VDAB

samen sterk voor werk

HTML &CSS

theorie

Deze cursus is eigendom van de VDAB

Inhoudsopgave

1 INLEIDING 12

1.1 Clients, servers en URL's 12
1.2 Browsers 13

2 OVER DEZE CURSUS 16

2.1 Voorkennis 16
2.2 Afspraken 16

3 WEB STANDAARDEN 17

3.1 HTML standaarden 17
3.2 CSS standaarden 21
3.3 Good practices 21
3.4 HTML5, CSS3 ondersteuning 23

4 HTML 24

4.1 De HTML5 versie 24
4.2 De XHTML versie 28
4.3 Het Document Object Model 30
4.3.1 Javascript 30
4.3.2 De document tree zichtbaar maken 31
4.3.3 Request en Response 32
4.4 Een HTML document 34
4.4.1 Elementen en tags 35
4.4.2 Attributen 36
4.4.3 data- attributen 38

5 DE HTML ELEMENTEN 39

5.1 Een HTML document 39
5.1.1 doctype 39
5.1.2 html 39
5.1.3 head 39
5.1.4 meta-data 39
5.2 Paginastructuur elementen 40
5.2.1 body 41
5.2.2 header 41

5.2.3 section 41
5.2.4 article 42
5.2.5 nav 42
5.2.6 aside 43
5.2.7 titels: h1..h6 43
5.2.8 footer 43
5.2.9 address 44
5.3 blokelementen 44
5.3.1 p 44
5.3.2 pre 44
5.3.3 blockquote 45
5.3.4 ol 45
5.3.5 ul 45
5.3.6 li 45
5.3.7 dl 46
5.3.8 dt 47
5.3.9 dd 47
5.3.10 figure en figurecaption 47
5.3.11 div 47
5.4 Structuur aanbrengen in een HTML pagina 49
5.4.1 Content first 49
5.4.2 Sectioning en outlines 51
5.4.3 de lay-out heeft zijn eigen structuur 52
5.5 Tekstelementen 56
5.5.1 Visueel benadrukken: `b` 56
5.5.2 Inhoudelijk benadrukken: `strong` 56
5.5.3 Andere stemming aangeven: `i` 56
5.5.4 Andere betekenis aangeven: `em` 57
5.5.5 Klein: `small` 57
5.5.6 Titel werk: `cite` 57
5.5.7 Woordgroep: `span` 57
5.5.8 Link: `a` 57
5.5.9 time en datetime 58
5.5.10 onderlijnen met `u` 58
5.5.11 programmeercode aangeven met code 58

5.5.12 sub 58
5.5.13 sup 58
5.5.14 aanduiden met mark 59
5.5.15 br 59
5.6 afbeeldingen met img 59
5.6.1 attribuut src 59
5.6.2 attribuut alt 59
5.6.3 attribuut title 60
5.6.4 een afbeelding als hyperlink 60
5.6.5 bestandsformaten bij beelden 60
5.7 Tabellen 62
5.7.1 eenvoudige tabel 62
5.7.2 tabelrand: attribuut border 62
5.7.3 lege cel 63
5.7.4 rij- en kolomkoppen 63
5.7.5 cellen samenvoegen: de attributen colspan en rowspan 64
5.7.6 bijschrift 65
5.7.7 de twee content-modellen van table 65
5.7.8 col en colgroup 67
5.8 Formulier 68
5.8.1 form 69
5.8.2 label 70
5.8.3 fieldset 71
5.8.4 legend 71
5.8.5 input 71
5.8.6 textarea 76
5.8.7 select 77
5.8.8 optgroup 78
5.8.9 button 79
5.8.10 input attributen 79
5.9 HMTL5 video en audio 82
5.9.1 element video/audio 82
5.9.2 attribuut source 83
5.9.3 attribuut width 83
5.9.4 attribuut heigth 83

5.9.5 attribuut autoplay 83
5.9.6 attribuut controls 83
5.9.7 attribuut loop 83
5.9.8 attribuut poster 83
5.9.9 Bestandsformaten video 83
5.10 HTML5 canvas 84
5.10.1 Canvas voorbereiden 84
5.10.2 Context initiatiseren 85
5.10.3 Vulkleur/randkleur instellen 85
5.10.4 Vierkant tekenen 85
5.10.5 Vullen met een patroon 86
5.10.6 Afbeelding toevoegen 86

6 CSS 88

6.1 Syntax 88
6.2 De CSS eigenschappen 89
6.3 CSS en HTML 89
6.3.1 Een extern stylesheet koppelen 90
6.3.2 Intern stylesheet 91
6.3.3 Inline styles 91
6.3.4 @import 92
6.3.5 CSS Overload? 92
6.4 CSS grammatica 93
6.5 Selectoren 94
6.5.1 Element-selectoren 94
6.5.2 Uittesten 95

7 BASIS CSS 96

7.1 Letter-opmaak 96
7.1.1 font-family 96
7.1.2 font-weight 97
7.1.3 font-style 97
7.1.4 font-size 98
7.1.5 Eenheden 98
7.1.6 font-variant 102
7.1.7 line-height 103

7.1.8 Font ... 103
7.1.9 word-spacing ... 104
7.1.10 Letter-spacing ... 104
7.1.11 Text-decoration ... 105
7.1.12 Text-transform ... 106
7.2 Alinea-opmaak ... 106
7.2.1 Margin ... 106
7.2.2 margin ... 107
7.2.3 Text-align ... 107
7.2.4 Text-indent ... 108
7.2.5 White-space ... 108
7.2.6 Teksteigenschappen samengevat ... 109
7.3 Kleur en achtergrond met CSS ... 109
7.3.1 Kleurnamen ... 109
7.3.2 RGB-kleuren ... 110
7.3.3 HSL-kleuren ... 110
7.3.4 transparent en currentColor ... 111
7.3.5 color ... 112
7.3.6 De kleuren van hyperlinks ... 112
7.3.7 background-color ... 113
7.3.8 background-image ... 113
7.3.9 background-repeat ... 114
7.3.10 background-position ... 114
7.3.11 background-attachment ... 116
7.3.12 background ... 116
7.4 Randen ... 116
7.4.1 border-style ... 116
7.4.2 border-color ... 117
7.4.3 border-width ... 117
7.4.4 Gecombineerde definities ... 117
7.4.5 Kortere schrijfwijze ... 118
7.4.6 outline ... 119
7.5 Lijsten opmaken ... 119
7.5.1 list-style-type ... 119
7.5.2 list-style-image ... 120

7.5.3 list-style-position 121
7.5.4 list-style 121

8 MEER SELECTOREN... 122

8.1 universal selector 122
8.2 Gebruik van class en id 123
8.2.1 class versus id: 124
8.3 class selector 124
8.4 id selector 125
8.5 selectoren combineren 125
8.5.1 Descendant combinator 125
8.5.2 child combinator 126
8.5.3 adjacent combinator 127
8.6 attribuutselectoren 127
8.7 Pseudo-classes 128
8.7.1 De dynamische pseudo-classes 128
8.7.2 De :focus pseudo-class 129
8.7.3 De :target pseudo-class 129
8.7.4 De :lang() pseudo-class 129
8.7.5 structurele pseudo-classes 129
8.7.6 De UI States pseudo-classes :enabled :focus :disabled :checked :intermediate 131
8.7.7 De :not negatie pseudo-class 131
8.8 Pseudo-elements 131
8.8.1 Het ::first-line pseudo-element 132
8.8.2 Het ::first-letter pseudo-element 132
8.8.3 De ::before en ::after pseudo-elementen 132

9 INHERITANCE EN DE CASCADE 133

9.1.1 Oorsprong van stylesheets 133
9.2 Hoe wordt CSS toegepast door de browser? 133
9.3 De Cascade 134
9.3.1 Volgorde 134
9.3.2 Specificiteit 134
9.3.3 berekenen van de specificiteit 135
9.3.4 Specificiteit van hyperlinks 137

9.3.5 !important 137
9.3.6 Inheritance 138
9.3.7 De waarde inherit 140
9.3.8 Nog enkele voorbeelden 141

10 HET BOX-MODEL 145

10.1.1 width berekenen 146
10.1.2 voorbeeld met absolute eenheden: 146
10.1.3 voorbeeld met relatieve eenheden: 149

11 HET VISUAL FORMATTING MODEL 150

11.1.1 Block elementen en inline elementen 150
11.1.2 display 150
11.1.3 background 152
11.1.4 Margin 153
11.1.5 Margin-collapse 154
11.1.6 Border 157
11.1.7 Padding 157
11.1.8 Content-area 158
11.1.9 width, height 158
11.1.10 min-width, min-height, max-width, max-height 159
11.1.11 margin of padding gebruiken? 159
11.2 Een box plaatsen 160
11.2.1 de Normal Flow 161
11.2.2 Een relatief geplaatste box 163
11.2.3 Een absoluut geplaatste box 164
11.2.4 Een fixed geplaatste box 165
11.2.5 Floats 166
11.2.6 lay-out problemen met floats 168
11.2.7 volgorde van lagen 170
11.3 Zichtbaarheid van de box 170
11.3.1 Visibility, tonen of niet tonen 170
11.3.2 Overflow: kleine box, grote inhoud 172
11.3.3 Clip, een stukje box afknippen 173

12 CSS VOOR LIJSTEN 175

12.1 De geordende of ongeordende lijst 175
12.2 Definition list 177
12.3 Background-image i.p.v. bullets 178
12.4 Lijsten worden menu's 179

13 CSS VOOR TABELLEN 182

13.1 Randen, kleuren en achtergronden 183
13.2 Uitlijning en ruimte 185
13.2.1 Tabel-breedte 186
13.2.2 Kolom-breedte met col en colgroup 187
13.2.3 De caption 188
13.2.4 Dynamische effecten op rij en kolom 189

14 CSS VOOR FORMULIEREN 190

14.1 Formulier en Formulier elementen 190
14.2 Form lay-out 190
14.2.1 Het label element 191
14.2.2 het form element zelf 191
14.2.3 de container-rijen 192
14.2.4 radiobuttons en checkboxes 194
14.2.5 textarea 196
14.2.6 Knoppen 196
14.2.7 Verplicht label 197
14.2.8 Gebruik de :focus 198

15 CSS VOOR PAGED MEDIA 200

15.1 Afdrukken 200
15.1.1 De Page box 201
15.1.2 Page breaks 202
15.1.3 Orphans 203
15.1.4 Widows 203
Deze eigenschap bepaalt het minimum aantal lijnen van een alinea dat aan de 203

16 CSS GENERATED CONTENT 204

16.1 de ::before en ::after pseudoelementen 204
16.2 content 204

16.3 De eigenschap quotes 205
16.4 Automatische nummering 205

17 CSS3 FEATURES 207

17.1 browser ondersteuning 207
17.2 box-sizing 208
17.3 Gradients 210
17.4 Transparantie 213
17.5 Backgrounds 214
17.6 Borders 226
17.7 Text shadow 230
17.8 Web fonts 233
17.8.1 Lettertype bestanden 233
17.8.2 @font-face 233
17.8.3 cross-browser @ff syntax 234
17.8.4 FOUT 235
17.8.5 Webfonts en mobile 235
17.8.6 Google Web Fonts API 236
17.9 CSS Columns 237
17.10 Transforms 244
17.11 Transitions 248
17.12 Animations 251
17.13 Flex box 257

18 MOBILE WEB DESIGN 271

18.1 Ontwikkelen voor mobile 271
18.2 browsers 272
18.3 viewport 272
18.4 Responsive Web Design 273
18.4.1 Design principes 274
18.4.2 Fluid vs Fixed lay-outs 275
18.4.3 images in RWD 277
18.4.4 media queries 282
18.4.5 Hoe media queries gebruiken 287
18.4.6 breakpoints 289

19 CSS FRAMEWORKS, SHIMS, PREPROCESSORS 291

19.1 Modernizr, Shims en Polyfills 291

19.2 Boilerplates en Frameworks 293

19.3 CSS preprocessors 294

20 BIJLAGE : INTERNET LINKS..... 298

21 COLOFON .....ERROR! BOOKMARK NOT DEFINED.

1 INLEIDING

HTML is de taal waaruit webpagina's bestaan. CSS is de taal die de opmaak van de HTML bepaalt. Typisch voor HTML is dat het **hyperlinks** bevat die doorverwijzen naar andere pagina's of bronnen.

Webpagina's worden door een webontwikkelaar gemaakt:

- als **statische** pagina: de ontwikkelaar schrijf rechtstreeks HTML
- als **dynamische** pagina: ontwikkeld in een programmeertaal zoals Javascript, PHP, Java of C#, waarbij die talen zelf HTML produceren als eindresultaat

Als webontwikkelaar of webdesigner stel je eisen aan je werk:

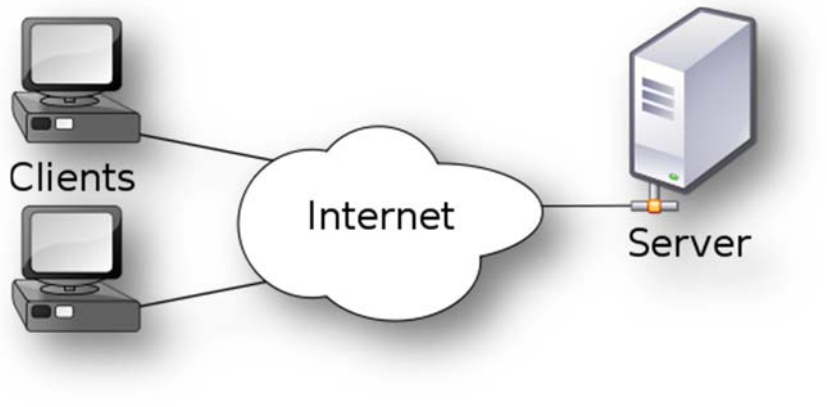
- aantrekkelijk, gebruiksvriendelijk en aangepast aan zijn doelpubliek
- bruikbaar voor iedereen, ongeacht browser of besturingssysteem en toegankelijk voor mensen met lichamelijke beperkingen
- makkelijk te onderhouden zowel qua inhoud als qua opmaak

Door HTML en CSS correct te gebruiken in geldig validerende webpagina's kunnen we dit bereiken.

Met deze cursus proberen we je de verschillende technologieën, web standaarden en talloze *good practice* tips aan te leren door je effectief een aantal webpagina's en sites te laten bouwen.

1.1 Clients, servers en URL's

Het Internet is een voorbeeld van een **client-server-netwerk**, zoals de meeste bedrijfsnetwerken (alleen op veel grotere schaal). Dit betekent dat sommige computers in het netwerk een **server**-functie gaan hebben en dus informatie gaan **aanbieden** aan andere computers. Zulke servers die permanent met het Internet verbonden zijn noemt men ook wel **hosts**. De computers die die informatie **opvragen** – je computer thuis bijvoorbeeld - worden dan de **clients** genoemd.



Webpagina's bevatten hyperlinks die verwijzen naar een **URL** (of URI): een webadres. Een URL zoals

http://www.example.com/admin/login/index.php

bestaat uit een aantal componenten:

URL = protocol + host + path

de componenten zijn

- ***protocol*** (of *scheme*):
  is de communicatiemethode tussen de client en de server
  `http` en `https` zijn veelgebruikte internet protocollen

- ***host*** (of *domain*):
  is de unieke DNS naam van de webserver, hier is dat `www.example.com`

- ***path***:
  is de samenstelling van de mappen en de bestandsnaam van de pagina, hier admin/login/index.php

Dit is een eenvoudig voorbeeld, een URL kan meer componenten hebben.

Enkele speciale, veel voorkomende gevallen:

- Het is ook mogelijk dat je in een URL geen bestandsnaam ziet staan:

  www.vdab.be

  dan zorgt de hostserver ervoor dat er een standaard pagina geladen wordt, hier bijvoorbeeld

  www.vdab.be/default.aspx.

- bij ***User friendly URL's*** zie je ook geen spoor van bestandsnamen:

  www. example.com/products/memory/

  daar zal de server zelf het *user-friendly url* omvormen naar een werkelijk url met querystring:

  www. example.com/products.php?prod=memory/

1.2 Browsers

Op de *client* heb je nog een ***browser*** nodig om een webpagina te bekijken.

Er zijn tegenwoordig heel wat browsers, dikwijls platform gebonden: *FireFox, Chrome, Internet Explorer, Safari, Opera*, etc…








Omdat de markt zo snel evolueert, kan je beter direct online hun marktaandeel bekijken bv. op http://www.netmarketshare.com/.



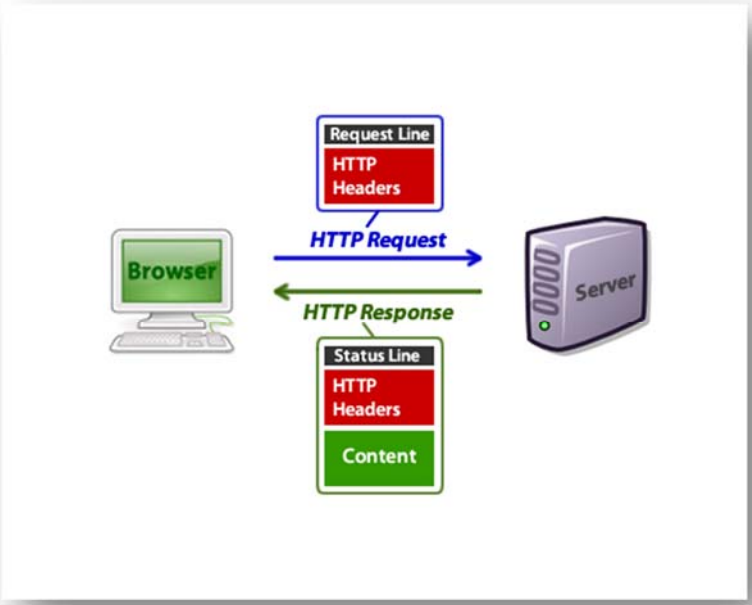
Eén ding is zeker: als ontwikkelaar MOET je evenredig rekening houden met de browsers die het publiek gebruikt!

Het is dan ook normaal dat jij als developer de meest courante browsers installeert!

Hoe werkt een browser?

De browser doet een *HTTP Request* naar een URL met de vraag om een pagina te krijgen.

De server antwoordt en stuurt de gevraagde HTML pagina terug, de *HTTP Response*. De server kan ook een foutbericht terugsturen als *Response*.



Aan de server-zijde

- kan een *statische* HTML pagina onmiddellijk teruggestuurd worden
- wordt een HTML pagina *dynamisch aangemaakt* met een **server-side script** (geschreven in een taal zoals C# of PHP of Java, …) en waarbij dikwijls ook **databanken** geraadpleegd worden om de gevraagde pagina te **genereren**.



Wat de server ook doet, de *HTPP Response* die de browser ontvangt zal altijd in HTML zijn.

Aan de client-zijde

- Eenmaal de HTML pagina ontvangen, maakt de browser (parser) een ***DOM tree***: dat is een boomstructuur van `element` objecten. Het is deze *tree* die getoond wordt op het scherm.
- De DOM tree kan dan nog aan de kant van de *client*, door *Javascript* gewijzigd worden: dat noemt men ***client-side scripting*** omdat het op de PC van de ontvanger uitgevoerd wordt, niet door de server.

2 OVER DEZE CURSUS

Wat moet je weten voor je met deze cursus verder gaat?

2.1 Voorkennis

Als enige voorkennis voor de cursus wordt verondersteld dat je **vertrouwd bent met het gebruik van het internet: surfen, mailen, zoeken, …**

Er is verder geen specifieke voorkennis vereist.

2.2 Afspraken

De tekst in deze handleiding bevat soms *elementen* met *attributen* zoals een `div` element een `class` attribuut kan hebben, die worden aangeduid met een passend kleurtje en lettertype.

Soms vind je korte stukken code zoals `font-family:Verdana, Geneva, sans-serif;` die worden ook aangeduid in een andere kleur en lettertype.

Omdat we in deze cursus zowel de klassieke HTML elementen/attributen gebruiken als de nieuwe HTML5 elementen/attributen, worden deze met een ikoontje aangegeven:

 `canvas`

Ook nieuwe CSS3 features (properties/selectors) worden met een ikoon aangegeven:

 `{background-radius: 0.2em}`

Grotere stukken code staan in een kadertje, zoals

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset=utf-8>
<title>JS PF project: het Document Object Model</title>
<style media=all>
  body {font-family:Verdana, Geneva, sans-serif;color:#333;}
  img {float:right;}
  #container {width:800px;margin: 0 auto;}
  h2{font-size:1.3em;font-style:italic;}
</style>
</head>
```

Soms gebeurt het dat een lange lijn code niet in één keer op deze pagina kan. Dan zetten we er een ⏎ teken tussen en splitsen de lijn, bijvoorbeeld:

```
<p><img src="../images/linda.jpg" alt="Linda, een front-end programmeur" ⏎
                title="Linda, een front-end programmeur" > ⏎
                       De meeste programmeurs <em>leven</em> op koffie</p>
```

3 WEB STANDAARDEN

Het toepassen van web standaarden betekent in de eerste plaats het **correct toepassen van technologieën**.

Op het web – meestal op het W3C http://www.w3.org/standards - vind je beschrijvingen van hoe je die technologieën moet toepassen, bijvoorbeeld

- *XML* op http://www.w3.org/standards/xml/
- *Semantic Web* technologieën op http://www.w3.org/standards/semanticweb/
- *Web design and applications* op http://www.w3.org/standards/webdesign/

Je moet een web standaard een beetje zien als de" bijbel" van een techniek.

Zo'n standaard

- bepaalt **welke elementen of attributen je mag gebruiken**:
  bijvoorbeeld, om een `canvas` element of een `data-set` attribuut te gebruiken moet je in HTML5 werken.
- zegt **wat je wel en niet mag doen**:
  bijvoorbeeld in alle HTML standaarden staat dat een `ul` element minstens één `li` element moet bevatten en geen enkel ander element.
- zegt ook of iets ***deprecated*** is: dat betekent "afgevoerd", met andere woorden dat je het niet meer moet gebruiken.
  Bijvoorbeeld in XHTML en HTML5 staat dat de `font` tag *deprecated* is

Voor deze cursus zijn de webstandaarden voor HTML en CSS van belang.

3.1 HTML standaarden

Er bestaan verschillende HTML standaarden.

Als je een webpagina maakt kies je voor één van deze web standaarden. Je doet dat door een `DOCTYPE` **declaratie** als eerste regel in de pagina te zetten, vóór het `html` element:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
…
```

Deze pagina werd gedeclareerd als *XHTML1.0 Transitional.*

Van HTML4 en XHTML 1.0 bestaan dan nog eens een *Transitional* en een *Strict* versie. Bij de *Strict* versie mag je geen fouten maken tegen de standaard, bij de *Transitional* worden fouten vergeven. je kiest dus best voor deze laatste.

Bemerk de verwijzing naar het *.dtd* document: daar staan de regels in.

De meeste editors laten je kiezen welke standaard je wil en zetten deze declaratie automatisch in de pagina, zoniet moet je hem zelf typen met aandacht voor Hoofd-en kleine letters.

HTML4

Dit was de meest gangbare specificatie een tiental jaar terug, het is de basis van de andere standaarden. Van nature uit "vergevend": fouten worden door de vingers gezien.

