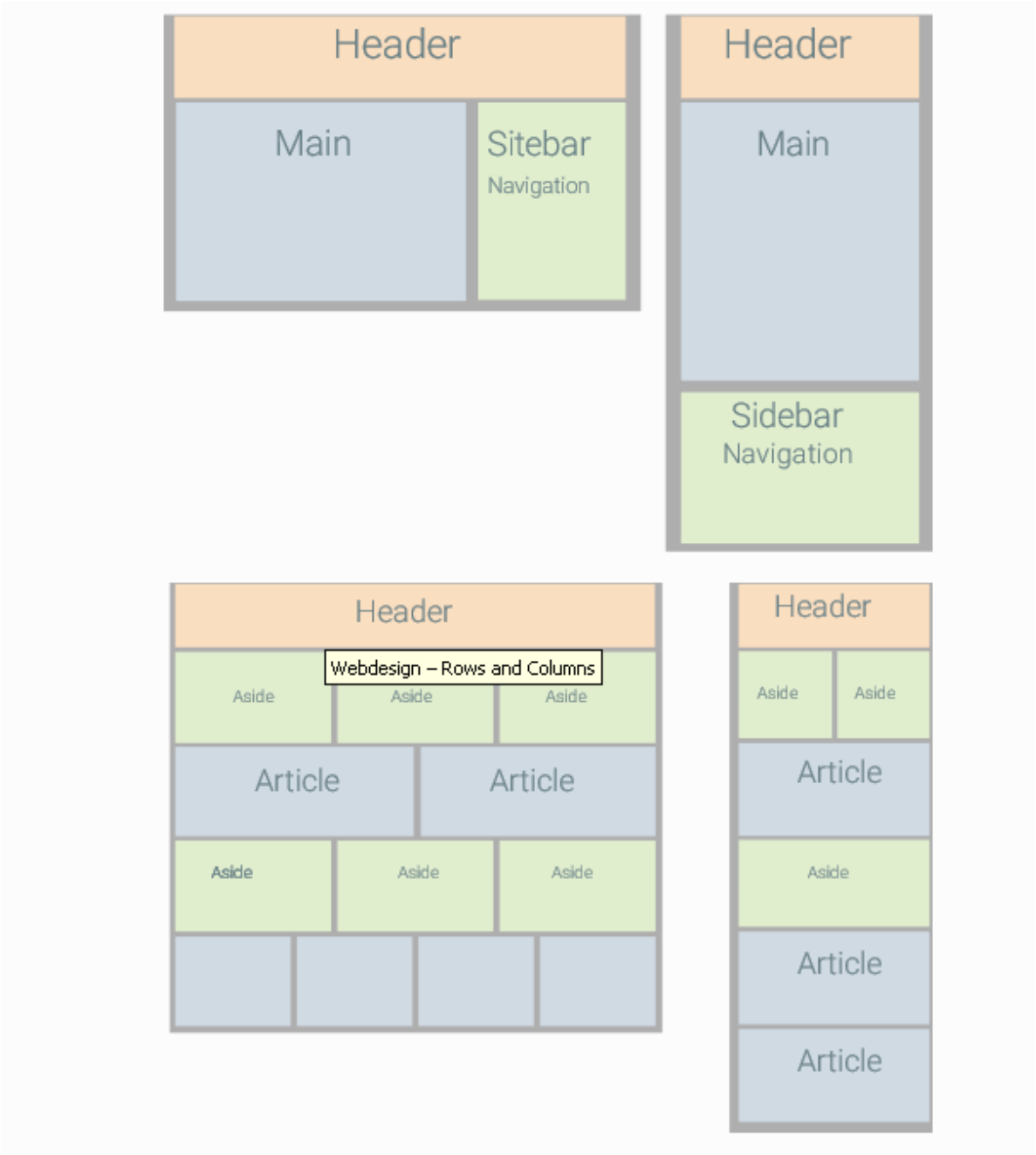
CSS Positionieren: Grid Systeme

==========

Ein Layout mit zwei, drei oder vier Spalten mit den modernen Browsern oder
der langjährigen Erfahrung im Umgang mit CSS-Eigenschaften für Positionierung – vor allem float – keine große Herausforderung mehr.
Aber heute gehört responsives Webdesign zu den grundlegenden Anforderungen an einen Internet-Auftritt. Auf kleinen Monitoren sollen die Spalten kollabieren und untereinander rutschen: An dieser Stelle kommen Grid-Systeme ins Spiel. Grid-Systeme sind CSS-Packs, die ein zeilen- und spaltenorientieres Layout vereinfachen. Typischerweise haben Grid-Systeme 12 bis 16 Spalten und ein einfaches Zuweisen von CSS-Klassen reicht, um komplexe responsive Layouts aufzuziehen. Das Grid-System kümmert sich um all die Ecken und Kanten der Browser.
Beim traditionellen Spaltenlayout rutscht die komplette Navigation auf kleinen Monitoren unter den Inhalt und erscheint dann schnell zusammenhanglos.



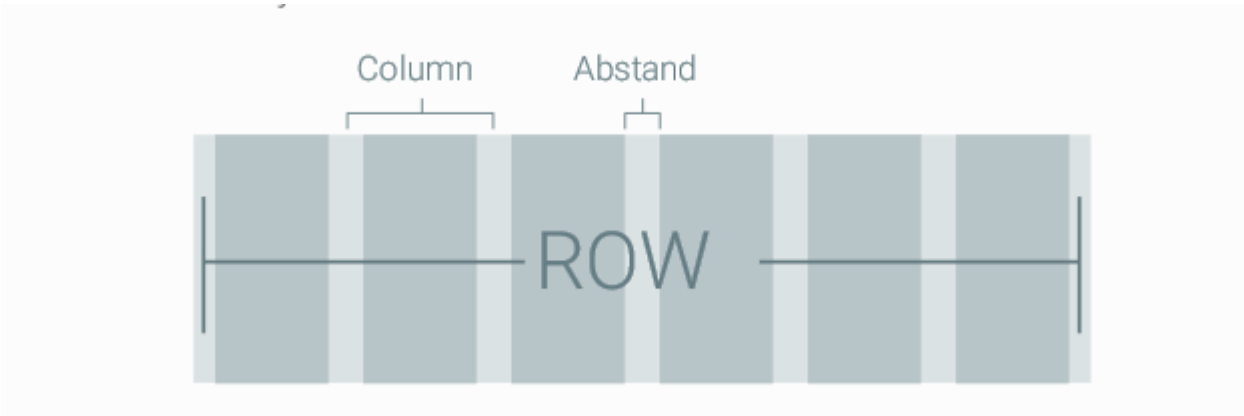
Das zeilenorientierte Raster eines Grid-Systems kann die Inhalte auflockern und gleichzeitig passende Inhalte beisammen halten.

Webseiten-Layout: Trends

Mit CSS Flex oder Flexbox bekommen wir ein Model für das Design von Webseiten, das uns völlig neue Sichtweisen auf das Positionieren von Blöcken erleichtert.
Noch leidet CSS Flexbox unter Browser-Bugs und kleinen Inkompatibilitäten. Es gibt einen weiteren Ansatz für das Positionieren und Ausrichten von Blöcken – CSS Grid. CSS Grid wird bislang nur von Internet Explorer unterstützt und steht weder bei Chrome, Firefox oder Safari auf dem Plan. Am 12. Januar 2016 fällt IE9 aus dem Support von Microsoft, wird aber sicher noch für viele Webseiten auf der Liste der unterstützten Browser bleiben. So sind die klassischen Grid-Systeme noch eine Weile die beste Erleichterung beim zeilen- und Spalten-orientierten Webdesign.

Positionieren mit CSS-Grid-Systemen

Jedes Grid System hat seine individuelle Syntax, aber das grundlegende Konzept ist bei allen Grid Systemen dasselbe.



Grid-Systeme haben natürlich auch ihre Schattenseiten: Die Zeilen und Spalten erfordern zusätzliches – nicht semantisches – HTML-Markup.
Die großen Frameworks wie Foundation und Bootstrap bringen immer ein Grid-System mit. Wer die überwältigenden Frameworks scheut, kann auf unzählige unabhängige Grid-Systeme zurückgreifen.

- 960 Grid System ein frühes Grid-System, noch nicht responsive. Das 960 Grid System ist das Mutterschiff der Grid-Systeme: Seine Syntax ist von vielen moderneren Grid-Systemen übernommen worden.
- Golden Grid – Minimales Grid, das Spalten auf kleinen Monitoren zusammenfaltet.
- Fluid Baseline Grid – schon fast ein Framework – mit großem Gewicht auf Typographie.
- Das Grid faltet sind zu einer Spalte auf kleinen Monitoren, zwei Spalten bei Tabletts und drei Spalten auf dem Desktop. Mit 18 KB ist Fluid Baseline Grid – kurz FBG – ausgesprochen sparsam im Verbrauch. Responsive Grid System hat 12, 16 oder 24 Spalten im Angebot und zieht ältere Browser durch Respond.js als Polyfill mit.
- Responsive Grid System. Hier muss nicht das ganze CSS-Paket geladen werden, sondern wenn klar ist, dass 3 oder 4 Spalten für das Layout reichen, konfektioniert der Grid Generator das CSS.
  So klein, bietet aber schon Spalten gleicher Höhe – allerdings muss dafür jQuery vorliegen.
- SKELETON A dead simple, responsive boilerplate
  Die komplette Dokumentation auf einer Seite – eine hervorragende Startposition.

