Box model en positionering

Om de CSS-opmaaktaal daadwerkelijk voor het **lay-outen** van webpagina's in te schakelen, is het erg belangrijk om de **schikkingsmethoden** van de verschillende HTML-elementen te kennen. Weet dat niet iedere browser elke CSS- en HTML-wetmatigheid in acht neemt en sommige instructies dan ook verkeerd uitgevoerd worden. Bovendien laat het W3C hier en daar wat ruimte voor interpretatie – browserfabrikanten zijn in die zin soms wat vrijer om het één of ander op een bepaalde manier te implementeren.

Dit cursusdeel biedt je een **overzicht van de verschillende methoden en regels** die het schikken van deze elementen verwezenlijken. Daarnaast lijst het deel enkele typische browsereigenaardigheden en -oplossingen op.

Het box model

Om ook maar iets van de verschillende schikkingsmethoden te kunnen begrijpen, dien je allereerst het **box model** te doorgronden. Dat systeem is gelukkig niet al te moeilijk. Als je weet dat **elk element in een HTML-document** als een **rechthoekige vorm** beschouwd wordt, ben je al redelijk ver. Elk van deze 'boxes' of rechthoeken heeft een **inhoudsgebied - 'content',** waarrond wat **opvulling of binnenste ademruimte – 'padding',** dus – zit, een **rand of boord – 'border'**-definities – en wat **marges of buitenste ademruimte – 'margin'.**

De eigenschappen die een box samenstellen:

- width: de breedte van het element
- height: de hoogte van het element
- border: de rand van het element
- padding: de ademruimte binnen het element
- margin: de ademruimte buiten het element

Van binnen naar buiten toe bestaat de box dus uit

1. de inhoud (content) van het element
2. de padding
3. de border
4. de margin

Enkele **wetenswaardigheden**…

1 **Margins** zijn steeds **transparant**.

2 **Borders** bestaan er in een heel arsenaal aan **stijlen** – zo heb je niet enkel ononderbroken randen, maar ook gestipte of gestreepte randjes. Voor een volledig overzicht van randstijlen, verwijzen we je graag naar de site van w3schools. Naast deze randstijl kunnen boorden ook een **kleur** en een **dikte** krijgen.

3 **Achtergrondkleuren of -afbeeldingen worden op het hele door de border omsloten gebied toegepast.** Ook de padding wordt dus met zulke achtergrondkleuren gevuld.

4 De door de browser **gereserveerde breedte- en hoogteruimte** van elke box zijn gelijk aan de breedte en hoogte van dat gebied dat **door de margins begrensd** wordt. Merk op dat dat vaak niet met de breedte van het zichtbare inhoudsgebied overeenstemt – marges zijn immers steeds transparant.

5 Merk op dat je voor de **boven-, rechter-, onder- en linkerzijde verschillende margin-, border- en padding** kan instellen. **Margins kan je met negatieve waarden opzadelen, maar voor borders en padding lukt dat niet.**

Je bent niet verplicht margins, borders en padding te definiëren. Omdat **elke browser** wel eens **verschillende standaardwaarden** voor margins, borders en padding hanteert, is het een goed idee deze waarden toch steeds te expliciteren. Dat kan op enkele manieren.

1 Je kan voor elk HTML-element margin-, border- en padding waarden opgeven, en dat **handmatig**. De kans is echter groot dat je enkele elementen over het hoofd ziet en je alsnog met verschillende browserweergaven zal kampen.

2 Je kan met een eenvoudig stukje CSS-code **alle margin-, border- en padding-breedtes annuleren**.

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

Omdat nu ook **niet elk element margin-, border- of padding-breedtes** heeft – zo kunnen de ietwat administratievere <html>-, <head>-, <title>-, <meta>-, <style>-, <script>-, <base>- en <param>-tags noch margin-breedtes, noch border-breedtes, noch padding-breedtes krijgen, maar ook concretere afbeeldingen kunnen geen opvullingsbreedtes krijgen – reken je nu ook weer op de tolerantie van de browser en boet je zelfs een beetje aan snelheid in. Dat verlies is echter zo goed als verwaarloosbaar.

Reset- en normalisatie bestanden

De **meest elegante oplossing** houdt het gebruik van een **reset bestand** in. Ook hier kan je enkele richtingen uit.

1 **Gebruik een nogal harde reset methode die je als ontwerper verplicht om elke stijl op te geven.** Het voordeel moge duidelijk zijn – het gros van de browserverschillen wordt ongedaan gemaakt én **je houdt als ontwerper alle touwtjes stevig in handen**. De tijd die je in het veelvuldig definiëren van je margin-, border- en padding-breedtes stopt, win je tijdens het cross browser testen terug. Eric Meyer ontwikkelde zo'n krachtig resetbestand.

2 **Gebruik een normaliseer bestand.** Net zoals een reset bestand, stelt dit ding allerlei margin-, border- en padding-breedtes op eenzelfde waarde in. Daarenboven zijn de **opgegeven CSS-definities nuttiger** in die zin dat ze niet enkel de optische browserverschillen wegstrijken, ook zit je meteen met een **startsituatie** die vormelijk al wat **aantrekkelijker** is en waarmee je meteen aan de slag kan. Het 'normalize.css'-bestand doet mooi dienst.

3 In een verder **gevorderd** stadium kan je ook een **eigen reset bestand** definiëren!

Een box kan op zijn beurt ook een willekeurig aantal boxes bevatten. Op die manier ontstaat er een hiërarchie – 'nesten' heet dat in vaktermen. Het browservenster of het <body>-element werpt zich als de **parent van elk van deze boxes in die hiërarchie** op.

Block-level-elementen en inline-elementen

Men maakt het onderscheid tussen een **tweetal box typen**. Het verschil tussen **block-level- en inline-elementen** is vooral belangrijk voor de standaardweergave van paginaonderdelen.

1 Op webpagina's neemt een **block-level-element** steeds een **rechthoekig gebied** in beslag. De letterlijke vertaling is blokniveau-element of blok-element. Veelgebruikte block-level-elementen zijn -elementen of 'divisions' voor delen van een webpagina, de <h1>- tot en met <h6>-elementen voor koppen, de <p>-tag voor alinea's en -, <ul> en <li> voor respectievelijk genummerde en ongenummerde lijsten en hun lijstitems. **Block-level-elementen nemen standaard de volledige beschikbare breedte in beslag.** Staan ze in de <body>-tag, dan zijn ze dus even breed als het browserscherm. Bevinden zulke block-level-elementen zich in een ander block-level-element, dan wordt de breedte van het child element door diens parent bepaald. **Een block-level-element verdraagt geen tweede block-level-element op dezelfde regel of op hetzelfde horizontale niveau - ze worden op elkaar gestapeld.**

2 **Een inline-element staat ergens op een tekstregel of tussen andere elementen.** Zulke elementen bevinden zich in principe steeds binnen een block-level-element. **De belangrijkste eigenschap is dat een inline-element zowel horizontaal als verticaal kan worden verplaatst als de indeling van de omringende tekst of de positie van het block-level-element verandert.** In die zin houdt het dus steek dat **inline-elementen** zich **niet volgens de regels van het box model** gedragen. Bij deze elementen kan je **geen breedte of hoogte instellen en geen margin gebruiken**. Je kan enkel een beetje padding en een border gebruiken. Veelgebruikte inline-elementen zijn em ('emphasis') voor nadruk en strong voor extra nadruk, span, a en img.

