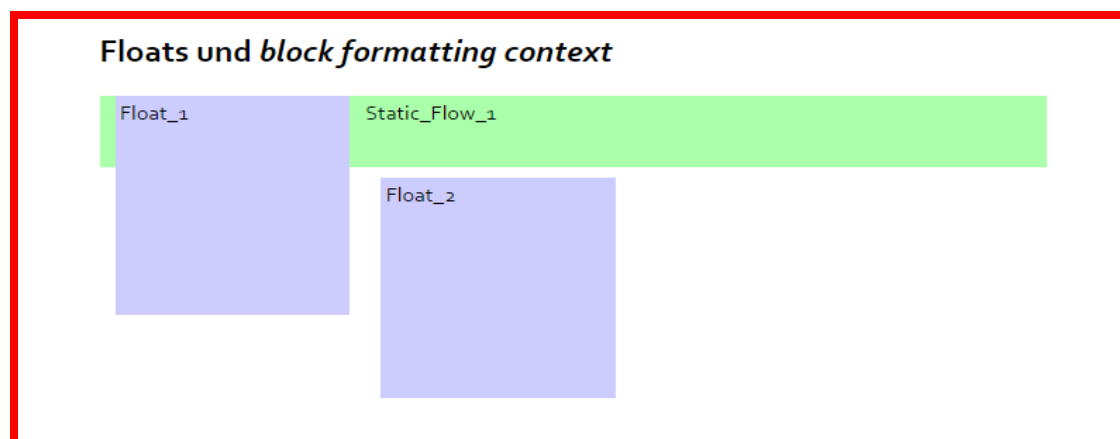
Der HTML Code (Auszug):

```
<div class="fl">Float_1</div>
<div class="static">Static_Flow_1</div>
<div class="fl">Float_2</div>
```

Das zugehörige CSS (Auszug):

```
.fl {
  float: left;
  background: #CCF;
  min-height: 10em;
  min-width: 10em;
  margin: 0 0.7em;
}
.static {
  background: #AFA;
  min-height: 3em;
  margin-bottom: 0.5em;
}
```

Im Browser:



Die statischen Folgebox (grün) verhält sich wie oben beschrieben: *Float\_1* "schwebt" über der grünen Box und verkürzt deren Textzeilen.

Die Folgebox '*Float\_2*' besitzt die Eigenschaft `float: left` und erzeugt damit einen neuen Kontext. Es gibt deshalb keine Überlappung zwischen '*Float\_1*' und '*Float\_2*' und es findet kein Umfließen statt. Vielmehr wird implizit ein linker Außenrand für '*Float\_2*' erzeugt. Es entsteht der Eindruck, als würde '*Float\_2*' hängen bleiben.

Nicht nur *Floats* erzeugen einen neuen Kontext. Da das Verhalten eines *BFC* von den 'normalen' Regeln abweicht (insbesondere in Zusammenhang mit *Floats*), ist folgende Frage nicht ganz unwichtig:

*Welche Elemente erzeugen einen neuen BFC?*

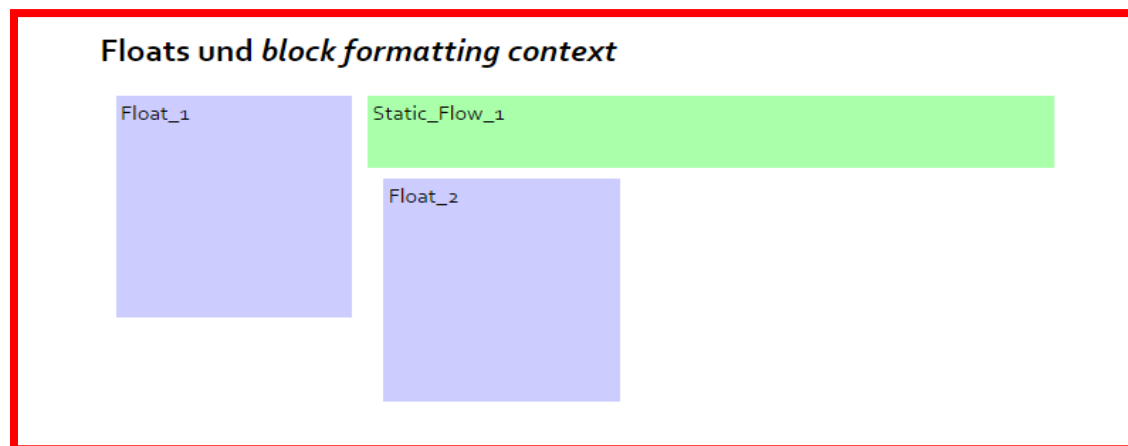
- Das Stammelement `html`
- *Floats*, d.h. Elemente mit Eigenschaft `float: left | right`
- Absolut positionierte Elemente (mit Eigenschaft `position: absolute | fixed`)
- Elemente deren Eigenschaft `display` einen der folgenden Werte besitzt
  - `inline-block`
  - `table-cell` oder `table-caption`
  - `flex` oder `inline-flex`
- Elemente mit Eigenschaft `overflow: hidden | scroll | auto`