Features von Grid-Systemen

Die meisten Grid-Systeme sind durchgehend flexible – d.h., dass auch Text bei jeder Veränderung des Browserfensters neu umgebrochen wird.
Wenn ein Grid-System »Folding« beherrscht, können die Blöcke bei mittleren und großen Monitoren neu angeordnet werden. Z.B.

- Auf großen Monitoren 4 Blöcke in einer Zeile, auf mittelgroßen Monitoren 2 Blöcke in zwei Reihen.
- Auf großen Monitoren 6 Blöcke in einer Zeile, auf mittelgroßen 3 Blöcke in zwei Reihen, auf kleinen Monitoren 2 Blöcke in drei Reihen.

Ein hochgerüstetes Grid-System kann Zeilen in Spalten verschachteln.
Grid-Systeme, die alle Schmankerln beherrschen, sind rar. Die komfortabelsten Grid-Systeme finden wir erst in den großen Frameworks wie Bootstrap und Foundation, aber dort muss man mit großen CSS-Dateien und heftigen Javascript plus jQuery-Einsatz rechnen.

Spalten positionieren

CSS gibt uns *float*, *display:inline-block*, *display-table* und *display-flex* für ein spalten-orientiertes Webdesign. *display: flex* ist die eleganteste Technik, aber noch von Browser-Inkompatibiliät belastet. IE9 unterstützt *display: flex* noch nicht.
CSS float ist und bleibt die griffigste Methode – darum setzen alle Grid-Systeme, CSS-Frameworks und Boilerplates auf float.
Grid-Systeme unterteilen die Zeilen allesamt ähnlich in Spalten:

```
[class*='colspan'] { width: 100%; float: left; box-sizing: border-box }

.col-1 { width: 16.66% }
.col-2 { width: 33.33% }
.col-3 { width: 50% }
.col-4 { width: 66.664% }
.col-5 { width: 83.33% }
.col-6 { width: 100% }
```

[class*='colspan'] ist ein »fast«-Universal-Selector: Alle CSS-Klassen, der Bezeichnung mit 'colspan' anfängt.
Bevor CSS box-sizing auf den Plan trat, war ein spalten-orientiertes Design auf Prozent-Basis harte Arbeit mit stetem Anlass zu frustrierenden Fehlern. Bei *box-sizing: border-box* werden padding und border mit zur Spaltenbreite gezählt. *box-sizing: border-box* wird bereits von IE8 unterstützt.

Zeilen erzeugen

Zeilen verhindern, dass die Spalten in die nächste Reihe überfließen und werden durch ein Wrapper-Element ROW umgesetzt.

```
.row:after {
   content: "";
   display: table;
   clear: both
}
```

Die Regel für das HTML-Markup von ROWS und COLUMNS lautet: In jeder ROW müssen sich die Spaltenbreiten zu 100% ergänzen.

```
<div class="row">
   <div class="col-1"><p>Spaltenbreite 16,66%</p></div>
   <div class="col-3"><p>Spaltenbreite 50%</p></div>
   <div class="col-2"><p>Spaltenbreite 33,33%</p></div>
</div>
```

Das Grid selber ist also einfach gestrickt. Damit das Grid-System responsive wird, muss festgelegt werden, wie die Spalten bei den Breakpoints kollabieren.
Für die kleinsten Monitore bleibt die Spaltenbreite immer 100%: Alle Spalten rutschen untereinander.
Ab 480px Breite gibt es zwei Spalten, ab 720px Breite drei Spalten, ab 1260px Breite treten 6 Spalten in Kraft.

```
body {background: #efefef}
.row:after { content: ""; display: table; clear: both  }
[class*='colspan'] {
   width: 100%;
   float: left;
   box-sizing: border-box;
   padding: 0.5em 1em;
   border:1px solid silver;
   background: white
}

@media (min-width:400px) {
   .colspan1 { width: 50%; }
   .row .colspan5 ~ .colspan1{ width: 100%; }
}

@media (min-width:720px) {
   .colspan1 { width: 33.33%; }
   .colspan2 { width: 50%; }
   .colspan3 { width: 50%; }

 .row .colspan2:last-of-type { width: 100% }
}
```

```
@media (min-width:1260px) {
    .colspan1 { width: 16.66%; }
    .colspan2 { width: 33.33%; }
    .colspan3 { width: 50%; }
    .colspan4 { width: 66.66%; }
    .colspan5 { width: 83.33%; }
    .colspan6 { width: 100%; }
    .row .colspan5 ~ .colspan1{ width: 16.66%; }
    .row .colspan2:last-of-type { width: 33.33%; }
}
```

Bis auf ein paar Ausnahmeregeln war's das. Ausnahmen treten auf bei

@media (min-width:480px)

Wenn Spaltenbreite 1 und Spaltenbreite 5 einer einer Zeile sitzen (in der Sprache der CSS-Selektoren indirekte Nachbarn sind). In so einem Fall muss Spaltenbreite 1 auf 100% gedehnt werden.

```
.row .colspan5 ~ .colspan1 { width: 100% }
```

@media (min-width:720px)

Wenn drei Spalten mit Spaltenbreite 2 zusammen in einer Zeile sitzen

```
.row .colspan2:last-of-type { width: 100% }
```

In @media (min-width:1260px) müssen die Ausnahmen wieder zurückgenommen werden.

```
 .row .colspan5 ~ .colspan1{ width: 16.66%; }
 .row .colspan2:last-of-type { width: 33.33%; }
```

Die Ausnahmen müssen im nächsten Breakpoint zurück genommen werden.

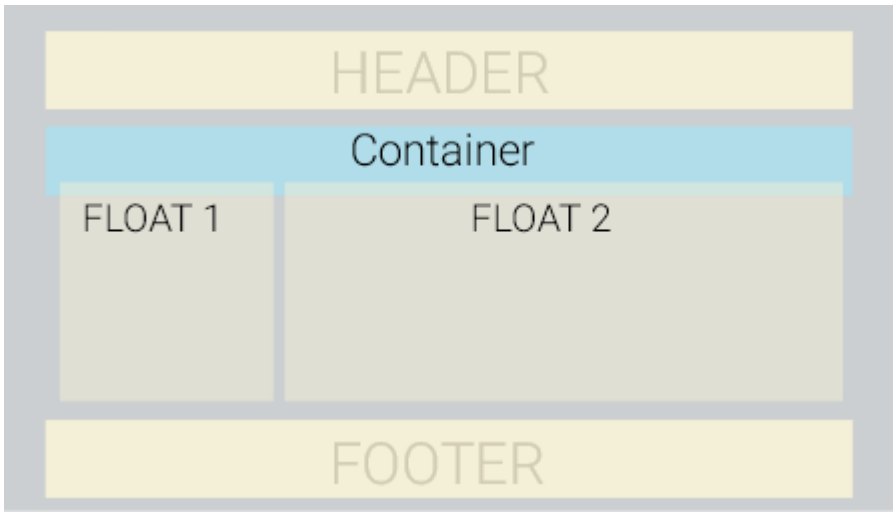
Demo: Spartian Grid

Zusammen ist das nur die Spitze des Eisbergs bis zu einem zuverlässigen responsiven Grid-System. Am Ende sind die Grids der großen Frameworks Bootstrap und Foundation das Null Plus Ultra, wenn auch zu hohen Kosten.