Specificatie: http://www.w3.org/TR/1999/REC-html401-19991224/

`DOCTYPE` (Transitional):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
```

XHTML1.0

Hier wordt HTML een onderdeel van XML, zodat aan de strenge regels van XML voldaan moet worden, zeker in de *Strict* versie, waar de browser gedwongen wordt een document te weigeren als er ook maar één foutje in staat.
De *Transitional* versie is vergevend. XHTML 1.1 is hetzelfde als XHTML1.0 Strict.

Specificatie: http://www.w3.org/TR/2002/REC-xhtml1-20020801/

`DOCTYPE` (Transitional):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
               "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

de belangrijkste kernpunten i.v.m. syntax:

- alle tags moeten **afgesloten** worden, ook de *empty* elementen:
  `<span></span>`, `<meta />`, `<br />`, `<img />`
- Elementen moeten **correct genest** zijn: overlapping (cross-over) mag niet
  `<b><i>inhoud</i></b>` is juist, `<b><i>inhoud</b></i>` is fout.
- Elementen en attributen worden steeds in **kleine letters** geschreven: vermits XML hoofdlettergevoelig is, wordt dit de afspraak.
- Attribuutwaarden zijn steeds in **aanhalingstekens** (enkel of dubbel) vervat, ook al lijkt de waarde numeriek.
  `<div class="links">…</div>`
- Attribuutafkorting (**minimisatie**) is niet toegelaten,
  bv. `<select multiple>` wordt `<select multiple="multiple">`.
- Attribuutwaarden zijn hoofdlettergevoelig:
  `UTF-8` is niet gelijk aan `utf-8`
- Om een element te identificeren mag enkel nog het attribuut `id` gebruikt worden, `name` wordt afgeschaft, met uitzondering van formuliercontrols.
- `<![CDATA[...]]>` secties voor tekst die zou kunnen geïnterpreteerd worden door de browser, zoals de inhoud van een `script` tag
- **Hexadecimale waarden** moeten in kleine letters geschreven worden: vroeger kon #XX5555; in XHTML enkel #xx5555;


in XHTML **moet** veel, weinig **mag** ...

HTML5

is de nieuwe *"living standard"*, wat betekent dat er geen upgrades zullen zijn (geen 5.01): alle updates blijven verwijzen naar deze standaard.
Is van nature zeer vergevend en vrij, zet zich af tegen de strikte regels van XHTML, streeft naar eenvoud. Er is geen *Transitional* of *Strict*. Er is wel een XHTML5 voor XML integratie. Bevat een aantal nieuwe elementen en attributen

Specificatie: http://www.w3.org/TR/html5/

DOCTYPE:

```
<!DOCTYPE HTML>
```

de **belangrijkste** kernpunten i.v.m. syntax:

- empty tags moeten **niet afgesloten** worden:
  `<meta>, <br>, <img>`
- Elementen moeten nog steeds **correct genest** zijn
- Elementen en attributen kunnen zowel in **kleine als in hoofdletters** geschreven worden. Toch raden we je aan consequent te zijn, wij verkiezen alles in kleine letters te zetten
- Attribuutwaarden moeten **niet in aanhalingstekens** vervat worden:
  `<div class=links>…</div>`
  Toch raden we je aan om ze **wel** in aanhalingstekens te zetten, want anders krijg je problemen als er meerdere tokens gebruikt worden:
  `<div class="links boven">…</div>`
- Attribuutafkorting (**minimisatie**) mag:
  `<select multiple>`
- Attribuutwaarden zijn **niet** hoofdlettergevoelig:
  `UTF-8` = `utf-8`
- Om een element te identificeren gebruik je steeds `id` , `name` wordt enkel op formuliercontrols gebruikt.
- geen `<![CDATA[...]]>` secties nodig voor een `script` tag


in HTML **mag** veel, weinig **moet** ...

Het is duidelijk dat HTML5 *bedoeld* eenvoudiger en lakser is op gebied van syntax, er wordt meer verantwoordelijkheid afgeschoven op de browser zelf: die moet het maar aankunnen. Je mag echter nog steeds een meer strikte stijl aanhouden, je kiest zelf maar.

HTML5 bevat ook **nieuwe (semantische) elementen en nieuwe attributen**, die zien we verderop.

Validatie

als je wil weten of je goed bezig bent, valideer dan je pagina. De validator zal aan de hand van je `DOCTYPE` je pagina beoordelen.

Je kan je HTML valideren op http://validator.w3.org/

Veel editors en developer tools hebben een ingebouwde validator.

Welke HTML standaard kiezen?

- *gebruikt je doelpubliek een oudere browser*?

  Een belangrijke vraag, bijvoorbeeld bij een bedrijf waar alle toestellen IE7 geïnstalleerd hebben. Als ontwikkelaar ben je de vrijheid gewoon om te installeren wat je wil. Die vrijheid is er meestal niet in bedrijven waar alle toestellen een besturingssysteem en browser hebben waar de gebruiker niets aan kan veranderen.
  Je moet nagaan wat de browser kan/niet kan.
  De kans zit er in dat je geen *HTML5* ondersteuning hebt

- *gebruik je nieuwe elementen/attributen zoals* `canvas`?

  dan moet je *HTML5* gebruiken

- *gebruik je geen nieuwe elementen/attributen*?

  doe verder in *XHTML1.0 Transitional* of
  gebruik *HTML5* want die is volledig *backward compatible* en geeft geen enkel verschil met *XHTML 1.0* of *HTML4*

- *Ik begrijp het gebruik van sectioning elementen (*`article, section`*) niet goed, moet ik ze nu gebruiken in mijn lay-out?*

  Heel wat ontwikkelaars twijfelen over het overschakelen naar de zogenaamde *sectioning* elementen, etc.. en het gebruik daarvan in hun designs. Ze kunnen goed werken met `div`'s maar zijn onzeker over hoe het nu verder moet.
  Gebruik *HTML5*: je leert het in deze cursus

- *moet ik nu ook CSS3 gebruiken, dat ken ik niet*?

  CSS3 heeft in principe niets te maken met *HTML5*, je kan het net zo goed toepassen op *XHTML*. Je bent ook niet verplicht *CSS3* toe te passen.

"HTML5" is meer dan html

De term "HTML5" dekt een grotere lading dan enkel html.

Naast het "html" gedeelte, met de nieuwe elementen en syntax, worden ook een aantal gerelateerde specificaties onder deze noemer gezet. Deze bevatten o.a. de *Web API's*

- *Canvas 2D*
- *Cross-document messaging*
- *Web storage*

- *Offline storage*
- *Geolocation*
- *Drag & Drop*
- *WebWorkers*
- *Websockets*
- *XHR2*
- *File API*

Daarnaast worden ook dikwijls CSS3 en de nieuwe specificaties van Javascript onder de noemer HTML5 geschoven.

Deze (heel interessante) Web API's hebben echter niets met html te maken, maar alles met *cliënt-side Javascript programming*, daarom komen ze hier helemaal niet aan bod.

In deze cursus houden we ons enkel bezig met HTML en CSS .

3.2 CSS standaarden

CSS is een taal waarmee je de **opmaak** van HTML (en ook XML) elementen controleert.

Ook CSS heeft standaarden, de huidige standaard vind je op http://www.w3.org/standards/techs/css#w3c_all.

Er zijn 3 standaarden in CSS:

- CSS1 (verouderd)
- CSS2, in modules opgesplitst (http://www.w3.org/TR/CSS2/)
- CSS3 : de verdere uitbouw op basis van CSS2.1.

CSS3 is geen nieuwe vorm van CSS, er is wezenlijk niets veranderd, er zijn enkel nieuwe modules, doorgedreven selectors, nieuwe properties en wat wijzigingen.

Ook je CSS kan je **valideren** op http://jigsaw.w3.org/css-validator/

Wat moet ik er van kennen?

1. ***iedere ontwikkelaar*** moet de basis van CSS kennen, dat is ± de CSS2.1 syntax, selectors, cascading & inheritance, media types, box model, visual formatting model, colors, backgrounds, fonts, text, tables
2. ***webontwikkelaars*** moeten een diepere kennis hebben waarin o.a., de CSS3 modules Selectors, media queries, het nieuwe box model aan bod komen
3. een ***front-end ontwikkelaar*** moet een CSS specialist zijn en naast het voorgaande perfect op de hoogte zijn van de verschillende lay-out mogelijkheden, ontwikkeling voor *mobile* en alle nieuwigheden op de voet volgen

Afhankelijk van de **dot brochure** die je hanteert, zul je deze zaken in deze cursus ook leren.

3.3 Good practices

Een goede website maken, betekent meer dan enkel de web standaarden toepassen.

De belangrijkste “*Good practice*” richtlijnen:

- **Accessibility**:
  de website moet de aanbevelingen voor **Toegankelijkheid** (Accessibility) toepassen .
  - Semantisch structureren van de pagina, headers, lijsten, `title`,...
  - Gebruik van alternatieve tekst (`alt` en `title` attributen) bij figuren, multimedia- en formulier elementen.
  - Correcte tabellenstructuur en gebruik van `th`, `caption` elementen
  - Duidelijke hyperlink-teksten
  - Voldoende kleurcontrast
  - Gebruik van het `label` element in formulieren
  - Taalaanduiding in de webpagina

- **Scheiding van opmaak en inhoud**:
  - HTML code mag geen opmaak bevatten:
    - weg met alle elementen die enkel voor opmaak dienen: `font, hr`
    - weg met alle attributen die voor opmaak dienen: `bgcolor, width, height`,… (tenzij je niet anders kan)
  - alle opmaak via CSS
    - de CSS mag intern zijn (in een `style` tag), maar
    - nog beter in een extern *.css* stylesheet (via een `link` element)
- **Scheiding van scripting en inhoud**:
  - de HTML code mag geen scripting bevatten, dus geen `onclick, onsubmit` meer in een element. Dit is natuurlijk het onderwerp van de Javascript cursus
  - Enkel `script` tags mogen aanwezig zijn
- **Het semantisch web**:
  - gebruik zoveel mogelijk **semantische elementen**

    "Semantiek" wil zeggen "*betekenis", "semantische elementen"* zijn dus *"betekenisvolle elementen"*.

    in het verleden dachten ontwikkelaars vooral aan de opmaak van hun site: *hoe ziet er het eruit*? daarbij vormden ze algemene elementen om met CSS, vb `<div class="kop">Titel</div>` tot iets dat eigenlijk al bestaat, nl. een `h1`

    denk eerst aan de **inhoud**: gebruik **toepasselijke** elementen als ze bestaan:

`h1, h2, code, address, article, section, aside, header, time`…

- HTML5 bevat **nieuwe semantische elementen**: `article, section, header, footer, aside, nav`
- Er zijn ook specifieke vocabulaires voor bepaalde inhouden: *microdata, microformats* en *RDF*, zie http://schema.org/
  Deze formaten ondersteunen een "web of data" waarbij de elementen die je gebruikt zowel voor machine als voor mens begrijpelijk zijn

We steken niet graag een waarschuwend vingertje op, maar toch één "***Not done***" principe:



gebruik geen tabellen voor lay-out: tabellen dienen enkel voor gegevens.
Vind je dit moeilijk, vraag dan raad aan je instructeur webontwikkeling, die toont je het betere werk!

3.4 HTML5, CSS3 ondersteuning

Als je HTML5 of CSS3 wil gebruiken, moet je minstens nagaan of de belangrijkste browsers dit wel ondersteunen.

Er zijn enkele websites waar je daarover informatie kunt vinden:

- www.html5test.com test jouw browser en geeft overzichten van andere
- http://www.quirksmode.org/css/contents.html is een goede website voor alle CSS/Javascript ondersteuning
- http://caniuse.com/ geeft een overzicht van CSS ondersteuning
- http://css3test.com/ test je browser specifiek voor CSS3

Voor browsers die HTML5 helemaal niet ondersteunen zijn er toch oplossingen in de vorm van scripts, templates en stylesheets die de ontbrekende elementen, attributen en functionaliteit detecteren en toevoegen aan je pagina:

- http://code.google.com/p/html5shiv/ is de Google CDN versie van het *HTML5shiv* (of *shim*) script dat automatisch de nieuwe HTML5 elementen aanmaakt voor een browser die ze niet kent (IE6). Dit stopt foutberichten en laat je ook toe ze te stylen met CSS, maar verwacht niet dat tags zoals `canvas` en `video` ook werken!
- http://modernizr.com is een javascript dat ondersteuning detecteert in de browser. Je kan dan via instellingen de ontbrekende features activeren
- http://html5boilerplate.com/ is een html/css/js template voor HTML5

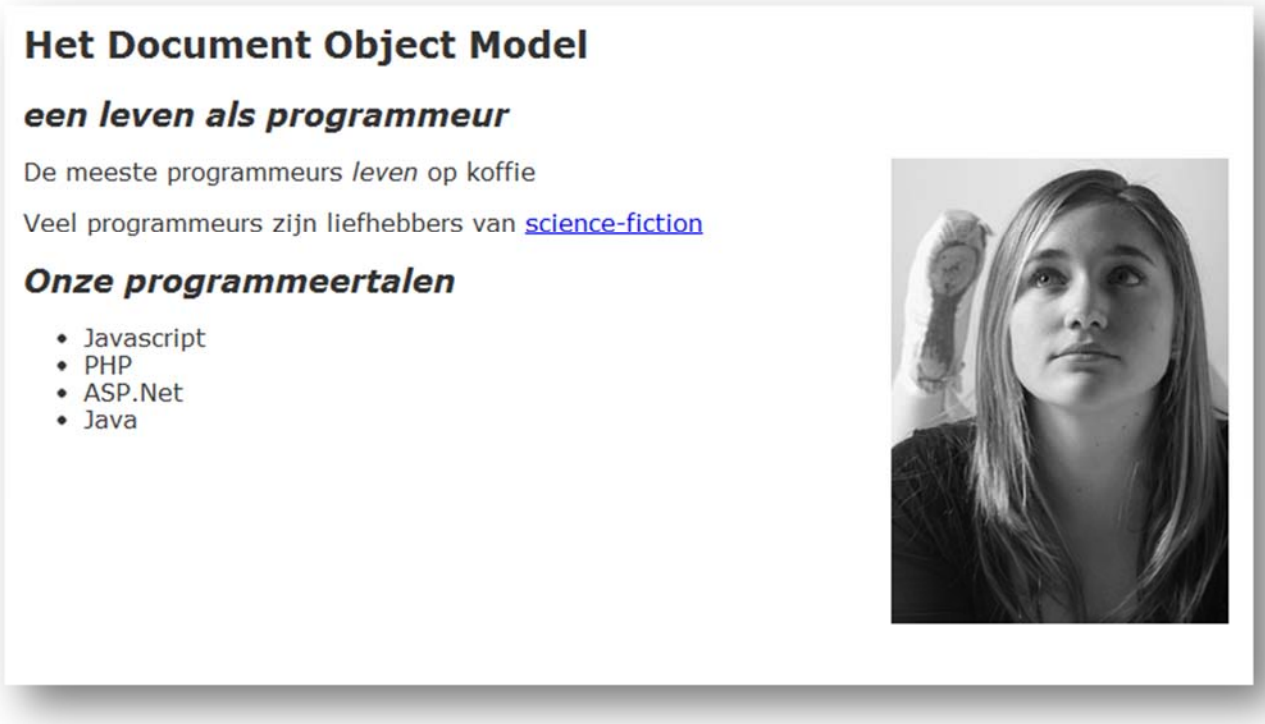
Je hoeft deze libraries nog niet onmiddellijk uit te pluizen, we gebruiken sommige verderop in deze cursus.

4 HTML

De snelste manier om HTML te leren is via een voorbeeldpagina.

4.1 De HTML5 versie

Hier zie je de pagina *basis_html5.html* (zie startbestanden):



De code voor deze eenvoudige pagina is zo:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset=utf-8>
<title>JS PF project: het Document Object Model</title>
<style media=all>
  body {font-family:Verdana, Geneva, sans-serif;color:#333;}
  img {float:right;}
  #container {width:800px;margin: 0 auto;}
  h2{font-size:1.3em;font-style:italic;}
</style>
</head>
<body>
<div id=container>
  <article>
     <h1>Het Document Object Model</h1>
     <h2>een leven als programmeur</h2>

    <!-- dit is HTML commentaar onzichtbaar in het beeld-->
```

```
    <p><img src="../img/linda.jpg" alt="Linda, een front-end programmeur" title="Linda,
  een front-end programmeur" >De meeste programmeurs <em>leven</em> op koffie</p>
    <p>Veel programmeurs zijn liefhebbers van <a
  href="http://en.wikipedia.org/wiki/Science-fiction">science-fiction</a></p>
    <aside>
      <h2>Onze programmeertalen</h2>
      <ul>
        <li>Javascript</li>
        <li>PHP</li>
        <li>ASP.Net</li>
        <li>Java</li>
      </ul>
    </aside>
  </article>
</div>
</body>
</html>
```

Dit document is opgemaakt volgens HTML5, hierna vergelijken we met dezelfde pagina in XHTML1.0 Transitional.

Achtereenvolgens zien we in deze code de volgende statements en elementen:

```
<!DOCTYPE HTML>
```

Deze eenvoudige `doctype` declareert deze pagina als HTML5

```
<html>
```

Het `html` element is het **root element** van het document: het opent als eerste en sluit af als laatste. Er mag maar 1 `html` element zijn.

```
<head>
```

Het `head` element bevat zelf elementen die niet zichtbaar zijn, maar wel van belang: *meta-informatie.*
Net voor het `body` element wordt het afgesloten.

```
<meta charset=utf-8>
```

Een `meta` element bevat een attribuut `charset` die de **karakterset** bevat: dit document is opgeslagen in `utf-8`.
`utf-8` is een internationale karakterset die lettertekens van zowat alle talen ter wereld kan bevatten: dit document kan zowel Arabische als Chinese tekens bevatten.
Er kunnen nog andere `meta` elementen zijn met andere soorten informatie, zie verder bij ***meta-data***.

Bemerk de afwezigheid van aanhalingstekens en het niet afsluiten van deze *empty* tag!

```
<title>JS PF project: het Document Object Model</title>
```

Het `title` element bevat de tekst die je bovenaan het browservenster ziet. Vergeet dit nooit in te vullen!

Deze tekst is zeer belangrijk voor SEO (zoekresultaten in Google etc…) en moet kort en duidelijk zijn.

```
<style media=all>
```

De `style` tag bevat een *intern stylesheet*: een aantal CSS statements tussen de *open* en de *close* tag die gelden voor deze pagina alleen.

Deze tag heeft een `media` attribuut met de waarde "*all*": dat betekent dat dit stylesheet geldt voor alle media: *screen, print,…*

```
<body>
```

Het `body` element bevat de werkelijke inhoud van de pagina. Deze tag wordt afgesloten net voor de afsluittag van het `html` element.

De meeste elementen die hierin vervat zitten, zullen zichtbaar zijn op de pagina. De opmaak (kleur, plaats, grootte,…) die ze zullen hebben, wordt bepaald door CSS.

```
<div id=container>
```

Een `div` element is een **neutrale container**, een *doos.* Het wordt meestal gebruikt om andere inhoud in te verpakken om greep te krijgen op de opmaak ervan.

Dit voorbeeld heeft een `id` attribuut met de waarde "*container*". Deze waarde moet uniek zijn, er mag geen tweede "*container*" in de pagina zitten. Het kan gebruikt worden om er via CSS opmaak op te zetten (het stylesheet bevat een `#container` selector).

```
<article>
```

 Een `article` element bevat een **inhoud als geheel**, een *artikel*, een *nieuwsbericht*, een *commentaarbericht*, een *tutorial*, een *collectie foto's*, een *blogbericht*, etc… De bedoeling is het als geheel te kunnen identificeren. Het element heeft geen opmaakfunctie

```
<h1><h2>
```

`h1,h2, … , h6` elementen zijn **koppen**.

```
<!-- dit is HTML commentaar onzichtbaar in het beeld-->
```

Alles tussen `<!--` en `-->` wordt niet getoond op de pagina. Het is **commentaar** in de code. Je gebruikt het om dingen aan te duiden voor jezelf of andere ontwikkelaars.

```
<p>
```

Een `p` element is een **alinea** (*paragraph*) die tekst en andere elementen kan bevatten.

```
<img>
```

Een `img` element is een **figuur**: foto, tekening, icoon, etc…
Het heeft een `src` attribuut met het pad van het beeldbestand.
Het heeft ook een `alt` attribuut met een tekst die getoond wordt als het beeld om de een of andere reden niet kan geladen worden.

Diezelfde tekst vinden we terug in een `title` attribuut, die zorgt voor een *tooltip* als je met de muis erover beweegt.

Hier hebben we de waarde wel voorzien van aanhalingstekens omdat het meerdere woorden (tokens) betreft.

```
<aside>
```

 Een `aside` element is een **zijbalkje**. Het kan een tekst bevatten of een menu, of wat je maar wil, maar is bedoeld voor inhoud die niet tot de hoofdinhoud behoort. Het krijgt dikwijls een opmaak die het opzij op de pagina zet.

4.2 De XHTML versie

Deze pagina ziet er identiek uit (zie *basis_xhtml.html*), maar de code verschilt enigszins:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>JS PF project: het Document Object Model</title>
<style type="text/css" media="all">
  body {font-family:Verdana, Geneva, sans-serif;color:#333;}
  img {float:right;}
  #container {width:800px;margin: 0 auto;}
  h2{font-size:1.3em;font-style:italic;}
</style>
</head>
<body>
<div id="container">
      <h1>Het Document Object Model</h1>
      <h2>een leven als programmeur</h2>
    <!-- dit is HTML commentaar onzichtbaar in het beeld-->
    <p><img src="../img/linda.jpg" alt="Linda, een front-end programmeur" title="Linda,
  een front-end programmeur" />De meeste programmeurs <em>leven</em> op koffie</p>
    <p>Veel programmeurs zijn liefhebbers van <a
  href="http://en.wikipedia.org/wiki/Science-fiction">science-fiction</a></p>
    <div id="sidebar">
      <h2>Onze programmeertalen</h2>
      <ul>
        <li>Javascript</li>
        <li>PHP</li>
        <li>ASP.Net</li>
        <li>Java</li>
      </ul>
    </div>
</div>
</body>
</html>
```

De verschillen:

- de DOCTYPE:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

  declareert de pagina als XHTML 1.0 Transitional.

- het `html` element heeft een `xmlns` attribuut:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

Is niet verplicht, maar geeft aan dat deze tags tot XHTML behoren.

- de `meta` tag is anders:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

het bevat een `http-equiv` en een `content` attribuut die het MIME type (text/html) en de karakterset bevatten

- het `style` element heeft een `type` attribuut:

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

is in XHTML een **verplicht** attribuut voor `style`. Het geeft aan dat dit stylesheet gemaakt is met CSS (er bestaan nog andere opmaaktalen)

- lege elementen zijn afgesloten:

```
<meta /> <img />
```

is verplicht!

- alle attribuutwaarden zijn in aanhalingstekens geplaatst:

```
<div id="container">
```

verplicht (en ook veilig)

- geen `article`, `nav` of `aside` elementen:

```
<div id="sidebar">
```

Deze elementen zijn ongekend in XHTML. Waar nodig zijn ze vervangen door neutrale `div` elementen met een `class` of een `id` attribuut om ze te kunnen identificeren en opmaken.

4.3 Het Document Object Model

Nu we idee hebben van HTML bedenken we nog eens wat een browser doet om dit allemaal op je scherm te krijgen. Als je de vorige pagina wil bekijken door op een hyperlink te klikken:

Wat gebeurt er in/door de browser?

1. de browser richt een `HTTP Request`, aan de webserver met URL voor die pagina
2. de webserver antwoordt: hij stuurt de HTML pagina terug: de `HTTP Response`
3. in de browser zit een *parser* (programma) die de HTML codes vertaalt naar nodes(elementen) en ermee een ***document tree*** bouwt in zijn intern geheugen
4. deze *tree* wordt gebruikt om er een beeld mee op te bouwen

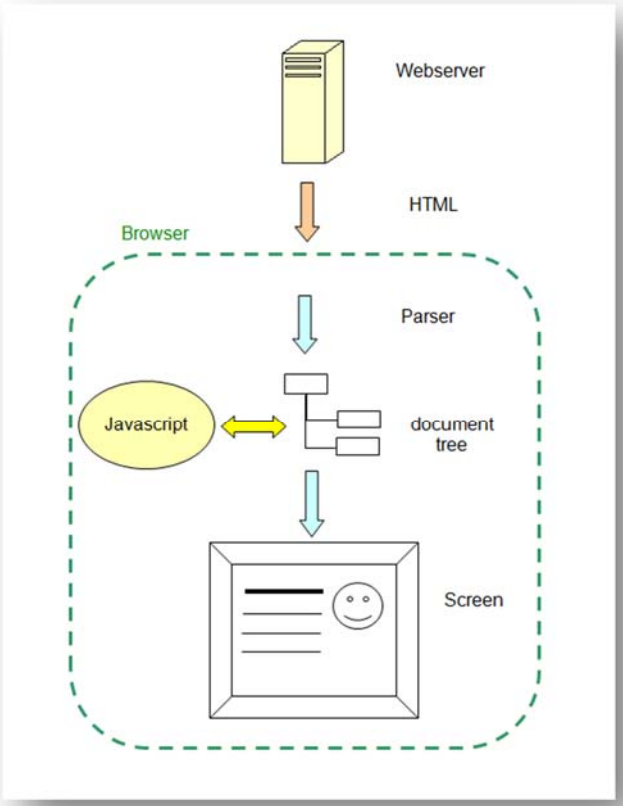
Het beeld op het scherm is dus **niet** direct gekoppeld aan de HTML code, wel aan de *document tree*.

We spreken van het **Document Object Model** (DOM), dat is het model dat gevolgd wordt om de document tree te maken.

4.3.1 Javascript

Elke browser heeft een *Javascript engine* ingebouwd. Met Javascript kan je – op de *client* PC - de *document tree* rechtstreeks manipuleren, zodat dit een onmiddellijk effect heeft op het beeld.

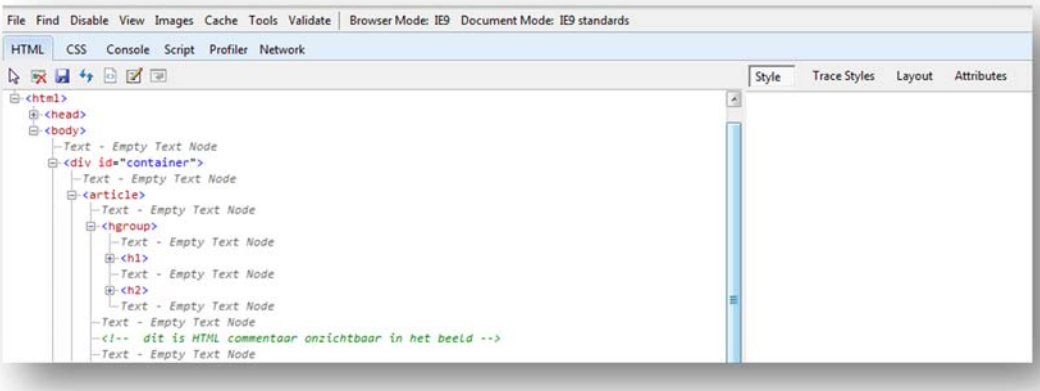
Javascript stelt ons dus in staat de oorspronkelijke HTML pagina helemaal te wijzigen **nadat** deze ingelezen werd. Dit is het onderwerp van de JS cursus.



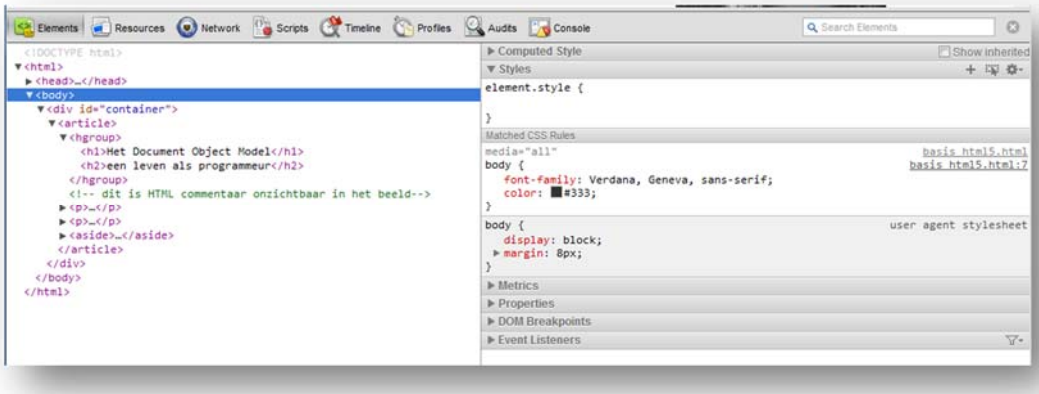
4.3.2 De document tree zichtbaar maken

We kunnen de *document tree* bekijken met een **webdeveloper tool**: gebruik de developer tool in elke browser (klik op de relevante tab) om de volgende beelden te krijgen (*basis_html5.html):*

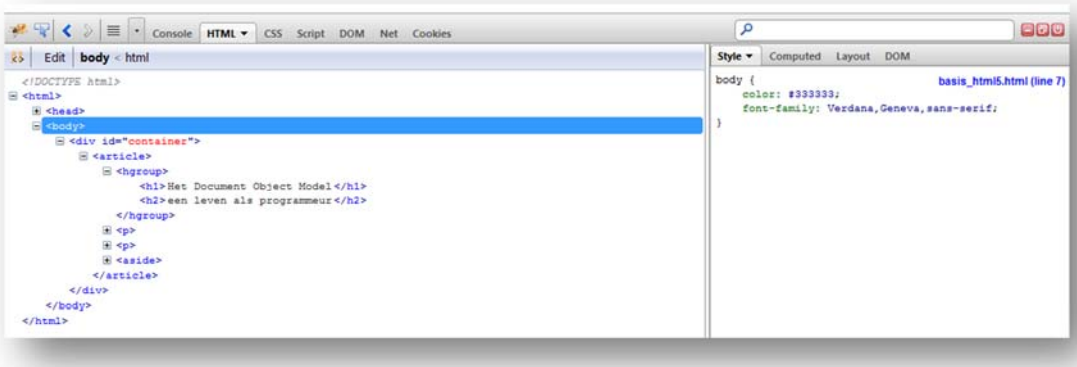
Developer tools in IE (*html* panel)



Developer tools in Chrome (*Elements* panel)



Firebug in FireFox (*html* panel)



Er zijn uiteraard verschillen, maar allemaal laten ze je toe te navigeren in de verschillende *takken* van de *document tree* om de structuur te volgen.

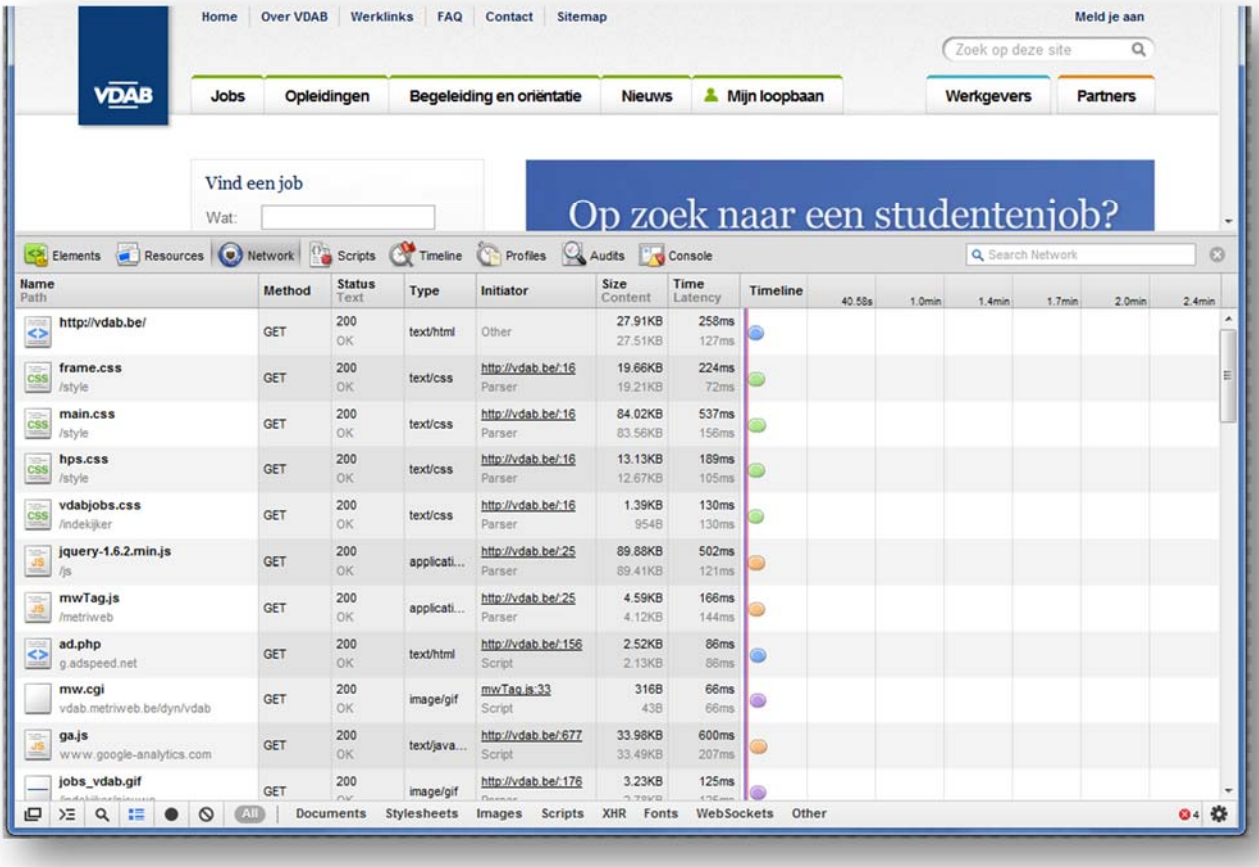
In deze *document trees* zie je dus de html elementen als **element** en niet als **tag**. Je ziet hoe ze zich verhouden tot elkaar: zitten ze in een ander element? Zijn ze broertje van elkaar? (*sibling*). Je ziet ook hun attributen.

Aan de rechterzijde zie je ook de CSS *styles* die op een geselecteerd element van toepassing zijn.

4.3.3 **Request en Response**

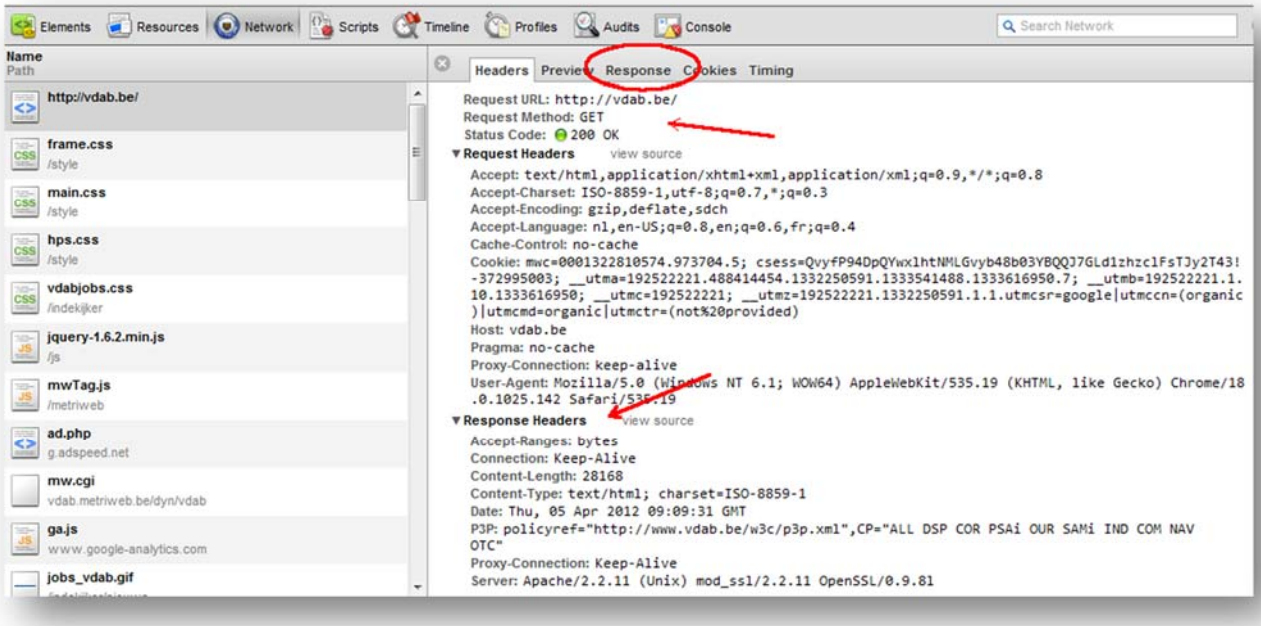
Puur ter info: terwijl we nu toch bezig zijn in een web developer tool (hier in Chrome), kan je ook eens kijken naar het netwerkverkeer van de browser en de Http Request en Response.

Activeer de developer tools op het *Network* panel en surf naar www.vdab.be. Je ziet iets in deze aard:



Wat je hier ziet zijn in feite **alle bronnen** die geladen worden voor deze pagina. Je bemerkt dat er veel meer is dan de html pagina zelf (de eerste resource): er zijn meerdere stylesheets, javascripts en een heleboel figuren. Je ziet rechts ook de tijd die het duurt om zo'n bron te laden.

Klik je nu op de bovenste bron http://vdab.be, dan zie je iets dergelijks:



Je ziet de Request header met al zijn (ingewikkelde) parameters en als je bovenaan op de Response tab klikt zie je de HTML code die teruggestuurd werd. In welke standaard is deze pagina gemaakt?

Misschien komt deze informatie ons nog eens van pas.

4.4 Een HTML document

Een HTML-document is een tekstdocument met een `.html` of `.htm` extensie. De extensie kan ook anders zijn, wat belangrijk is, is de inhoud die bestaat uit pure tekst met een bepaalde karakterencodering: die is meestal `utf-8`.

De tekst bevat code in de vorm van **tags**. Ook scripts (PHP, Java, …) kunnen html pagina's genereren.

Een html document wordt dus door de browser in een *document tree* omgezet volgens het **Document Object Model** (DOM). Om dit te kunnen moeten alle onderdelen aanwezig (ingelezen) zijn:

Een html document bestaat uit:

- een HTML **Declaration** bovenaan
- html **elementen**
- andere **nodes** zoals commentaar
- **extra bronnen** (figuren, stylesheets, scripts,…) die door de browser worden ingeladen

Het is pas als al deze zaken in een tree omgezet zijn (*parsing*) dat het document volledig getoond kan worden.

4.4.1 Elementen en tags

Een html **element** schrijf jij als een **tag**: een tag is een code tussen **scherpe haakjes** (groter-dan en kleiner-dan teken) :

Bijvoorbeeld: het `h1` element schrijf je als

```
<h1>Het Document Object Model</h1>
```

en dit wordt door een browser getoond als



Schrijf tags bij voorkeur in kleine letters.

Er bestaan twee soorten elementen (en ook tags):

- **Containerelementen (-tags)**
  Dit zijn elementen met een **inhoud**: hun tag bestaat uit een **begintag** en een **eindtag** met de inhoud ertussen. De eindtag is identiek aan de begintag, maar begint met een slash (/).

  Een voorbeeld is een alinea (`p`) met een tekst waarvan een deel benadrukt is door een `em` element:

  ```
  <p>In deze zin is dit <em>heel belangrijk</em></p>
  ```

  De alinea is vervat tussen de begin- en eindtag van het `p` element, het `em` element omvat het te benadrukken gedeelte van de tekst.

  Let erop dat je **geen *cross-over*** maakt! veronderstel dat er nog een alinea volgt en dat het eerste gedeelte ook benadrukt wordt, dan mag het `em` element niet doorlopen. Dit is fout:

  ```
  <p>In deze zin is dit <em>heel</p>
  <p> belangrijk</em> en dit weer minder</p>
  ```

  Dit is correct:

  ```
  <p>In deze zin is dit <em>heel</em></p>
  <p><em> belangrijk</em> en dit weer minder</p>
  ```

  Hoe deze tekst er uit zal zien hangt af van de CSS: die bepaalt het uitzicht van zowel `p` als `em`. Bij gebrek aan CSS zal de browser zijn **standaardinstellingen** voor die tags toepassen.

- **Lege elementen (open tags)**
  Dit zijn elementen zonder inhoud: er zit geen tekst in en ook geen andere elementen. Ze hebben geen eindtag nodig hebben en komen dus “alleen" voor.

Afhankelijk van de HTML standaard die je hanteert moeten ze wel of niet met een slash aan het eind afgesloten worden.

Een voorbeeld is het *break* element `br` die een line-break in een lijn veroorzaakt:

XHTML: `<br />`

HTML5: `<br>`

Een ander voorbeeld is het `img` element dat enkel attributen heeft (hier xhtml):

```
<img src="kim.jpg" id="instructeur" alt=”foto van instructeur Kim” />
```

Opmerkingen:

- Er is één speciaaltje: de `script` tag kan als lege tag geschreven worden als hij een extern javascript laadt:

  ```
  <script src="../js/ mijnscript.js" />
  ```

  doe dat echter nooit, hij zal niet meer werken.
  Schrijf een `script` tag altijd met begin- en eindtag!

  ```
  <script src="../js/ mijnscript.js"></script>
  ```

Witruimte

Alle witruimte in een html document wordt genegeerd. Alle spaties, tabs, *Enters* en alle aanverwante codes worden door de parser(browser) gewoon genegeerd. Een enkele spatie wordt wel weergegeven, maar elke reeks spaties wordt herleid tot één enkele spatie.

Foute tags

Wanneer een browser een tag tegenkomt die niet herkend wordt (bvb `<bold>` in de plaats van `<b>`) dan wordt deze tag gewoon genegeerd. Op dezelfde manier worden foute of ongeldige attributen ook genegeerd.

Standaardopmaak

Ook goed om te weten is dat HTML tags van een browser een standaard-opmaak krijgen: titels zullen bijvoorbeeld groter en in het vet worden weergegeven, boven en onder elke paragraaf tekst wordt een beetje ruimte gelaten, een link wordt onderlijnd, de achtergrondkleur van je pagina is wit, etc... Het is belangrijk om je te realiseren dat deze standaard opmaak in elke browser anders kan zijn.

Met CSS kan je die opmaak dan wijzigen.

4.4.2 Attributen

Alle elementen kan je bepaalde parameters meegeven via een attribuut. Ze worden geschreven tussen de haakjes van de begintag:

```
<tag attrib1="waarde" attrib2="waarde1 waarde2">inhoud</tag>
```

Eerst de **naam** van het attribuut, een **gelijkheidsteken** en tenslotte de **waarde** voor het attribuut. Je hebt ook al een paar keer gehoord dat in HTML5 aanhalingstekens niet vereist zijn en waarom wij ze wel schrijven.

Attributen voegen extra informatie toe aan een element.

Een voorbeeld:

```
<a href="http://www.vdab.be" title="de werkwinkel">VDAB</a>
```

In een `a` element is het `href` attribuut de verwijzing naar een URL voor de hyperlink naar de website van de vdab. Het `title` attribuut toont een tekstje als je met de muis over de link zweeft.

Core en eigen attributen

Veel attributen zijn *eigen* aan het element, zo is het `href` attribuut eigen aan een `a` element en kan je het niet gebruiken in een `img` element. Als je niet zeker bent moet je dus nagaan of een attribuut mag gebruikt worden in een element.

Andere attributen, de zogenaamde ***core*** (ook *global*) ***attributes***, kan je **in elk element** gebruiken, de belangrijkste zijn:

- `id`: om een unieke *identifier* toe te kennen aan het element
- `class`: om één of meerdere CSS *classes* toe te kennen
- `style`: om direct CSS *style rules* toe te passen
- `title`: om een tooltiptekst te voorzien
- `lang`: om een *language* (taal) aan te geven
- `data-`: HTML5 attrib om gegevens aan een element te koppelen (zie verderop)

Foute attributen bestaan niet

Attributen die de browser niet herkend worden, worden gewoon genegeerd. Als je merkt dat een attribuut niet werkt, komt dat waarschijnlijk door een schrijffoutje.

Er is niets *mis* mee een attribuut in een element te plaatsen dat niet beschreven is in de HTML standaard. Wil je bijvoorbeeld een attribuut `betekenis` in een element plaatsen dan zal de browser daar niet op afknappen, hij zal het gewoon negeren.

Zo zijn er *auteurs* en *Front-End Frameworks* die volop gebruik maken van eigen of semi-standaard attributen om meer semantiek te kunnen geven aan elementen. Een voorbeeld is het (oude XHTML) `role` attribuut:

```
<ul role="navigation sitemap">
    <li href="downloads">Downloads</li>
    <li href="docs">Documentation</li>
    <li href="news">News</li>
</ul>
```

Hier duidt de auteur de betekenis van de `ul` aan als *navigatie* en *sitemap*.

Meestal is het gebruik van zo'n attribuut enkel de eerste stap en zal een *Framework* nadien met Javascript iets gaat doen met alle elementen die dit attribuut hebben.

Een ander voorbeeld is het gebruik van meta-data voor Facebook verwijzingen. Sommige websites bevatten een aantal meta-data (*Open Graph*) die dienen om goed opgenomen te worden in Facebook previews etc…

```
<meta property="og:title" content="Jobs voor werkzoekenden" />
<meta property="og:site_name" content="vdab Jobs"/>
```

Het attribuut `property` is strikt gezien geen geldig HTML5 attribuut, maar dat weerhoudt websites er niet van ze te gebruiken: een fout zal het niet geven.



Attributen voor opmaak zijn verbannen

In HTML4 werden veel attributen gebruikt voor opmaak (`bgcolor` etc..): die gebruiken we nu **nooit meer**: alle opmaak gaat via CSS.

4.4.3 data- attributen

In HTML5 werd het *dataSet* attribuut (ook `data-*`) geïntroduceerd. Het bestaat uit de naam (met koppelteken) `data-` gevolgd door eender welke zelf gekozen term:

`data-role, data-naam, data-stad`, enzovoort. Een voorbeeld:

```
<li  class="user" data-name="John Resig" data-city="Boston"
       data-lang="js" data-food="Bacon">
    <b> John says:</b> <span> Hello, how are you?</span>
  </li>
```

Dit laat ons toe geldige HTML te schrijven en toch (onzichtbaar) extra gegevens in te bedden in een element. De bedoeling hiervan is die data te gebruiken met Javascript. Verschillende *Frameworks* (vb. jQuery) steunen op het gebruik van dataSet attributen. Hoe je precies dit attribuut in Javascript gebruikt, valt buiten de scope van deze cursus.

5 DE HTML ELEMENTEN

5.1 Een HTML document

5.1.1 doctype

Het `doctype` hebben we eerder besproken: je bepaalt er de gebruikte html standaard mee: HTML4, XHTML of HTML5

5.1.2 html

Is het *root* element van het document. Er kan maar één `html` element aanwezig zijn.

5.1.3 head

Ook het `head` element bespraken we hierboven: het bevat *meta-data*.

5.1.4 meta-data

Het `head` element bevat *child* elementen waarvan we de inhoud niet **zien** op de pagina, maar die wel belangrijke informatie bevatten. `title` is er zo eentje, `script` koppelt naar een javascript, `link` koppelt naar andere bronnen o.a. naar een stylesheet.

Een ander belangrijk element is `meta` die exclusief informatie bevat over de pagina. Die informatie kan essentieel zijn – zoals de karakterset in ons voorbeeld – of vrijblijvend: *voor wie het zou kunnen interesseren: de informatie is hier.* De twee standaard properties van `meta` zijn `name` en `content`, waarbij `name` de naam van inhoud geeft en `content` de eigenlijke inhoud.

Al deze onzichtbare informatie noemen we de **meta-data**.

***Welke informatie kan je in meta-data stoppen*?**
in principe alles wat je maar wil, maar de **klassiekers** zijn:

- *content-type en characterset*: zoals eerder getoond

```
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
```

- *description*:

```
<meta name="description" content="National Geographic provides free maps, photos, videos
  and…">
```

- *keywords*:

```
<meta name="keywords" content="national geographic society, national geographics, nat
  geo, national geografics, magazine, channel,…">
```

- *author*: de auteur

```
<meta name="author" content="National Geographic Society" />
```

- *robots*: voor zoekrobotten

```
<meta name="robots" content="all" />
```

- het site fav-ikoontje via een `link` element

```
  <link rel="icon" type="image/x-icon" href="http://images.nationalgeographic.com/
/favicon-cb1274471343.ico" />
```

Maar ook heel andere soort informatie kan meegegeven worden, bijvoorbeeld voor *Facebook* integratie gebruikt met de Open Graph metadata:

```
<meta property="fb:page_id" content="23497828950" />
<meta property="fb:admins" content="1442858697,506432430,104276,7800037" />
<meta property="og:site_name" content="National Geographic" />
<meta property="og:title" content="National Geographic - Inspiring People to Care About
 the Planet Since 1888" />
<meta property="og:description" content="National Geographic provides free maps, photos,
 videos and daily news stories, ….">
<meta property="og:type" content="article" />
<meta property="og:url" content="http://www.nationalgeographic.com/" />
<meta property="og:image" content="http://images.nationalgeographic.com/wpf/sites/...
 cb1343821768.png"/>
```

of voor *Twitter*:

```
<meta name="twitter:card" content="summary"/>
<meta name="twitter:site" content="@NatGeo"/>
<meta name="twitter:site:id" content="17471979"/>
```

Voor meta-data is *The sky the limit*…

5.2 Paginastructuur elementen

HTML5 is er voornamelijk op gericht om **structuur** aan een document te geven d.m.v. zelfbeschrijvende of **semantische elementen**. De volgende zijn nieuw in HTML5: `header, section, article, nav, aside, footer`.
De elementen `div` en `span` zijn algemene HTML elementen.

De nieuwe semantische HTML5 elementen geven ons de kans een duidelijker structuur in de webpagina aan te brengen.
Het is de bedoeling dat iedereen deze elementen gaat gebruiken.

5.2.1 body

Het `body` element bevat het zichtbare deel van de pagina onder de vorm van andere elementen.

```
<!DOCTYPE HTML>
<html>
 <head>
  <title>HTML5</title>
 </head>
 <body>
  <header>
         <h1>mijn HTML pagina</h1>
  </header>
 </body>
</html>
```

5.2.2 header


Het gedeelte bovenaan op de website is de `header`. Het zou minstens één of meerdere koppen (`h1`, `h2`,…) moeten bevatten, maar het kan ook navigatie zijn, een logo, een slogan, een zoekvak of links naar sociale media.

5.2.3 section


Een `section` bevat een groepering van verschillende onderwerpen die over 1 bepaald thema of onderwerp gaat. 1 sectie kan onderverdeeld zijn in meerdere secties.

Een sectie kan nieuwsonderwerpen bevatten, een sectie kan contactgegevens bevatten, een sectie kan terms of service bevatten, of andere `section` elementen.

Waarvoor ***geen*** sectie gebruiken?

- voor scripting,
- enkel voor lay-out en styling


Een `section` dient voornamelijk om een documentstructuur aan te geven en niet om te stylen. Dat betekent niet dat je er geen style aan mag geven... Gebruik eerder een `div` als het enkel dient om op te maken.

```
<aside>
  <section>
         <h1>Overspel</h1>
         <p>Waarom niet eens proberen? er zijn voldoende kandidaten! <a href="">klik
  hier...</a>
  </section>
  <section>
         <h1>Vervroegd pensioen</h1>
         <p>Geef er de brui aan! <a href="">klik hier...</a>
```

```
  </section>
</aside>
```

5.2.4 article


Een `article` omvat een deel van de inhoud van een webpagina. Het artikel vormt een afzonderlijke teksteenheid.

Inhoud van het type artikel kan makkelijk gepubliceerd worden via sociale media, Twitter, Facebook.

Op een pagina kunnen meerdere `article` elementen staan, ze kunnen genest worden. Ze kunnen allemaal een h1 en/of h2 omvatten.

Voorbeelden:

- nieuwsartikel
- commentaar op een artikel
- blog
- forum post

```
<article>
    <header>
            <h1>Sectioning elementen</h1>
            <h2>nodig voor de outline</h2>
    </header>
    <section>
            <h1>het SECTION element</h1>
            <p>blablabla</p>
            ...
    </section>
    <section>
            <h1>het SECTION element</h1>
            <p>blablabla</p>
            ...
    </section>
    <footer>

            <p>Oostende, de <time pubdate datetime="2011-09-05">5 september 2011</time></p>
    </footer>
</article>
```

5.2.5 nav


Dit element is bedoeld voor de navigatie die links bevat naar andere onderdelen op de website. Een pagina kan meerdere `nav` elementen bevatten en ze kunnen zowel horizontaal als verticaal gestyled zijn.

```
<nav>
        <ul>
                <li>...</li>
                ...
        </ul>
  </nav>
```

5.2.6 aside

Dit element bevat inhoud die op zichzelf staat. Ze houdt geen verband met de inhoud van de artikels die eromheen staan.

Dit element kan opzij staan maar ook gewoon midden in een blok dat voorzien is voor een artikel. Vaak heeft het een aparte vormgeving, vb een kader of een kleur.

Voorbeelden

- getuigenis van een klant
- kan een lijst met linkjes bevatten
- kan een blok reclame zijn
- kan een blok zijn met inschrijven op een nieuwsbrief

5.2.7 titels: h1..h6

Titels of kopregels duiden we aan met `h1` t.e.m. `h6`. h1 is de belangrijkste kopregel, `h2` de subkop, ...

```
<header>
  <h1>Sectioning elementen</h1>
  <h2>nodig voor de outline</h2>
</header>
```

5.2.8 footer

Dit onderdeel staat onderaan de pagina en kan allerlei info omvatten.

Een `footer` kan ook onderaan ieder artikel staan en bevat afsluitende informatie die bij dat artikel hoort.

```
<footer>
  <section>
        <h2>Copyright informatie</h2>
  </section>
  <address>
        <a href="http://...">Joerie's blog</a>
  </address>
  <nav>
        <ul>
                <li>...</li>
```

```
                ...
        </ul>
  </nav>
</footer>
```

5.2.9 address

Het `address` element bevat **contactinformatie**, e-mailadres of postadres, **over de auteur**.

Deze informatie kan horen bij een artikel of een sectie of gelden voor het volledige document. Elk article kan dus een eigen element address bevatten.

```
<footer>
 <address>
  Contacteer:
  <a href="mailto:js@example.com">John Smith</a>.
 </address>
 <p><small>© copyright 2038 Example Corp.</small></p>
</footer>
```

5.3 blokelementen

De volgende elementen worden gebruikt om tekst te **groeperen** of te **ordenen**:
`p, pre, blockquote, ol, ul, li, dl, dt, dd, figure, figcaption, div.`

5.3.1 p

Met het `p` element maak je een alinea aan.

```
<p>De kat krabt de krollen van de trap</p>
```

5.3.2 pre

Met behulp van het `pre` element kan tekst in een vaste opmaak (preformatted) worden opgenomen, zonder dat de browser daar iets aan mag veranderen. Bijvoorbeeld als je iets in tabelvorm moet opnemen, maar geen echte tabel wilt gebruiken. Het `pre` element kan daarmee ook gebruikt worden om meerdere blanco regels achter elkaar vast te leggen.

```
<pre>                maxling

it is with a          heart
               heavy

that i admit loss of a feline
        so           loved

a friend lost to the
        unknown
```

```
                                (night)

~cdr 11dec07</pre>
```

5.3.3 blockquote

Het `blockquote` element wordt gebruikt om aan te geven dat het gaat om een blok tekst, dat uit een andere bron geciteerd wordt.

```
<blockquote>
 hier staat de quote
</blockquote>
```

5.3.4 ol

Wanneer de **volgorde** van de items in een lijst **belangrijk** is dan gebruik je **een geordende lijst**. Elk item in de lijst wordt voorafgegaan door vb. een cijfer, letter, ...

```
<ol>
  <li>item een</li>
  <li>item twee</li>
  <li>item drie</li>
</ol>
```

`ol` en `ul` elementen mogen niet leeg zijn: ze moeten minstens één `li` element bevatten (en mogen ook niets anders bevatten)

5.3.5 ul

Een `ul` wordt gebruikt om items op te sommen waarvan **de volgorde geen rol** speelt. Elk item in de lijst wordt voorafgegaan door **een symbool,** een bolletje, vierkantje, ... zijn.



Een navigatie is ook een ul.