De display eigenschap laat je toe HTML-elementen van weergavetype te veranderen. Met CSS kan je bijvoorbeeld een block-level-element als een inline-element weergeven.

```
h1 {
      display: inline;
}
strong {
      display: block;
}
```

Merk op dat wanneer je de display eigenschap op de 'none'-waarde instelt, de browser geen box tekent. De browser doet alsof het element niet bestaat en verbergt het. Wanneer deze bewuste box nog andere elementen bevat, worden deze ook verborgen, ook al definieerde je voor deze child elementen andere waarden voor de display eigenschap.

Wanneer je de waarde van de display eigenschap op 'inline-block' instelt, combineer je beide systemen. Zulke elementen staan nog steeds op een horizontale regel, maar vormen wèl een nieuwe box. Die dingen kunnen dus weer van **verticale padding en margin** voorzien

worden. Deze eigenschap is dus een beetje the best of both worlds. Het zal de elementen in lijn met elkaar plaatsen, én je kan de box model eigenschappen gebruiken. We passen dit vaak toe op horizontale navigatiebalken.

Meestal laat je de display waarde ongemoeid. Hoewel er voor tabellen en lijsten nog andere waarden voor deze display eigenschap bestaan, behandelt dit cursusdeel ze niet. Uiteindelijk worden deze specialere weergaven op een sterk vergelijkbare manier als block-level-elementen of inline-elementen behandeld – w3schools licht je verder in.

Geneste block-level-elementen

Block-level-elementen gedragen zich als een **container** voor eender welke box erin.

```
<div>
     Lorem ipsum dolor sit amet, consectetur adipiscing elit.
     <p>Aliquam nisi libero, iaculis eu commodo vel, porta at tortor.
     </p>
</div>
```

Koppel dit HTML-codestukje bijvoorbeeld aan volgende CSS-regels.

```
     div {
          width: 150px;
          height: 150px;
          background-color: beige;
     }
     p {
          background-color: green;
     }
```

Bekijk voorbeeld 1 in je browser.

Het -element voorziet een omsluitende blok voor zowel de paragraaf als het eerste 'losse' zinnetje. De paragraaf is op haar beurt opnieuw een omvattende blok voor het stukje tekst dat ze omsluit.
Merk op dat het **losse zinnetje** een **inline-element** betreft, maar de volgende **paragraaf** alsnog op een **nieuwe regel** begint. De losse zin wordt met andere woorden ook door een 'block box' omsloten! Men noemt dit **anonieme block boxes**, en ze worden gebruikt om het positioneringsproces te vereenvoudigen. Klinkt ingewikkeld, maar als je weet dat er standaard niets aan de linker- en rechterkant van elk block-level-element kan staan, zit het al snor.
Het omsluitende -element wordt gebruikt om beide ingesloten elementen te positioneren en de breedtewaarden ervan te bepalen. Wanneer we de paragraaf een breedte van 75 procent meegeven, zal het ding slechts de drie vierde van de breedte van de parent- in beslag nemen.

```
<div>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      <p>Aliquam nisi libero, iaculis eu commodo vel, porta at tortor.
      </p>
</div>
      div {
            width: 150px;
            height: 150px;
            background-color: beige;
      }
      p {
            width: 75%;
            background-color: green;
      }
```

Bekijk voorbeeld 2 in je browser.

De box-sizing eigenschap

Wat is nu **de uiteindelijke breedte** van een element? Waarom is dit belangrijk?
Als we **elementen naast elkaar willen positioneren** is het belangrijk dat we weten hoeveel plaats de elementen innemen.

Stel, we maken een webpagina van 800px breed. Als we twee kolommen willen maken met een ruimte van 20px tussen de kolommen, hoe breed kunnen deze kolommen dan zijn? We willen kolom 1 bijvoorbeeld 600px breed maken.

800px-600px-20px = 180px

We hebben dus 180px over voor de tweede kolom. Maar wat als we nu ook nog padding willen toevoegen? Gaat de berekening dan hetzelfde zijn? Of moeten we hier rekening mee houden?

box-sizing: content-box;

Standaard krijgen elementen de box-sizing waarde content-box mee. In dit geval bepaalt de width eigenschap van een element **de breedte van de inhoud** van dat element. We moeten er **zelf nog de borders en paddings en margins bij optellen**.

Wat houdt dit in voor bovenstaand scenario? We hebben een webpagina van 800px breed. We willen 20px tussenruimte (margin), we willen 10px ademruimte (padding) aan elke kant, en de kolommen moeten een rand (border) van 2px hebben. Kolom 1 heeft een breedte van 600px.

- Breedte pagina: 800px
- Randen: 4 borders van 2px: 8px
- Padding: 4 paddings van 10px: 40px
- Margin: één tussenruimte van 20px
- Kolom 1: 600px

Breedte kolom 1:
600px-4px-20px = 576px

Breedte kolom 2:
800px-600px-20px-(4px+20px) = 156px

De totale breedte komt dan uit op op een som van de borders, paddings, margins, en de breedtes van de elementen.

800px = (2px+10px + 576px + 10px+2px) + 20px + (2px+10px + 156px + 10px+2px)

box-sizing: border-box;

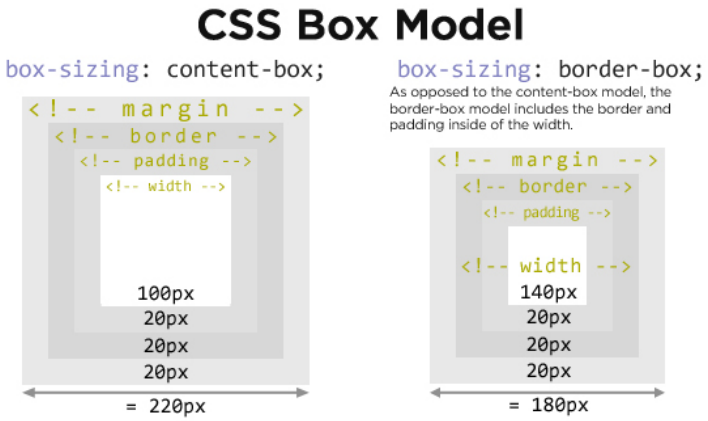
Een gemakkelijkere methode is de **border-box berekening**. Met deze box-sizing waarde zal de width eigenschap **de volledige breedte van het element** bepalen; de **browser zal zelf de berekeningen voor border en padding** uitvoeren.