Beispiel:

Der CSS Code des letzten Beispiels wird geringfügig ergänzt:

```
.static {  
    overflow: hidden  
}
```

Diese kleine Ergänzung bewirkt, dass die Folgebox 'Static-Flow\_1' einen neuen BFC erzeugt (mit der Nebenwirkung eines impliziten linken Außenrandes in der Breite von 'Float\_1') und sich somit die beiden Bereiche nicht mehr überlappen. Im Browser:



*Welche weiteren Nebenwirkungen erzielt ein BFC?*

- Die vertikalen Außenabstände (`margin-top | margin-bottom`) kollabieren nicht
- In einem *BFC* enthaltene *Floats* werden von dem *BFC* vollständig umschlossen. Mit anderen Worten: *Floats* ragen nie aus ihrem *BFC* heraus. (vgl: 'Containing Floats', unten)

#### 4.1.4 Containing Floats

Befinden sich alle Elemente im normalen Fluss, so wird die Höhe einer Block-Box standardmäßig aus dem Inhalt berechnet, z.B. aus der Summe der Höhe der enthaltenen Textzeilen.

*Floats* werden bei dieser automatischen Berechnung jedoch nicht mitgezählt, mit anderen Worten: *Floats* können aus ihrem umschließenden Container ragen!

Beispiel: Containing Floats ([test\\_floats/Containing\\_Floats.html](#))

In diesem Beispiel soll neben einer links fließenden Navigation eine Bildergalerie dargestellt werden. Zu jedem Bild der Galerie gehört ein Beschreibungstext, der auf der rechten Seite des jeweiligen Bildes erscheinen soll. Bild und Text sollen in jeweils einem Container *section* zusammengefasst werden.

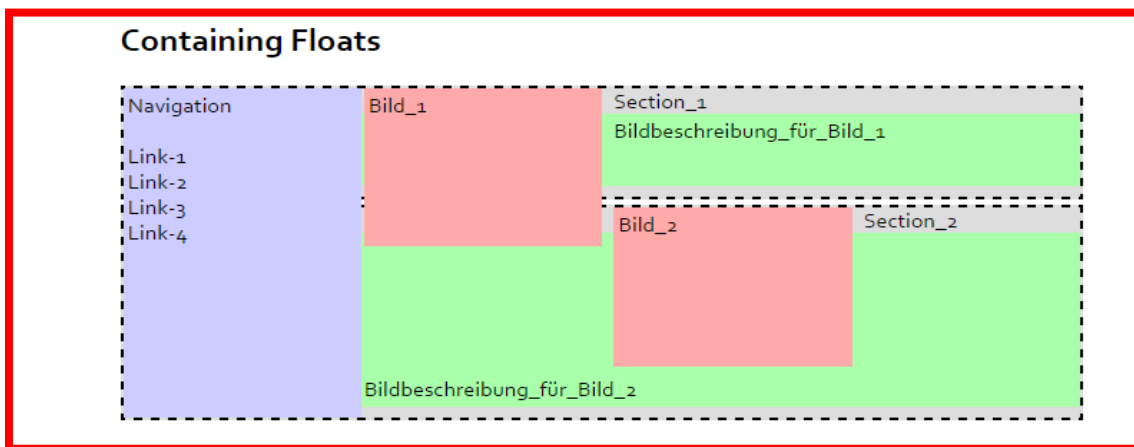
Der HTML Code (Auszug):

```
<nav>
  Navigation<br><br>
  Link-1<br>
  Link-2<br>
  Link-3<br>
  Link-4
</nav>
<section>
  Section_1
  <div class="fl">Bild_1</div>
  <div class="static">Bildbeschreibung_für_Bild_1</div>
</section>
<section>
  Section_2
  <div class="fl">Bild_2</div>
  <div class="static">Bildbeschreibung_für_Bild_2</div>
</section>
```

Das zugehörige CSS (Auszug):

```
nav {
  float: left;
  width: 10em;
  height: 15em;
  margin: 2px;
  padding: 0.2em;
  background: #CCF;
}
.fl {
  float: left;
  background: #FAA;
  min-height: 7em;
  min-width: 10em;
  margin: 0 0.5em 0.5em 0;
}
.static {
  background: #AFA;
  min-height: 3em;
  margin-bottom: 0.5em;
}
section {
  background: #DDD;
  border: dashed 2px;
  margin-bottom: 4px;
}
```

Im Browser:



Das Ergebnis entspricht nicht gerade unseren Erwartungen?