CSS float

==========

CSS *float* setzt ein HTML-Element an den rechten oder linken Rand des umfassenden HTML-Blocks. *float* verdrängt den Text der folgenden Blöcke und ahmt so das Umfließen aus Quark, InDesign und Word nach.
float eignet sich für individuelle Elemente wie ein schwebendes Bild, ist aber auch die Grundlage für das Spalten-Raster (Grid) von HTML-Seiten.




```
.col:nth-child(1) { float: left; width: 30%; }
.col:nth-child(2) { float: right; width: 60%; }
```

Ein Element mit *float* wird nur so breit wie sein Inhalt, auch wenn es ein Block-Element ist. Elemente mit *CSS float* brauchen darum immer die Angabe der Breite durch *width*.

float und center

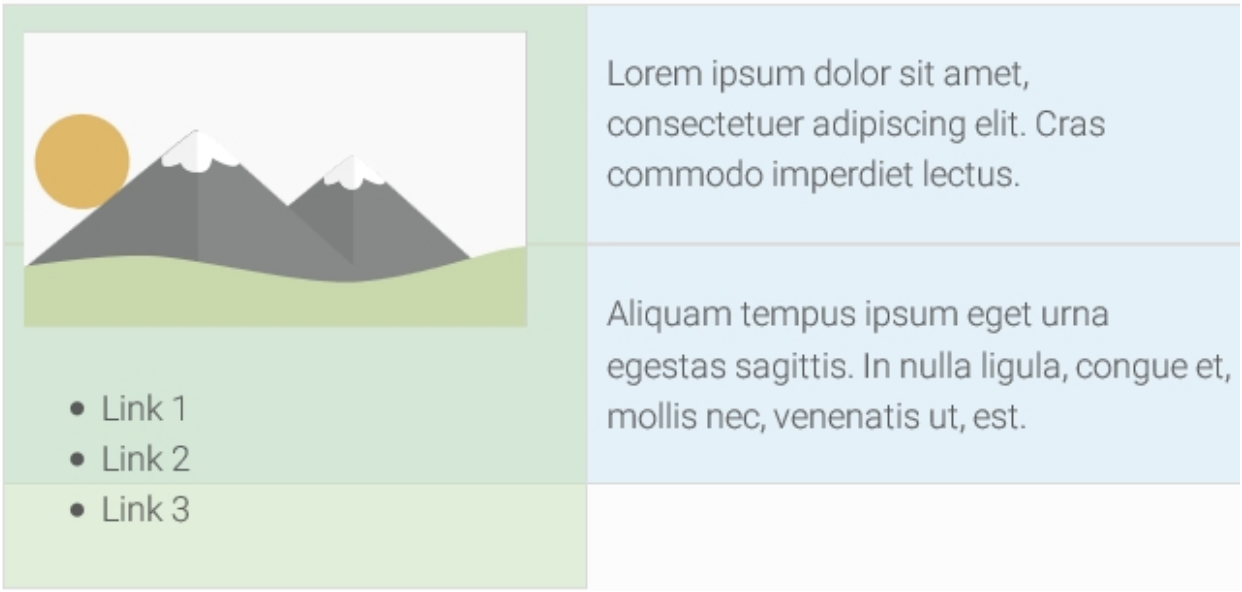
float hat nur diese beiden Werte: *float: left* und *float: right*. Es gibt kein float: center. Um Elemente – z.B. eine Seitennavigation mit next und prev – in der Mitte ihres Blocks zu zentrieren, eignet sich display: inline-block besser. Um die Spalten eines Layouts zu zentrieren, muss der umfassende Block selber zentriert werden. Erst display: flex ist eine solide Lösung für das Zentrieren von Spalten.

Blöcke mit float

float setzt Blöcke an den rechten oder linken Rand – typischerweise Logos, die Sidebar, Inhalt oder Bilder. Der übrige Platz bleibt frei und wenn weitere Elemente platziert werden, fließt ihr Inhalt in diesen freien Platz.
Die Lage im Text bestimmt die obere Kante des float-Elements. CSS *margin* legt den Abstand innerhalb des umfassenden Blocks fest.

- Link 1
- Link 2
- Link 3

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Cras commodo imperdiet lectus.
Aliquam tempus ipsum eget urna egestas sagittis. In nulla ligula, congue et, mollis nec, venenatis ut, est.




```
.green { float: left; width: 30%; margin-right: 10px; }
…
<div class="green">
   <img src="…" width="…" />
   <ul> … </ul>
</div>

<section">Lorem ipsum dolor sit amet … </section>

<section>Aliquam tempus ipsum … </section>
```

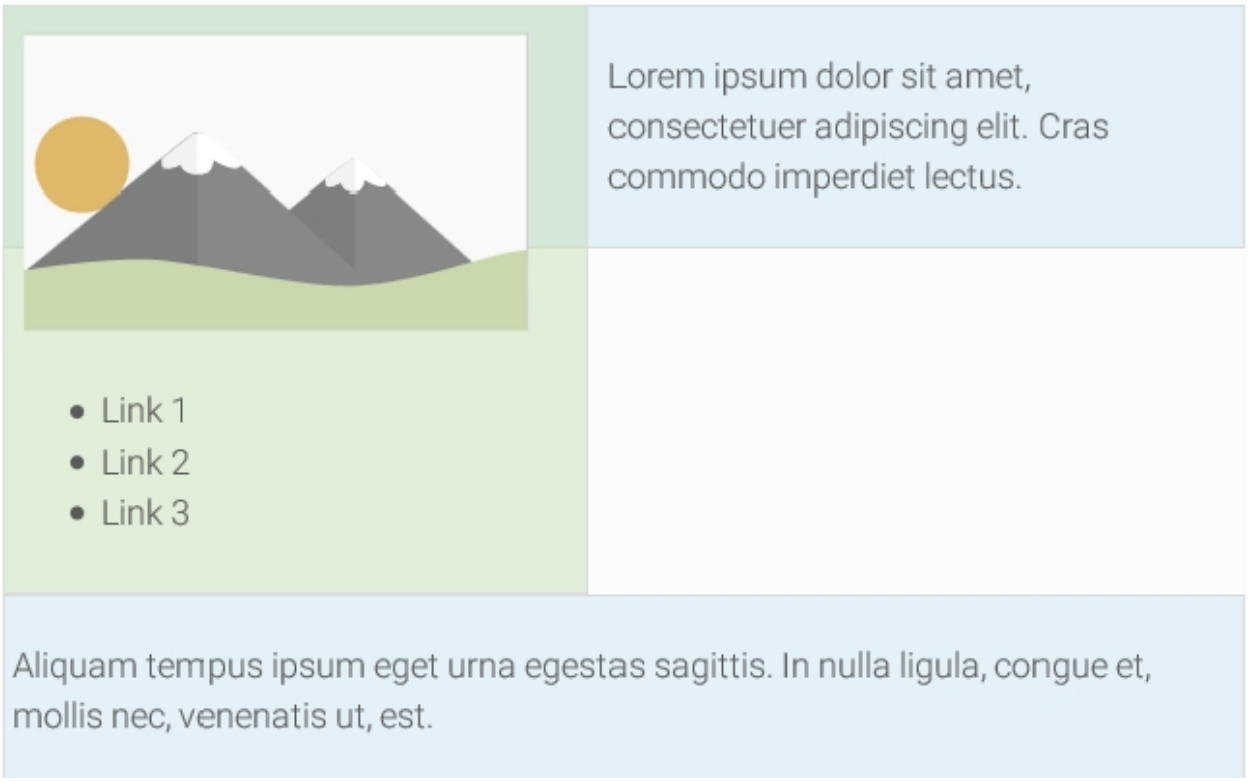
clear nach float: Umfließen beenden

Mit *float* positionierte Elemente verdrängen den Inhalt der folgenden Blöcke. Die Höhe des float-Elements wird durch seinen Inhalt oder durch die Angabe von *height* bestimmt. Am Ende des float-Blocks laufen die folgenden Elemente wieder über die gesamte Breite des umfassenden Blocks.
CSS clear hebt zwar das Umfließen auf, aber clear wird nicht auf den umfassenden Block und auch nicht auf das floating Element selbst angewendet. Die Zusammenarbeit zwischen float und clear ist nicht gerade intuitiv.
Erst CSS *clear* für ein Element, das im HTML-Code auf einen float-Block folgt, zwingt den Block unter den schwebenden Block. Dann läuft der Inhalt eines folgenden Blocks wieder über die gesamte Breite des umfassenden Blocks.

- Link 1
- Link 2
- Link 3

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Cras commodo imperdiet lectus.

Aliquam tempus ipsum eget urna egestas sagittis. In nulla ligula, congue et, mollis nec, venenatis ut, est.




```
.green { float: left; margin-right: 10px; }
.setclear { clear: both; }
…
<div class="green">
   <img src="…" width="…" />
   <ul> … </ul>
</div>

<section">Lorem ipsum dolor sit amet … </section>

<section class="setclear">Aliquam tempus ipsum … </section>
```

Problematisch wird die Kombination aus *float* und *clear* in den Editoren der Content Management Systeme, wenn Autoren ohne CSS-Kenntnisse ihre Bild rechts oder links schweben lassen: Dann zeigen sich Absätze u.U. als Treppen.

float und clearfix

Wenn in einem Container nur Elemente mit *float* liegen, entsteht eine hinterhältige Falle: Die Höhe des umfassenden Containers collabiert.
Mit wachsender Erfahrung entstanden Gegenmaßnahmen: Zuerst setzte man ein div mit *clear:both* an das Ende des Containers, dann ein *hr* mit *clear:both* und *visibility:hidden*.
Die modernest Technik heißt *clearfix*:

```
.clearfix:after {
  content: "";
  display: table;
  clear: both;
}
```

clearfix ist die CSS Klasse für das umfassende Element. Das ist die einfachste und sicherste Methode im Umgang mit Blöcken, die Element mit CSS float enthalten. clearfix wird allerdings nicht für das floating Element selbst angesetzt, sondern für den umfassenden Block.
Für IE 6 / 7 brauchte das clearfix noch eine Erweiterung:

```
.clearfix {
  *zoom: 1;
}
```

Diese Erweiterung können wir uns heute sparen.