```
<ul>
  <li>item één</li>
  <li>item twee</li>
  <li>item drie</li>
</ul>
```

5.3.6 li

Het `li` element definieert een item van een ongeordende of een geordende lijst.

Zie voorbeeld hierboven.

5.3.6.1 geneste lijst

Een geneste lijst is een lijst in een andere lijst:

```
<ul>
  <li>Dit is het eerste item van de lijst</li>
  <li>Dit is het tweede item van de lijst
        <ul>
               <li>Dit is het eerste subitem</li>
               <li>Dit is het tweede subitem</li>
        </ul>
  </li>
  <li>Dit is het laatste item van de lijst</li>
</ul>
```

5.3.6.2 standaardwaarde bij opsomming: attribuut `start`

Met het attribuut `start` kan je aangeven dat de lijst moet beginnen met een andere waarde dan de standaardwaarde 1. **Dit werkt alleen met een geordende lijst.**

Deze manier van werken is nuttig wanneer je je lijst wilt onderbreken met een alinea tekst en daarna verder wilt gaan met dezelfde lijst.



Dit attribuut was afgekeurd in HTML4 maar is bij HTML5 weer opnieuw in gebruik

```
<ol start="3">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ol>
```

5.3.7 dl

Het `dl` element definieert het begin en einde van een **definitielijst**, welke bestaat uit termen en bijbehorende beschrijvingen.

```
<dl>
 <dt> Auteurs
 <dd> John
 <dd> Luke
 <dt> Editor
 <dd> Frank
</dl>
```

5.3.8 dt

Het `dt` element definieert een term in een definitielijst, welke bestaat uit termen en bijbehorende beschrijvingen. De definitielijst wordt gedefinieerd met het `dl` element, een beschrijving met het `dd` element.

Code hierboven

5.3.9 dd

Het `dd` element definieert een beschrijving in een definitielijst, welke bestaat uit termen en bijbehorende beschrijvingen (descriptions). De definitielijst wordt gedefinieerd met het `dl` element, een term met het `dt` element.

Code hierboven

5.3.10 figure en figurecaption

Het `figure` element verwijst naar een onderdeel dat een stuk tekst illustreert. Die illustratie kan van allerlei aard zijn; een grafiek, een foto, een logo, een gedicht, ...

De `figcaption` geeft informatie over het `figure` element.

Voor HMTL5 was er het element `img` om een afbeelding in te voegen.

Het element figure verwijst naar een illustratie die **van allerlei aard** zijn. Het element **img** verwijst naar een **afbeelding** die gelinkt is met het document.

```
<figure>
        <figcaption> Overwerk: de sporen gaan een leven mee...</figcaption>
        <img src="../img/jan_vandorpe.jpg" alt="Jan, herstellende na de crash">

          <p>U bemerkt: </p>
          <ul>
            <li>de wallen onder de ogen</li>
            <li>de grijze haren</li>
            <li>de varkensoogjes</li>
          </ul>
   </figure>
```

5.3.11 div

`div` staat voor *division*, en dient enkel om bepaalde stukken code te verzamelen in een blok.

```
<div>
<h2>Dit is h2</h2>
<p>Deze tekst is ingesloten door het P element.</p>
</div>
```



Voor de komst van HTML5 werd dit element voornamelijk gebruikt om structuur aan te brengen. Om echter de overbodige markup tot een minimum te beperken, mag je enkel een div element gebruiken als er niet reeds een bestaand element (HTML5) aanwezig is dat de taak van deze div kan overnemen.

5.4 Structuur aanbrengen in een HTML pagina

Als *'absolute beginner'* voel je je waarschijnlijk een beetje als een student tandheelkunde (of timmerman, elektricien, vroedvrouw,…) die net uitleg gekregen heeft over de werking van 101 verschillende instrumenten, maar die helemaal niet weet **wat ermee te doen!**

"Help! om deze navelstreng door te knippen gebruik ik een schaar, een scalpel of een laserstraal ???"



Hoe bouw je een HTML pagina op?
welke elementen gebruiken we best en hoe organiseren we de DOM tree?

We hebben het hier enkel over de visuele inhoud van de pagina en de architectuur ervan.

Om dat te kunnen doen, moet je een aantal zaken voor ogen houden:

1. welke **soort inhoud** bevat de pagina?

   zijn dat enkel foto's, of nieuwsartikels, productfiches? Een lijst van webshopproducten? is het een forum, een blog?

2. hoe is de **lay-out** van de pagina?

   hoe ziet de design van de pagina er uit? Je hebt misschien een design gekregen of anders moet je die zelf maken. Is responsive design een vereiste?

Deze twee vragen moeten vooraf beantwoord zijn: je moet weten wat de inhoud zal zijn van de (elke) pagina en je moet een lay-out bepaald hebben.

5.4.1 Content first

De inhoud is het belangrijkste: denk in termen van **objecten/entiteiten**:

de pagina zal
één / een aantal / een lijst van
blogartikel(s) / nieuwsartikel(s) / productbeschrijving(en) /…
bevatten.

Om ons object te *verpakken* hebben we een **container** nodig die

a. **herkenbaar** is als object

b. uniek **identificeerbaar** is

Daarvoor hebben we 3 middelen:

1. kies het meest geschikte **type element**:

   a. indien mogelijk gebruik een **semantisch** element:
   `nav, article, section, header, aside, p, h2, quote, time.`

   b. Bij gebrek daaraan gebruiken we een neutraal element:
      `div, li, span, …`

2. het `class` **attribuut** kan bij neutrale elementen aangeven wat het object is:
   `<div class="vacature">, <li class="commentaar_item">,…`

3. het `id` **attribuut** gebruiken we om elk object **uniek** te kunnen identificeren:
   `<div class="vacature" id="rfc_553312">, <article id="763427943">`

Eenmaal het object een **container** heeft, kunnen we nadenken over de **attributen** (onderdelen) ervan en hoe we die willen tonen:

Is er een titel? Een beschrijving? Begeleidende foto's? Hyperlinks, etc…

Deze brengen we dus onder als *children:*

```
<div class="vacature" id="rfc_553312">
 <h3>1 Php developer & programmer (M/V) (GENT)</h3>
 <h4>Functieomschrijving</h4>
 <ul class="functieomschrijving">
   <li>Je werkt voor een gerenommeerd e-commerce bedrijf.</li>
   <li>Je komt terecht in een tof team van web-designers, front-& back-end
  developers.</li>
 </ul>
 <h4>Verwacht profiel</h4>
 <ul class="profiel">
   <li>PHP en MySQL</li>
   <li>HTML5, CSS3, Javascript, JSON, SOAP </li>
 </ul>
 <h4>Solliciteren;</h4>
 <p>
 <a href="http://www.careerbuilder.com/jobseeker/jobs/RedirectAOL.aspx?
  job_did=J3F5MZ5Y8LLGJMTZ0HK&siteid=int_BEVDAB&show=yes">via onze website</a>. Vermeld
  de referentie: J3F5MZ5Y8LLGJMTZ0HK</p>
 <time datetime="08/10/2012" pubdate="true">Vacaturedatum: 09/08/2012</time>
</div>
```

Eenmaal we tevreden zijn over ons object en zijn attributen, dan kunnen we nadenken over *meervouden* en de grotere structuur:

Hoe verpakken we meerdere objecten? In een andere container? In een lijst? Is er een Hoofdobject en ondergeschikte objecten?

Met andere woorden: wat heb je voor ogen met een pagina over deze objecten?…

Bijvoorbeeld:

```
<ul class="vacature_lijst">
  <li><a href="detail.aspx?id=rfc_553312">1 Php developer & programmer </a>
     <p>bij STARWEB GATEWAY <br> in GENT</p>
```

```
  </li>
  <li>  …  </li>
  <li>  …  </li>
  …
</ul>
```

en op dezelfde manier werken we ons omhoog tot alle objecten verpakt zijn en het tijd is ook de **lay-out** erbij te betrekken.

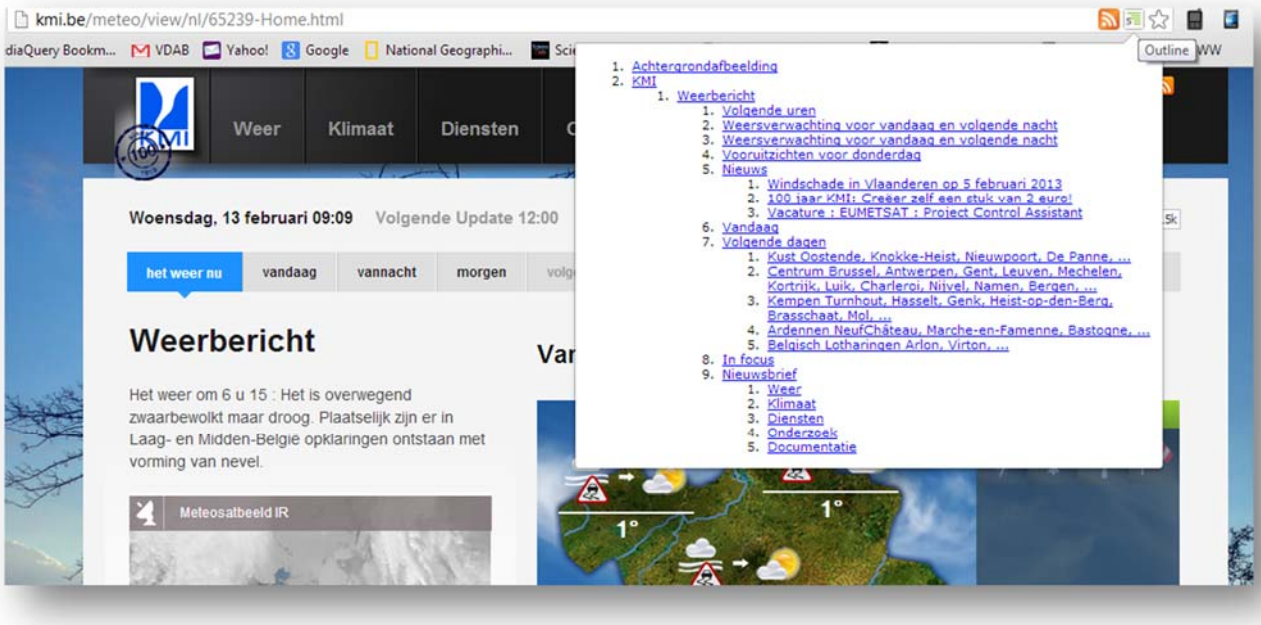
In deze fase hebben we nog niet aan opmaak gedacht, maar door onze goed gestructureerde entiteiten zal dat eenvoudig zijn: voornamelijk via `elementen` en `class` attributen zullen we gemakkelijk CSS kunnen toepassen.

5.4.2 Sectioning en outlines

Sommige types websites hebben nood aan een inhoudstafel of een automatisch genummerde structuur. Dat kan gemaakt worden via scripting of via CSS, maar waarop zal deze inhoudstafel of nummering zich baseren?

HTML5 komt daar beter aan tegemoet door te stellen dat ***sectioning elements*** zullen gebruikt worden in de ***outline.***

De bedoeling is dat de ontwikkelaar kan aangeven wat belangrijk is in een website en wat slechts "randinformatie" is: kop, inhoud, voet, zijbalk



Hierboven zie je de outline van de KMI website zichtbaar gemaakt met de Chrome extension "HTML5 outliner".

Met het juiste script of de juiste CSS kunnen we dan een inhoudstafel of nummering automatisch genereren. Hoe dat precies kan valt echter buiten de scope van deze cursus.

Toch moeten we weten hoe het werkt en welke *sectioning elements* er zijn:

`body, section, article, nav, aside, div`

Elk sectioning element kan zijn eigen `header`, `footer` en *inhoud* hebben en als hij ook een kop (`hx`) heeft dan zal die gebruikt worden om hem aan te geven in de outline.

Bijvoorbeeld deze HTML structuur

```
<body>
<header>
<h1>De Gazet</h1>
</header>
<section id="nieuws">
    <h2>Nieuws</h2>
    <article class="toparticle">
        <h2><a href="artikel.html">Vanaf Oktober meer files</a></h2>
                <p>…</p>
    </article>
  <section class="span2 verbergTitel" id="meernieuws">
      <h2>Meer nieuws</h2>
        <article>
          <h2><a href="artikel.html">Gemeenteraad Genk wil kerncentrale dicht</a></h2>
                <p>…</p>
         </article>
  </section>
</section>
</body>
```

geeft deze outline:

1. De Gazet
    1. Nieuws
        1. Vanaf oktober meer files
        2. Meer nieuws
            1. Gemeenteraad genk wil kerncentrale dicht

De juiste semantische sectioning gebruiken is ook een facet van *Content First*.

5.4.3 de lay-out heeft zijn eigen structuur



In veel webbedrijven gaat de workflow als volgt:

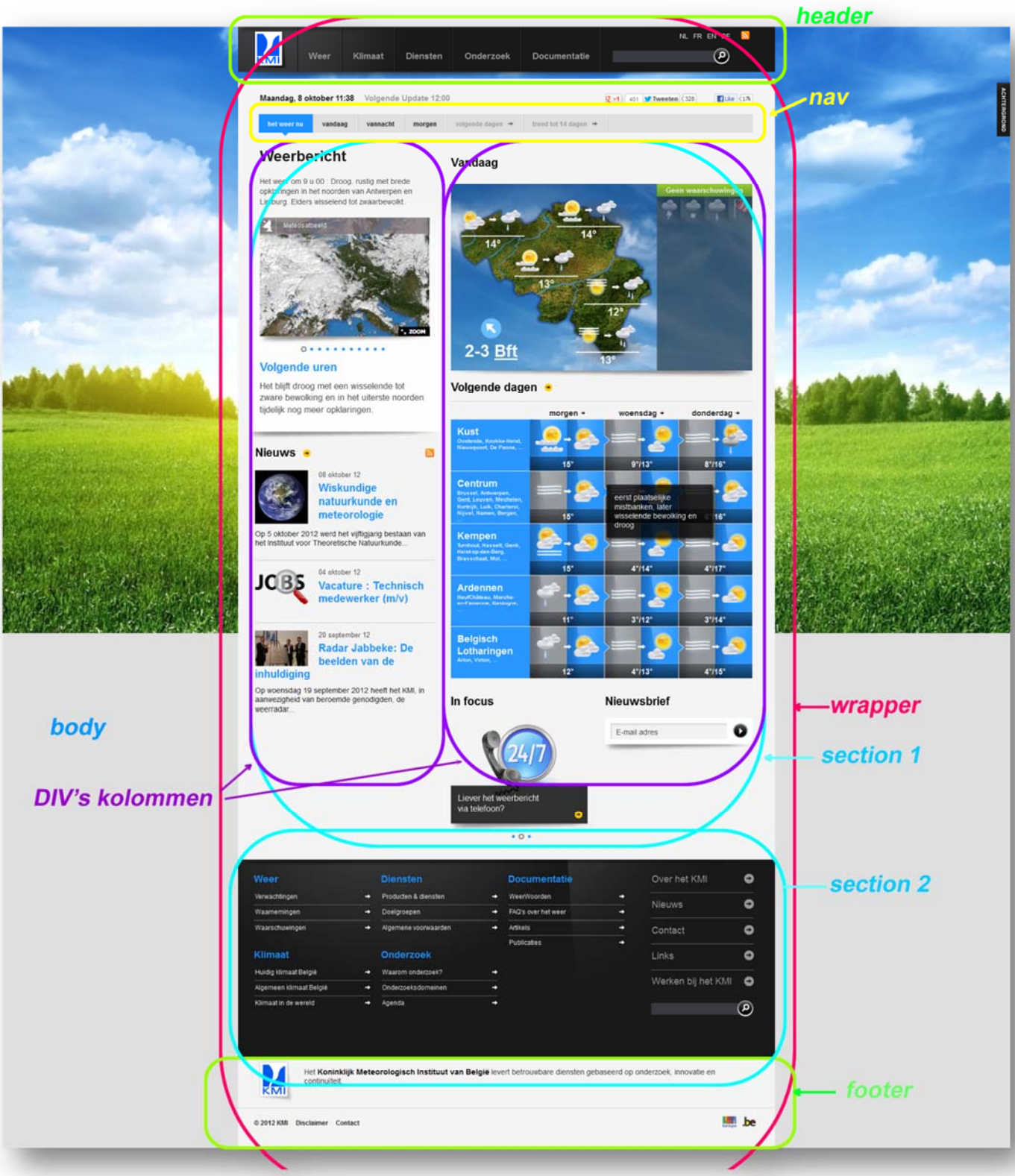
`design -> HTML structuur -> inhoud`

De stap van design (in Photoshop) naar HTML gebeurt door een "*slicer*" of een

front-end ontwikkelaar. Dit zijn experten op gebied van HTML en CSS die in korte tijd een design vertalen naar code. Het is een job die veel kennis maar ook ervaring vergt.

En "*Content First*" komt niet noodzakelijk in conflict met lay-out en design. De designer denkt vooral in termen van ruimte, kleuren en vormgeving, maar een goede slicer weet dat hij de inhoud op een goede structurele manier moet verpakken (of verpakt toegeleverd krijgt)

Maar als "*absolute beginner*" moeten we zelf een "mock-up" of schets maken, die we dan in een structuur zullen vertalen. Laten we een voorbeeld nemen, de home pagina van het KMI:



Door de pagina te verdelen in stukken - van groot naar klein – "*slicen*" we het design. Een mogelijk en vereenvoudigd resultaat van deze pagina kan zijn:

```
<body>
  <div class="wrapper">
        <header>
              <h1 id="logo">KMI</h1>
              <nav id="topmenu">…</nav>
        </header>
        <div class="inhoud">
              <nav id="hoofdmenu">…</nav>
              <section id="sectie1">
                    <div class="kol_2"></div>
                    <div class="kol_3"></div>
              </section>
              <section id="sectie2"></section>
        </div>
        <footer></footer>
</div>
</body>
```

Bespreking:

- de `body` kan gebruikt worden om er een achtergrondfiguur op te plaatsen
- een "wrapper" wordt dikwijls gebruikt om alle inhoud in te vervatten: op die manier kan je –zoals hier – de breedte ervan beperken
- een `header` element is hier de "kop" van de pagina en bevat een `h1` element en een `nav` als navigatiebalk. Hou in gedachten dat `header` evengoed gebruikt kan worden om de "kop" van een `section` of een `article` te omvatten
- de `div` met class inhoud bevat de eigenlijke inhoud en heeft zelf een nav element en aantal `section` elementen als children
- de eerste `section` is zelf opgedeeld in twee div's die als kolommen gestyled zullen worden
- een `footer` bevat infomatie onderaan de pagina
- PS: de werkelijke webpagina heeft waarschijnlijk een andere en complexere structuur

In `div.kol_2` kunnen we dan een aantal `article` elementen plaatsen die een "artikel" bevatten.

In het projectenboek zitten meerdere projecten waar je leert structuren aan te maken.

Conclusie:

HTML-structuur via `elementen`, `class` en `id` is nodig:

- om de server-side scripts entiteiten in te laten verpakken en ze op de pagina te kunnen herkennen als "groep"
- om die entiteiten individueel te kunnen identificeren op de pagina

- om de entiteiten te kunnen opmaken met CSS
- om de lay-out van de pagina mogelijk te maken via CSS

Wat nu nog rest is veel oefenen en ervaring!

5.5 Tekstelementen

Tekstelementen of inline elementen vormen geen visuele blokken. het zijn stukken tekst die na elekaar kunnen komen. ze stapelen niet boven elkaar.

5.5.1 Visueel benadrukken: `b`

Het element `b` wordt gebruikt om aan te geven dat een stuk tekst in een **andere stijl** wordt weergegeven dan de omringende tekst, **zonder dat de tekst extra belang heeft**.

Het wordt gebruikt om kernwoorden in een tekst aan te duiden of een inleidende tekst voor een artikel.

Via CSS wordt dit element van een stijl voorzien. Zonder CSS wordt b weergegevens als vet. Indien je CSS gebruikt dan kan het b element anders weergegeven worden.

```
<b>Bold text</b>
```

5.5.2 Inhoudelijk benadrukken: `strong`

Het element `strong` wordt gebruikt om aan te duiden dat **een stuk tekst inhoudelijk belangrijker is** dan de omringende tekst.

Via CSS wordt dit element van een stijl voorzien. Zonder CSS wordt strong weergegeven als vet. Indien je CSS gebruikt dan kan het strong element anders weergegeven worden.

```
<strong>Strong text</strong>
```

5.5.3 Andere stemming aangeven: `i`

Het element `i` wordt gebruikt om aan te geven dat een stuk tekst **een andere toon of stemming heeft** dan de de omringde tekst. Het zou kunnen gaan om een technisch woord, een spreekwoord, een naam.

Via CSS wordt dit element van een stijl voorzien. Zonder CSS wordt i weergegevens als schuin. Indien je CSS gebruikt dan kan het i element anders weergegeven worden.

```
<i>Italic text</i>
```

5.5.4 Andere betekenis aangeven: em

Het element em wordt gebruikt om het stuk tekst in een zin te benadrukken **dat de betekenis van de zin op een subtiele wijze verandert.**



Via CSS wordt dit element van een stijl voorzien. Zonder CSS wordt em weergegeven als schuin. Indien je CSS gebruikt dan kan het em element anders weergegeven worden.

```
<em>Emphasized text</em>
```

5.5.5 Klein: small



Het element small wordt gebruikt voor teksten zoals voorwaarden, privacyreglement, copyright. Op een pagina met het allerlei tekst is small **bedoeld voor het afwijkend gedeelte**, meestal aan de zijkant of onder de hoofdtekst of in de footer.

```
<small>Small text</small>
```

5.5.6 Titel werk: cite



Het element cite dient om de **titel van een werk** op te geven. Dit kan gaan om: een boek, lied, gedicht, schilderij.



Gebruik het element **q** voor een **citaat**.

```
<cite>Citation</cite>
```

5.5.7 Woordgroep: span

span wordt gebruikt worden om inline elementen**,** bepaalde woorden**,** te **groeperen of identificeren.** Via CSS of JavaScript kunnen ze gelinkt worden naar een bepaalde opmaak.

```
<h2>20 jaar VDAB?</h2>
<p>Gepubliceerd op <time>22 maart 2009</time> door <span class=”author”>Jean Smits</span></p>
```

Het wordt meestal gebruikt als er geen andere semantische element voorhanden zijn.

5.5.8 Link: a

Een hyperlink is een woord of tekst, of afbeelding waarop je kan klikken om je te verplaatsen naar een andere locatie die binnen of buiten de huidige webpagina kan zijn.

Dit wordt opgebouwd met een a element en daarbinnen een href-attribuut. Een a element is pas een hyperlink als het een href attribuut bevat.

Het gebruik van het a element is met HTML5 uitgebreid. Het kan nu ook wijzen op een plaatsbepaling waar een hyperlink had kunnen staan.

```
<a href="http://www.destandaard.be/">de Standaard</a>
```

5.5.9 time en datetime

Het time element dient om een **tijd** of een **datum** weer te geven. Datum/tijd komen er niet automatisch in te staan: jij moet ze er zelf in plaatsen. Negatieve datums, zoals bij de Gregoriaanse kalender zijn niet mogelijk, dus geen BC datums.

Met het attribuut datetime kan je de datum laten weergeven. De vorm is steeds YYYY-MM-DD.
Een tijdstip toevoegen kan door de hoofdletter T toe te voegen achter de datum. Twee cijfers voor het uur in het 24-uurssysteem en twee cijfers voor het aantal minuten.


Vermeld steeds de volledige datum (dus de dag, de maand en het jaar) bij een artikel

```
<p>We open at <time>10:00</time> every morning.</p>

  <p>I have a date on <time datetime="2008-02-14">Valentines day</time>.</p>
```

5.5.10 onderlijnen met u

Het u element wordt gebruikt om tekst onderstreept (Underlined) weer te geven.

```
<p>Do not <u>underline</u> text if it is not a hyperlink!</p>
```

5.5.11 programmeercode aangeven met code

Dit element wordt gebruikt om aan te geven dat het gaat om computercode. Standaard wordt de inhoud dan weergegeven met een lettertype met een vaste letterafstand.

```
<code>A piece of computer code</code>
```

5.5.12 sub

Het sub element wordt gebruikt om tekst in ***subscript*** (iets onder de regel) weer te geven en bovendien vaak in een iets kleinere letter.

```
<p>This text contains <sub>subscript</sub> text.</p>
```

5.5.13 sup

Het sup element wordt gebruikt om tekst in ***superscript*** (iets boven de regel) weer te geven en bovendien vaak in een iets kleinere letter.

```
<p>This text contains <sup>superscript</sup> text.</p>
```

5.5.14 aanduiden met `mark`

Het `mark` element annoteert een stuk tekst met een kleur (net als met een markeerstift). De kleur kan bepaald worden via CSS.

```
<p>Toen Grote Smurf terugkwam van zijn middagdutje, betrapte hij Groene Smurf samen met
  de Smurfin aan het <mark>rollebollen</mark> in het gras!</p>
```

5.5.15 `br`

Het `br` element (break) wordt gebruikt om een nieuwe regel te maken. Bij een nieuwe alinea zal er meer tussenruimte zijn.

```
<p>This is<br />a para<br />graph with line breaks</p>
```

5.6 afbeeldingen met `img`

Om een beeld te tonen wordt het element `img` gebruikt. De afbeelding is altijd een apart extern bestand (JPG, GIF, PNG, …).

Verwar `img` niet met `figure` (HTML5) dat een blokelement is die als container voor een `img` (of stuk code,etc…) en bijschrijft dient. `img` is het beeld zelf.

Ook `canvas` kan gebruikt worden om een beeld te tonen, maar dient eerder voor gescripte afbeeldingen.

Het `img` element heeft twee verplichte attributen: `src` en `alt`, en enkele optionele.

5.6.1 attribuut `src`

De `src` (source) verwijst naar de bron. De waarde van het attribuut is een absolute of relatieve URI.

```
<img src="../img/liesbeth.jpg" alt="Liesbeth" title="Liesbeth is onze webdesigner"/>
```

5.6.2 attribuut `alt`

Het attribuut `alt` is een verplicht attribuut. De waarde is een **tekstomschrijving** van de afbeelding.

Ze wordt getoond **als de afbeelding niet geladen wordt** en in een tekstgeoriënteerde browser. Je kan dat zelf makkelijk controleren door het tonen van afbeeldingen in je browser eens af te zetten.

Ook tekstlezers gebruiken de inhoud van `alt` om die voor te lezen aan een persoon met gezichtsbeperking.



In heel wat browsers wordt de `alt` getoond als je met de muis over de afbeelding beweegt. Dit is niet de primaire bedoeling van `alt` maar wel die van `title`.

```
<img src="lolcat.jpg" alt="Verontwaardigd kijkende kat voor ronde sneeuwfiguur met
  kattekop, zegt: Is it ur way to tell me that im fat? - Just say it" />
```

5.6.3 attribuut `title`

Het `title` attribuut toont een zogenaamde *tooltiptekst* als je er met de muiscursor op rust. Het `title` attribuut is niet verplicht. Deze tekst kan een hele uitleg geven over het beeld.