Wat houdt dit in voor bovenstaand scenario?
800px = 600px + 20px + 180px

Het enige nadeel is dat deze eigenschap niet standaard wordt toegepast. Je kan deze eigenschap toepassen op elementen naar keuze, maar je kan het ook toepassen op alle elementen van een pagina door onderstaande **code-snippet bovenaan je CSS bestand** te plaatsen.

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

Een variant vind je hier terug.



Samengevat
Met de box-sizing eigenschap kan je bepalen hoe boxen gemeten worden.
De box-sizing eigenschap kan twee waarden aannemen:

- **content-box:** (standaard) bij het berekenen van de grootte van een box moet je padding en border en margin erbij optellen om de eigenlijke hoogte en breedte van het element te berekenen.
- **border-box:** padding en border worden mee gerekend in de hoogte en breedte van een element.

De meest intuïtieve manier om met boxen te werken is border-box.

Positionering

Men onderscheidt een vijftal positionering schema's of methoden – **statische positionering**, **zwevende (float) positionering, relatieve positionering**, **absolute positionering en verankerde/fixed positionering**. Elk van deze systemen wordt door **eigen regels** gekenmerkt. Elke box volgt één van deze regels. **Positionering methoden worden niet overgeërfd!** (daartoe dien je position: inherit; te gebruiken)

De position eigenschap kan volgende waarden hebben:

- static (default)
- relative
- absolute
- fixed
- inherit

Statische positionering, normale volgorde of 'normal flow'

De statische positionering is het **standaard positionering schema**. Het wordt **op elk element toegepast** zolang je geen CSS-definities inzake positionering opgeeft. Volgend codestukje is dan niet foutief, maar wel een beetje **overbodig**.

```
div#huls {
      position: static;
}
```

In dit schema volgen **block-level-elementen** elkaar **verticaal** op, te beginnen vanaf de bovenkant van de parent container en elkaar rechtstreeks onder elkaar opvolgend. Neem even dit HTML-codestukje.

```
<div id="boter">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      In ornare ligula sed lectus varius dapibus rutrum ante imperdiet.
</div>
<div id="kaas">
      Curabitur bibendum mi tincidunt risus dapibus tristique
condimentum est convallis.
      In ornare interdum nisi, et hendrerit augue accumsan id.
</div>
<div id="eieren">
      Phasellus volutpat enim risus.
      Morbi quis nisl lorem.
</div>

      div#boter {
            background-color: beige;
      }
      div#kaas {
            background-color: green;
      }
      div#eieren {
            color: #fff;
            background-color: brown;
      }
```

Bekijk voorbeeld 3 in je browser.

Merk op dat **verticale marges** – boven- en ondermarges, dus – **niet** worden **samengeteld**. De onderste marge van het bovenste element wordt dus niet bij de bovenste marge van het onderste element geteld om zo een onderlinge tussenruimte te voorzien. **Enkel de grootste waarde wordt gehanteerd.** Horizontaal-zijdelingse marges worden echter wèl samengeteld. Wanneer we bovenstaande codestukjes uitbreiden, zitten we bijvoorbeeld met dit.

```
<div id="boter">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      In ornare ligula sed lectus varius dapibus rutrum ante imperdiet.
</div>
<div id="kaas">
      Curabitur bibendum mi tincidunt risus dapibus tristique
   condimentum est convallis.
```

```
     In ornare interdum nisi, et hendrerit augue accumsan id.
</div>
<div id="eieren">
     Phasellus volutpat enim risus.
     Morbi quis nisl lorem.
</div>


     div#boter {
          margin-bottom: 25px;
          background-color: beige;
     }
     div#kaas {
          margin-top: 50px;
          margin-bottom: 25px;
          background-color: green;
     }
     div#eieren {
          color: #fff;
          background-color: brown;
     }
```

Bekijk voorbeeld 4 in je browser.

Hoewel je zou verwachten dat de ondermarge van 25 pixels van het -element met als ID 'boter' bij die van de bovenmarge van het -element met als ID 'kaas' wordt opgeteld, blijft enkel de grootste margewaarde overeind.

Inline-elementen doen het omgekeerde en volgen elkaar **van links naar rechts** op.

```
<p><strong>Boter</strong>, <em>kaas</em> en <del>eieren</del>.</p>
```

Boter, *kaas* en ~~eieren~~.

Inline-elementen worden van een **automatische regelterugloop** voorzien – past de tekst of het prentje niet op de regel, dan zakt het ding één regel naar beneden, op voorwaarde dat de beschikbare breedte de plaatsing van dit bewuste element toelaat. Dat kan lelijke resultaten opleveren.

```
<div id="kattebelletje">
     <p>Koop je <em>boter, kaas en eieren</em>? Ook zitten we bijna
     door onze <em>melkvoorraad</em> heen.</p>
</div>

     div#kattebelletje {
          width: 150px;
          background-image: url('images/plaknotitie.png');
     }
```

```
div#kattebelletje p em {
	border: 1px solid green;
	background-color: yellow;
}
```

Bekijk voorbeeld 5 in je browser.

Volgens de W3C-standaarden dient de browser inderdaad slechts tweemaal drie randen te tekenen. Niemand zei dat standaarden altijd mooi zijn.

Meestal willen we bij het lay-outen een andere schikking dan de statische standaard stapelvolgorde van de normal flow verwezenlijken. We willen elementen ook kunnen verplaatsen, ze naast mekaar positioneren, kolommen creëren, elkaar laten overlappen, en noem maar op.
We gaan nu de verschillende mogelijkheden bekijken om een aantrekkelijkere lay-out te bouwen.

Zweven of 'float'

Je kan een HTML-element doen zweven door de **float eigenschap** op **'left' of 'right'** in te stellen. Vanaf dan reageren deze elementen een klein beetje anders.
Verticaal gezien wordt de box **conform de regels van de statische positionering** geplaatst, maar **horizontaal** wordt het element **binnen diens container zo ver mogelijk naar links of naar rechts verschoven**, nog steeds rekening houdend met de binnenste ademruimte/padding van het parent element. Naburige inhoud vloeit dan netjes langs de vrije kant van het zwevende element.

```
<div id="boter">
	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
	Cras at nisl vel ligula iaculis placerat eu at orci.
</div>
<div id="kaas">
	Aliquam erat orci, tincidunt in lacinia vel, molestie in est.
	Vestibulum posuere lorem lacinia metus aliquam dictum.
</div>
<div id="eieren">
	Nunc massa libero, varius vel aliquet a, porta quis dui.
	Vivamus vitae dui dui, non convallis orci.
</div>
```

```
	div#boter, div#kaas, div#eieren {
		width: 150px;
		height: 150px;
		float: left;
	}
```

```
div#boter {
      background-color: beige;
}
div#kaas {
      background-color: green;
}
div#eieren {
      color: #fff;
      background-color: brown;
}
```

Bekijk voorbeeld 6 in je browser.