CSS float

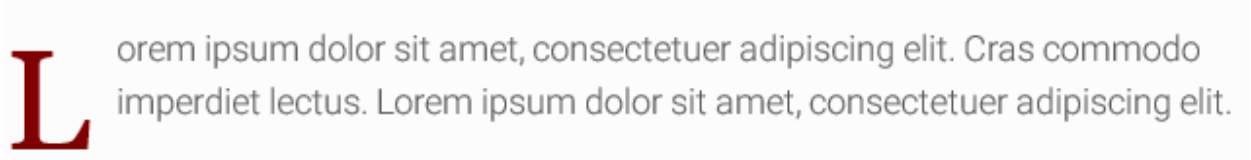
Erblich: Nein
float
bestimmt, dass ein Element an den linken bzw. rechten Rand des umfassenden Elements versetzt wird – ähnlich dem "Umfließen" von Bildern im Satzprogramm und in der Textverarbeitung. *float* ist eine von zwei Eigenschaften, mit denen die Positionierung eines Elements gesteuert werden kann. Die andere Eigenschaft ist *position*.
Werte
left | right | *none*
Bilder und Texte, die eingebettet in Texten oder anderen Elementen dargestellt werden, werden als "schwebende Elemente" (floating elements) bezeichnet.
left
platziert ein Element auf der linken Seite des Textes.
right
platziert ein Element auf der rechten Seite des Textes.
none
ist die Vorgabe und stellt ein Element dort dar, wo es im Text steht.
Für eine Box mit CSS float:left oder float:right muss immer auch eine Breite angegeben werden, ansonsten würde das schwebende Element immer so breit wie der umfassende Block statt rechts oder links am Rand zu sitzen. Bei Bildern mit float ist die Breitenangabe im <img>-Tag bereits enthalten; bei schwebenden Texten muss die Breite explizit angegeben werden.

Bilder mit float

Zu den klassischen Anwendungen von CSS *float* gehört das Umfließen von Bildern mit Text – ähnlich wie beim Layout in InDesign, Quark oder Word.

```
<style>
.right {
   float: right;
   margin-left: 1em;
}
</style>
…
<p><img class="right" … /> Zu den klassischen Anwendungen …</p>
```

Initiale mit float




```
.initiale {
    float: left;
    font: normal 4em/110% Georgia, Times, serif;
    color: maroon;
    margin-right: 0.2em;
}
…
<p><span class="initiale">L</span>orem ipsum dolor sit amet, …
```

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Cras commodo imperdiet lectus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Hinweis: Die Initiale kann auch ohne Eingriff in den HTML-Code durch das Pseudoelement first-letter erzeugt werden.

CSS *float* nimmt Blöcke aus dem normalen Fluss des Dokuments und setzt sie an die linke oder rechte Seite des umfassenden Blocks, während der restliche Text des umfassenden Blocks um sie herumfließt.
Obwohl CSS uns mit *position* ein speziellen Mechanismus für die Positionierung zur Hand gegeben wurde, ist *float* die eleganteste Alternative für das Layout von HTML-Seiten.
Dabei war CSS float ursprünglich nicht für die Positionierung gedacht, sondern für das klassische »Bild-wird-vom-Text-umflossen«. float hat sich zur beliebtesten Technik für ein Layout-Raster entwickelt:

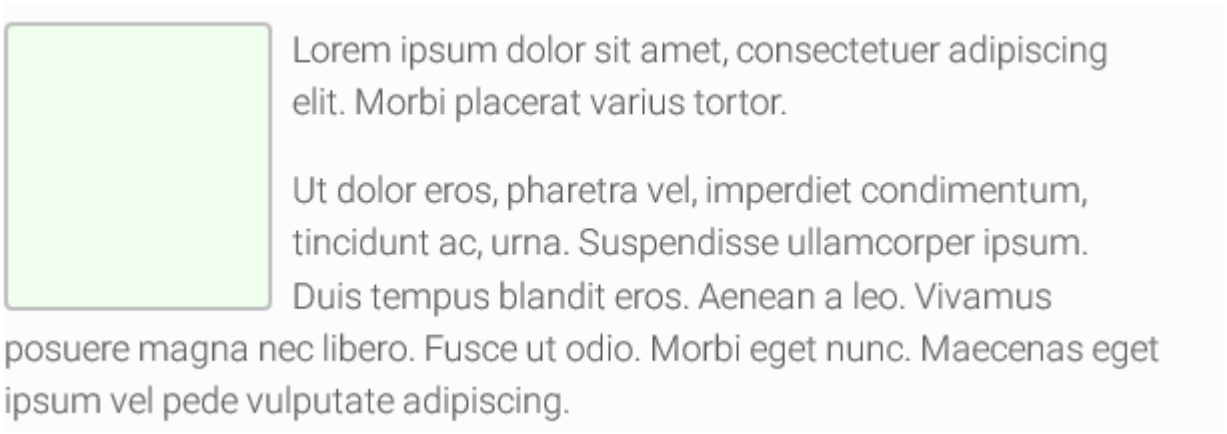
1. weil *float* intuitiver ist und immer weniger unter Browserfehlern gelitten hat als *position*.
   Heute sind sich die Browser sehr nah gekommen und die Browserfehler spielen kaum noch eine Rolle, aber *position* eignet sich nicht mehr für ein modernes Layout-Raster.
2. weil kleinere Blöcke aus Text und Bildern bei Änderungen der Seite einen besseren Zusammenhang behalten.
   Blöcke mit float ändern ihre Position zusammen mit dem Text automatisch – sie »fließen« mit.
3. weil Bilder, Sidebars oder andere Blöcke an der linken oder rechten Seite ihres umfassenden Blocks und an der richtigen Position bleiben, egal wie lang die Blöcke werden.
4. weil *clear* alles unter alle schwebenden Blöcke zwingt – ganz gleich, wie lang der vorangehende Block wird,
5. weil Blöcke mit CSS float an jeder beliebigen Stelle des HTML-Dokuments auftauchen dürfen.

Bilder von Text umfließen lassen

float darf so ziemlich alles positionieren: Blockelemente wie Textabsätze und div-Container genauso wie Inlineelemente, gleich ob Bilder und *span*.

```
<p style="float: left; width: 250px; ">
<div style="float:left; width 300px">
<img style="float:left; margin-right: 10px">
```

Schwebende Bilder halten auch in flexibel breiten Blöcken, wenn das Browserfenster verkleinert oder vergrößert wird.

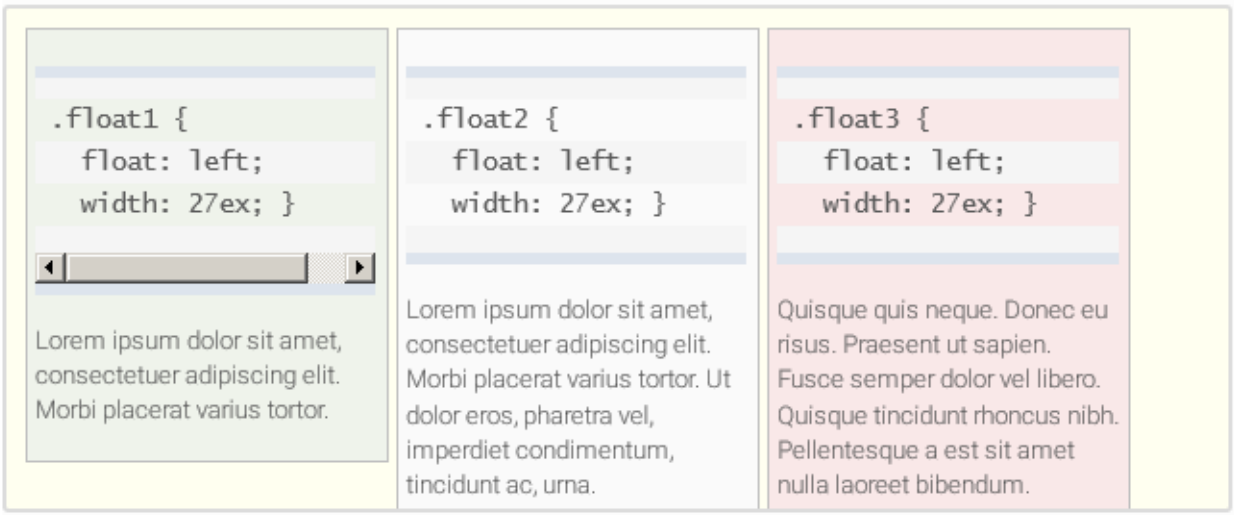



```
<!DOCTYPE html>
<html lang="de">
<head>
<title> CSS-Eigenschaft float für Bilder </title>
<style>
    body { margin: 0 9em; }
    img { float: left; margin-right: 1em; }
</style>
</head>
<body>
   <p> <img … width="240" height="182" />Lorem ipsum dolor …</p>
   <p>In pharetra. Praesent aliquet neque. Integer  … </p>
</body>
</html>
```