```
<img src="lolcat.jpg" alt="Verontwaardigd kijkende kat voor ronde sneeuwfiguur met
  kattekop, zegt: Is it ur way to tell me that im fat? - Just say it"
  title="Verontwaardigd kijkende kat voor ronde sneeuwfiguur met kattekop, zegt: Is it ur
  way to tell me that im fat? - Just say it" />
```

Oude HTML attributen die niet meer gebruikt worden zijn

- `width` en `height`
  Vroeger gaf men de breedte en de hoogte van de afbeelding in met deze twee extra attributen, dit heeft echter met lay-out te maken en wordt nu dus met CSS gedaan. Meestal is dit ook onnodig
- `align`
  de uitlijning van een beeld wordt nu ook via CSS gedaan

5.6.4 een afbeelding als hyperlink

Een afbeelding kan je aanklikbaar maken en laten linken naar een bepaalde URL.

Daarvoor plaats je een `img` element **in** een `a` element.

```
<a href="http://icanhascheezburger.com/">
  <img src="lolcat.jpg" alt="Verontwaardigd kijkende kat voor ronde
        sneeuwfiguur met kattekop, zegt: Is it ur way to tell me that im
        fat? - Just say it" />
</a>
```



Het blauwe of paarse randje dat om de afbeelding verschijnt indiceert dat het een aanklikbare afbeelding of een link is. Dit kan je uiteraard aanpassen of verwijderen met CSS.

5.6.5 bestandsformaten bij beelden

`.jpg` (***Joint Photographic Experts Group***)

voor foto's en voor afbeeldingen met rijke kleurschakeringen en gradients. Gebruik bij voorkeur een .jpg voor online afbeeldingen, alle browsers ondersteunen .jpg.

`.gif` (***Graphics Interchange Format***)

voor eenvoudige afbeeldingen en lijntekeningen. De bestandsgrootte kan zeer laag zijn, maar de kwaliteit van een .gif is vooral wat kleur en randen (hier krijg je dikwijls een gekarteld effect) betreft niet zo goed als bij .jpg bestanden.

Een .gif ondersteunt wel transparantie, maar beperkt. Een eenvoudig cirkelvormig logo bijvoorbeeld, sla je indien mogelijk op als een .gif bestand. Een pixel in een .gif is ofwel volledig transparant ofwel niet transparant, maar nooit deels transparant, bv. een opaciteit of doorzichtigheid van 20% kan niet.

`.png` (***Portable Network Graphics***)

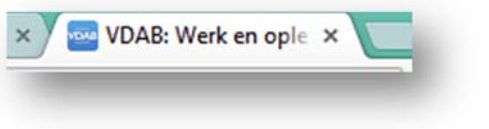
De kleur- en compressiemogelijkheden van dit formaat zijn zeer goed, en het ondersteunt ook -veel beter dan .gif- transparantie. Oudere browsers, zoals Internet Explorer 6, ondersteunen dit formaat niet volledig. Deze browser toont een storend grijs kader op de plaatsen waar de afbeelding transparant hoort te zijn.

`.svg` (***Scalable Vector Graphics***)

beschrijft afbeeldingen die bestaan uit tekst, vectoren en rasterafbeeldingen. SVG is een XML gebaseerd bestandsformaat, met als grote voordeel dat de tekst erin “live" blijft, dus een onderdeel van de DOM is en dus kan gemanipuleerd worden.Met HTML5 komt SVG weer meer in gebruik.

5.6.6 Favicon

Een Favicon is een ikoontje dat je associeert met je website. Het is zichtbaar in de titelbalk of het tabblad van het browservenster of in de bookmarks, etc…



Dit pictogrammetje is standaard van het bestandstype *.ico*, maar kan ook een .png, of.gif zijn. Groottes kunnen 16 X 16, 32 X 32 of 64 X 64 zijn.

Een browser zal standaard zoeken naar een bestand *favicon.ico* (ico bestandstype) in de **root** van je website.

Indien dat je daarvan afwijkt kan je een `link` element in de `head` van het document gebruiken:

```
<link rel="icon" type="image/x-icon" href="/images/mijnikoon.ico"/>
```

Of

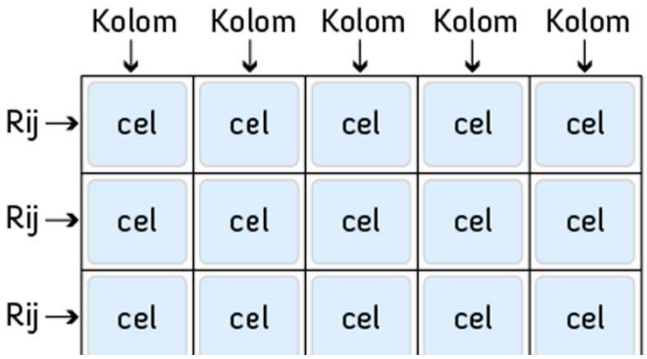
```
<link rel="icon" type="image/png" href="/images/mijnikoon.png"/>
```

5.7 Tabellen

0m **gegevens in een tabelstructuur te tonen, bv**. **cijfergegevens of een adressenlijst**, kun je een tabel gebruiken.

Een tabel is opgebouwd uit **rijen** en **kolommen**. De kruising van een rij met een kolom noemen we een **cel.**

Het voorbeeld hieronder toont een tabel met 3 rijen en 5 kolommen.



5.7.1 eenvoudige tabel

Een tabel wordt gemaakt met het `table` element. Daarbinnen worden **rijen** gedefinieerd met een `tr` element **.** Binnen elke rij worden dan nog eens cellen gedefinieerd met `td` elementen. Code mag je laten inspringen om de tabelstructuur zichtbaar te houden.

```
<table>
  <tr>
        <td>cel 1</td>
        <td>cel 2</td>
  </tr>
  <tr>
        <td>cel 3</td>
        <td>cel 4</td>
  </tr>
</table>
```

5.7.2 tabelrand: attribuut border

Als we gebruik maken van het attribuut `border` (mag je voorlopig nog gebruiken) dan krijgen we een beter beeld van de tabel en zijn structuur.

```
<table border="1">
  <tr>
        <td>cel 1</td>
        <td>cel 2</td>
  </tr>
  <tr>
```

```
        <td>cel 3</td>
        <td>cel 4</td>
  </tr>
</table>
```

5.7.3 lege cel

Elke cel in de tabel wordt dus voorgesteld door de tags `<td>…</td>`. Indien een cel geen waarde bevat dan wordt deze niet gegenereerd. Plaats er dan ` ` (een vaste spatie) in zodat de tabel toch zonder problemen getoond zal worden. Als je dat niet doet, krijg je onvoorspelbare resultaten:

5.7.4 rij- en kolomkoppen

Cellen die zich bovenaan een aantal rijen bevinden kunnen gedefinieerd worden als **rijkop**. Dit doe je door een cel te markeren met `th` elementen in plaats van `td`. Hetzelfde kan verkregen worden voorafgaand aan een aantal kolommen. We spreken dan van een **kolomkop**.

```
<table border="1">
  <tr>
        <th>$nbsp;</th>
        <th>kolomtitel</th>
        <th>kolomtitel</th>
        <th>kolomtitel</th>
  </tr>
  <tr>
        <th>rijtitel</th>
        <td>Gewone cel</td>
        <td>Gewone cel</td>
        <td>Gewone cel</td>
  </tr>
  <tr>
        <th>rijtitel</th>
        <td>Gewone cel</td>
        <td>Gewone cel</td>
        <td>Gewone cel</td>
  </tr>
  <tr>
        <th>rijtitel</th>
        <td>Gewone cel</td>
        <td>Gewone cel</td>
        <td>Gewone cel</td>
  </tr>
</table>
```

5.7.5 cellen samenvoegen: de attributen `colspan` en `rowspan`

Meerdere cellen samenvoegen gebeurt met de attributen `colspan` en `rowspan`.
Beide attributen kunnen gebruikt worden bij het `td` en `th` element.

De betekenis van deze attributen is:

`colspan="x"`
voegt x cellen horizontaal samen, dus voegt de huidige cel samen met x-1 cellen rechts ervan.

`rowspan="y"`
voegt y cellen verticaal samen, dus voegt de huidige cel samen met y-1 cellen eronder.

Het resultaat is dat de **drie cellen** in de eerste kolom worden **samengevoegd.** In de tweede en derde rij moeten dus nog slechts twee cellen worden gedefinieerd.

```
<table border="1">
  <tr>
        <td rowspan="3">abc1</td>
        <td>a2</td>
        <td>a3</td>
  </tr>
  <tr>
        <td>b2</td>
        <td>b3</td>
  </tr>
  <tr>
        <td>c2</td>
        <td>c3</td>
  </tr>
</table>
```

Syntax met colspan:

```
<table border="1">
  <tr>
        <td rowspan="3">abc1</td>
        <td>a2</td>
        <td>a3</td>
  </tr>
  <tr>
        <td colspan="2">b23</td>
  </tr>
  <tr>
        <td>c2</td>
        <td>c3</td>
  </tr>
</table>
```

`rowpsan` en `colspan` magen samen in één gebruikt worden.

Een cel kan niet meer dan één rijgroep of kolomgroep overspannen (`tbody, thead, tfoot, colgroup`).

5.7.6 bijschrift

Om een bijschrift of meer uitleg over de inhoud van de tabel te geven, kan je `caption` gebruiken. Dit element moet direct na de openingstag van de tabel komen. Via CSS kan de `caption` op een andere plaats getoond worden.

```
<table border="1">
  <caption>Test 123</caption>
  <tr>
        <td>Cel 1</td>
        <td>Cel 2</td>
  </tr>
  <tr>
        <td>Cel 3</td>
        <td>Cel 4</td>
  </tr>
</table>
```

5.7.7 de twee content-modellen van table

De web standaard definieert 2 modellen voor `table`:

- een eenvoudig model , de "*Basic Table Module*"
- een wat complexer model de "*Table Module*"

Het *Basic Table* model:

Dit is het model dat we tot nu toe bespraken. Het bestaat uit:

```
<table>
 <caption></caption>
 <tr>
   <th></th>
   <th></th>
   <th></th>
 </tr>
 <tr>
   <th></th>
   <td></td>
   <td></td>
 </tr>
</table>
```

Enkel de elementen `tr` en `td`/`th` zijn verplicht in dit model. Je mag dus gerust een tabel op deze manier maken.

Het volledige *Table model*:

De volledige Table module bevat meer optionele elementen en attributen.

Het skelet:

```
<table>
 <caption></caption>
 <colgroup />
 <colgroup>
  <col />
  <col />
 </colgroup>
 <thead>
   <tr>
          <th></th>
          <th></th>
          <th></th>
   </tr>
 </thead>
 <tfoot>
   <tr>
          <th></th>
          <td></td>
          <td></td>
   </tr>
 </tfoot>
 <tbody>
   <tr>
          <th></th>
          <td></td>
          <td></td>
   </tr>
 </tbody>
</table>
```

Enkele bemerkingen:

- Enkel de elementen `tbody`, `tr` en `td`/`th` zijn verplicht in dit model
- Let er op dat `tfoot` steeds vóór een `tbody` moeten komen
- `thead` en `tfoot` mogen slechts éénmaal voorkomen
- `tbody` kan meerdere malen voorkomen in een `table`
- De lege elementen `col` kunnen enkel als onmiddellijk child van `table` voorkomen als er geen `colgroup` gebruikt is, is dat wel zo dan mogen ze enkel nog in deze laatste voorkomen

Om rijen te groeperen gebruiken we dus `thead, tfoot, tbody`.

- `thead` groepeert header elementen van de tabel.
- `tfoot` groepeert de footer informatie van de tabel.
- `tbody` groepeert de eigenlijke inhoud van een tabel.

5.7.8 col en colgroup

Om kolommen te groeperen gebruiken we `colgroup` en `col`.

- `colgroup` groepeert kolommen; het attribuut `span` duidt dan het aantal kolommen aan dat in deze colgroup zit
- `col` groepeert dan weer één of een aantal kolommen binnen een `colgroup`

```
<table>
  <caption>Tabel</caption>
  <colgroup id="titels" />
  <colgroup span="2" id="midden">
        <col class="dec" />
        <col class="int" />
  </colgroup>
  <colgroup id="totaal" />
  <thead>
        <tr>
              <th colspan="4">Francine's vrolijke fruitmandje</th>
        </tr>
  </thead>
  <tfoot>
        <tr>
              <td colspan="3">Totaal</td>
              <td>651.5</td>
        </tr>
  </tfoot>
  <tbody id="subkop">
        <tr>
              <th>Produkt</th>
              <th>Prijs</th>
              <th>Aantal</th>
              <th>Totaal</th>
        </tr>
  </tbody>
  <tbody>
        <tr>
              <th>Appels</th>
              <td>1,25</td>
              <td>2</td>
              <td>2,50</td>
        </tr>
        <tr>
              <th>Banaan</th>
              <td>2,25</td>
              <td>4</td>
              <td>9</td>
        </tr>
```

```
        <tr>
                <th>Qumquat</th>
                <td>10,40</td>
                <td>100</td>
                <td></td>
        </tr>
  </tbody>
</table>
```



Gebruik tabellen **enkel voor gegevens**: cijfers, overzichten, summiere samenvattingen. Gebruik tabellen **nooit voor lay-out**, daar kan veel beter via andere elementen samen met CSS. Dus niet voor paginalay-out, kolommenlay-out, formulierlay-out, rasters met thumbnails, etc.

Gebruik ook zo weinig mogelijk `rowspan` en `colspan` in dynamische gegenereerde tabellen (uit een database): ze zorgen enkel voor heel wat extra kopzorgen.

5.8 Formulier

We willen een gebruiker de mogelijkheid bieden om vlot informatie in te voeren. Hiervoor bouwen we een formulier. HTML voorziet hiervoor heel wat syntax.

De ingevoerde informatie wordt verwerkt door een script of een stuk programmacode.

Deze cursus behandelt enkel het aanmaken van het formulier. De verwerking van het formulier hoort thuis in cursus PHP of Java of .NET



Een formulier bestaat uit **invoervelden** zoals **tekstveld, keuzelijst**. Aan **een volledig formulier** kunnen **extra kenmerken** of **attributen** in de HTML code voorzien worden. Ook **apart aan een invoerveld** kunnen **attributen** voorzien worden.

Een groot aantal elementen waren reeds in gebruik bij eerdere HTML versies.



HTML5 voegt een aantal nieuwe elementen toe.

De volgende onderwerpen komen aan bod in deze cursus:

- het `form` element en zijn attributen
    a. `name` attribuut
    b. `id` attribuut
    c. `action` attribuut

d. `method` attribuut:
GET-POST

- de invoertypes `input` en `select`:
tekstvak(`text`), paswoordveld(`password`), verborgen veld(`hidden`), bestandskeuzeveld(`file`), verzendknop(`submit`), wisknop(`reset`), keuzerondjes(`radio`), selectievak(checkbx), keuzelijst(drop-down list)
- nieuwe html 5 attributen
`search, email, url, tel, datetime, date, month, week, time, datetime-local, number, range, color`
- invoertype `button`
- gewijzigd invoertype `textarea`
- nieuw invoertype `optgroup`
- HTML5 en zijn nieuwe formulier attributen
`required, placeholder, pattern, autocomplete, multiple, datalist en element list, autofocus, min/max, step`

5.8.1 form

Een formulier wordt gemaakt met de container tag `<form>`. Via CSS kan opmaak toegevoegd worden. De volgende attributen in het form element zijn mogelijk: `name`, `id`, `action`, `method`, `enctype`.

```
<form method="post|get" action="http://www.ict.teno.be/formtest.php" name="frmInloggen"
  id="frmInloggen">
    ...
</form>
```

5.8.1.1 `name` attribuut

Het `name` attribuut geeft een naam aan het veld. Het is deze naam die doorgegeven wordt aan het server-side script. Als je bijvoorbeeld een veld hebt

```
<input type="text" name="bedrijf" />
```

en je vult "*vdab*" in, dan zal de waarde `bedrijf=vdab` doorgegeven worden

5.8.1.2 `id` attribuut

Aan elk element *kan* een `id` attribuut toegevoegd worden. `id` heeft geen invloed op de waarde die doorgegeven wordt. Het wordt meestal gebruikt in CSS of met JavaScript. Dit `id` kan, maar moet niet hetzelfde zijn als `name`. In een groep velden met dezelfde `name`, zoals radio buttons, kan het `id` voor elke radio button uniek zijn.

5.8.1.3 `action` attribuut

Het `action` attribuut bevat een absolute of relatieve url (=bestemmings-url) die een rol speelt bij de afhandeling.

Syntax:

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
```

```
...
</form>
```

5.8.1.4 `method` attribuut

De `<form>`-tag bevat het attribuut method. Tot HTML 4.01 werden enkele de waarden `GET` of `POST` gebruikt.

```
<form method="post" action="http://www.ict.teno.be/formtest.php" name="frmInloggen"
  id="frmInloggen">
    ...
</form>
```

`GET`

Achter de url (bestemmings-url) worden de formuliergegevens geplakt. Deze url wordt geladen in de browser. Dit noemt men ook wel eens een query-string. Alle waarden zijn zichtbaar.

`POST`

De formuliergegevens worden verzameld in een bestand. Dat bestand wordt doorgestuurd naar de server. De bestemmings-url bevat een verwijzing naar de form handler. Deze laatste zorgt ervoor dat de formuliergegevens verwerkt worden naar de database. De waarden zijn onzichtbaar.

5.8.1.5 `enctype` attribuut

Dit attribuut is optioneel. Een formulier wordt normaal altijd doorgestuurd met een standaard `enctype` waarde van "*application/x-www-form-urlencoded*. Maar dat moet je dus niet schrijven tenzij je er een speciale reden toe hebt.

Zo'n geval gebeurt als je een veld `input type="file"` gebruikt om bestanden te uploaden. In dat geval moet je `form` element een `enctype` hebben met de waarde "*multipart/form-data*".

```
<form enctype="multipart/form-data" action="uploader.php" method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Kies een bestand om te uploaden: <input name="uploadedfile" type="file" />
<input type="submit" value="Upload Bestand" />
</form>
```

5.8.2 label

Een `label` geeft een **verklarende tekst voor een formuliercontrol.**

Door gebruik te maken van het `label` element **associeer** je de bijhorende tekst met de *form control*, hierdoor verbeter je de toegankelijkheid van je formulieren.

Een `label` kan op 2 verschillende manieren aan zo'n control gekoppeld worden:

- **impliciet**, door de control **in** het `label` element te zetten

```
<label>Voornaam: <input type="text" name="voornaam" />
  </label>
```

- **expliciet**, door het met zijn `for` attribuut te koppelen aan de `id` van de control.

```
<label for="voornaam">Voornaam: </label>
<input type="text" name="voornaam" id="voornaam" />
```

Als je op een gekoppeld `label` klikt, zie je dat automatisch het inputveld *voornaam* geactiveerd wordt. De `focus` komt erop te staan: er verschijnt een rand omheen, of de achtergrondkleur verandert, de muiscursor staat in het invulveld.

5.8.3 fieldset

Het `fieldset` element kan gebruikt worden om een aantal (bijvoorbeeld thematisch gerelateerde) controls van een formulier te groeperen door er een kader om te plaatsen.

```
<form>
    <fieldset>
      <legend>Personalia:</legend>
      Name: <input type="text" /><br />
      Email: <input type="text" /><br />
      Date of birth: <input type="text" />
    </fieldset>
  </form>
```

5.8.4 legend

Het `legend` element kan in een formulier gebruikt worden om een bijschrift te plaatsen bij het kader dat gemaakt is met het `fieldset` element.

```
<form>
    <fieldset>
      <legend>Personalia:</legend>
      Name: <input type="text" size="30" /><br />
      Email: <input type="text" size="30" /><br />
      Date of birth: <input type="text" size="10" />
    </fieldset>
  </form>
```

5.8.5 input

Het element `input` is een manusje van alles en kan zowel een invulveld als een andere vorm aannemen. Alles hangt af van de waarde van het `type` attribuut.

De attributen `type`, `name`, `id` en `value` kunnen meegegeven worden.

- `type`: het type veld, verplicht, zie verder
- `name`: de naam die de control krijgt, verplicht
- `value`: de waarde van de control (vooraf ingevuld)
- `id`: voor opmaak via CSS of scripting

```
<input type=”...“ id=”...” name=”...” value=”...”>
```

5.8.5.1 tekstvak (text)

`<input type=”text”>` is een invoervak voor tekst bv. een voornaam. Name is verplicht. Id mag.

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <h3>Een invulformulier</h3>
  <label for="vnaam">Geef je voornaam in: </label>
  <input type="text" name="vnaam" id="vnaam" />
</form>
```

5.8.5.2 Een paswoordveld (password)

`<input type=”password”>`. Het paswoord dat de gebruiker intikt wordt weergegeven in bolletjes. De gegevens worden echter niet gecodeerd of versleuteld. Gebruik dit veld daarom met de nodige voorzichtigheid. Zo wordt in de adresbalk van een formulier met als action GET formulier het wachtwoord getoond.

```
<form method="post" action="http://www.ict.teno.be/formtest.php">  <label
  for="jeNaam">Een tekstvak: </label>
  <input type="text" name="jeNaam" id="jeNaam" />
  <label for="jePas">Een paswoordvak: </label>
  <input type="password" name="jePas" id="jePas" />
<input type="submit" />
</form>
```

5.8.5.3 Een verborgen veld(hidden)

`<input type=”hidden”>`. Een control die de inhoud onzichtbaar maakt voor de gebruiker wordt aangemaakt door input van het type "hidden" te maken. De inhoud (zijn `value`) van die control wordt ook doorgegeven aan het script, maar de gebruiker ziet er niets van. Dit kan gebruikt worden om allerlei instellingen van het formulier aan de server door te geven, bv. de taal, de vorm waarin je antwoord verwacht, de naam van dit formulier, etc…

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <label for="zoekterm">U zoekt: </label>
  <input type="text" name="zoekterm" id="zoekterm" />

<input type="hidden" name="searchSetting" id="searchSetting" value="images" />
<input type="hidden" name="lang" id="lang" value="nl" />
<input type="submit" />
</form>
```

5.8.5.4 Een bestandskeuzeveld (file)

`<input type="file">` Is een inputveld met een *browse* knop, dat kan dienen om bestanden te uploaden is een bestandskeuzeveld. In onderstaand voorbeeld hebben we uiteraard ook het serverside script *upload.php* nodig om dit te doen werken.
Merk ook het `enctype` van het `form` element op.

```
<form method="post" enctype="multipart/form-data"
  action="http://www.ict.teno.be/upload.php">
  <label for="bestand">upload: </label>
  <input type="file" name="bestand" id="bestand" />
<input type="submit" />
</form>
```

5.8.5.5 Een verzendknop(submit)

`<input type="submit">` Een formulier verzenden kan via een submit-knop (of verzendknop). Via het attribuut `value` geef je aan welke tekst op de knop moet worden getoond (laat je dit weg, dan zullen de meeste browsers zelf een tekst plaatsen, bv. Send).

```
<form method="post" action="">
  <label for="zoekterm">U zoekt: </label>
  <input type="text" name="zoekterm" id="zoekterm" />
<input type="hidden" name="searchSetting" id="searchSetting" value="images" />
<input type="hidden" name="lang" id="lang" value="nl" />
  <label><input type="submit" value="Versturen" /></label>
</form>
```

5.8.5.6 Een wisknop(reset) - <input type="reset">

`<input type="reset">` Op je formulier kan je een knop voorzien die de ingevoerde gegevens op het formulier wist. Dit wordt een reset-knop genoemd.

Via het attribuut `value` geef je aan welke tekst op de knop moet worden getoond. Laat je dit weg, dan zullen de meeste browsers zelf een tekst plaatsen.

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <label for="jeNaam">Geef je naam in: </label>
  <input type="text" name="jeNaam" id="jeNaam" />

  <label for="isVerborgen">Een verborgen tekstvak: </label>
  <input type="hidden" name="isVerborgen" id="isVerborgen" value="hier de waarde voor het
  verborgen veld" />

  <label for="jePas">Een paswoordvak: </label>
  <input type="password" name="jePas" id="jePas" />
```

```
  <label><input type="submit" value="Versturen" /></label>
  <label><input type="reset" value="Wissen" /></label>
</form>
```

5.8.5.7 Keuzerondjes(radio buttons)



Keuzerondjes (of radiobuttons) worden gebruikt om een lijst met opties te geven waaruit **één mogelijkheid** geselecteerd kan worden. Elke optie wordt aangegeven met een rondje, het geselecteerde item wordt een gevuld rondje:

Een keuzerondje maak je met `<input type="radio">`. Afhankelijk van je besturingssysteem kunnen controls, dus ook deze rondjes, er een beetje anders uitzien, bovenstaande rondjes zijn radiobuttons op Mac.

Wanneer je meerdere keuzerondjes **dezelfde** `name` geeft, worden deze beschouwd als één control, de waarde van het geselecteerde rondje wordt de waarde voor de volledige control.
Merk op dat je wel degelijk verschillende waarden moet opgeven bij `value` ! Het is deze waarde die zal doorgegeven worden.

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <h4>Wat is je favoriete kleur?</h4>
  <label for="kiesRo">Rood </label>
  <input type="radio" name="kiesKleur" id="kiesRo" value="rood" />
  <label for="kiesGr">Groen </label>
  <input type="radio" name="kiesKleur" id="kiesGr" value="groen" />
  <label for="kiesBl">Blauw </label>
  <input type="radio" name="kiesKleur" id="kiesBl" value="blauw" />
  <input type="submit" value="Versturen" />
</form>
```

Je kan in je HTML-code ook al op voorhand opgeven welke van de rondjes de standaardwaarde is, of dus al standaard staat aangevinkt, dat rondje wordt dan meteen ook zwart weergegeven. Dit doe je door het attribuut `checked` toe te voegen aan het keuzerondje in kwestie. Merk op dat het attribuut `checked` maar één waarde kent, nl. "checked".

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <h4>Wat is je favoriete kleur?</h4>
  <label for="kiesRo">Rood </label>
  <input type="radio" name="kiesKleur" id="kiesRo" value="rood" checked="checked" />
  <label for="kiesGr">Groen </label>
  <input type="radio" name="kiesKleur" id="kiesGr" value="groen" />
  <label for="kiesBl">Blauw </label>
```

```
  <input type="radio" name="kiesKleur" id="kiesBl" value="blauw" />
  <input type="submit" value="Versturen" />
</form>
```

5.8.5.8 Aankruisvakjes(checkboxes)



Een aankruisvakje (of *checkbox*) wordt gebruikt om een optie aan te geven die de gebruiker kan aan- of uit zetten.
Een checkbox maak je met `<input type="checkbox">`.
Grafisch wordt dit weergegeven met een vierkantje. Als erop wordt geklikt komt er (afhankelijk van je besturingssysteem en browser) een vinkje of een kruisje in te staan. Nogmaals klikken zet de optie weer uit.

Het grote verschil met radiobuttons is dat je bij radiobuttons normaal gezien één keuze maakt, terwijl je bij checkboxes meer keuzes kan aanvinken. Checkboxes moeten dus niet samenwerken en krijgen dus niet dezelfde `name`.

Bij het attribuut `value` geef je de waarde op die door het script kan gebruikt worden indien de checkbox is aangevinkt.