- Die Navigation (blau) befindet ganz links (✓)
- Section\_1 umschließt seinen enthaltenen Float (rot) nicht. Bild\_1 hängt aus seinem Container (grau, mit gestrichelter Linie) (–)
- Bild\_2 bleibt an Bild\_1 hängen (–)
- Die Bildbeschreibung\_für\_Bild\_2 steht *nicht* rechts von Bild\_2 (–)

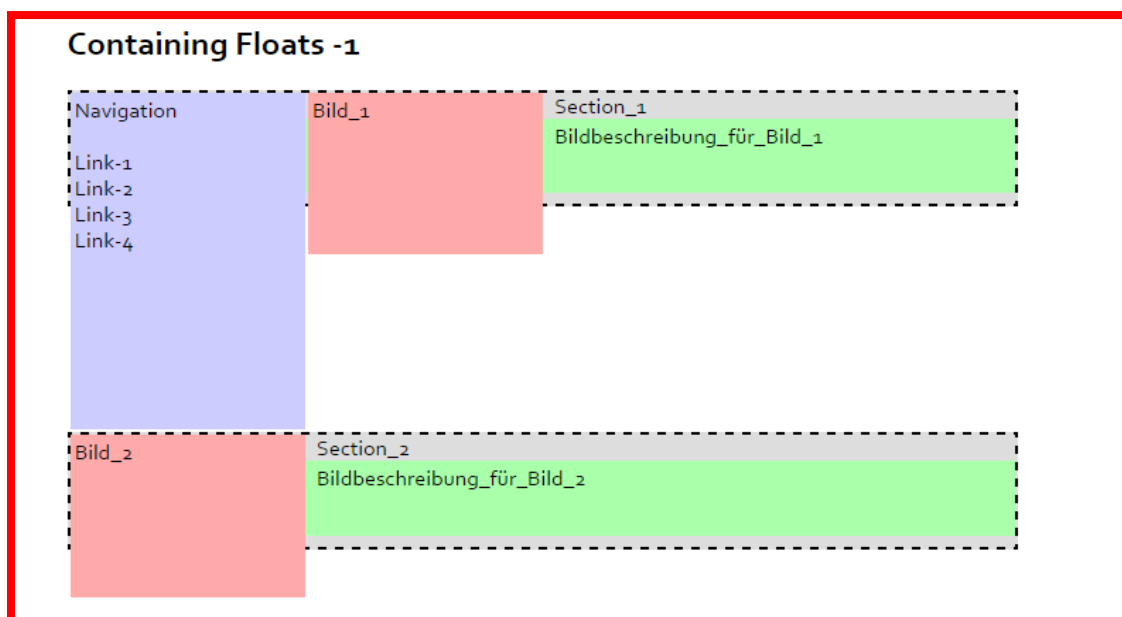
Offensichtlich haben wir das Clearing von Section\_2 vergessen. Als Zutat etwas CSS ([test\\_floats/Containing\\_Floats\\_1.html](#)):

```
.clearing {  
    clear: left;  
}
```

Die neue Klasse wird dem zweiten Element section zugewiesen:

```
<section class="clearing">  
  Section_2  
    <div class="fl">Bild_2</div>  
    <div class="static">Bildbeschreibung_für_Bild_2</div>  
</section>
```

Im Browser:



Ergebnis:

- Die Navigation (blau) befindet ganz links (✓)
- Section\_1 umschließt seinen enthaltenen Float (rot) nicht. Bild\_1 hängt aus seinem Container (grau, mit gestrichelter Linie) (–)
- Bild\_2 bleibt *nicht* an Bild\_1 hängen (✓)
- Die Bildbeschreibung\_für\_Bild\_2 steht nicht rechts von Bild\_2 (✓)
- Section\_2 befindet sich nun unterhalb der Navigation (–). Zuviel gecleart?

Das *Clearing* von Section\_2 bewirkt, dass *alle* linken *Floats* beendet werden, da sich alle *Floats* im gleichen *block formatting context* (dem Stammelement `html`) befinden!