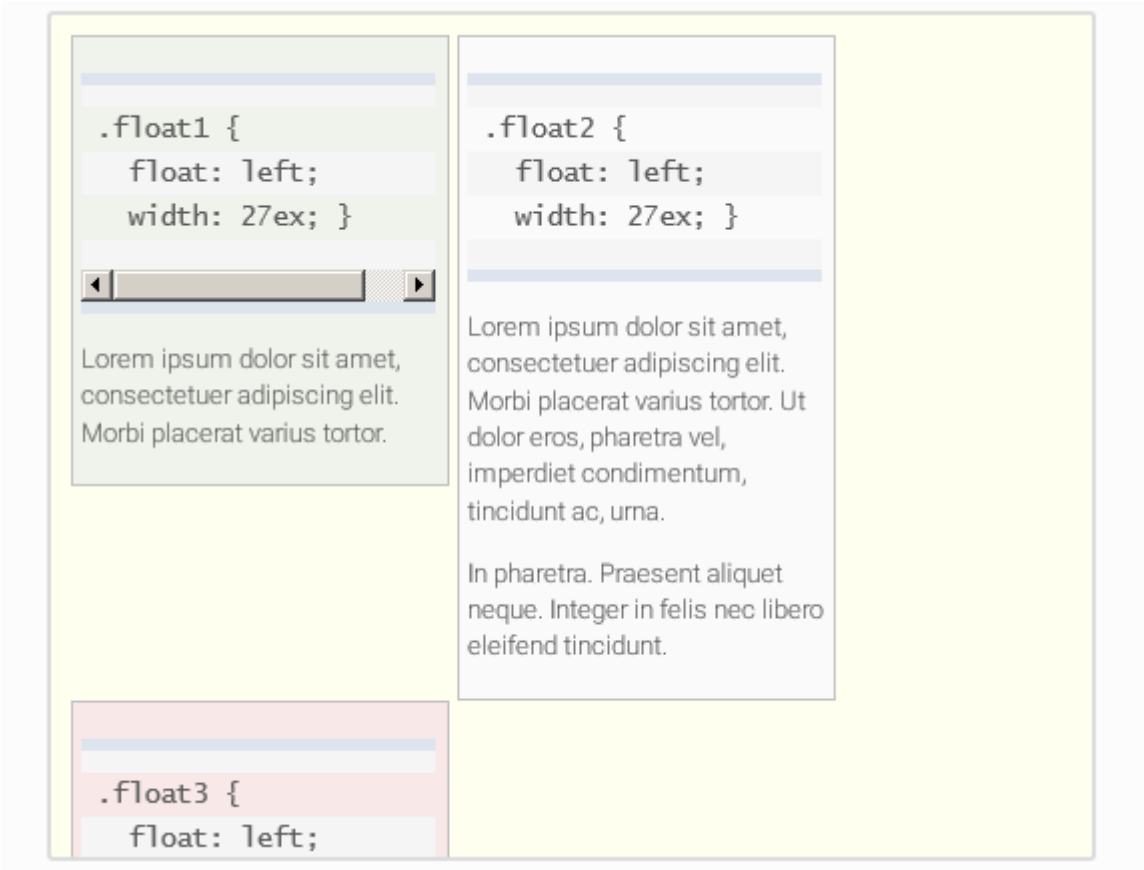
Wichtig! Damit z.B. eine Überschrift unter dem Text wieder in Form kommt – sprich: wieder am rechten Rand beginnt, muss *style="clear: both;"* für das HTML-Element (z.B. h3 { clear: both; }) notiert werden.

Spalten-Layout mit CSS float

Auf diese Weise lassen sich auch HTML-Blöcke für das Layout der Seite für große Monitore nebeneinander setzen. Wenn das Browserfenster kleiner ist oder die Auflösung nicht für alle Spalten reicht, kollabieren die Spalten und erscheinen untereinander.



CSS float wurde schnell zu unserem beliebtesten Layoutwerkzeug, denn *float* kann unseren Besuchern mit großen Monitoren ein Drei- oder Vier-Spalten-Layout bieten, während das Layout für Besucher mit kleinen Monitoren zu einem Zwei-Spalten-Layout mutiert.




```
<style type="text/css" media="screen">
    .float1 { float: left; width: 12em; }
    .float2 { float: left; width: 24em; }
    .float3 { float: left; width: 12em; margin-bottom: 1em; }
</style>
…
<div class="float1"> …  </div>

<div class="float2">
```

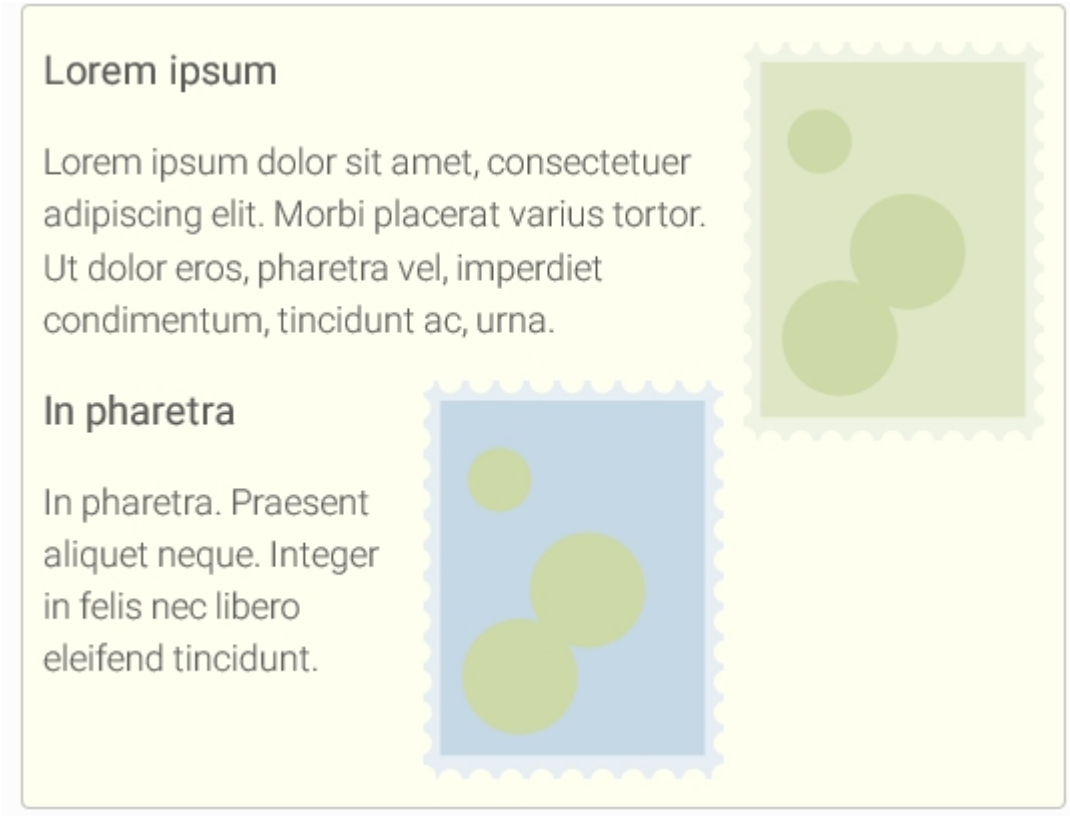
```
   <p>Lorem ipsum   </p>
   <p>In pharetra.  </p>
   <p>Pellentesque  </p>
</div>

<div class="float3"> …  </div>

<div class="float4">  …  </div>
```

CSS float: Treppen beim Umfließen von Bildern

Wenn Textabsätze mit links oder rechts schwebenden Bildern kürzer als das Bild mit float sind, entstehen in den folgenden Textblöcken Stufen.



Damit der folgende Absatz wieder bis an den rechten (oder linken Rand rutscht), braucht er ein *clear*.
clear weist vier Werte auf: *left*, *right*, *both* und *none*. *clear: right* als Stil für die Überschrift sorgt dafür, dass eine Überschrift so weit nach unten geschoben wird, dass sie unter einem rechts schwebenden Element landet und wieder bis zur äußerst rechten Seite des umfassenden Blocks läuft.