Je kan de waarde van de float eigenschap op 'left', 'right', 'none' of 'inherit' instellen. Die laatste twee waarden heffen respectievelijk het zweven op of volgen het zwevende gedrag van het parent element.

Enkele belangrijke **opmerkingen** in verband met zwevende elementen…

1. **De te zweven box dient over een specifieke breedtewaarde te beschikken.** Je kan die breedtewaarde rechtstreeks opgeven met de 'width'-eigenschap. Doe je dit niet, dan neemt het zwevende element alsnog de hele beschikbare breedteruimte van het parent element in beslag. **In die zin houdt het dus steek dat de browser zwevende boxes steeds als block-level-elementen behandelt, ook al zijn het in werkelijkheid bijvoorbeeld inline-elementen.**

2. Wanneer twee of meer naburige elementen toegelaten worden te zweven, dan worden hun **bovenste zijden op gelijke hoogte uitgelijnd** wanneer er voldoende horizontale ruimte is om hen naast elkaar te plaatsen. Is dat niet het geval, dan verschuift het laatste element naar dat gebied waar er wèl genoeg plaats is.

3. Dit gedrag tekent zich ook af wanneer een enkel zwevend element te breed is om binnen de breedteafmetingen van zijn parent te passen wanneer er reeds naburige inhoud aanwezig was.

4. De **verticale margewaarden** van zwevende boxes klappen niet samen en worden dus **wèl netjes opgeteld**.

5. Een zwevende box kan naburige, statische gepositioneerde block-level-elementen **overlappen**. In onderstaand voorbeeld bevat het zwevende blok meer inhoud dan diens container, waardoor het zwevende blok ook op de onderste blok uit de normale volgorde zal ingrijpen. **Merk op dat de tekst in het onderste blok ook met het zwevende element rekening houdt en er dus mooi langs 'vloeit'.**

```
<div id="boter">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      Suspendisse id suscipit metus.
      Morbi tempus odio nec lacus feugiat blandit.
      Proin convallis hendrerit tortor nec ultrices.
      Pellentesque et lectus enim, vel sollicitudin lacus.
      <div id="kaas">
            Mauris pretium hendrerit turpis, a porttitor risus tincidunt
            a. Proin vulputate neque et elit ultricies in aliquam quam
            pellentesque.
      </div>
</div>
<div id="eieren">
      Donec accumsan metus at neque vulputate sed vulputate justo
      malesuada. Nunc convallis magna nec lacus blandit aliquam.
      Pellentesque id felis magna. Phasellus libero felis, tempus non
      imperdiet in, tempus at urna.
</div>

      div#boter {
            width: 300px;
            background-color: beige;
      }
      div#kaas {
            width: 150px;
            float: right;
            background-color: green;
      }
      div#eieren {
            width: 150px;
            color: #fff;
            background-color: brown;
      }
```

Bekijk voorbeeld 7 in je browser.

Wil je vermijden dat het onderste blok door het zwevende blok gehinderd wordt, dan dien je je op de 'clear'-eigenschap te beroepen. Door de waarde van deze eigenschap op **'right'** in te stellen, instrueer je de browser de **rechterruimte langs dit bewuste blok te vrijwaren** – er mag in de parent container niets meer langs dit -element staan. In concreto doet deze instructie het onderste blok dan ook gewoon zakken tot het niet meer met het zwevende element interfereert.

```
      div#boter {
            width: 300px;
            background-color: beige;
      }
```

```
div#kaas {
      width: 150px;
      float: right;
      background-color: green;
}
div#eieren {
      width: 150px;
      clear: right;
      color: #fff;
      background-color: brown;
}
```

Bekijk voorbeeld 8 in je browser.

Je kan de waarde van de clear eigenschap op 'left', 'right', 'none' of 'both' instellen. Die laatste twee waarden heffen respectievelijk het vrijwaren op of zorgen voor een vrijwaring aan beide zijden van het te vrijwaren element. **De clear eigenschap heeft enkel zin voor block-level-elementen.**

Relatieve positionering

Wanneer een HTML-element via CSS van een **'position: relative;'-eigenschap** wordt voorzien, wordt dat element aanvankelijk volgens de regels van de statische positionering geplaatst. Ook de ingesloten boxes volgen dit normale positionering schema.
Met verschuivingswaarden of 'offset values' kan je het block-level-element ten opzichte van haar normale, statische positie doen verschuiven. Deze verschuiving is louter visueel, de originele positie van het element maakt nog steeds deel uit van de normal flow.

```
div#boter, div#kaas, div#eieren {
      width: 150px;
      height: 150px;
}
div#boter {
      background-color: beige;
}
div#kaas {
      position: relative;
      top: -25px;
      left: 25px;
      background-color: green;
}
div#eieren {
      color: #fff;
      background-color: brown;
}
```

```
<div id="boter">
     Lorem ipsum dolor sit amet, consectetur adipiscing elit.
     Nulla sodales porttitor consectetur.
</div>
<div id="kaas">
     Integer suscipit adipiscing posuere. Etiam pretium sem vitae
     metus condimentum ut tempus augue imperdiet.
</div>
<div id="eieren">
     Pellentesque dapibus, diam rutrum consequat dignissim, ante purus
     interdum metus, at iaculis velit leo egestas justo.
     Proin purus enim, tempus vel suscipit commodo, tempor non ligula.
</div>
```

Bekijk voorbeeld 9 in je browser.

Bovenstaand block-level-element zal rekening houden met de omringende elementen en **volgens de normale positioneringsregels geplaatst** worden. **Vervolgens schuift het blokje vijfentwintig pixels naar rechts en boven.**

De **offset values** worden middels een **combinatie** van de **'top, right'-, 'left'- en 'bottom'-stijleigenschappen** bepaald. Elke afstand wordt geïnterpreteerd als de afstand die de box vanaf diens uiterste rand ten opzichte van haar normale positie uit de normal flow moet verschuiven. Je maakt een **keuze** tussen de 'left'- en 'right'-eigenschap en de 'top'- en 'bottom'-eigenschap.