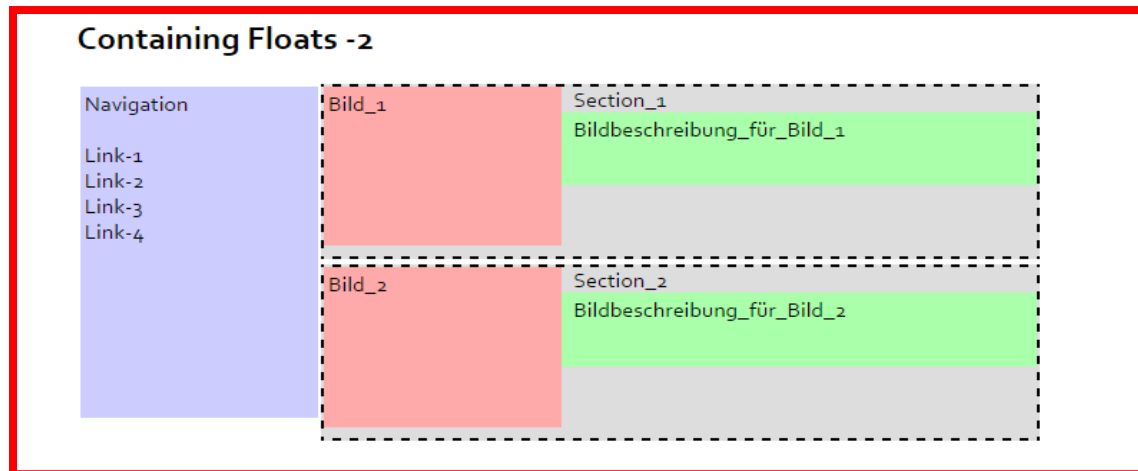


Ein anderer Ansatz: Wir machen aus dem Container `section` einen BFC, z.B. mit der Eigenschaft `overflow: hidden`. Ein *BFC* umschließt stets seine enthaltenen *Floats*!

Beispiel: Containing Floats -2 ([test\\_floats/Containing\\_Floats\\_2.html](test_floats/Containing_Floats_2.html))

```
section {  
    overflow: hidden;  
}
```

im Browser:



Ergebnis:

- Die Navigation (blau) befindet ganz links (✓)
- Section\_1 und Section\_2 umschließen ihre enthaltenen *Floats*. Bild\_1 und Bild\_2 (rot) hängen *nicht* aus ihren Containern (grau, mit gestrichelter Linie) (✓)
- Section\_2 befindet sich nun ebenfalls neben der Navigation, jedoch unterhalb von Section\_1. (✓)

### 4.1.5 Der Micro Clearfix Hack

Von Nicolas Gallagher entwickelte Methode, Floats zu umschließen, hier in vereinfachter Form (für modernere Browser, IE ab Version 8), bei der das Element keinen BFC erzeugt:

```
.clearfix:after {  
    content: "";  
    display: table; /* weniger Nebenwirkungen als display: block */  
    clear: both;  
}
```

Einsatz der Klasse im HTML-Code

```
<section class="clearfix">  
    <!-- die Floats, eingeschachtelt -->  
</section>  
<section>  
    <!-- die Inhalte dieser Section sind gecleart -->  
</section>
```

Da der Clearfix Hack aus dem Element keinen BFC erzeugt, kann zur Vermeidung kollabierender vertikaler Margin folgende Regel hinzugefügt werden:

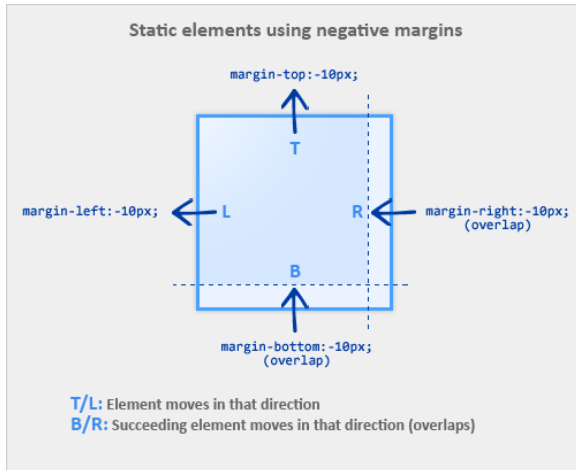
```
.clearfix:before {  
    content: "";  
    display: table;  
}
```

Zum Thema *Containing Floats* gibt es reichlich Literatur im Internet. Dort werden verschiedene Techniken diskutiert und gegeneinander abgewogen, z.B. bei <http://www.sitepoint.com/clearing-floats-overview-different-clearfix-methods/>.