CSS clear: float beenden

CSS *clear: left* schiebt ein Element so weit nach unten, dass es unter einem links schwebenden Element steht, clear: both versetzt das Element sowohl unter rechts als auch unter links schwebende Boxen. Wenn die CSS-Eigenschaft float in einem Dokument benutzt wird, definiert man z.B. alle Überschriften vorsichtshalber mit der Eigenschaft *clear: both*, damit sie nicht neben schwebenden Blöcken, sondern immer unter ihnen landen.


```
<style type="text/css" media="screen">
   img { float: right; margin-left: 1em; }
   h3 {clear: right; }
</style>
…
<h3><img … width="150" height="163" /> Lorem ipsum</h3>
<p>Lorem ipsum dolor sit amet, … </p>
<h3><img … width="150" height="120" /> In pharetra</h3>
<p>In pharetra. Praesent aliquet neque. … </p>
…
```

Das Umfließen von Bildern ist auf responsiven Webseiten komplex. Wir müssen das Verhältnis zwischen Bildern und Text im Auge behalten, wenn die Seite auf einem kleinen Monitor angezeigt wird.
Wenn Bilder vom Text umflossen werden sollen, ist es besser, *CSS float* erst bei größeren Viewports anzuwenden (dazu mehr unter CSS Media Queries).

display: flex als genereller Ansatz

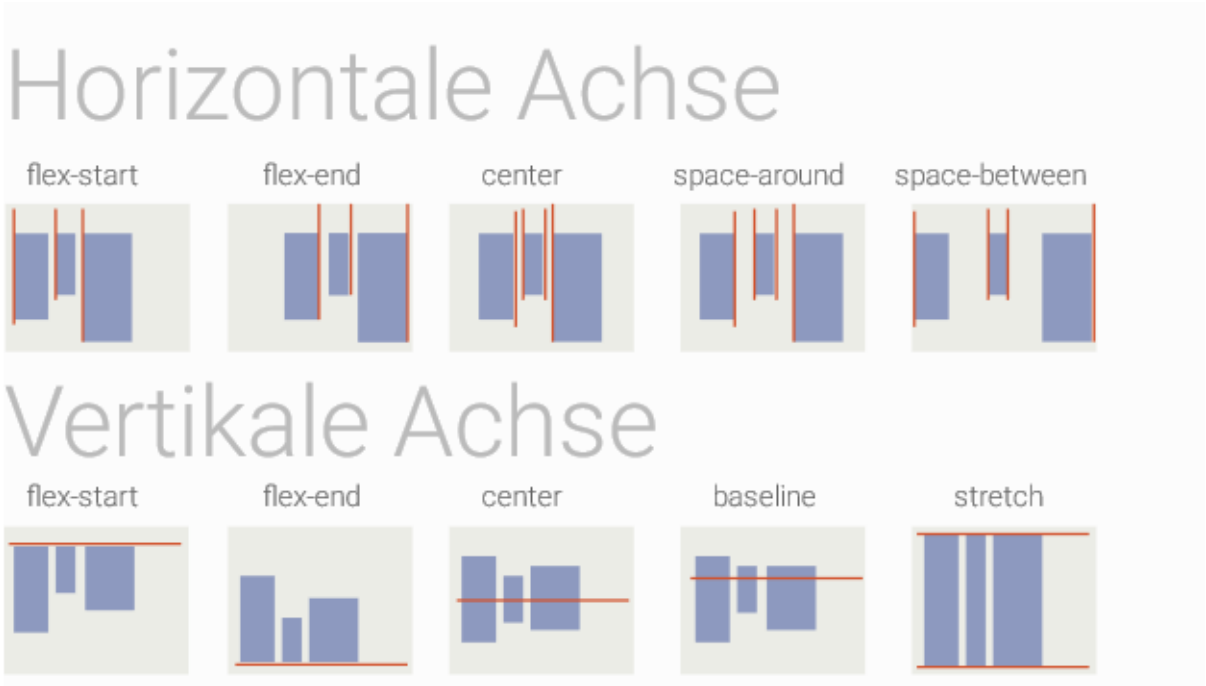
==========

Der modernste Ansatz für das Positionieren – sowohl von Layout-Elementen als auch von Boxen wie Preisblöcken – bringt mit einfachen CSS-Regeln alle Elemente in ein Layout-Raster (das Grid), berechnet die Abstände zwischen den Boxen automatisch und bringt alle Boxen synchron auf eine Höhe.
display: flex bzw. *display: flexbox* (alte Syntax) ist das erste Werkzeug in CSS, das speziell für ein Layout-Raster (Grid) geplant wurde.
Der Wermutstropfen hier liegt auf IE8/IE9, die *display: flex* noch nicht unterstützen. Ein mögliches Fallback für IE8/IE9 ist *float* oder *display: inline-block*.

display:flex erzeugt Flexboxen aus allen direkten Nachkommen des Elements und richtet die flexiblen Boxen links, rechts, zentriert, oben oder unten aus – und zwar entlang der horizontalen (row) und vertikalen Achse (column).
Mit *space-around* und *space-between* wird der freie Raum zwischen den Flexboxen verteilt.
Positionieren mit dem Boxmodell war immer Fummelei. Erst durch jahrelange Erfahrung haben wir das Puzzle aus *display:absolute*, *display:relative* und *float* in den Griff bekommen.
display:flex bietet endlich eine einfache Lösung für das Problem, Elemente vertikal zu zentrieren. Und während das horizontale Zentrieren von Elementen mit CSS float ein Spagat war, kostet es mit display:flex gerade mal eine Zeile.
Alle modernen Browser bieten Support für display flex (ohne Browser-Präfix). Internet Explorer unterstützt flex ab IE10, allerdings in einer älteren Version der Syntax und nur mit Browser-Präfix.

Positionieren mit dem Flexmodell

Das Flexmodell hat das Ausrichten und Verteilen von Boxen und freiem Raum aus der Grafik abgekupfert, und basiert ebenso auf einer horizontalen und einer vertikalen Achse:



So sieht das Flexmodell für eine zeilenorientierte Verteilung (flex-direction:row) der Flexboxen aus. Äquivalent gibt es auch noch die spaltenorientierte Verteilung von Flexboxen (flex-direction: column).
Alle direkten Nachkommen eines Containers mit CSS *display:flex* werden zu Flex-Items und reihen sich in einer Zeile auf. Die Nachkommen müssen nicht einmal vom selben Typ sein. Per Vorgabe erzeugt *display:flex* eine Zeile von Boxen.
Die Eigenschaften des Flexmodells werden dem umfassenden Container zugewiesen und verteilen Flexboxen in Zeilen oder Spalten, geben allen Flexboxen dieselbe Höhe und nehmen uns so nebenbei die Berechnungen für den Platz zwischen den Boxen ab.
flex-direction
row (default)| column
flex-flow (Kurzschrift für flex-direction und flex-wrap)
row wrap | column wrap
flex-wrap
nowrap (default) | wrap | wrap-reverse
justify-content
flex-start (default) | flex-end | center | space-between | space-around
align-items
flex-start (default)| flex-end | center | baseline | stretch
align-content
flex-start | flex-end | center | space-between | space-around | stretch (default)
Die Reihenfolge der Elemente einer Flexbox kann durch *order* beliebig festgelegt werden.
order
.flexitem:nth-child(n) {order: m} (Beispiel für flex order)