Merk op dat tegensprekende verschuivingen elkaar beperken – wanneer de waarde voor de 'left'-eigenschap bijvoorbeeld niet het precieze negatief van de waarde voor de 'right'-eigenschap is, wordt de rechterverschuiving genegeerd. **Het spreekt voor zich dat het combineren van linker- en rechter en boven- en onderwaarden voor onvoorspelbare situaties zal zorgen – niet doen, dus.**
Browsers durven wel eens verschillen in de manier waarop ze relatief gepositioneerde elementen die andere elementen overlappen tonen. Jammer genoeg bieden de standaarden geen echte duidelijkheid over het gewenste gedrag.

Stapelvolgorde is een stevig pijnpunt wanneer je relatief of absoluut positioneert. Probeer met de **'z-index'-eigenschap** (zie verderop) de elementen om te wisselen tot je het gewenste overlappingsresultaat verkrijgt. Wanneer je voor verschillende -elementen een waarde voor de z-index eigenschap instelt, dan komt het **element met de hoogste z-index waarde bovenaan**. Hebben overlappende -elementen **eenzelfde z-index waarde**, dan zal het **element dat het laatst in de HTML-code gedefinieerd wordt bovenaan** komen te liggen.

```
     div#boter, div#kaas, div#eieren {
          width: 150px;
          height: 150px;
     }
     div#boter {
```

```
            background-color: beige;
      }
      div#kaas {
            position: relative;
            top: -25px;
            left: 25px;
            z-index: -1;
            background-color: green;
      }
      div#eieren {
            color: #fff;
            background-color: brown;
      }

<div id="boter">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      Vestibulum et sagittis dolor. Mauris bibendum erat nec sem luctus
      tincidunt.
</div>
<div id="kaas">
      Sed imperdiet ultrices elit, vitae tincidunt elit dictum ac.
      Vestibulum aliquam consequat eleifend.
</div>
<div id="eieren">
      Fusce nulla diam, commodo et vehicula nec, laoreet quis nisi.
</div>
```

Bekijk voorbeeld 10 in je browser.

Afstammeling positionering

Relatief gepositioneerde elementen kunnen zich als als nieuwe container voor ingesloten elementen opwerpen. Deze elementen volgen standaard dan weer de statische positioneringsregels – nogmaals, positioneringsdefinities worden niet overgeërfd!

Wanneer het relatief gepositioneerde element een block-level-element is, zal dit element zich als een nieuwe container gedragen. De daarin vervatte elementen houden rekening met de offset values van de relatief gepositioneerde moeder.
Relatief gepositioneerde inline-elementen reageren in verschillende browsers nogal anders, en dat omwille van enkele standaardonduidelijkheden en -wijzigingen. Net om die reden verdient het aanbeveling inline-elementen niet relatief te positioneren en het **relatief positioneren enkel voor block-level-elementen te houden**.

Absolute positionering

Wanneer je de position eigenschap op de 'absolute'-waarde instelt, haal je de elementen uit hun normale paginavolgorde, dus uit de normal flow. Het mag dan ook duidelijk zijn – absoluut gepositioneerde elementen **oefenen dan ook geen invloed meer op de andere pagina elementen uit**. Wanneer je elementen absoluut positioneert, worden ze, net zoals zwevende elementen, als block-level-elementen behandeld. Op die manier werpen zulke absoluut gepositioneerde elementen zich als nieuwe containerblokken voor hun child elementen op. **Die child elementen volgen op hun beurt weer de normale volgorde in het absoluut gepositioneerde element, op voorwaarde dat je deze child elementen niet met position waardes opzadelt.**

De precieze plaats van een absoluut gepositioneerd element wordt door diens **offset values** bepaald. De **offset values** worden middels een **combinatie** van de **'top', 'right'-, 'left'- en 'bottom'-stijleigenschappen** bepaald.

Terwijl de **plaatsing van relatief gepositioneerde elementen ten opzichte van hun oorspronkelijke positie in de normale paginavolgorde gebeurt**, definiëren de offset values voor absoluut gepositioneerde elementen een **verplaatsing ten opzichte van het bewuste containerblok**. **Dergelijk containerblok dient op zijn beurt weer uit de normale volgorde van een pagina gerukt te zijn, en dus relatief, absoluut of gefixeerd gepositioneerd zijn.**

Een absoluut gepositioneerd blok zal zich met andere woorden dus steeds ten opzichte van zijn dichtsomringende absoluut, relatief of gefixeerde parent blok verplaatsen. **Is er geen dergelijk parent element, dan zal het allesomvattende containerblok gebruikt worden – het browserscherm, dus.**

Hoewel absoluut gepositioneerde elementen zeker over **marges** kunnen beschikken en de verschuivingen ook steeds zulke margewaarden in rekenschap nemen, is het verwerken van **marges redelijk zinloos** omdat absoluut gepositioneerde elementen toch uit hun normale volgorde getrokken worden en **naburige elementen in de normale volgorde niet op de hoogte** zijn van de plaatsing van zulke absoluut gepositioneerde elementen.

```
<div id="boter">
     Lorem ipsum dolor sit amet, consectetur adipiscing elit.
     Sed rutrum urna non ligula ornare vitae porta urna porttitor.
     <div id="kaas">
          Nullam in mattis nibh.
          Etiam consequat vehicula tortor dapibus congue.
     </div>
</div>
<div id="eieren">
```

```
      Vivamus lorem justo, faucibus quis lobortis nec, interdum vitae
      nisi. Mauris orci lacus, feugiat pharetra bibendum sit amet,
      auctor vitae lorem.
</div>

      div#boter {
            width: 300px;
            position: relative;
            background-color: beige;
      }
      div#kaas {
            width: 150px;
            position: absolute;
            left: 25px;
            background-color: green;
      }
      div#eieren {
            width: 150px;
            color: #fff;
            background-color: brown;
      }
```

Bekijk voorbeeld 11 in je browser.

Absoluut gepositioneerde elementen worden ten opzichte van één van de hoeken van hun parent elementen gepositioneerd. De parents kunnen **zowel block-level- als inline-elementen** zijn. In beide situaties gebeurt er iets verschillends.
Wanneer het parent element een **block-level-element** is, gebeurt de **verschuiving** van het absoluut gepositioneerde child element **ten opzichte van de eventuele padding van diens ouderelement**, en niet ten opzichte van de border.