```
Akkoord? <input type="checkbox" name="chkAkkoord" value="Akkoord" />
```

Je *kan* meerdere aankruisvakjes groeperen, wanneer ze samen moeten gezien worden als één keuze-item, door ze dezelfde `name` te geven. De gebruiker van het formulier kan dan meerdere items tegelijk selecteren. De waarde van de groep van aankruisvakjes is de groepering van de waarden van alle geselecteerde vakjes.

Ook bij een aankruisvakje kan je aangeven dat dit bij aanvang al moet geselecteerd zijn, door het attribuut `checked` toe te voegen. Vanzelfsprekend kan je in dit geval ook meerdere items binnen dezelfde groep "`checked`" maken!

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <input type="checkbox" name="nieuws" id="nb1" value="nb1" checked="checked" />
  <label for="nb1">Ja, ik wil Nieuwsbrief 1 ontvangen!</label>
  <input type="checkbox" name="nieuws" id="nb2" value="nb2" checked="checked" />
  <label for="nb2">Ja, ik wil Nieuwsbrief 2 ontvangen!</label>
  <input type="checkbox" name="nieuws" id="rml" value="roml" checked="checked" />
  <label for="rml">Ja, ik wil nog andere rommel ontvangen!</label>
  <input type="submit" value="versturen" />
</form>
```

5.8.5.9 type = email



Het `type=email` voorziet de mogelijkheid om een e-mailadres in te voeren. Het ziet eruit als een gewoon tekstveld.

De controle op de invoer bestaat alleen uit de controle op de aanwezigheid van het apenstaartje en de aanwezigheid van een punt. Er wordt niet gekeken of de domeinnaam of het opgegeven e-mailadres bestaat.

Een browser die het type niet herkent maakt er een type `text` van.

```
<input name=”e-mailadres” type=”email”>
```

5.8.5.10 type = url

Om een webadres te kunnen ingeven gebruiken we het `type=url`. Een url bestaat uit verschillende delen die gescheiden zijn door een slash en punten.

Een browser die het type niet herkent maakt er een type text van.

```
<input name=”webadres” type=”url”>
```

5.8.5.11 type = number

Dit type geeft aan dat een getal moet ingevoerd worden worden in een invoerveld.

```
<input name=”aantal” type=”number”>
```

5.8.5.12 type = range

Dit type maakt het mogelijk om een cijferwaarde in te voeren via een schuifregelaar. De browser toont enkel de regelaar en niet de waarde. Via JavaScript kan dit aangepast worden.

```
<input name=”aantal” type=”range”>
```

5.8.5.13 type = search

Voor het ingeven van een zoekterm gebruiken we het `type=search`. Het zoekveld is van belang voor de opmaak. Enkel voor in Safari bij een Mac wordt een zoekvak anders getoond.

```
<input name=”zoekterm” type=”search”>
```

5.8.6 textarea

Een `textarea` element geeft een tekstvak dat kan gebruikt worden om de bezoeker een onbeperkte hoeveelheid tekst te laten invoeren. Het kan zo groot gemaakt worden als je wil aan de hand van de attributen `rows` (aantal tekstrijen) en `cols` (aantal karakters horizontaal). In tegenstelling tot `input` heeft het een begin- en een eindtag.

```
<textarea rows="2" cols="20">
  At W3Schools you will find all the Web-building tutorials you need, from basic HTML to
  advanced XML, SQL, ASP, and PHP.
  </textarea>
```

5.8.7 select

Een keuzelijst (*dropdown*) geeft de gebruiker de mogelijkheid te kiezen uit een groot aantal waarden. In HTML creëer je een dergelijke keuzelijst met het element `select`. De verschillende opties binnen de keuzelijst geef je aan met `option` elementen.

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <label for="landkeuze">Uit welk land kom je?</label>
  <select name="landkeuze" id="landkeuze">
        <option>Belgium</option>
        <option>Germany</option>
        <option>France</option>
        <option>Other</option>
  </select>
  <input type="submit" value="submit" />
</form>
```

Een `option` element kan wel of niet een `value` attribuut hebben. Hebben de options geen value, dan wordt de **tekst** van het geselecteerde `option` doorgegeven aan de server. Hebben ze wel een `value` (zoals in het voorbeeld hieronder) dan wordt niet de tekst maar wel de `value` doorgegeven.

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <label for="landkeuze">Uit welk land kom je?</label>
  <select name="landkeuze" id="landkeuze">
        <option value="">--- Kies ---</option>
        <option value="BE">Belgium</option>
        <option value="DE">Germany</option>
        <option value="FR">France</option>
        <option value="??">Other</option>
  </select>
  <input type="submit" value="submit" />
</form>
```

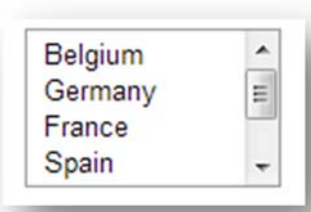
Indien je in dit voorbeeld "Germany" uit de lijst zou kiezen, wordt de waarde "DE" naar de server gestuurd.

- Bemerk dat we een standaardkeuze hebben: *"--- Kies ---"*.
  Merk ook op dat we de `value` van deze `option` de waarde **""** gegeven hebben, een lege string.
  Dit is handig om te **valideren**: als de gebruiker niets kiest, dan blijft de eerste `option` gekozen en ontvangt de server een lege string als waarde. Daar is gemakkelijk voor te testen.
- Je kan op voorhand aangeven welke optie de **standaardwaarde** is, door aan die `option` het attribuut `selected` toe te voegen. Als je die niet gebruikt is de eerste optie altijd geselecteerd.
  in XHTML moet je schrijven `selected="selected"`,
  in HTML5 is `selected` voldoende

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <label for="landkeuze">Uit welk land kom je?</label>
  <select name="landkeuze" id="landkeuze">
          <option value="">--- Kies ---</option>
          <option value="BE">Belgium</option>
          <option value="DE">Germany</option>
          <option value="FR" selected="selected">France</option>
          <option value="??">Other</option>
  </select>
  <input type="submit" value="submit" />
</form>
```

Je kan een keuzelijst ook "open" tonen, met andere woorden: niet als uitklaplijst maar met meerdere opties zichtbaar. Gebruik daarvoor het attribuut `size` in het `select` element.

```
<form method="post" action="http://www.ict.teno.be/formtest.php">
  <label for="landkeuze">Uit welk land kom je?</label>
  <select name="landkeuze" id="landkeuze" size="4">
          <option value="BE">Belgium</option>
          <option value="DE">Germany</option>
          <option value="FR">France</option>
          <option value="SP">Spain</option>
          <option value="UK">United Kingdom</option>
          <option value="??">Other</option>
  </select>
  <input type="submit" value="submit" />
</form>
```

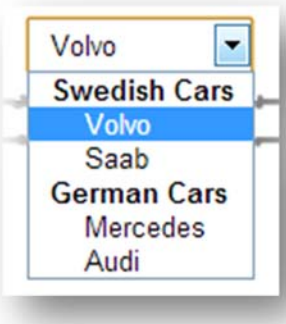


5.8.8 optgroup

Het `optgroup` element kan je gebruiken om een aantal `option` elementen te groeperen. Het heeft een attribuut `label` waarmee je een tussentekst maakt (die niet selecteerbaar is).

```
<select>
    <optgroup label="Swedish Cars">
      <option value="volvo">Volvo</option>
      <option value="saab">Saab</option>
    </optgroup>
    <optgroup label="German Cars">
```

```
      <option value="mercedes">Mercedes</option>
      <option value="audi">Audi</option>
    </optgroup>
  </select>
```



5.8.9 button

Het `button` element is een manusje van alles dat je kan gebruiken voor scripting (met Javascript). Het hoeft niet in een `form` element te staan, maar als je het er in zet kan je het ook gebruiken als submit-knop.

Het heeft een `type` attribuut dat verschillende waarden kan hebben:

- `button`: werkt als algemene knop voor Javascript
- `reset`: werkt als een `input type=reset`
- `submit`: werkt als een `input type=submit`

Opmerking: als je het `type` attribuut vergeet, en het `button` element staat in een `form` dan werkt het standaard als submit-knop!

Een voorbeeld als een scripting knop:

```
<button type="button" id="scriptKnop">start slideshow</button>
```

Hoe deze knop een script zal starten zien we in de Javascript cursus.

5.8.10 input attributen

HTML5 voorziet een aantal nieuwe formulierattributen die gebruikt worden voor validatie (=controle) op de invoer. In eerdere HTML versies werd beroep gedaan op JavaScript om de ingevoerde waarde te controleren. Voor oudere browsers is deze vorm van validering nog steeds nodig.

De volgende attributen worden besproken: `required, placeholder, pattern, autocomplete, multiple,` het `datalist` element en `list, autofocus, min max, step` .



Met de komst van deze attributen is bijkomende programmatie niet meer nodig, tenzij je een oudere browser gebruikt. Deze attributen worden in oudere browsers niet ingelezen en veroorzaken geen problemen.

5.8.10.1 required

Met het attribuut `required` kan aangegeven worden of een veld **verplicht in te vullen** is. Dit attribuut kan toegevoegd worden aan ieder element van het type <input> behalve op `button`, en een `hidden` element



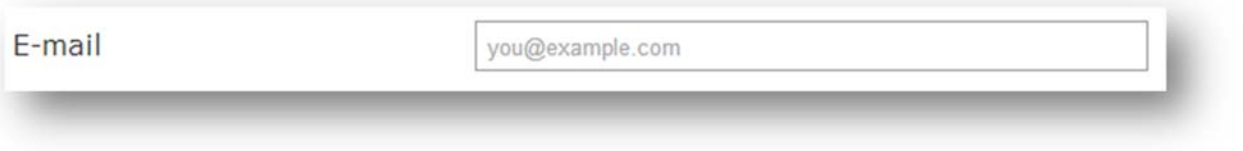
Je mag ook niet vergeten de gebruiker duidelijk te maken dat het veld verplicht is: gebruik bv. een sterretje.

```
<input id=”naam” name=”naam” type=”text" required>
```

5.8.10.2 placeholder

In een invoervak waar de gebruiker zijn bv. zijn naam dient in te voeren kan reeds een plaatsvervangende tekst getoond worden in het grijs. Na invoer verdijnt de `placeholder` tekst. `placeholder` wordt toegepast op velden van het type <input>. Browsers die het attribuut `placeholder` niet kennen negeren het. De inhoud van placeholder kan enkel tekst zijn. Opmaak is niet mogelijk, niet via CSS en niet via HTML.

```
<input id="email" name="email" type="email" placeholder="you@example.com" required>
```



5.8.10.3 pattern

Het attribuut `pattern` maakt het mogelijk om een controle te doen op de ingevoerde waarde, bv. op een e-mail, nummer, url, datum en tijd. De invoercontrole gebeurt via reguliere expressies of regex. Een voorbeeld van zo een expressie die dient om te controleren of een wachtwoord uit ten minste 6 tekens bestaat, zonder spatie: `\s{6,}`



Reguliere expressies worden in deze cursus niet verder uitgelegd. Een goede tutorial over reguliere expressie is:
http://net.tutsplus.com/tutorials/other/8-regular-expressions-you-should-know/

```
<input id=”password” name=”password” pattern="\S{6,}" type=”password”>
```

5.8.10.4 autocomplete

Het attribuut `autocomplete` gaat over het automatisch invullen door de browser. Het kan de waarde “on” (=standaardwaarde) of “off” hebben. Bij invoer van gevoelige informatie is het beter om de waarde op “off” te zetten.

```
<input id=”pincode” name=”pincode” autocomplete=off type=”password”>
```

5.8.10.5 multiple

`multiple` verwijst naar de mogelijkheid om meerdere opties te kunnen selecteren uit een lijst. Bij het element `select` was dat al mogelijk. Het kan nu ook bij het `input` element van het type `file` zodat de gebruiker meerdere bestanden tegelijk kan selecteren die hij wenst te uploaden.

```
<input multiple type=”file”>
```

5.8.10.6 autofocus

Het attribuut `autofocus` zorgt ervoor dat de cursor in één veld komt te staan: het veld krijgt de `focus`. Dus de gebruiker kan meteen de waarde invoeren. Dit effect kan ook op andere manieren bekomen bv. via JavaScript.

```
<input id=”naam” autofocus name=”naam” type=”text”>
```

5.8.10.7 min/max

De attributen `min` en `max` voorzien de mogelijkheid om een maximum en minimum waarde in te stellen voor een veld. De waarde kan een cijfer, datum of tijd zijn. De validatie zal erop inspelen.

```
<input id=”jaartal” max=”2012-12-31” min=”2012-01-01” name=”jaartal” type=”text”>
```

5.8.10.8 datalist element en list

Eén van de meest in Javascript geprogrammeerde controls is de zogenaamde "autocomplet widget", zoals je bijvoorbeeld in Google Search hebt. Met HTML5 is dit ook bereikbaar zonder scripting, enkel in HTML.

Het `datalist` element wordt gecombineerd met een `input` element om een automatisch aanvullende lijst te creeëren.

Een `datalist` lijkt erg op een `select` element omdat het bestaat uit een reeks `option` elementen met mogelijke keuzes. Het heeft een `id` attribuut dat dezelfde waarde heeft als een `list` attribuut in het `input` element.

Browsers die geen HTML5 kennen slaan het element `datalist` over zodat het `input` element normaal werkt.

Een voorbeeld

```
<input name="gemeente" list="gemeentelijst" type="text">
<datalist id="gemeentelijst">
        <option value="Genk">
        <option value="Gent">
        <option value="Geraardsbergen">
        <option value="Tongeren">
        <option value="Terneuzen">
</datalist>
```



Niet alle browsers passen het perfect toe

5.9 HMTL5 video en audio

5.9.1 element video/audio

- Het element `video` maakt het mogelijk om een video in te voegen. `video` heeft een sluittag. De tekst tussen de open- en de sluittag wordt getoond als de video niet geladen kan worden, ingeval de browser het element video niet ondersteunt. De `width` en `height` opgeven is niet verplicht omdat de browser de afmetingen uit het bestand kan halen. Via CSS geef je vorm aan het element video.

```
<video controls poster="../img/big-buck-bunny_poster.jpg" width="640" height="360">
        <source src="../img/big_buck_bunny.mp4" type="video/mp4" />
        <source src="../img/big_buck_bunny.webm" type="video/webm" />
        <source src="../img/big_buck_bunny.ogv" type="video/ogg" />
    <p>Download de video als <a href="../img/big_buck_bunny.mp4">MP4</a>,
        <a href="../img/big_buck_bunny.webm">WebM</a>,
        of <a href="../img/big_buck_bunny.ogv">Ogg</a>.
    </p>
</video>
```

5.9.2 attribuut source

Dit attribuut geeft de url op van de video.



Aan het video element kunnen 2 bronnen opgegeven worden. Indien de ene niet inlaadt, dan de andere hopelijk wel.

5.9.3 attribuut width

breedte

5.9.4 attribuut heigth

hoogte

5.9.5 attribuut autoplay


Dit attribuut laat de video uit zichzelf starten.

5.9.6 attribuut controls


Een regelbalk verschijnt onder het filmbeeld zodat de gebruiker via knoppen, vb. volumeregelaar, start/pauze-knop, ... het video-element kan regelen.

5.9.7 attribuut loop


Als het attribuut loop aanwezig is dan wordt het afspelen herhaald.

5.9.8 attribuut poster


Als het een zware video is kan het even duren voordat alles is ingelezen. Zolang er nog geen beeld van de video beschikbaar is zal een afbeelding getoond worden. Dit doen we met het attribuut poster.

5.9.9 Bestandsformaten video

Op dit moment worden er twee (uitstekende) formaten gebruikt in HTML5:

- Het MP4-formaat, met daarin de H.264 video codec en de AAC audio codec.
- Het WebM formaat, met daarin de VP8 video codec en de Vorbis audio codec.

Qua kwaliteit doen de twee formaten niet veel onder voor elkaar. Het grote verschil zit hem in hun IP (intellectual property):

- MP4 (H264/AAC) wordt breed ondersteund in de video-industrie, maar is volledig dicht gepatenteerd. Zowel voor encoderen, publiceren als decoderen zijn licenties nodig.
- WebM (VP8/Vorbis) is daarentegen patent-vrij en open source (net als alle andere W3C technologieën). Het formaat is vrij nieuw, met een klein ecosysteem van encoders en decoders.

In deze afweging tussen pragmatisme en patenten hebben sommige browsers gekozen voor het eerste (MP4) en anderen voor het laatste (WebM). Het resultaat is dat je nu je video twee keer moet encoderen om alle HTML5 browsers te ondersteunen. Als WebM op termijn meer tractie krijgt, zal dat veranderen, maar dit is een langzaam proces.

Er is trouwens nog een derde HTML5 formaat: Ogg. Het is kwalitatief stukken minder dan MP4/WebM, maar werd ondersteund door Firefox en Opera voordat Google WebM vrijgaf. De enige browser die nog Ogg vereist, is het snel verdwijnende Firefox 3.6.

5.10 HTML5 canvas

In HTML5 is `canvas` een element dat een rechthoekig gebied op de webpagina voorziet. Met JavaScript kunnen tekeningen gemaakt worden.

De breedte en de hoogte zijn niet verplicht mee te geven. Als ze niet worden meegegeven dan krijgt het canvas de standaardafmeting van 300 bij 150 pixels. Ook met JavaScript kunnen de afmetingen opgegeven worden.

Tussen de begin- en eindtag van het canvas kan een zin worden voorzien. Deze wordt getoond als het canvas element niet herkend wordt.

Een `canvas` element zonder tekenopdracht is een transparante rechthoek zonder inhoud.

De opmaak van het canvas doen we met CSS. Om het `canvas` element te kunnen aanspreken is een `id` toegevoegd.

```
<canvas id="myCanvas" width="200" height="200">
      Sorry, je browser ondersteunt het element canvas niet.
</canvas>
```

CSS:

```
#myCanvas {border: dotted 2px black;}
```

Hier tonen we ter illustratie een eenvoudig voorbeeld van een script dat een tekening maakt.

Dit hoort allemaal thuis in de cursus Javascript.

5.10.1 Canvas voorbereiden

Via een JavaScript function kunnen we tekenen op het canvas. Door een JQuery `document.ready` function toe te voegen roepen we de JavaScript function *draw* op.

JQuery document ready function:

```
<script>
   $('document').ready(function(){
    draw();
   });
</script>
```

JavaScript draw function:

```
<script>
⋮
function draw() {
var canvas = document.getElementById("myCanvas");
```

```
}
</script>
```

5.10.2 Context initiatiseren

Voorlopig laten de meeste browsers enkel het tekenen in twee dimensies (2d) toe. Het canvas moet ingesteld worden. Hiervoor roepen we de JavaScript methode getContext() aan.

JavaScript methode:

```
function draw() {
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
}
```

5.10.3 Vulkleur/randkleur instellen

We gaan op het canvas een halfdoorzichtig blauw vierkant tekenen met rode rand. We stellen eerst de randkleur en de vulkleur in.

Kleuren kunnen toegevoegd worden met een hexadecimale waarde `#00FFFF`, rgba `rgba(0, 0, 255, 0.5)` of kleurnamen `red` or `blue`.

JavaScript:

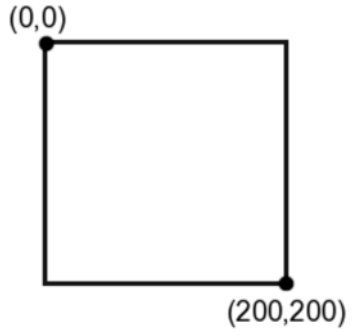
```
function draw() {
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
context.strokeStyle = "red";
context.fillStyle = "rgba(0, 0, 255, 0.5)";
}
```

5.10.4 Vierkant tekenen

Het starten met tekenen doen we door de JavaScript methodes fillRect en strokeRect aan te roepen. Deze methode voorziet X en Y coördinaten om begin- en eindpunt op te geven en de breedte en de hoogte van het vierkant.

JavaScript:

```
function draw() {
⋮
context.fillRect(10,10,100,100);
context.strokeRect(10,10,100,100);
}
```



In het canvas coördinatiesysteem bevindt de hoek linksboven zich op positie (0,0). Stel dat het canvas 200

pixels groot zou zijn dan bevindt de hoek rechtsonder zich op positie (200,200).

5.10.5 Vullen met een patroon

Een canvas kan gevuld worden met een patroon. De methode die hiervoor aangeroepen wordt is de createPattern method. We hebben een afbeelding nodig en instellingen om de afbeelding te herhalen. Eerst maken we een afbeeldingobject en we voegen een broneigenschap toe.

JavaScript:

```
function draw() {
⋮
var img = new Image();
img.src = "../images/bg-bike.png";
}
```

Het patroon wordt gegenereerd na het laden van de afbeelding:

```
function draw() {
⋮
var img = new Image();
img.src = "../images/bg-bike.png";
img.onload = function() {
pattern = context.createPattern(img, "repeat");
context.fillStyle = pattern;
context.fillRect(10,10,100,100);
context.strokeRect(10,10,100,100);
};
}
```

5.10.6 Afbeelding toevoegen

Een afbeelding die zich op de pagina bevindt kan je laten verschijnen in het canvas op een bepaalde positie. De image tag krijgt een id mee. Met JavaScript roepen we de image aan. Met de method drawImage wordt de afbeelding gepositioneerd.

HTML:

```
<canvas id="myCanvas" width="200" height="200">
Your browser does not support canvas.
</canvas>
<img src="../images/html5-logo.png" id="myImageElem">
```

CSS:

```
#myCanvas {
border: dotted 2px black;
margin: 0px 20px;
}
```

JQuery:

```
<script>
   $('document').ready(function(){
    draw();
   });
</script>
```

JavaScript:

```
function draw() {
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
var image = document.getElementById("myImageElem");
context.drawImage(image, 68, 68);
}
```

6 CSS

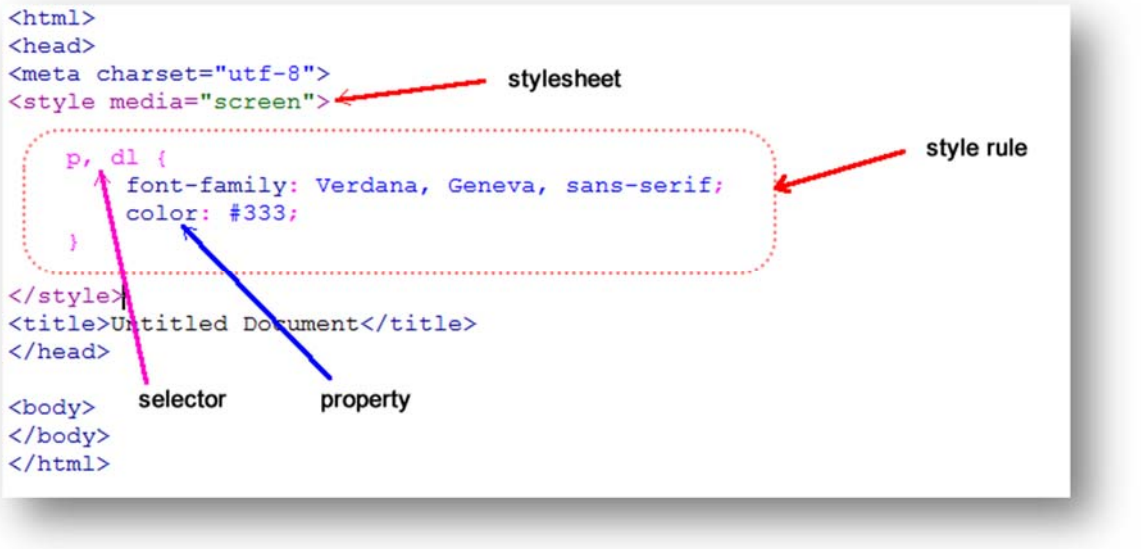
Cascading Stylesheets (CSS) is een taal om opmaak toe te passen op HTML.

HTML bevat de inhoud, CSS zorgt voor de opmaak: hiermee passen we het principe "*scheiding van inhoud en opmaak*" toe.

CSS wordt toegepast met een **stylesheet**, die bestaat uit een aantal **stylerules:** dit zijn één of meer **selectors** elk gevolgd door een **declaration block** met CSS properties.

Een stylesheet kan **intern** zijn, d.w.z. vervat in een `style` element in de HTML pagina zelf, of **extern** in een eigen .CSS bestand en gekoppeld via een `link` element.

Een stylesheet kan gericht zijn op één bepaald **medium**, vb. *screen*, of op meerdere. Dit gebeurt via een `media` attribuut.



Hierboven zie je een klein intern stylesheet voor het medium *screen*, met één style rule bestaande uit een selector `p,dl` en twee CSS properties `font-family` en `color`.

6.1 Syntax

De basis syntax is eenvoudig: een **style rule** bestaat uit een **selector** gevolgd door een **declaration block**.

Dit block zit in een paar **accolades** waarin 1 of meerdere **stylerules** staan, die gescheiden worden door **punt-komma's**. Elke stylerule wordt gevolgd door een **dubbele punt** en een **waarde**.

Een style rule:

```
selector {
                property1: waarde1;
                property2: waarde2;
        }

/* commentaar hier */
```

- alle witruimte wordt genegeerd.

- commentaar kan enkel in /* */

Er zijn nog andere rules mogelijk, maar dat zien we verder.

6.2 De CSS eigenschappen

Het is via een CSS property (eigenschap) dat de opmaak van de geselecteerde elementen wordt beïnvloed.
We gebruiken de **naam** van de eigenschap gevolgd door een **dubbelpunt** en een **waarde**.