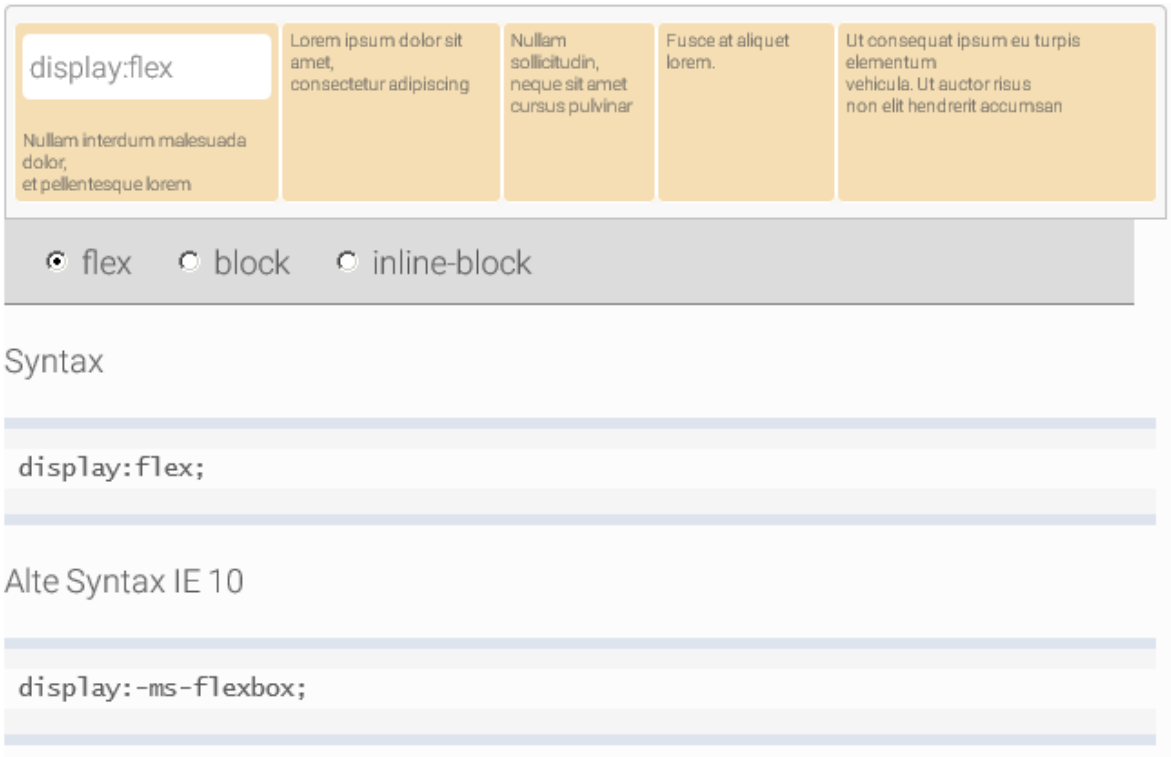
Flexmodell und Browserunterstützung

- IE ab Version 10 (alte Syntax)
- Firefox ab Version 28
- Chrome ab Version 33
- Safari ab Version 7
- Opera ab Version 22

Das Flexbox-Modell ist noch nicht in allen Browsern vertreten (nicht IE8 und IE9) und ist noch nicht vollständig ratifiziert, da die Standardisierer noch über die Funktionen diskutieren.
IE 10 braucht den Browser-Präfix und agiert zudem noch nach der alten Syntax : Flexible Box-Layout (Flexbox) in Internet Explorer 10

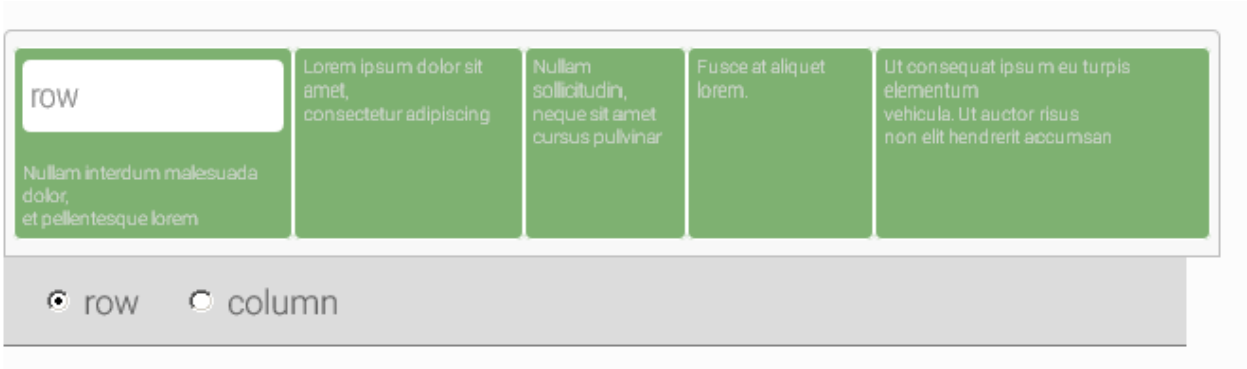
display:flex

Die Eigenschaften *display:block* und display:*inline-block* kennen wir schon aus CSS2. *display:flex* ist in CSS3 aufgenommen worden und muss für Webkit-Browser und IE noch mit dem Browser-Präfix geschrieben werden.
Ohne jegliche weitere Eigenschaft werden Flexboxen in einer Zeile aufgereiht. Wenn der Viewport des Browsers kleiner wird, werden die Flexitems zusammengequetscht.
Wenn der Platz reicht und die Größe der Boxen nicht durch CSS begrenzt ist, hängen die Flexboxen auf der linken Seite des Flex-Containers.



flex-direction

Verteilt alle direkten Nachkommen in einer Zeile (bzw. in einer Spalte, wenn zusätzlich *flex-direction:column* gilt).



flex-wrap

Packt alle Flexboxen in eine Zeile bzw. Spalte (nowrap), bricht die Zeile / Spalte um (wrap) oder kehrt die Reihenfolge der Flexboxen um (wrap-reverse).

IE 10 braucht display:inline-block für die Flex-Items, damit -ms-flex-flow: wrap funktioniert.

justify-content

richtet Flexboxen entlang der Hauptachse (links, rechts, zentriert) und verteilt den freien Raum zwischen den Flexboxen mit *space-between* bzw. *space-around*.

Neue Syntax

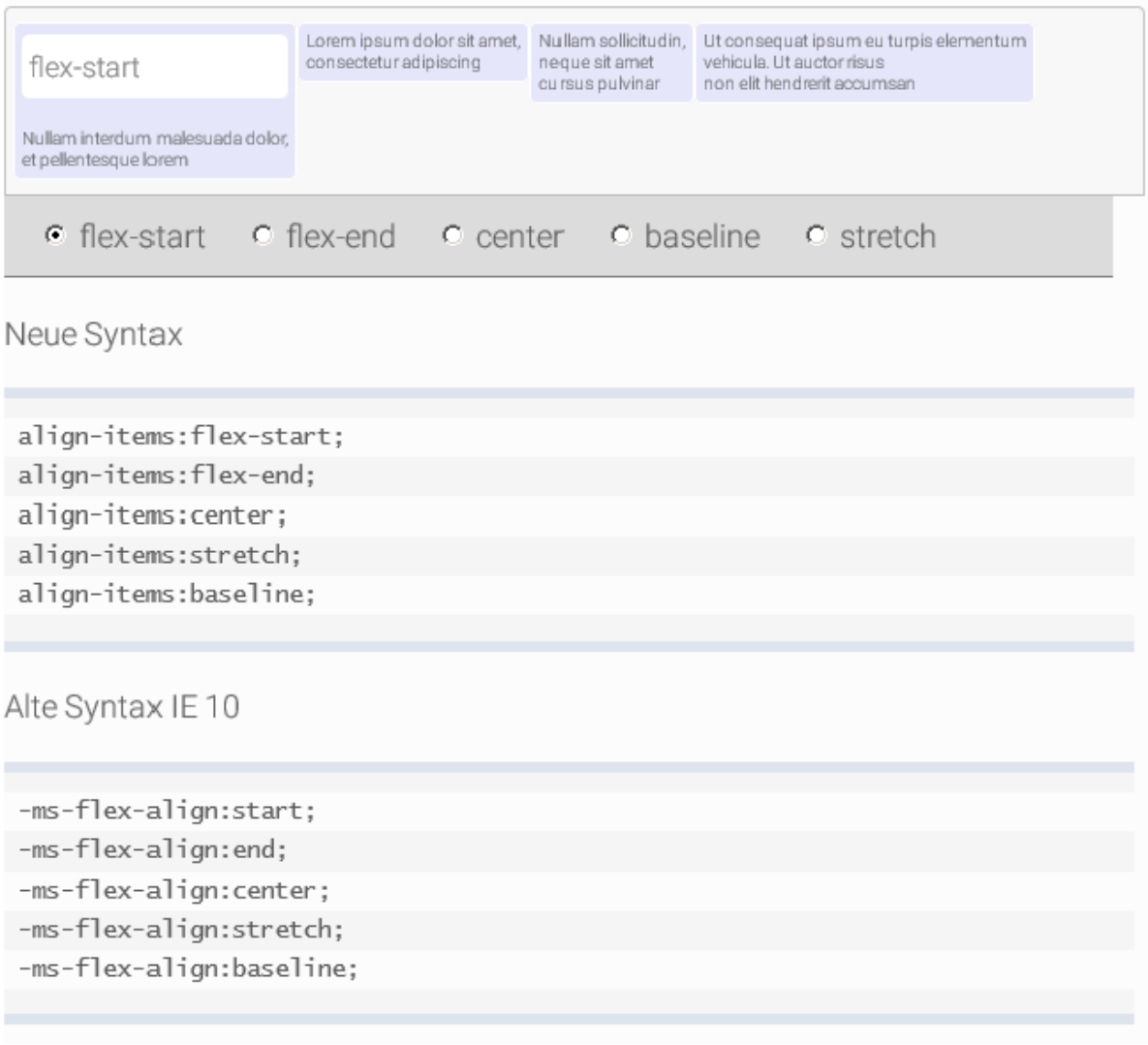
```
justify-content:flex-start;
justify-content:flex-end;
justify-content:center;
justify-content:space-between;
justify-content:space-around;
```