1. Wanneer je voor het moederelement **geen padding** definieerde, lijkt de **verschuiving wèl ten opzichte van de border** van het moederelement te gebeuren.
2. Geef je **geen offset values** op, blijft de **padding** van het moederelement **gehandhaafd**.
3. Stel je de **offset values** op **nulwaarden** in, zal het absoluut gepositioneerde element zich wel **terug tegen de rand van het parent element** plaatsen.

```
      div#boter {
            width: 250px;
            padding: 25px;
            position: relative;
            background-color: beige;
      }
      div#kaas {
            width: 150px;
            position: absolute;
            left: 0;
```

```
            top: 25px;
            background-color: green;
      }
      div#eieren {
            width: 200px;
            position: absolute;
            right: 0;
            bottom: 0;
            color: #fff;
            background-color: brown;
      }

<div id="boter">
      <div id="kaas">
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
            Suspendisse nec nisi nulla, eu aliquam tellus.
      </div>
      <div id="eieren">
            Curabitur tortor mi, tincidunt eget consequat vel, tempus
            vitae est. Integer scelerisque, urna nec vehicula gravida,
            nulla metus venenatis erat, non adipiscing velit odio vitae
            justo.
      </div>
      In at sem lorem.
      Aliquam ultrices venenatis ligula, vel tempor lacus tincidunt
sed.
</div>
```

Bekijk voorbeeld 12 in je browser.

Wanneer het parent element zich volgens de **'inline'-regelgeving** gedraagt, wordt het een beetje moeilijker, en dat omdat zulke inline-elementen **meerdere lijnen** in beslag kunnen nemen. In dat geval gebeurt de **linker- en bovenverschuiving** van het absoluut gepositioneerde child element **ten opzichte van de linkerbovenkant van de eerste box** die door het element voorzien wordt. Een bottom- of right-verschuiving gebeurt **ten opzichte van de rechterbenedenhoek van de laatste box** die door het element voorzien wordt.

```
      div#boter {
            width: 300px;
            height: 300px;
            background-color: beige;
      }
      span#kaas {
            position: relative;
            background-color: green;
      }
      div#eieren {
            width: 150px;
```

```
            position: absolute;
            right: 25px;
            top: 25px;
            color: #fff;
            background-color: brown;
      }

<div id="boter">
      <span id="kaas">
            <div id="eieren">
                  Lorem ipsum dolor sit amet, consectetur adipiscing
                  elit. Aenean at velit magna.
            </div>
            Aliquam euismod enim ut mi congue ac tempor velit blandit.
            In sollicitudin, ipsum vestibulum viverra imperdiet ut
            venenatis ante lectus quis leo.
      </span>
      Nam ut magna vel ante lobortis imperdiet. Praesent placerat
      ultrices dui, ac pellentesque elit laoreet nec.
</div>
```

Bekijk voorbeeld 13 in je browser.

Omdat niet elke browser relatief gepositioneerde elementen op eenzelfde manier behandelt, houdt het steek dat er zich ook browserverschillen aftekenen wanneer je absoluut gepositioneerde elementen in zulke relatief gepositioneerde blokken verwerkt. Controleer je CSS-experimenten dus steeds in verschillende browsers en browserversies.

Voorts is het mogelijk om absoluut gepositioneerde elementen **wederom met absoluut gepositioneerde elementen vullen**. Zulke child elementen zullen dan, afhankelijk van de offset values, al dan niet **binnen of buiten diens parent element** vertoeven.

Verankerde/fixed positionering

Het verankerde of fixed positioneringssysteem is een bijzondere variant van absolute positionering. Het **browserscherm (de viewport)** werpt zich steeds als containerblok van verankerde elementen op. **Een fixed element beweegt niet wanneer je door een webpagina scrolt.** Wanneer je de webpagina zou afdrukken, dient het verankerde element op elke pagina te verschijnen.

Bekijk het element in de hoek rechtsonder.

Een codevoorbeeld:

```
#fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 50px;
  height: 50px;
  background-color: white;

}
```

Stapelvolgorde

Broncodevolgorde, z-index eigenschap en microstapelcontext

Absoluut gepositioneerde elementen kunnen niet-gepositioneerde elementen én elkaar overlappen. Die **onderlinge bedekking** wordt niet enkel door de **HTML-codevolgorde** en de getalwaarde van de **z-index eigenschap** bepaald, ook de **microstapelcontext** speelt een belangrijke rol.
Een element met een **hogere z-index waarde** verschijnt **voor een element met een lagere z-index waarde**. Wanneer twee elementen eenzelfde stapelvolgordewaarde hebben of er voor geen van beide elementen een z-index werd opgegeven, zullen de later in de HTML-code gedefinieerde blokken hogerop liggen. **Codegeordend**, noemt men dat.

De z-index waarde kan negatief zijn. Een element met een negatieve stapelvolgordewaarde wordt achter een element zonder z-index waarde of met een element met een positieve z-index waarde uit eenzelfde stapelcontext geplaatst.

Absoluut gepositioneerde elementen kunnen een eigen, lokale microstapelcontext creëren die dan enkel op de absoluut gepositioneerde elementen erin van toepassing is. De stapelvolgorderegels in deze microcontexten blijven echter hetzelfde.

Elk absoluut gepositioneerd element behoort tot een stapelcontext. De omvattende containerblok genereert een **initiële stapelcontext**. Absoluut gepositioneerde elementen binnen eenzelfde stapelcontext worden volgens diens z-index waarde getoond. Een **lokale stapelvolgorde** maak je aan door een absoluut gepositioneerd, omvattend blok een numerieke z-index waarde mee te geven. Vanaf dan zal elk absoluut gepositioneerd element in die stapelvolgorde in die lokale stapelvolgorde deelnemen.
Het wordt ingewikkeld wanneer je meerdere blokken uit **verschillende stapelcontexten** wil stapelen – hogere z-index waarden garanderen dan niet altijd een hogere stapelvolgordepositie.

```
<div id="kattebelletje">
     <div id="boter">
          Lorem ipsum dolor sit amet, consectetur adipiscing elit.
```

```
            Aliquam tincidunt metus et sem consequat rhoncus.
            <div id="kaas">
               Donec rhoncus nunc id mi hendrerit vitae ultricies sem
               posuere. Proin dui nulla, lacinia a gravida ut,
               pellentesque id mi.
            </div>
      </div>
      <div id="eieren">
            Duis varius nibh vel massa posuere eget vulputate quam
            interdum. Vivamus molestie accumsan eros, in aliquam turpis
            porttitor in.
      </div>
</div>

      div#kattebelletje {
            width: 300px;
            height: 300px;
            position: relative;
            background-color: yellow;
      }
      div#boter {
            width: 150px;
            height: 150px;
            position: absolute;
            left: 50px;
            top: 25px;
            z-index: 100;
            background-color: beige;
      }
      div#kaas {
            width: 150px;
            height: 150px;
            position: absolute;
            top: 50px;
            left: 25px;
            z-index: 300;
            background-color: green;
      }
      div#eieren {
            width: 150px;
            height: 150px;
            position: absolute;
            left: 25px;
            top: 100px;
            z-index: 200;
            color: #fff;
            background-color: brown;
      }
```

Bekijk voorbeeld 14 in je browser.