### 4.1.6 Exkurs: Negative Margins

Negative Margins bewirken bei **statischem Inhalt**, dass...

- ...für negative `margin-top` bzw. `margin-left` das Element selbst verschoben wird und es dadurch zu Überlappungen mit dem Vorgängerelement kommen kann.
- ...für negative `margin-bottom` bzw. `margin-right` das Element selbst stehen bleibt, jedoch nachfolgende Elemente auf dieses Element gezogen werden können.



Der Effekt kann beispielsweise dazu genutzt werden, nebeneinander stehende durch Floats erzeugte Spalten optisch gleich hoch zu machen ("Durchfärben").

Beispiel (Auszug)

```
/* Regeln:
1. Ein negativer margin(-right|-bottom) zieht nachfolgende Elemente über das Element.
2. Ist padding-bottom = -(margin-bottom) wird das nachfolgende Element über den gesamten
   Padding-Bereich gezogen.
3. Ist das Element von einem Container (hier .wrap) mit Eigenschaft overflow: hidden
   umschlossen, wird überstehender padding abgeschnitten.
*/
/* zu 1. und 2. (.wrap umschließt alle Spalten) */
.wrap > * {
    padding-bottom: 30000em;
    margin-bottom: -30000em;
}
/* zu 3. */
.wrap {
    overflow: hidden;
}
```

Negative Margins bewirken bei **Floats**, dass

- für negatives `margin-top` das Element selbst verschoben wird
- für negatives `margin-bottom` das Element selbst stehen bleibt, jedoch nachfolgende Elemente auf dieses Element gezogen werden können.
- für horizontale Margins (`margin-left` / `margin-right`) *in* Float-Richtung wird das Element selbst verschoben.
- für horizontale Margins (`margin-left`/`margin-right`) *gegen die* Float-Richtung werden nachfolgende Elemente über diesen Float gezogen. Diese Regel kann genutzt werden, um Spalten in einer anderen Reihenfolge darzustellen, als im HTML-Code vorgegeben.

## 4.2 Übersicht CSS Tabellen

| HTML Element | Wert von "display"        |
|--------------|---------------------------|
| table        | <i>table</i>              |
| tr           | <i>table-row</i>          |
| td, th       | <i>table-cell</i>         |
| thead        | <i>table-header-group</i> |
| tfoot        | <i>table-footer-group</i> |
| tbody        | <i>table-row-group</i>    |
| caption      | <i>table-caption</i>      |
| col          | <i>table-column</i>       |
| colgroup     | <i>table-column-group</i> |

Weitere Eigenschaften

**border-collapse:** *separate* | *collapse*

Rahnen benachbarter Zellen sind getrennt (*separate*) oder zusammenfallend (*collapse*).

**border-spacing:** *<width>* [*<width>*] | *0*

Abstand zwischen Tabellenzellen, *nur* wenn auch '**border-collapse:** *separate*' ist. Bei zwei Werten von **border-spacing** legt der erste Wert den Horizontalabstand, der zweite Wert den Vertikalabstand fest.

**table-layout:** *auto* | *fixed*

Anpassung der Zellbreite (*auto*) oder feste Spaltenbreite (*fixed*). Ist die Breite der Spalten bei festem Tabellenlayout nicht angegeben, sind alle Spalten gleich breit. Das feste Tabellenlayout wird nur wirksam, wenn der Tabelle selbst eine Breite zugeordnet wurde.

**caption-side:** *top* | *bottom*

Anordnung der Tabellenbeschriftung über (*top*) oder unter (*bottom*) den Tabellenzeilen.

Die Verwendung von CSS Tabellen zu Layoutzwecken wird kontrovers diskutiert (vgl.

<http://adamschwartz.co/magic-of-css/chapters/3-tables/>,

<http://colintoh.com/blog/display-table-anti-hero>).

## 4.3 Übersicht: Flexbox

Eigenschaften des *Flexcontainers* (defaults sind unterstrichen)

```
display : flex | inline-flex

flex-flow : <flex-direction> || <flex-wrap>

    flex-direction : row | column | row-reverse | column-reverse

    flex-wrap : nowrap | wrap | wrap-reverse

justify-content : flex-start | flex-end | center | space-between | space-around

align-items : flex-start | flex-end | center | baseline | stretch

align-content : flex-start | flex-end | center | space-between | space-around
                | stretch
```

Mit der Eigenschaft **display**: flex wird das Flex-Layout für den *Flexcontainer* aktiviert.