Alte Syntax für IE10

```
-ms-flex-pack:start;
-ms-flex-pack:end;
-ms-flex-pack:center;
-ms-flex-pack:justify;
-ms-flex-pack:distribute;
```

align-items

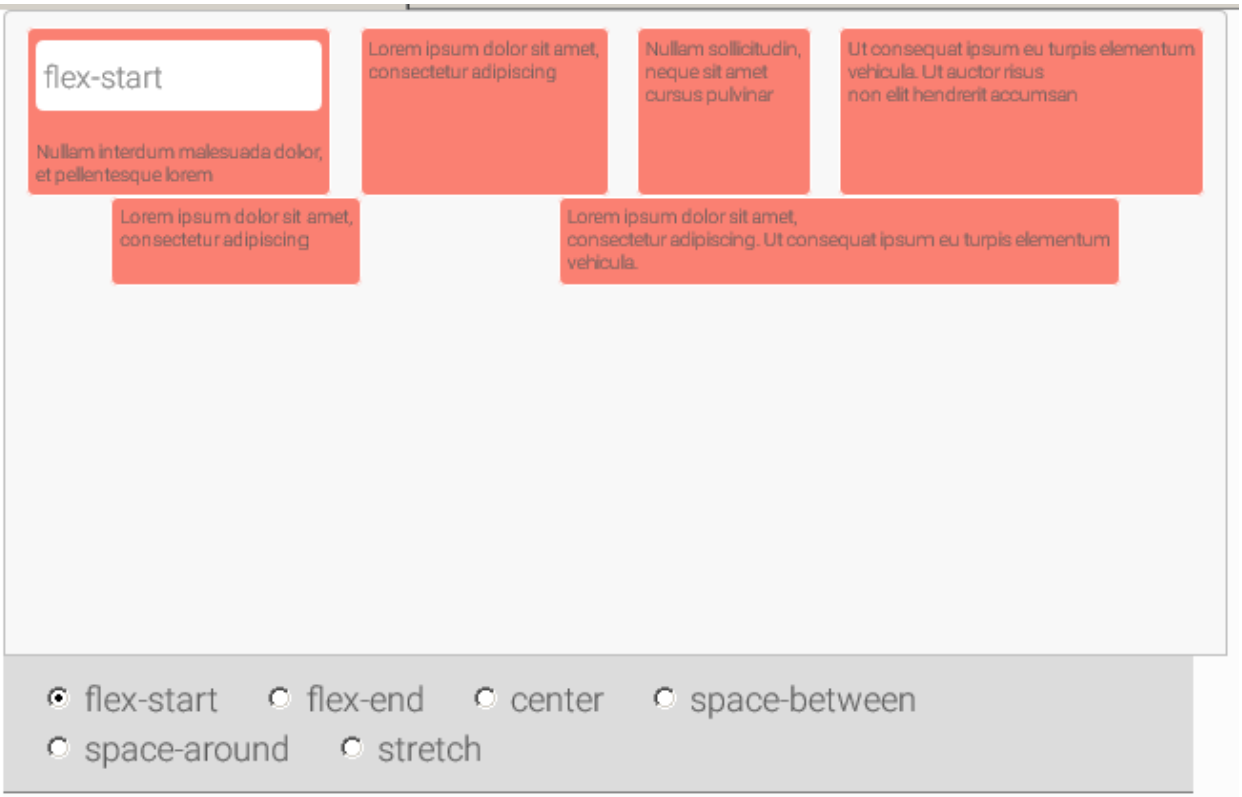
richtet Flexboxen an den Kanten oben, unten oder zentriert entlang der zweiten Achse aus (so wie justify-content entlang der Hauptachse).



align-content

verteilt die Flexboxen anhand ihres Inhalts auf der zweiten Achse (so wie *justify-content* die Boxen auf der Hauptachse verteilt). Mit *flex-direction:row* verteilt *align-content* die Boxen also in der Vertikalen.
Einen sichtbaren Effekt hat *align-content* nur, wenn mehr als eine Reihe bzw. eine Spalte vorhanden ist.
Die Vorgabe ist stretch, so dass die Boxen die volle Höhe des Flex-Containers einnehmen.

```
#acbox {
   height:320px;
   display:flex;
      display:-ms-flexbox;
   flex-flow: row wrap;
      -ms-flex-wrap: row-wrap;
   justify-content:space-around;
      -ms-flex-pack:justify;
   align-content:flex-start;
      -ms-flex-line-pack:start;
}
```



flex-start flex-end center space-between
space-around stretch

Neue Syntax

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

Alte Syntax IE10

```
-ms-flex-line-pack:start;
-ms-flex-line-pack:end;
-ms-flex-line-pack:center;
-ms-flex-line-pack:justify;
-ms-flex-line-pack:distribute;
-ms-flex-line-pack:stretch;
```

float vs inline-block vs display:flex

==========

Positionieren mit *CSS float* war lange Zeit das Lieblingsinstrument der Front-End-Entwickler. Aber die kleinen Fallen und das Frickeln mit dem clearfix setzen einen Trend zu *display:inline-block*.
Die vorgegebene Breite von Blöckem mit display:inline-block hängt vom Inhalt ab und nicht von der Breite des umfassenden Containers. Aber auch das Positionieren mit *display:inline-block* hat seine Schattenseiten: Die Inline-Eigenschaften führen dazu, dass ein Leerzeichen zwischen den HTML-Blöcken zu einem Abstand führt.
Also überschlagen wir display:inline-block und setzen auf display:flex. Ein weiterer Vorteil von *display: flex* – Blöcke mit *display: flex* werden automatisch gleich hoch bzw. gleich breit.

Positionieren mit *float* ist der Standard im Webdesign. float ist aber auch die häufigste Ursache für Kopfschmerzen und Schwindelgefühl.
Das responsive Webdesign profitiert *CSS float*: float bietet Besuchern mit großen Monitoren ein Vier-Spalten-Layout, das auf kleineren Monitoren automatisch zum Zwei-Spalten-Layout mutiert.
float nimmt HTML-Blöcke aus dem Fluss des Dokuments. Eine Reihe von zusätzlichen CSS-Regeln (die alles andere als intuitiv sind) muss darum kümmern, dass nach einem Block mit *float* der normale Fluss des Dokuments wieder aufgenommen wird.
display:inline-block ist eine Alternative zum Positionieren mit *float*. Der umfassende Container wächst mit, so dass Tricks und Hacks wie ein clearfix außen vor bleiben können.

Boxen mit *display: inline-block* liegen wie Text auf einer Grundlinie unten. Unterschiedlich hohe Boxen führen darum zu einem dynamischen Auf- und Ab. Das einfach Mittel gegen das Auf und Ab der Boxen ist *vertical-align: top* – das hängt die Boxen an die obere Grundlinie.

Kein Layout-Werkzeug ohne Tücken

Beim Positionieren von Boxen mit *display:inline-block* zählen die Leerzeichen zwischen den Blöcken. Also muss entweder das vollständige Markup in eine Zeile geschrieben werden oder ein zweifelhafter Comment Hack herhalten.

```
<div id="cbox">
    <div class="cbox">
        <div></div>
        <h5>BOX 1</h5>
        <p>Als es klingelte, …</p>
    </div><div class="cbox">
        …
    </div><div class="cbox">
        …
    </div>
</div>
<div id="cbox">
    <div class="cbox">
        <div></div>
        <h5>BOX 1</h5>
        <p>Als es klingelte, …</p>
    </div><!--
    --><div class="cbox">
        …
    </div><!--
    --><div class="cbox">
        …
    </div>
</div>
```

Ein Plus gegenüber *float* hat das Positionieren mit *display:inline-block*:

```
#cbox { text-align: center}
```

zentriert die Boxen in ihrem umfassenden Block – das wirkt adrett, wenn der umfassende Block in einem responsiven Design schmaler wird.

Genauso wie beim Positionieren mit *float* rutschen die Boxen automatisch untereinander.

Blöcke mit *inline-block* wachsen – genauso wie bei Boxen mit *float* – nicht automatisch synchron auf dieselbe Höhe. Immerhin spannen sie den umfassenden Block stets zuverlässig auf.

```
#cbox {
    text-align: center
}
#cbox {
    margin: 20px auto
}
.cbox {
    display: inline-block;
    width:90%; min-width: 160px; max-width: 190px;
```

```
    border:1px solid silver;box-sizing: border-box;
    margin: 10px
}
.cbox h5, .cbox p {
    text-align: left
}

@media only screen and (min-width:460px){
    .cbox { width:30%; }
    .cbox { vertical-align: top; }
}

@media only screen and (min-width:680px){
    .cbox:last-child { margin: 10px 0 10px 10px}
    .cbox:first-child { margin: 10px 10px 10px 0}
}
```

Auch *display: inline-block* ist kein Allheilmittel in einem modernen Grid-Layout für ein responsives Layout, sondern eine Alternative zu float. Weder float noch display:inline-block waren als Layout-Werkzeuge geplant.