Sommige pagina-elementen liggen dwars wanneer ze door gepositioneerde elementen overlapt worden. **Wanneer je met Flashfilmpjes, formuliervelden, exotische Java-applets of andere plug-ins werkt, zit de kans er dik in dat je vroeg of laat op stapelvolgordeproblemen stoot.** Zelfs **veelvuldig gesjoemel** met z-index waarden **haalt dan niet veel uit**.
Daarnaast verschilt het doorschemeren van zulke onstapelbare elementen ook vaak van browser tot browser. In het gros van deze gevallen is het een goed idee **overlappingen** te **vermijden** of je op **JavaScriptoplossingen** te beroepen.

De 'overflow: hidden;'- en 'clearfix'-hacks

Het is je vast al opgevallen dat wanneer je -elementen in een omsluitende parent- stuk voor stuk doet zweven, het parent element geen enkele ruimte meer in beslag neemt – het ding lijkt samen te klappen. In sommige gevallen is dat niet zo heel erg, maar vaak wil je het omvattende parent element wèl laten meegroeien.
Zowel de 'overflow: hidden;'- als de 'clearfix'-hacks zijn erop gericht parent elementen diens children te laten bevatten.

De overflow: hidden; hack

Werp even een blik op onderstaand HTML-codestukje.

```
<div id="boter">
	<div id="kaas">
		Lorem ipsum dolor sit amet, consectetur adipiscing elit.
		Mauris ultrices dictum tincidunt.
		Etiam scelerisque pellentesque mi.
		Sed vestibulum venenatis nulla sed sodales.
	</div>
	<div id="eieren">
		Quisque diam urna, porta ac posuere quis, luctus in velit.
		Proin vulputate porttitor ultrices.
	</div>
</div>

	div#boter {
		width: 300px;
		background-color: beige;
	}
	div#kaas {
		width: 150px;
		height: 150px;
		float: left;
		background-color: green;
```

```
}
div#eieren {
      width: 100px;
      height: 150px;
      float: right;
      color: #fff;
      background-color: brown;
}
```

Bekijk voorbeeld 15 in je browser.

Hoewel de -elementen met als ID 'kaas' en 'eieren' door een **verschillende hoogte** gekenmerkt worden, lijkt de **achtergrondkleur** van het omsluitende parent element **niet mee te 'groeien'**. **Dit is standaardgedrag – wanneer een parent element louter uit zwevende child elementen bestaat, klapt het parent element samen en neemt het dus geen ruimte meer in beslag.** Dit gedrag is enkel waarneembaar indien dit parent element van een **achtergrondkleur of -afbeelding** voorzien werd.
Wanneer we het **omsluitende element** een **willekeurige hoogte** van 50 pixels zouden meegeven, komt de **achtergrond** echter wèl tevoorschijn.

```
div#boter {
      width: 300px;
      height: 50px;
      background-color: beige;
}
div#kaas {
      width: 150px;
      height: 150px;
      float: left;
      background-color: green;
}
div#eieren {
      width: 100px;
      height: 150px;
      float: right;
      color: #fff;
      background-color: brown;
}
```

Bekijk voorbeeld 16 in je browser.

Wanneer je de **hoogte** van een bepaald element **niet precies kan voorspellen**, is het **geen goed idee** deze hoogte in je CSS-specificaties **vast te leggen**. Zonder nadrukkelijke definitie wijst de browser dan **automatisch een minimumhoogte** toe. **Merk ook op dat de twee child elementen over de voor de parent vastgelegde grenzen vloeien.** Wanneer we de

stijldefinities van het 'div'-element met als ID 'boter' met een **'overflow: hidden;'-regel** verrijken, drukken we het overvloeien van de child elementen de kop in.

```
div#boter {
	width: 300px;
	height: 50px;
	overflow: hidden;
	background-color: beige;
}
```

Bekijk voorbeeld 17 in je browser.

Dat helpt ons echter niet meteen verder. **Het probleem zit in het gegeven dat we de eigenlijke hoogte van deze drie div elementen nooit precies kennen.** Wanneer HTML-elementen tekst bevatten, is de **hoogte niet enkel afhankelijk van de hoeveelheid tekst**, maar bijvoorbeeld ook van het gebruikte **lettertype**, de **zoominstellingen** van de browser van de gebruiker,… Wanneer we de **hoogtedefinitie** voor het parent element verwijderen, zitten we met onderstaande **CSS-code**.

```
div#boter {
	width: 300px;
	overflow: hidden;
	background-color: beige;
}
```

De **achtergrondkleur** van het parent element **loopt** nu wel **door**, alsook de **child elementen** worden volledig **getoond**.

Bekijk voorbeeld 18 in je browser.

Mooi, zo.

De clearfix hack

Het overflow: hidden; trucje brengt in sommige configuraties echter een ander probleem met zich mee. Wanneer we om één of andere reden een **extra -element in de omsluitende -container** met als ID 'boter' willen opnemen en dit wat **aflopend** – buiten de grenzen van diens parent tredend, dus – willen weergeven? We passen ons **HTML-codestukje** even als volgt aan.

```
<div id="boter">
	<div id="kaas">
		Lorem ipsum dolor sit amet, consectetur adipiscing elit.
		Aenean vitae tellus justo, vitae congue justo.
```

```
            Etiam et dui quis sapien dapibus dictum.
            Nunc semper tempus nibh, nec posuere orci sodales quis.
      </div>
      <div id="eieren">
            Donec consequat dignissim sem, sed blandit erat feugiat sed.
            Proin tellus est, laoreet vel euismod non, interdum ac orci.
      </div>
</div>
```

Omdat we het -element met als ID 'eieren' ten opzichte van het parent element willen verplaatsen, dienen we diezelfde parent de relatieve positioneringsregels te laten volgen. Pas dan kunnen we het **child element absoluut positioneren**.

```
      div#boter {
            width: 300px;
            position: relative;
            overflow: hidden;
            background-color: beige;
      }
      div#kaas {
            width: 150px;
            height: 150px;
            float: left;
            background-color: green;
      }
      div#eieren {
            width: 100px;
            height: 150px;
            position: absolute;
            top: 25px;
            right: -25px;
            color: #fff;
            background-color: brown;
      }
```

Bekijk voorbeeld 19 in je browser.