Die Hauptachse (**flex-direction**) und Fähigkeit zu mehreren Flexzeilen (**flex-wrap**) kann mit dem Kürzel **flex-flow** festgelegt werden, *default* ist **flex-flow**: row nowrap.

Die Ausrichtung der Items in der Flexzeile entlang der Hauptachse wird mit **justify-content**, entlang seiner Kreuzachse mit **align-items** festgelegt.

Die Verteilung der Zwischenräume bei *mehrzeiligen* Flex-Containern ist durch die Eigenschaft **align-content** festgelegt.

Eigenschaften des *Flexitems*

```
flex : none | auto | [<flex-grow> <flex-shrink>? || <flex-basis>]

    flex-grow : <number> | 0

    flex-shrink : <number> | 1

    flex-basis : auto | <width>

order : <integer> | 0

align-self : auto | flex-start | flex-end | center | baseline | stretch
```

Die Basisgröße (**flex-basis**) sowie das Verhalten beim Vergrößern (**flex-grow**) bzw. beim Verkleinern (**flex-shrink**) kann mit dem Kürzel **flex** festgelegt werden, *default* ist **flex**: 0 1 auto.

Die Reihenfolge, in der das Flexitem in der Flexzeile auftaucht, ist durch die Eigenschaft **order** definiert, *default* ist **order**: 0.

Die Eigenschaft **align-self** überschreibt die Vorgabe **align-items** (des Elternelements) zur Ausrichtung entlang der Kreuzachse, *default* ist **align-self**: auto.

## 5 Mediaqueries


### 5.1 Syntax

Allgemeine Syntax:

media-query = media-type condition\*

Beispiel

```
<link rel="stylesheet" media="screen and (color)" href="example.css" />
oder
@import url(color.css) screen and (color);
```



Der Medientyp **all** kann weggelassen werden. Identisch sind.

```
@media all and (min-width:500px) { ... }
@media (min-width:500px) { ... }
```

Logische "oder" Verknüpfung durch Kommaseparierte Liste

```
media-query, media-query, ...
@media screen and (color), projection and (color) { ... }
```

Verneinung des Ausdrucks durch vorangestelltes **not**

```
<link rel="stylesheet" media="not screen and (color)" href="example.css" />
```

Da HTML4-Browser vor dem Medientyp kein anderes Schlüsselwort erwarten, wird das Stylesheet aus dem vorigen Beispiel in HTML4-Browsern nicht ausgeführt. Zunutze macht dies die Syntax mit vorangestelltem '**only**'; HTML4 Browser verweigern wie bei 'not' das Ausführen des Stylesheets, während HTML5 Browser das Schlüsselwort 'only' ignorieren sollen und das Stylesheet dann ausführen:

```
<link rel="stylesheet" media="only screen and (color)" href="example.css" />
```

Wenn der Medientyp nicht zur nachfolgenden Bedingung passt, ist der Ausdruck false. Beispiel: ein aurales Medium kennt kein Seitenverhältnis (device-aspect-ratio). Daher ist folgender Ausdruck immer false:

```
<link rel="stylesheet" media="aural and (device-aspect-ratio: 16/9)"
href="example.css" />
```

## 5.2 Übersicht Features

| Name                | Value                | Media           | min/max? |
|---------------------|----------------------|-----------------|----------|
| width               | <length>             | visual, tactile | ✓        |
| height              | <length>             | visual, tactile | ✓        |
| device-width        | <length>             | visual, tactile | ✓        |
| device-height       | <length>             | visual, tactile | ✓        |
| orientation         | portrait   landscape | bitmap          | -        |
| device-aspect-ratio | <ratio>              | bitmap          | ✓        |
| resolution          | <resolution>         | visual          | ✓        |

### Beispiele

print and (min-width: 25cm)

screen and (min-width: 400px) and (max-width: 700px)

handheld and (min-width: 20em), screen and (min-width: 20em)

Die folgenden zwei Beispiele erzeugen das gleiche Ergebnis:

```
@media screen and (device-aspect-ratio: 16/9) { ... }
```

```
@media screen and (device-aspect-ratio: 32/18) { ... }
```


print and (min-resolution: 300dpi)

print and (min-resolution: 118dpcm)

## 6 Übersicht: CSS Einheiten

### 6.1 CSS Längeneinheiten

Für die Eigenschaften `font-size`, `padding`, `margin` sowie die Angabe der Rahmendicke bei `border` ist der Wert eine Größenangabe. CSS verwendet die folgenden Längeneinheiten:

- Die gebräuchlichsten Längeneinheiten  
**px** für **Pixel**. Auf einem Standard-Desktop System mit einer Pixeldichte von 96dpi entspricht dies 0,96 inches bzw. 0,26 mm  
**em** für **elementeigene Schrifthöhe** (relativ zur Schrifthöhe des Elternelements)  
**%** für **Prozent**. Der Bezug ist von der jeweiligen Eigenschaft abhängig.  
**pt** für **Punkt** (= 1/72 inches)
- Weitere Längeneinheiten  
**pc** für **Pica** (= 12 Punkt)  
**in** für **Inch** (= 2,54 cm)  
**mm** für **Millimeter**  
**cm** für **Zentimeter**  
**ex** für **elementeigene Höhe des Buchstabens x** (relative Angabe)
- neu in CSS3<sup>1</sup>  
 **rem** für 'root'-em (relativ zur Schriftgröße des Wurzelements)  
**vw** für 'viewport width': je 1% der Breite des Browserfensters  
**vh** für 'viewport height': je 1% der Höhe des Browserfensters  
**vmin** für 'viewport minimum': je 1% der schmalere Seite Browserfensters  
**vmax** für 'viewport height': je 1% der breiteren Seite des Browserfensters  
**ch** für 'character width' Breite des Zeichens 0 des aktuellen Fonts


Prozentuale Angaben werden je nach CSS-Eigenschaft unterschiedlich interpretiert. Für die Eigenschaft `font-size` gibt sie das Verhältnis zur Schriftgröße im Elternelement an. Bei `padding` sowie `margin` beziehen sich Angaben in Prozent auf den Wert von `width` des Elternelementes. Bei `border` wiederum ist eine prozentuale Angabe nicht möglich.

Für die Eigenschaft `font-size` kann die Angabe auch in `em`, `ex` erfolgen, dies bezieht sich dann aber auf die Schriftgröße im Elternelement.

Die Maßeinheit muss unmittelbar hinter die Zahl geschrieben werden, es darf kein Leerraum zwischen Größenangabe und Einheit stehen.

Das Dezimaltrennzeichen bei Größenangaben ist der Punkt.

### 6.2 CSS Winkel

- neu in CSS3  
 **deg** für 'Degree'. 360deg bilden einen Vollkreis.  
**grad** für 'Gradian'. 400grad bilden einen Vollkreis.  
**rad** für 'Radian'.  $2\pi$  Radian bilden einen Vollkreis.  
**turn** ein Turn bildet einen Vollkreis.

---

<sup>1</sup> zur Zeit nur teilweise Browserunterstützung (vgl. <http://caniuse.com/#feat=viewport-units>)




## 6.3 CSS Farben - RGB

Im *herkömmlichen* RGB-Farbmodell mischen sich die Farben aus den Grundwerten **Rot**, **Grün** und **Blau**. Jede dieser Komponenten kann einen Intensitätswert zwischen null (Komponente nicht enthalten) und 255 (Komponente maximal enthalten) annehmen. So ergibt die Kombination (0,0,0) die Farbe Schwarz, die Kombination (255,255,255) hingegen die Farbe Weiß. Die Kombinationen können...

1. ...als **Hexadezimalwert** geschrieben sein, bei dem jede Komponente zwei Hexadezimalziffern 0 bis F ausmacht. Browser erkennen solche Angaben an einem vorangestellten #-Zeichen, z.B. `#FF00FF`. Es kann gekürzt werden, wenn jede Komponente aus jeweils zwei gleichen Ziffern besteht; `#F0F` ergibt `#FF00FF`.
2. ...als **Funktion** `rgb` geschrieben sein, bei der die Komponenten, durch Komma getrennt, als Funktionsparameter geschrieben sind, z.B. `rgb(255,0,255)`. Die Farbintensität kann in der `rgb`-Funktion auch in Prozentwerten ausgedrückt werden, z.B. `rgb(100%,0%,100%)`.

RGB-Farben sind 'aus dem Gefühl heraus' nur sehr schwer abzuschätzen. Eine Hilfe bieten sog. *Colorpicker*, die als (online-)Tool, z.B. aus dem Web angeboten werden (Beispiel für ein Online-Colorpicker: <http://www.colorpicker.com/>).