```
eigenschap : waarde;
```

Bijvoorbeeld:

```
color : #33FFCC;
position: relative;
clear: left;
```



In deze cursus leer je het merendeel van de CSS eigenschappen kennen

Maar misschien heb je toch vragen:

Welke eigenschappen zijn er?

Er zijn tientallen eigenschappen. Je kan ze indelen volgens hun doel: tekstopmaak, positie, tabellen, box etc.. Sommige hebben heel specifieke bedoelingen (Ruby, Transition,…)

Welke waarden mag ik gebruiken?

Meestal heb je meerdere mogelijkheden (bv. `color`). Soms kan je enkel kiezen uit een lijstje van mogelijkheden, bv: `left | right`.
Foutieve waarden (`color:appelblauwzeegroen`) worden genegeerd of de volledige eigenschap wordt genegeerd.

Waar vind ik een overzicht?

CSS2 properties op http://www.w3.org/TR/CSS2/propidx.html

CSS2 + CSS3 https://developer.mozilla.org/en-US/docs/CSS/CSS_Reference

6.3 CSS en HTML

Op HTML kunnen 3 soorten stylesheets toegepast worden:

- **browser stylesheet**:

  een browser heeft **altijd** een *ingebouwd stylesheet* dat een standaardopmaak toepast op de HTML. Dit is bijvoorbeeld de reden dat een `h1` kop groter toont dan andere tekst. Je hoeft er niets voor te doen.

- **author stylesheet(s)**:

  dit zijn de stylesheets aangemaakt door de webontwikkelaar (jij dus!). Ze zijn intern of gekoppeld. Het zijn meestal deze styles die een webpagina maken wat ze is: met mooie kleuren, lay-out etc…

- **user stylesheet**:

  je wist het misschien niet, maar een user die een website bekijkt kan zijn eigen stylesheet erop toepassen. Waarom zou je dat doen? Toegegeven, het gebeurt zelden, maar in sommige situaties kan het nodig zijn, bv. mensen met gezichtsproblemen (kleurenblindheid, …) kunnen een hoog contrast stylesheet laden

Onthou dat in de meeste gevallen er **steeds 2 stylesheets inwerken op de HTML**: het ***browser stylesheet*** en het ***author stylesheet***.

Omdat het browser stylesheet nogal durft verschillen tussen de verschillende browsers en er allerlei ongewenste opmaak in staat, gebruiken sommige ontwikkelaars wel eens een *Reset* (of *Normalize*) stylesheet, die *tabula rasa* maakt van die standaardopmaak. Zo begin je met een "schone lei".

We kunnen onze eigen opmaak toepassen op 3 manieren:

1. met een **extern** stylesheet
2. met een **intern** stylesheet
3. via een `style` attribuut in een HTML element (**inline** style)

Alle combinaties van deze zijn ook mogelijk.

6.3.1 Een extern stylesheet koppelen

Een extern stylesheet wordt **gekoppeld** met een `link` element in de `head` van het document. Deze methode heeft de **voorkeur** want zo kan je met één stylesheet de opmaak van alle HTML pagina's in één site bepalen.

XHTML:

```
<link href="css/mijnstijl.css" rel="stylesheet" type="text/css" media="screen" />
```

- het `href` attribuut (verplicht) bevat het pad naar het CSS bestand
- het `rel` attribuut (verplicht) bevat de waarde "stylesheet". `link` elementen kunnen ook andere bestanden koppelen.
- het `type` attribuut (verplicht) bevat het MIME type "text/css"
- het `media` attribuut (optioneel) geeft aan voor welk medium het stylesheet bedoeld is

HTML5:

```
<link href="css/mijnstijl.css" rel="stylesheet" media="screen" >
```

- de `href` en `rel` attributen zijn ook hier nodig en verplicht
- het `type` attribuut mag je achterwege laten
- het `media` attribuut is ook hier optioneel en heeft dezelfde functie

Je kan ook meerdere stylesheets na elkaar koppelen:

```
…
<link href="css/huisstijl.css" rel="stylesheet" >
<link href="css/intro.css" rel="stylesheet" >
<link href="css/forms.css" rel="stylesheet" >
<script src="js/nuttig_lib.js"></script>
…
```

Plaats alle stylesheets (ook interne) vóór eventuele javascripts of js-koppelingen in het document.

6.3.2 Intern stylesheet

Een intern (ook *embedded*) stylesheet is een `style` element in de `head` van het HTML bestand met één of meerdere stylerules.

```
<style>
body {
        margin-left: 5px;
        color: #000000;
        font-family: Arial, Helvetica, sans-serif;
        background:#ffffff url(../images/archiExt_bg.gif) repeat-x scroll top left;
  }
</style>
```

Ook hier is een `type` en een `media` attribuut optioneel in HTML5, verplicht in XHTML.

Probeer zo weinig mogelijk interne stylesheets te gebruiken, wij doen het hier enkel in de kleine oefeningen om slechts één bestand te gebruiken.

6.3.3 Inline styles

Een **inline** style is een styledeclaratie die in een `style` **attribuut in het element zelf** gezet wordt zoals:

```
<p style="color:red">rode tekst</p>
```



Dit is dus geen stylesheet. **Vermijd deze manier van werken!**

Inline styles worden vooral toegepast via Javascript: doordat ze een grotere specificiteit (zie verder) hebben dan alle andere methodes (ze hebben altijd voorrang), worden ze door scripts gebruikt om onmiddellijk effect te hebben.

Voor een script is dat natuurlijk geen probleem want het heeft *geen blijvend effect* op de HTML.

6.3.4 @import

Met een `@import` rule kan je een stylesheet in een ander importeren:

```
<style>
@import url('forms.css');
body {
          margin-left: 5px;
          color: #000000;
          font-family: Arial, Helvetica, sans-serif;
          background:#ffffff url(../images/archiExt_bg.gif) repeat-x scroll top left;
  }
…
</style>
```

Alle CSS van *forms.css* wordt nu geïmporteerd in dit stylesheet.

Een `@import` regel **moet altijd vooraan** staan.

Deze methode heeft het voordeel dat je je CSS **modulair** kan scheiden, bv. de styles voor een formulier zijn zeer specifiek en kan je afzonderen. Je importeert ze dan waar je ze nodig hebt.

Het nadeel is dat dit net hetzelfde is als twee aparte stylesheets koppelen: er gebeuren dan 2 HTTP requests en dat zorgt voor vertraging.

6.3.5 CSS Overload?



Browsers moeten dus rekening houden met *browser*, *author* en *user stylesheets*!
en dan nog eens met *externe*, *interne stylesheets* en *inline styles* !!!

Hoe raakt een browser uiteindelijk wijs uit al die (tegenstrijdige) CSS rules? welke toepassen en welke niet?

De browser zal moeten beslissen als er zich **conflicten** voordoen:

- ? tussen verschillende stylesheets

```
h2 { font-size: 2.4em;}          /* browser stylesheet */

h2 { font-size: 2em;}            /* author stylesheet 1 */

h2 { font-size: 2.2em;}          /* author stylesheet 2 */
```

- ? tussen verschillende CSS rules binnen hetzelfde stylesheet

```
h2         { font-size: 2em;}

#inhoud h2 { font-size: 2.2em;}
```

- ? tussen stylesheets en inline styles

```
p { color:blue;}
…
<p style="color:red">tekst</p>
```



Welke CSS Rule uiteindelijk de voorrang krijgt, hangt af van
- de **Specificiteit** van de selector,
- de **Cascade**,
- de **volgorde**

We gaan er in detail op in in het hoofdstuk "*Inheritance en de Cascade*".

6.4 CSS grammatica

- CSS zelf is **niet hoofdlettergevoelig**, maar we geven de **voorkeur** aan kleine letters
- **waarden** van eigenschappen zijn wel hoofdlettegevoelig, zoals de naam van een `id` of een `class`, een lettertype of een url
- **sleutelwoorden** worden herkend door de parser; ze moeten niet in aanhalingstekens staan: `auto, none, transparent`

- **commentaar** kan enkel tussen `/* commentaar */`
- Een *backslash* karakter kan gebruikt worden om een waarde te escapen: `"\""`
- **witruimte** wordt genegeerd
- zet nooit een **spatie** tussen getal en eenheid: `14px`
- **decimalen** aanduiden met een punt-notatie: `1.2em`
- de waarde `0` moet je nooit een eenheid meegeven, dus bv. `0px` wordt gewoon als `0` geschreven.
- een eigenschap kan altijd de waarde `inherit` gegeven worden, waarmee de waarde van de *parent* overgeërfd wordt

6.5 Selectoren

Om opmaak op één of meerdere elementen toe te passen hebben we een **selector** nodig.

```
selector {
      eigenschap : waarde;
      eigenschap : waarde;
}
```

Je hebt eenvoudige selectors die één specifiek element selecteren (via zijn `id` attribuut), 1 soort element (via de HTML tag), of meer complexere die slechts een soort element selecteren enkel als dat voorkomt in een ander element, tot zeer complexe selectoren die bijvoorbeeld het derde *child* viseren van een bepaal type enkel als de muis eroverheen hovert…

We starten met de meest eenvoudige om onze voorbeelden te kunnen illustreren, later bekijken we ook de complexe selectors.

6.5.1 Element-selectoren

Voor een HTML-element-selector geldt de algemene vorm:

```
element { stijldeclaratie }
```

Elk element (elke tag dus) kan hierbij gebruikt worden als selector.

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

Indien meerdere selectoren dezelfde declaratie hebben kunnen zij gegroepeerd worden; hun namen worden dan gescheiden door **komma's**:

```
element1, element2, element3 { stijldeclaratie }
```

Je mag dit ook schrijven als

```
element1,
element2,
element3 {
      stijldeclaratie
 }
```

zolang je er maar op let dat de komma's aanwezig zijn op het einde van de regel.

Dit is geen gecombineerde selector, dit zijn gewoon meerdere selectoren met dezelfde opmaak.

Een vaak gebruikt voorbeeld: stel dat je alle koppen binnen je document rood wilt weergeven, dan kan je natuurlijk achtereenvolgens stijlen voor `h1` t.e.m. `h6` definiëren:

```
h1 { color: red; }
h2 { color: red; }
h3 { color: red; }
h4 { color: red; }
h5 { color: red; }
h6 { color: red; }
```

Maar korter zou je dit kunnen noteren als:

```
h1, h2, h3, h4, h5, h6 { color: red; }
```

Bij inline-styles wordt geen expliciete selector opgegeven, aangezien de style gedefiniëerd is binnen de tag waarop hij van toepassing is.

6.5.2 Uittesten

Sommigen onder jullie willen deze korte lijntjes code wel eens uitproberen op HTML tussen de projecten in. Je kan daar het oefenbestand *testbestand.html* voor gebruiken.

Kopieer en plak de CSS in de `style` tag in de `head` van het document:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<base target="_blank">
<style>
/* css hier */
</style>
<title>Startbestand demo oostende, Save as</title>
</head>
```

Bekijk het resultaat daarna in een browser.



Maar vergeet niet: je werkt best vanuit het **projectenboek**! de code in deze theorie bestaat slechts uit losse voorbeeldjes !

7 BASIS CSS

We beginnen met de meest gebruikte opmaak voor tekst, box en lijst.

7.1 Letter-opmaak

7.1.1 font-family

Met de eigenschap `font-family` wordt ingesteld welk **lettertype** gebruikt moet worden voor een bepaald element.

Dit kan de naam zijn van een bepaald lettertype:

```
font-family: Arial;
```

Dit veronderstelt dat het opgegeven lettertype op de computer van de gebruiker aanwezig is. Nu is bijvoorbeeld het lettertype *Helvetica* op alle Mac-systemen aanwezig, maar niet op Windows-computers. Op die toestellen bestaat echter wel een quasi identiek lettertype met de naam *Arial*. Je kan al raden wat er gebeurt als de browser het gewenste lettertype niet vindt: dan wordt inderdaad het standaardlettertype gebruikt.

Je mag daarom ook meerdere lettertypes opgeven, gescheiden door komma's. Wanneer het eerste niet wordt gevonden, wordt het tweede gebruikt en zo verder.

Bijvoorbeeld:

```
font-family : Arial, Helvetica;
```

Om problemen met exacte namen te vermijden, zijn ook een aantal **generieke namen** voorzien: namen die altijd mogen gebruikt worden en die door elke browser worden geconverteerd naar op dat systeem aanwezige lettertypes:

`serif` : een lettertype met *schreef*, zoals *Times New Roman*

`sans-serif`: een lettertype zonder *schreef*, zoals Arial

`monospace` : een niet-proportioneel lettertype (met vaste afstand, zoals bij een schrijfmachine) zoals `Courier`

`cursive` : een handschrift-lettertype, zoals *Comic*

De beste werkwijze is eerst enkele benoemde lettertypes vermelden, gevolgd door het generiek lettertype:

```
body{
font-family: Arial, Helvetica, sans-serif;
}
```

Als je deze css uittest kan het gebeuren dat je geen enkel verschilt ziet met eerder: bedenk dat het *browser stylesheet* het standaard lettertype ook al op Arial, Helvetica gezet heeft.…

Een andere manier om lettertypes toe te passen is gebruik maken van **webfonts**: dat wordt verder uitgelegd.

7.1.2 font-weight

Met de eigenschap `font-weight` kan het **gewicht van een lettertype** (de dikte) bepaald worden.

De meest gebruikte waarden zijn `normal` en `bold`.

voorbeeld:

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

en bekijk daarna het resultaat in je browser, je ziet dat de paragrafen nu in het vet komen te staan.

7.1.3 font-style

Met de eigenschap `font-style` kan een **stijl** voor de letter worden opgegeven.

Mogelijke waarden zijn: `normal`, `italic` en `oblique`

De waarde `oblique` wordt vaak hetzelfde weergegeven als `italic`, dus cursief, maar eigenlijk staat `oblique` voor een schuin geplaatste normale letter.

Concreet toegepast: voeg onderstaande code toe aan je CSS:

```
h1 {
font-style : italic;
}
```

en bekijk daarna het resultaat in je browser. De titel “*Responsive webdesign*” komt nu cursief te staan.

> *Even tussendoor willen we je er al op attent maken dat het belangrijk is om de volgorde van je CSS regels in het oog te houden!*
>
> *Iedereen ontwikkelt zo’n beetje z’n eigen manier van werken, maar het is beter om er toch een zekere structuur in te steken. Zo kan je alle CSS voor de* `header` *bij elkaar zetten, alle CSS voor de content bij elkaar, alle CSS voor de* `footer` *bij elkaar, enzoverder...*
>
> *Je kan er anderzijds ook voor kiezen om de hiërarchie van je HTML elementen te volgen: je zet je CSS rules in dezelfde volgorde als de corresponderende elementen in je HTML document.*

Als je de HTML hiërarchie volgt ziet je complete CSS er nu zo uit:

```
<style type=”text/CSS”>
body {
font-family: Arial, Verdana, sans-serif;
}
h1 {
```

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

7.1.4 font-size

Met de eigenschap `font-size` kan vanzelfsprekend de **lettergrootte** bepaald worden.

Hiervoor kan je verschillende **eenheden** gebruiken:

- `px`
- `em`
- `rem`
- `%`
- `pt`

We bespreken de belangrijkste hieronder.



De **standaard lettergrootte** van een browser – als je dus niets instelt – is **16px**.

Browsers hebben ook een **zoom-functie** die je via het menu, met het muiswiel of met een shortcut key kan bereiken (Ctrl +, Ctrl -). Daarmee vergroot of verklein je de inhoud van het scherm, in eerste instantie de lettergrootte. Dit is uiteraard een ***Accesibility feature***, voor mensen met een visuele beperking of als je je bril vergeten bent…

7.1.5 Eenheden

pixels: `px`

Door het lettertype in te stellen met pixels zeg je **exact** hoeveel pixels hoog de tekst op je scherm moet zijn. Een pixel hangt vast aan de **schermresolutie**, de letters zullen groter lijken op een scherm met lagere resolutie.

Een `px` is dus een **absolute eenheid**. Dit is enkel nuttig als je exact weet hoe groot je scherm is.

De zoom functie van IE6 was niet in staat je tekst te vergroten als die uitgedrukt was in pixels. Nieuwere versies van IE hebben dit ernstig probleem niet meer.

Als je pixels gebruikt in je CSS stel je dus voor alle selectors vast wat de exacte grootte is van het lettertype, vb:

```
p       {font-size: 12px}
h1      {font-size: 26px}
h2      {font-size: 22px}
h3      {font-size: 20px}
```

dit heeft het voordeel dat je exact weet hoe groot een letter zal zijn. Er zijn echter meer nadelen dan voordelen:

- je moet voor **alles** de lettergrootte instellen anders wordt de standaardgrootte gebruikt…
- als je van gedacht verandert en alle iets groter/kleiner wil kan je herbeginnen…
- de groottes van de boxen zal je waarschijnlijk ook in pixels moeten gaan uitdrukken…dit is onmogelijk voor een *fluid lay-out*
- de lettergroottes zijn niet makkelijk te zetten in verhouding tot elkaar

em

Een `em` stond oorspronkelijk voor de breedte van de hoofdletter M, maar is infeite een volledig schaalbare eenheid.

We stellen

```
body { font-size: 100%; }      /* 1em = 16px */
```

Als je ervan uitgaat dat de **standaard lettergrootte** `16px` is (van de `body`), dan is `1em = 16px`. Vanaf dit punt worden alle andere groottes afgeleid in `em`, de `em` is dus een **relatieve eenheid**. De `font-size: 100%` is enkel nodig voor oudere IE versies, maar voor cross-browser support laten we het staan.
Een `em` kan een decimaal getal zijn tot maximum drie decimalen: `0.125em`

Een voorbeeld van een `div` die erft van de standaard lettergrootte `16px=1em`:

```
div { font-size:1.125em;       /* 1.125em * 16 = 18px */ }
```

Erg belangrijk is het verschil tussen de geërfde waarde voor `font-size` en alle andere CSS properties:

- `font-size` in `em` is relatief t.o.v. de `font-size` van zijn **parent**
- voor elke andere property (bv. `width`) die `em` gebruikt is dit relatief t.o.v. zijn **eigen** `font-size`

We leggen het best uit aan de hand van een voorbeeld. Veronderstel de volgende html structuur:

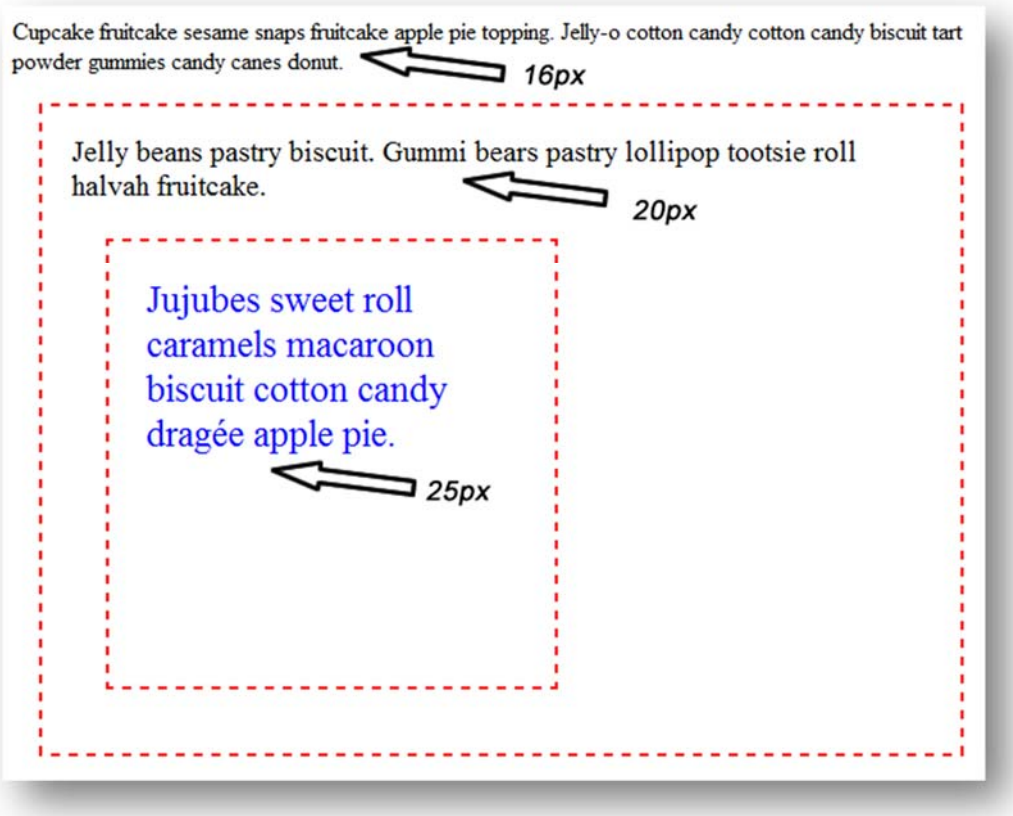
```
…
<body>
<p>Cupcake fruitcake sesame snaps fruitcake apple pie topping. Jelly-o cotton candy
  cotton candy biscuit tart powder gummies candy canes donut. </p>
<div id="outer>Jelly beans pastry biscuit. Gummi bears pastry lollipop tootsie roll
  halvah fruitcake.
  <div id="inner">Jujubes sweet roll caramels macaroon biscuit cotton candy dragée apple
  pie. </div>
</div>
</body>
…
```

een `body` element bevat 2 children: een `p` en een `div` element. De `div#outer` bevat zelf een andere `div#inner`.

We passen de volgende CSS hierop toe:

```
body { font-size: 100%; }      /* 1em = 16px */
div {
       font-size:1.25em;
       outline:2px dashed red;
       padding:1em;
       margin:1em;
  }
div#inner{
       color:blue;
       width:10em;
       height:10em;
  }
```

Het resultaat is zo:



Bespreking:

- de `outline` property, de `padding`, de `margin` en de `color` dienen enkel om dit vb te verduidelijken
- het `p` element heeft geen *style* gekregen, heeft dus de waarde `inherit` wat resulteert in een lettergrootte van `16px`
- de eerste `div` heeft `font-size: 1.25em`, dus

```
16px (parent=body) x 1.25 = 20px
```

- de tweede `div#inner` heeft ook `font-size: 1.25em`, dus

  `20px (parent=#outer) x 1.25 = 25px`

- de `div#inner` heeft ook `width` en `height: 10em`, dus

  `25px (eigen) x 10 = 250px`

Alle groottes in em zijn dus relatief ten opzichte van elkaar en het beginpunt is de body: verander je de waarde in de body (bv: 90%) dan wijzigt alles (relatief) mee.

Eén nadeel is dat als je eens een exacte pixelwaarde wil, je die moet omrekenen, bijvoorbeeld als je exact `100px` wil in bovenstaand voorbeeld moet je de berekening maken: `100 / 16 = 6.25em`.

Om die reden gaat een aantal ontwikkelaars anders tewerk om makkelijker te kunnen rekenen:

```
body { font-size: 62.5%; }
```

Dit stelt `1em = 10px`.

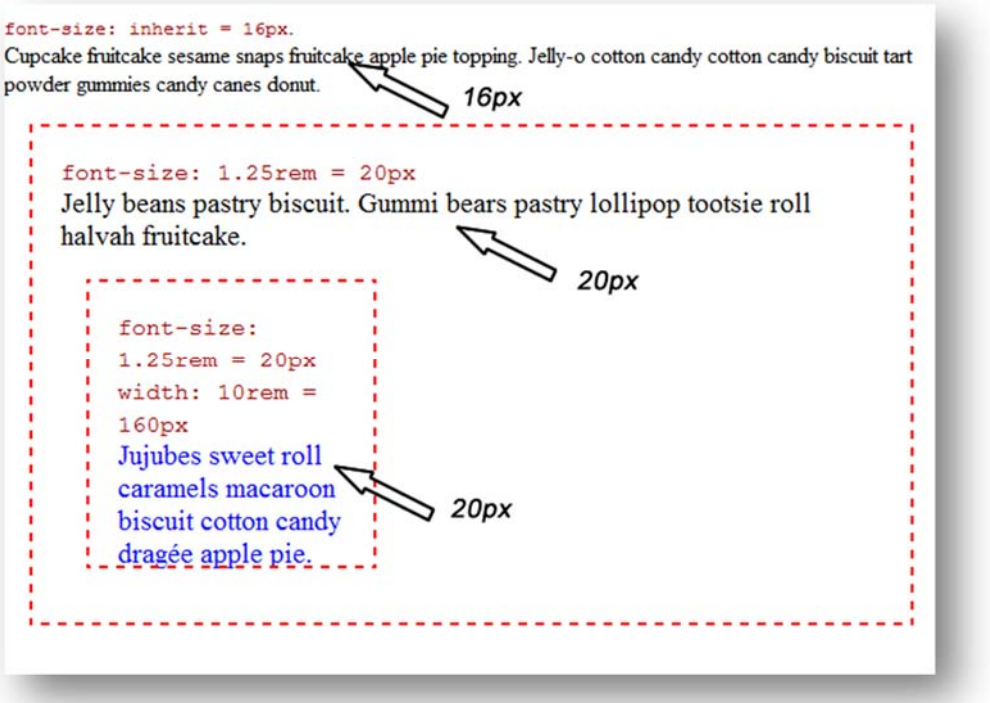
Nu is het kinderspel om een titel een waarde van `39px` te geven: em's zijn decimaal, dus de titel wordt `3.9em`, 3.9 x 10 pixels.

Je mag zelf kiezen welke methode je makkelijkst vindt.

rem

De `rem` (*root em*) is een nieuwe CSS3 eenheid identiek aan de `em`, met één groot verschil: de relatieve wijziging van de lettergrootte is steeds t.o.v. van het root element, M.a.w. de `body`.

Vervangen we in bovenstaand voorbeeld alle em's door rem's dan krijgen we:



- het `p` element heeft de waarde `inherit` dus `16px`
- de eerste `div` heeft `font-size: 1.25rem`, dus

```
16px (body) x 1.25 = 20px
```

- de tweede `div#inner` heeft ook `font-size: 1.25rem`, dus ook

```
16px (body) x 1.25 = 20px
```

- de `div#inner` heeft ook `width` en `height: 10rem`, dus

```
16px (body) x 10 = 160px
```

Dus alles relatief ten opzichte van de eerste instelling van de waarde in de `body`.

`rem` wordt ondersteund vanaf IE9, FF3, Chrome6, Safari5, O11.6, dus in alle moderne desktop browsers.

7.1.6 font-variant

Met de eigenschap `font-variant` kan worden ingesteld dat de tekst in **kleine hoofdletters** moet worden weergegeven. Dit zou normaal het volgende effect moeten hebben:

WELKOM BIJ VDAB!

Mogelijke waarden voor deze eigenschap zijn dus: `normal` en `small-caps`.

Maar niet alle lettertypes zijn in een “small-caps”-variant beschikbaar; in die gevallen worden doorgaans gewone hoofdletters weergegeven, maar iets kleiner dan de rest van de tekst.

7.1.7 line-height

En beetje een apart geval is de eigenschap `line-height` . Hiermee wordt de **afstand tussen de basislijnen** van twee regels tekst weergegeven.

Mogelijke waarden zijn:

`normal`	120% van het lettertype (opgelet: browserafhankelijk)
`getal`	3 betekent dat de regelhoogte gelijk is aan 3 keer de lettergrootte.
`lengte`	een getal en een eenheid bv. 16px of 1.5em
`percentage`	een getal en een percentteken bv. 120% betekent dat de regelafstand 1,2 keer de lettergrootte is.