Merk op dat ook dit ** -element met als ID 'eieren'** wordt **afgesneden** – niet goed. In dit specifieke geval kan je de klus niet met het overflow: hidden; trucje klaren.
Het W3C stelt voor een extra, zij het leeg -element met een 'clear: both;'-stijldefinitie te verwerken. Omdat dit element **niet meer zwevend** geplaatst wordt, wordt het **omsluitende -element gedwongen** dit **element** ook **op te nemen**, hetgeen in een **meegroeiende hoogte** lijkt te resulteren.

```
<div id="boter">
      <div id="kaas">
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
```

```
            Aenean vitae tellus justo, vitae congue justo.
            Etiam et dui quis sapien dapibus dictum.
            Nunc semper tempus nibh, nec posuere orci.
      </div>
      <div id="eieren">
            Donec consequat dignissim sem, sed blandit erat feugiat sed.
            Proin tellus est, laoreet vel euismod non, interdum ac orci.
      </div>
      <div class="clear"></div>
</div>

      div#boter {
            width: 300px;
            position: relative;
            background-color: beige;
      }
      div.clear {
            clear: both;
      }
```

Bekijk voorbeeld 20 in je browser.

Nu is het **weinig logisch of semantisch** om een **extra HTML-element** toe te voegen om de lay-out naar onze hand te zetten. Eén van de grootste voordelen van het gebruik van CSS behelst net de strikte **scheiding tussen inhoud en vorm**. Het zou onze HTML-code netjes moeten houden. We moeten dus op zoek gaan naar **alternatief CSS-codestukje**.

Het **':after'-pseudo-element** laat je toe om middels **CSS** wat **inhoud toe te voegen** aan de HTML-code, en dat **zonder zelf in je HTML-bestand te moeten sleutelen**. De inhoud wordt als het ware door enkele slimme CSS-instructies in **geïnjecteerd**. Hoewel je zulke door CSS toegevoegde inhoud niet kan stijlen, kan je wèl instructies inzake de **'clear'-eigenschap** meegeven.
We gebruiken het ':after'-pseudo-element om een **eenvoudig teken zoals een spatie** – '\0020' in unicode – in te voegen, waarna we dat zo goed als onzichtbare ding een **'clear: both;'-regel** meegeven. Omdat dit spatietekstje een **inline-element** is, dienen we het nog even naar een **block-level-element** om te zetten. Om de spatie te **verbergen**, gebruik je de **'visibility: hidden;'-eigenschap** en om het ding **geen hoogte** in beslag te laten nemen, stel je de **'height'- en 'font-size'-eigenschap** op 0 in.

Omdat de fossielen **Internet Explorer 6 en 7** het ':after'-pseudo-element **niet ondersteunen**, diende je indertijd aan de rits CSS-definities nog de **'zoom: 1;'-declaratie** toe te voegen, hetgeen tot iets Microsoftspecifieks als 'hasLayout' leidt. In dat geval wordt dat bewuste element verantwoordelijk voor diens eigen dimensies en positionering. Omdat de 'zoom'-eigenschap enkel door Internet Explorer begrepen wordt, wordt je **CSS-bestand** plotsklaps **ongeldig**. Het werkt op deze manier echter in elke browser. Afwegingen…

```
div#boter:before, div#boter:after {
	width: 0;
	height: 0;
	display: block;
	overflow: hidden;
	visibility: hidden;
	font-size: 0;
	content: '\0020';
}
div#boter:after {
	clear: both;
}
div#boter { /* als je codegeldigheid erg belangrijk vindt, kan je
deze kiezer en definities naar een Internet Explorerspecifiek
stijlblad verplaatsen */
	zoom: 1;
}
```

We kunnen het **lelijke -element** nu naar de **prullenmand** verwijzen.

```
<div id="boter">
	<div id="kaas">
		Lorem ipsum dolor sit amet, consectetur adipiscing elit.
		Aenean vitae tellus justo, vitae congue justo.
		Etiam et dui quis sapien dapibus dictum.
		Nunc semper tempus nibh, nec posuere orci.
	</div>
	<div id="eieren">
		Donec consequat dignissim sem, sed blandit erat feugiat sed.
		Proin tellus est, laoreet vel euismod non, interdum ac orci.
	</div>
</div>
```

Bekijk voorbeeld 21 in je browser.

Omdat het denkbaar is dat je deze zogenaamde **clearfix hack** wel eens **vaker** zal toepassen, houdt het steek er een **toegewijde klasse** voor te bakken. Deze kan je dan **aan meerdere elementen en elementsoorten 'hangen'**.

```
.clearfix:before, .clearfix:after {
	width: 0;
	height: 0;
	display: block;
	overflow: hidden;
	visibility: hidden;
	font-size: 0;
	content: '\0020';
}
```

```
      .clearfix:after {
            clear: both;
      }
      .clearfix { /* als je codegeldigheid erg belangrijk vindt, kan je
      deze kiezer en definities naar een Internet Explorerspecifiek
      stijlblad verplaatsen */
            zoom: 1;
      }

<div id="boter" class="clearfix">
      <div id="kaas">
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
            Aenean vitae tellus justo, vitae congue justo.
            Etiam et dui quis sapien dapibus dictum.
            Nunc semper tempus nibh, nec posuere orci.
      </div>
      <div id="eieren">
            Donec consequat dignissim sem, sed blandit erat feugiat sed.
            Proin tellus est, laoreet vel euismod non, interdum ac orci.
      </div>
</div>
```

Gebruik de clearfix truc enkel wanneer het veel eenvoudigere overflow: hidden; de klus niet zou klaren. Hoe **minder code** je produceert, hoe minder vlug deze kleine codehoeveelheden **conflicten** kunnen veroorzaken.

Bovenstaande versie van de hack is inmiddels (2017) verouderd. Momenteel volstaat volgende 'clearfix'-hack.

```
      .group:after {
        content: “”;
        display: table;
        clear: both;
      }

<div class="group">
  <div class="is-floated"></div>
  <div class="is-floated"></div>
  <div class="is-floated"></div>
</div>
```

Volg de laatste ontwikkelingen op CSS-Tricks.

Slotsom

Het moge duidelijk zijn dat verschillende browsers en browserversies HTML- en CSS-code net dat tikkeltje anders interpreteren. En we hebben het enkel nog maar over positionering gehad. De zaken worden al vlug gecompliceerder wanneer je weergavestijlen, formuliervelden, JavaScriptinteractie,… toevoegt. En zelfs wanneer elk browserverschil werd geëlimineerd, zullen ze nog steeds kleine **bugjes** bevatten. Voorts zijn de **specificaties** zoals het W3C ze dicteert ook **niet altijd even duidelijk**, hetgeen ruimte voor interpretatie laat, en dus wederom in verschillen resulteert.

Wanneer je je webontwerpen naar code converteert, kunnen codecontrolemechanismen eenvoudige scriptingfouten zoals ongesloten of verkeerd geneste tags opsporen en elimineren – de W3C-opmaakvalidatiedienst klaart de klus.
Voorts is het een goed idee om bepaalde testcases op te bouwen – **verwijder alle niet-noodzakelijke elementen en opmaak** en herleid je code tot het absolute minimum. Vergelijk dit resultaat in verschillende browsers, om zo het probleem goed te kunnen benoemen én op te lossen.