Für einige vordefinierte Farben können Farbnamen als *Schlüsselwörter* geschrieben werden. **Basisfarbnamen** aus CSS2:

| Farbe   | Schlüsselwort  | als Hex rgb | als Funktion                  |
|---|----------------|-------------|-------------------------------|
|  | <i>black</i>   | #000000     | <code>rgb(0,0,0)</code>       |
|  | <i>silver</i>  | #C0C0C0     | <code>rgb(192,192,192)</code> |
|  | <i>gray</i>    | #808080     | <code>rgb(128,128,128)</code> |
|  | <i>white</i>   | #FFFFFF     | <code>rgb(255,255,255)</code> |
|  | <i>maroon</i>  | #800000     | <code>rgb(128,0,0)</code>     |
|  | <i>red</i>     | #FF0000     | <code>rgb(255,0,0)</code>     |
|  | <i>purple</i>  | #800080     | <code>rgb(128,0,128)</code>   |
|  | <i>fuchsia</i> | #FF00FF     | <code>rgb(255,0,255)</code>   |
|  | <i>green</i>   | #008000     | <code>rgb(0,128,0)</code>     |
|  | <i>lime</i>    | #00FF00     | <code>rgb(0,255,0)</code>     |
|  | <i>olive</i>   | #808000     | <code>rgb(128,128,0)</code>   |
|  | <i>yellow</i>  | #FFFF00     | <code>rgb(255,255,0)</code>   |
|  | <i>navy</i>    | #000080     | <code>rgb(0,0,128)</code>     |
|  | <i>blue</i>    | #0000FF     | <code>rgb(0,0,255)</code>     |
|  | <i>teal</i>    | #008080     | <code>rgb(0,128,128)</code>   |
|  | <i>aqua</i>    | #00FFFF     | <code>rgb(0,255,255)</code>   |

Beispiel: Absatz mit Schrift- und Hintergrundfarbe

```
<p style="color: maroon; background-color: #008080"> Text.... </p>
```

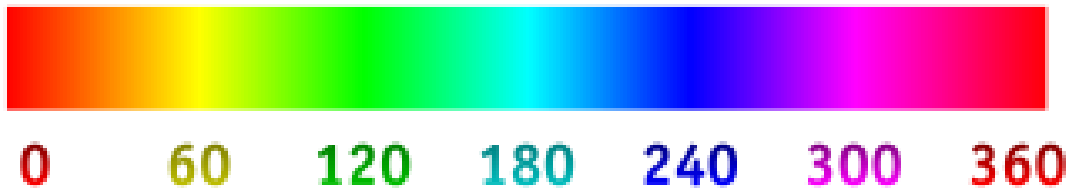


Ab CSS3 werden zusätzlich Farbnamen aus der Spezifikation von SVG (*scalable vector graphics*) unterstützt, vgl. <http://www.w3.org/TR/css3-color/#svg-color>.

## 6.4 CSS Farben - HSL



Ab CSS3 können Farben auch im HSL-Farbraum angegeben werden. Ein Farbton (*hue*) aus dem "Farbrad" zwischen 0 und 360 Grad wird kombiniert mit dem prozentualen Sättigungswert (*saturation*) und der prozentualen Helligkeit (*lightness*).



Farbtonskala mit Werten zwischen 0 und 360 Grad

Farbton 120 (grün) bei verschiedenen Helligkeitswerten (Zeilen) und Sättigungsstufen (Spalten).

|            |      | Sättigung |     |     |     |    |
|------------|------|-----------|-----|-----|-----|----|
|            |      | 100%      | 75% | 50% | 25% | 0% |
| Helligkeit | 100% |           |     |     |     |    |
|            | 88%  |           |     |     |     |    |
|            | 75%  |           |     |     |     |    |
|            | 63%  |           |     |     |     |    |
|            | 50%  |           |     |     |     |    |
|            | 38%  |           |     |     |     |    |
|            | 25%  |           |     |     |     |    |
|            | 13%  |           |     |     |     |    |
|            | 0%   |           |     |     |     |    |

Weitere Details rund um HSL (inklusive Colorpicker-Tool) finden Sie beispielsweise auf der Webseite <http://www.workwithcolor.com/>.

Beispiel: Absatz mit Hintergrundfarbe

```
<p style="background-color: hsl(120,25%,88%)"> Text.... </p>
```

## 6.5 Transparenz mit RGB/HSL



Ab CSS3 ist auch (Halb-)Transparenz in den Farbdefinitionen vorgesehen. RGB bzw. HSL-Farben erhalten eine vierte Komponente, den *Alphawert*. Diese Komponente ist ein Wert zwischen 0.0 (voll transparent, völlig durchscheinend) und 1.0 (nicht transparent).

Verwenden Sie dann die Funktionen *rgba(rot, grün, blau, transparenz)* bzw. *hsla(ton, sättigung, helligkeit, transparenz)*.

Beispiel: Absatz mit halbtransparentem Hintergrund

```
<p style="background-color: hsla(120,25%,88%,0.7)"> Text.... </p>
```