`line-height` controleert de minimale hoogte tussen de lijnen van een tekstblok

Denk eraan dat `line-height` moet bekeken worden in verhouding tot het lettertype: een vaste `line-height` instellen en daarna een ander lettertype kiezen, is geen goed idee.

De `line-height` kan zowel in lengte-eenheid als relatief (em, %, getal) ingegeven worden, toch wordt aangeraden dat steeds als getal te doen om problemen met inheritance te vermijden. vb. 1.4.

7.1.8 Font

De eigenschap `font` **combineert meerdere eigenschappen** voor het lettertype, we noemen dit een verkorte schrijfwijze. Zo kan de stijldefinitie:

```
font-weight: bold;
font-style: italic;
font-size: 14px;
font-family: Arial, Helvetica, sans-serif;
```

ook genoteerd worden als:

```
font: bold italic 14px Arial, Helvetica, sans-serif;
```

De algemene syntax is:

```
font:       [<font-style>||<font-variant>||<font-weight>]? ↵
            <font-size>[/<line-height>]? ↵
            <font-family> ]
```

Concreet toegepast: wijzig de code voor de body in je CSS als volgt:

```
body {
```

```
  text-align: justify;
  font: 10px/1.5em Verdana, sans-serif;
}
```

Bekijk daarna het resultaat in je browser.

Let ook op de notering `10px/1.5em` waarbij terzelfdertijd `font-size` en `line-height` ingesteld worden, een courant gebruikte methode.

Als je een verkorte schrijfwijze gebruikt, let dan goed op de **volgorde** van de sub-eigenschappen.

De eerste drie sub-eigenschappen zijn optioneel en hun volgorde t.o.v. elkaar is niet belangrijk. Een eventuele `font-size` moet er echter op volgen, net zoals de `font-family`.

De volgende voorbeelden toegepast op een fictieve selector `e` zijn geldig:

```
e { font: Arial, Helvetica, sans-serif }
e { font: large Arial, Helvetica, sans-serif }
e { font: bold italic large Arial, Helvetica, sans-serif }
e { font: italic bold large Arial, Helvetica, sans-serif }
e { font: 12px/14px sans-serif }
```

Deze echter zijn niet correct en zullen dan ook niet volledig uitgevoerd worden:

```
e { font: large bold italic Arial, Helvetica, sans-serif }
e { font: Georgia,”Times New Roman”, Times, serif small-caps}
```

7.1.9 word-spacing

Met de eigenschap `word-spacing` kan de **afstand tussen woorden** worden aangepast.

Mogelijke waarden zijn:

`normal`	
een lengte :	een getal en een eenheid bvb. 2em betekent 2x de normale afstand.

7.1.10 Letter-spacing

Met de eigenschap `letter-spacing` kan de **afstand tussen de afzonderlijke tekens** worden aangepast.

Mogelijke waarden zijn:

`normal`	
een lengte **:**	een getal en een eenheid, bvb. 2em betekent 2x de normale afstand

Negatieve waarden zorgen ervoor dat de letters dichter op elkaar geplakt worden. Enkele voorbeelden:

```
normale tekenafstand
vergrootte tekenafstand
verkleinde tekenafstand
```

Concreet toegepast: wijzig de CSS voor de `h1` in onderstaande code in je CSS:

```
h1 {
font-style : italic;
font-variant : normal;
letter-spacing : 1em;
}
```

en bekijk daarna het resultaat in je browser.

De property `letter-spacing` bepaalt dus de ruimte tussen de letters van een woord, terwijl `word-spacing` de ruimte tussen woorden instelt. Beide eigenschappen hebben een standaardwaarde `normal` die gepast is voor het gekozen lettertype en grootte.

7.1.11 Text-decoration

Met de eigenschap `text-decoration` kan je **speciale opmaak** aan je tekst geven.

Mogelijke waarden zijn

`none`	beginwaarde; geeft aan dat geen speciale effecten moeten worden toegepast
`underline`	onderlijnde tekst
`overline`	lijn boven de tekst
`line-through`	doorstreepte tekst
`blink`	flikkerende tekst



Gebruik **onderlijning enkel voor hyperlink,** nooit op andere tekst!

Omdat `a` elementen standaard onderlijnd zijn, verwacht een gebruiker dat een onderlijnde tekst een hyperlink is. Ga daar nooit tegenin en gebruik andere opmaak om een tekst te benadrukken.

De kleur van de lijn wordt afgeleid van de `color` eigenschap van de tekst.

Bijvoorbeeld:

```
text-decoration : line-through;
```

Niet alle browsers ondersteunen al deze waarden. `text-decoration` wordt het meest frequent gebruikt om de standaard onderlijning van hyperlinks weg te halen

```
(a { text-decoration: none: }
```

Nog een voorbeeld:

```
h1 {
font-style: italic;
font-variant: normal;
letter-spacing: 1em;
text-decoration: overline;
}
```

en bekijk daarna het resultaat in je browser.

7.1.12 Text-transform

Met de eigenschap `text-transform` kan het **hoofdlettergebruik** van een stuk tekst worden aangegeven.

Mogelijke waarden zijn:

`none`	beginwaarde tekst wordt weergegeven zoals ingetypt
`capitalize`	beginkapitalen eerste letter van elk woord wordt een hoofdletter
`uppercase`	alles in hoofdletters
`lowercase`	alles in kleine letters

De `text-transform` property laat omzettingen tussen hoofd- en kleine letters toe. De waarde capitalize geeft een enkele hoofdletter aan een woord, uppercase zet alle letters om naar hoofdletters, lowercase naar kleine letters terwijl none niets doet.

Voorbeeld:

```
h2 {
font-style: italic;
letter-spacing: 1em;
text-decoration: underline;
text-transform: capitalize;
}
```

en bekijk daarna het resultaat in je browser.

7.2 Alinea-opmaak

7.2.1 Margin

Volgens het box-model (zie verder) heeft elke box vier marges: top, right, bottom en left.

Je kan elke marge apart instellen met `margin-top`, `margin-right`, `margin-bottom`, `margin-left`, of alle samen met `margin.`

Je kan hiervoor eender welke lengteeenheid gebruiken en ook de waarde `auto`..

Bijvoorbeeld

```
p {
  margin-top: 0.6em;
  margin-right: 1em;
  margin-bottom: 0.4em;
  margin-left: 1em;
}
```

7.2.2 margin

De eigenschap `margin` is de verkorte schrijfwijze voor marges.
De juiste volgorde is TRBL (top, right, bottom,left).

Bovenstaand voorbeeld kan ook geschreven worden als

```
p {margin: 0.6em 1em 0.4em 1em;
```

Maar omdat marges dikwijls gelijk zijn, zijn er ook andere schrijfwijzen beschikbaar: met 1, 2, 3 of 4 getallen:

1 getal: *top=bottom=left=right*

2 getallen: *top=bottom left=right*

3 getallen: *top left=right bottom*

4 getallen: *top right bottom left*

Bijvoorbeeld:

```
margin: 0.6em 1em 0.4em;
```

Betekenis: boven-marge is 0.6em, de ondermarge is 0.4em, linker- en rechtermarge op 1em.

Als een block percentages als margewaarden heeft dan worden die berekend op de `width` van zijn **parentcontainer**. Bijvoorbeeld

```
div { margin: 5%;  }
```

krijgt 5% marge van de breedte van de container van de `div`. Dat betekent natuurlijk dat als die container verandert (bv. in een responsive design), deze marges mee veranderen.

Als een marge de waarde `auto` heeft , dan krijgt die een automatische waarde. Dit wordt dikwijls gebruikt om een blok horizontaal te centereren:

```
div { width:50%;  margin:0 auto; }
```

7.2.3 Text-align

Met de eigenschap `text-align` wordt de **uitlijning** van de inhoud ingesteld.

Mogelijke waarden zijn: `left, center, right` en `justify.` Links, rechts, gecentreerd of uitgevuld.

De eigenschap `text-align` bepaalt de horizontale uitlijning van de inline inhoud (niet enkel tekst) van een container.

Er bestaat geen betrouwbare manier om tekst vertikaal uit te lijnen in CSS2!

7.2.4 Text-indent

Met de eigenschap `text-indent` wordt **de insprong van de eerste regel** opgegeven.

Mogelijke waarden zijn: een **lengte** en een **percentage**

Negatieve waarden zijn ook toegelaten (voor “verkeerd-om” inspringen).

Voorbeeld

```
p {
text-align: justify;
text-indent: 2em;
}
```

Dit is een voorbeeld van een alinea waarvan de eerste regel een bepaalde afstand ingesprongen is. Alle andere regels reageren gelijk. Hier is een positive waarde gegeven, maar met een negatieve waarde krijg je een hangende regel.

7.2.5 White-space

HTML negeert witruimte in de code en drukt die samen tot één spatie. Het element `pre` is de enige uitzondering op die regel. Alle andere elementen hebben de waarde `normal`.

De CSS eigenschap `white-space` laat je toe witruimte te controleren op andere elementen.

Waarde	witruimte samengedrukt	nieuwe regel bij return (of “\A” in broncode *)	automatische terugloop
`normal`	ja	neen	ja
`pre`	neen	ja	neen
`nowrap`	ja	neen	neen
`pre-wrap`	neen	ja	ja
`pre-line`	ja	ja	ja

* De “`\A`” is enkel van toepassing in *generated content* (zie verder).

7.2.6 Teksteigenschappen samengevat

7.2.6.1 Lettertype

property	waarden
font-family	een lettertype of een groep lettertypes of als generisch lettertype
font-size	grootte als absolute of relatieve waarde, % of lengte
font-style	normal | italic | oblique
font-variant	normal | small-caps
font-weight	normal | bold
font	verkorte schrijfwijze voor de subeigenschappen

7.2.6.2 Andere teksteigenschappen

property	waarden
letter-spacing	normal | lengte
word-spacing	normal | lengte
line-height	normal | lengte | getal | %
text-align	left | right | center | justify
text-indent	lengte | %
text-decoration	none | underline | overline | line-through | blink
text-transform	capitalize | uppercase | lowercase | none
white-space	normal | pre | nowrap | pre-wrap | pre-line

7.3 Kleur en achtergrond met CSS

Kleur kan op verschillende manieren aangegeven worden in eigenschappen die kleur gebruiken.

7.3.1 Kleurnamen

In de HTML-standaard zijn kleurnamen voorzien, die je kan gebruiken i.p.v. RGB-waarden. Ze bevatten enkele zeer herkenbare woorden zoals `white, yellow, red, blue`, etc… maar er zijn er veel meer: `cornflowerblue, bisque, darkviolet`, etc… de volledige lijst vind je op de CSS3 Color module: http://www.w3.org/TR/css3-color/

Alle moderne browsers ondersteunen deze kleurnamen.

7.3.2 RGB-kleuren

Een andere mogelijkheid is het **RGB**-kleuren-model gebruiken. RGB staat voor **red-green- blue**, de basiskleuren waaruit schermkleuren worden samengesteld.

Een rgb waarde kan opgegeven worden als

- een hexadecimal waarde
- met een `rgb()` functie

Een **hexadecimale waarde** is een combinatie van drie **hexadecimale** getallen (gaande van 00 tot FF), één voor elke kleur rood, groen blauw. Deze combinatie wordt voorafgegaan door een `#` dat aangeeft dat het om een hexadecimaal getal gaat:

`#rrggbb`

Enkele veelgebruikte kleuren zijn:

`#000000`	zwart
`#ffffff`	wit
`#ff0000`	rood
`#00ff00`	groen
`#0000ff`	blauw
`#ffff00`	geel
`#00ffff`	cyaan
`#ff00ff`	magenta

Wanneer elk van de drie cijferparen (voor rood, groen en blauw) telkens bestaat uit twee gelijke cijfers (zoals in de acht eerste voorbeelden hierboven), dan kan de verkorte notatie worden gebruikt: #rgb De browser verdubbelt zelf elk van deze cijfers om tot de volledige kleurcode te komen.

Zo is `#ffff00` (geel) gelijk aan `#ff0`

in CSS kan je ook de functie `rgb()` gebruiken:

```
{ color: #ffff00 }   /* geel */
{ color: rgb(255,255,0) } /* geel */
```

Deze functie neemt ook drie getallen van 0-255 die respectievelijk rood-groen-blauw voorstellen. Je kan een hexadecimaal getal dus vertalen naar een rgb decimale waarde.

Een afgeleide kleurenruimte is het **RGBA** model waarbij de laatste waarde de transparantie bepaalt. Dit wordt uitgelegd in het hoofdstuk "CSS3 features"

7.3.3 HSL-kleuren

Nog een ander kleurenmodel is **HSL**, voor hue-saturation-lightness.

Een dergelijke waarde is een triplet van

- **hue**: de hoek van een kleurencirkel: min 0, max 360, 0=360=rood, groen=120, blauw=240, etc…
- **saturation**: een percentage, 100% is volledig verzadigd, 0% is grijs
- **lightness**: een percentage , hoe hoger hoe lichter. 50% is normaal, 0% is zwart, 100% is wit



Infeite is HSL veel intuïtiever dan RGB want je moet enkel het **kleurenwiel** bekijken.

We passen dit toe door middel van de functie `hsl()`:

```
{ color: hsl(0, 100%, 50%) }   /* rood */
{ color: hsl(120, 100%, 50%) } /* geelgroen */
{ color: hsl(120, 100%, 25%) } /* donker groen */
{ color: hsl(120, 100%, 75%) } /* licht groen */
```

Een afgeleide kleurenruimte is het **HSLA** model waarbij de laatste waarde de transparantie bepaalt. Dit wordt uitgelegd in het hoofdstuk "CSS3 features".

7.3.4 transparent en currentColor

Een speciale waarde is `transparent`, die expliciet de kleurwaarde op doorzichtig zet:

```
div { background-color: transparent; }
```

Nog een speciale waarde is `currentColor`: deze waarde verwijst naar de huidige waarde van de `color` eigenschap. Je gebruikt het in andere eigenschappen om expliciet dezelfde kleur te gebruiken en hem toch maar éénmaal in te stellen. Een voorbeeld:

```
div {
  color: #4682B4;
  border: 2px solid currentColor;
  box-shadow: 0 0 5px solid currentColor;
}
```

Opmerking: sommige eigenschappen erven de kleur van color uit zichzelf (via inheritance), `currentColor` is niet altijd nodig.

7.3.5 color

De eigenschap `color` wordt gebruikt om de **voorgrondkleur** (doorgaans de **tekstkleur**) op te geven.

Mogelijke waarden zijn:

- kleurnaam,
- een RGB-waarde, hexadecimaal of `rgb()`
- een HSL-waarde met `hsl()`
- `inherit`
- `currentcolor`

Dit betekent in eerste instantie tekstkleur maar andere eigenschappen erven de kleur, bijvoorbeeld border en text-decoration als ze niet gespecificeerd zijn.

Opmerkingen:

- `transparent` is hier ongeldig.
- doordat die **overgeërfd** wordt, is het voldoende die kleur in te stellen voor het root-element.
- denk aan **Accessibility**:
  - gebruik voldoende **contrast**: bekijk je pagina eens op een zwart-wit scherm!
  - gebruik **niet enkel kleur**: Wat heb je aan “Klik op de rode link” als je geen kleuren ziet? Laat ook `font-weight`, `font-style` en `font-size` variëren
  - denk aan veel voorkomende kleuren-blindheid: rood-groen,…

De volgende notaties zijn identiek:

```
color: #ffffcc;
color: rgb(100%, 100%, 80%);
color: rgb(255, 255, 204);
color: hsl(40,240,216)
```

7.3.6 De kleuren van hyperlinks

Het is je vast al opgevallen dat **hyperlinks in websites standaard blauw** zijn. Op het moment dat je op een hyperlink klikt wordt deze even rood en als je een pagina opent waarin hyperlinks voorkomen naar pagina’s die je reeds bezocht, worden de hyperlinks paars weergegeven. Dit zijn de standaardkleuren voor hyperlinks (blauw, rood, paars).

Je kan de **kleuren zelf beïnvloeden door gebruik te maken van CSS**.

Een hyperlink is uit zichzelf een dynamisch element. Het kan zich in verschillende *states* bevinden:

`a:link` (of gewoon `a`)	een nog niet bezochte link.
`a:visited`	een reeds aangeklikte link.
`a:hover`	een link waar men met de muis over beweegt.
`a:active`	een link op het moment dat erop geklikt wordt.

Als je even nadenkt en de regels van de Cascade in acht neemt, kom je ongetwijfeld tot het besef dat een `a:hover` nà `a:link` en `a:visited` moet komen om effect te hebben. Net zoals een `a:active` nà een `a:hover` moet komen!

Bijvoorbeeld:

```
a {
  color:#00cc33;
}
a:visited {
  color:#66ccff;
}
a:hover {
  background:#99ffff;
  color:#0000cc;
}
a:active {
  background:#99ffff;
  color:#0000cc;
}
```

Probeer hier zelf eens het één en ander mee uit!

7.3.7 background-color

De eigenschap `background-color` wordt gebruikt om de achtergrondkleur van een element op te geven. Een geldige kleurwaarde, inclusief `transparent`

```
body {
  background-color: #fcf;
}
```

7.3.8 background-image

Met de eigenschap `background-image` kan je instellen welke afbeelding gebruikt moet worden als achtergrond. Mogelijke waarden zijn:

- `none`
- een url (met de functie `url()`, bv. `url(img/clouds_groot.png)`)



U.R.I.
Een uri is een **Uniform Resource Identifiër** , een algemene term voor de locatie van

een document. Een uri wordt in CSS meegegeven met de functie `url()`

Het is best altijd de `background-image` eigenschap te combineren met de `background-color` eigenschap. De kleur wordt dan weergegeven als de afbeelding niet gevonden wordt, of als vulling van de transparante delen van de afbeelding.

Bijvoorbeeld:

```
body {
  background-color: #fcf;
  background-image: url(img/clouds_groot.png);
}
```

Bekijk het resultaat in je browser!

7.3.9 background-repeat

Met de eigenschap `background-repeat` kan je instellen of de **achtergrondafbeelding al dan niet herhaald** moet worden.

Mogelijke waarden zijn:

`repeat`	de afbeelding wordt herhaald zowel vertikaal als horizontaal. Dit is de standaard-instelling
`no-repeat`	de afbeelding wordt niet herhaald en slechts één keer getoond.
`repeat-x`	de afbeelding wordt enkel langs de x-as, of horizontaal, herhaald.
`repeat-y`	de afbeelding wordt enkel langs de y-as, of verticaal, herhaald.

Bijvoorbeeld:

```
body {
  background-color: #fcf;
  background-image: url(img/clouds_groot.png);
  background-repeat: no-repeat;
}
```

Bekijk het resultaat in je browser. Je zal zien dat de afbeelding nu slechts één keer op de achtergrond van de pagina staat, namelijk links bovenaan.

```
body {
  background-color: #fcf;
  background-image: url(img/clouds_groot.png);
  background-repeat: repeat-y;
}
```

De afbeelding wordt enkel vertikaal herhaald.

7.3.10 background-position

Bepaalt de uitlijning van de achtergrondfiguur t.o.v. de box. Met de eigenschap `background-position` kan je instellen **waar** de achtergrondafbeelding precies moet

komen te staan. Je geeft telkens twee waarden op, één voor de horizontale verplaatsing (x) en één voor de verticale verplaatsing (y).

Je kan dit op volgende manieren doen:

in procenten	bv. 20% 10%.
in pixels	bv. 400px 60px.
met keywords	bv. top left.

Indien niet gespecificeerd, wordt een achtergrondfiguur steeds in de linkerbovenhoek van de box uitgelijnd, maar met deze eigenschap kan je dat wijzigen.

Indien slechts één lengte opgegeven wordt, bepaalt die de horizontale positie en is de vertikale 50%, worden er twee opgegegeven dan wordt dat beshouwd als hor – vert

Bijvoorbeeld:

```
body {
  background-color: #fcf;
  background-image: url(img/clouds_groot.png);
  background-repeat: no-repeat;
  background-position: 50% 50%;
}
```

Bekijk dit in je browser en je ziet dat de achtergrondafbeelding zowel horizontaal als verticaal gecentreerd wordt op je pagina.

Bekijk ook dit even in je browser:

```
body {
  background-color: #fcf;
  background-image: url(img/clouds_groot.png);
  background-repeat: no-repeat;
  background-position: 20px 200px;
}
```

Je ziet dat de afbeelding nu 20px opschuift langs de x-as (horizontaal) en 200px zakt langs de y-as (verticaal). Het nulpunt van de assen is dus helemaal links bovenaan.

Let op als je een `repeat-x` of `repeat-y` meegeeft:

```
body {
  background-color: #fcf;
  background-image: url(img/clouds_groot.png);
  background-repeat: repeat-y;
  background-position: 20px 200px;
}
```

Door de `repeat-y` zie je niet dat de afbeelding eigenlijk 200px lager gepositioneerd wordt!

7.3.11 background-attachment

Met de eigenschap `background-attachment` kan je instellen of de **achtergrondafbeelding blijft staan of meescrollt** als je scrollt op de pagina.

De mogelijke waarden zijn:

`scroll`	de achtergrondafbeelding scrollt mee
`fixed`	de achtergrondafbeelding blijft op dezelfde plaats staan als er gescrolld wordt.

Bijvoorbeeld:

```
body {
  background-color: #fcf;
  background-image: url(img/clouds_groot.png);
  background-repeat: repeat-y;
  background-position: 20px 200px;
  background-attachment: scroll;
}
```

Probeer dit zelf even uit!

7.3.12 background

Er is ook de **verkorte notatie background** waarmee je alle eigenschappen in één keer kan opgeven.

De verkorte schrijfwijze voor de sub-eigenschappen is:

`background-color||background-image||background-repeat||background-attachment||background-position`

Zo zou je bovenstaand voorbeeld evengoed zo kunnen schrijven:

```
body {
  background: #fcf url(img/clouds_groot.png) repeat-y scroll 20px 200px;
}
```

7.4 Randen

Door gebruik te maken van de border kan **rond HTML-elementen** een rand worden geplaatst, hiermee kunnen die elementen extra benadrukt worden.

7.4.1 border-style

Met de eigenschap `border-style` kan het **type rand** worden opgegeven dat rond een element moet worden getoond.

Mogelijke waarden zijn:

`none`	standaardwaarde; geen rand
`dotted`	stippellijn

dashed	streepjeslijn
solid	volle lijn
double	dubbele volle lijn
groove	3D-effect
ridge	3D-effect
inset	3D-effect
outset	3D-effect

Er kunnen één tot vier waarden worden opgegeven:

4 waarden: wijzerzin vanaf top, right, bottom, left (**trbl**)

3 waarden: top, left=right en bottom

2 waarden: top=bottom en left=right

1 waarde: top=bottom=left=right

7.4.2 border-color

Met de eigenschap `border-color` wordt de **kleur** van de rand opgegeven, op één van de eerder besproken kleurenschema's.

Op dezelfde manier kunnen er één tot vier waarden worden opgegeven.

7.4.3 border-width

Met de eigenschap `border-width` wordt de **dikte** van de rand opgegeven. Mogelijke waarden zijn `thin`, `medium` en `thick`, maar geef bij voorkeur de breedte op met een lengteeenheid (bv. 3px).

Op dezelfde manier kunnen er één tot vier waarden worden opgegeven.

De breedte kan voor elke rand ook afzonderlijk worden opgegeven met de aparte eigenschappen `border-top-width, border-bottom-width, border-left-width, border-right-width.`

7.4.4 Gecombineerde definities

Om voor één van de vier randen een combinatie van kleur, stijl en dikte op te geven kunnen de verkorte notaties worden gebruikt:

`border-top, border-bottom, border-left, border-right.`

De volgorde voor de definitie is:

`breedte stijl kleur.`

Bijvoorbeeld:

```
h1 {
```

```
  border-bottom: 5px solid #ff3300;
}
```

Om deze combinatie voor de vier randen tegelijk te definiëren kan je de eigenschap `border` gebruiken, zoals hieronder uitgelegd

7.4.5 Kortere schrijfwijze

Een aantal eigenschappen zoals `border`, `margin` en `padding` kunnen verkort of uitgebreid geschreven worden.

Sommige omdat ze van toepassing zijn op de vier zijden van de container, andere zijn dan weer een verzameling zijn van een aantal sub-eigenschappen.

Beide schrijfwijzen kunnen juist hetzelfde betekenen:

```
p {border: 1px solid black}
p {
  border-top: 1px solid black;
  border-right: 1px solid black;
  border-bottom: 1px solid black;
  border-left: 1px solid black;
}
```

In bovenstaand voorbeeld wordt een eenvoudig zwart kader rond alle alineas geplaatst.

Toch zijn beide schrijfwijzen ook al een verkorting van drie sub-eigenschappen `border-width, border-color` en `border-style`:

```
p {
  border-width: 1px;
  border-style: solid;
  border-color: black;
}
```

dus is ook het volgende equivalent aan al het vorige:

```
p {
  border-top-width: 1px;
  border-top-style: solid;
  border-top-color: black;

  border-right-width: 1px;
  border-right-style: solid;
  border-right-color: black;

  border-bottom-width: 1px;
  border-bottom-style: solid;
  border-bottom-color: black;

  border-left-width: 1px;
  border-left-style: solid;
  border-left-color: black;
```

```
}
```

Geen haar op ons hoofd dat eraan denkt dit te gebruiken natuurlijk (je vindt het soms in stylesheets die gegenereerd zijn) maar door handig gebruik te maken van cascade, inheritance en specificity (zie volgend hoofdstuk voor detail), kan je de meest efficïente weg bereiken:

```
p {border: 1px solid black}
p.speciaal {
  border-bottom: double;
  border-top-style: double;
  border-bottom-color: red;
}
```

zal alle alineas een eenvoudig zwart kader bezorgen en diegenen met de class speciaal een zwart kader waarvan de bovenrand en de onderrand een dubbele lijn is en die onderaan ook rood is.

Let er even op dat `border-bottom: double` en `border-top-style: double` hier hetzelfde effect hebben en allebei geldig zijn: in de verkorte schrijfwijze border zijn de subeigenschappen width, style en color optioneel.

7.4.6 outline

`outline` is de verkorte notatie voor de eigenschappen `outline-style`, `outline-width` en `outline-color.`

Deze eigenschap lijkt sterk op `border` maar de lijn **neemt geen enkele ruimte in**. Elke `outline` wordt getekend bovenop de inhoud en heeft dus geen invloed op de totale breedte/hoogte van een box. Outlines kunnen niet-rechthoekig zijn.

`outline` worden veel gebruikt door ontwikkelaars tijdens het debuggen van de lay-out, maar bijvoorbeeld ook om aan te duiden waar de `focus` op ligt: typisch rond velden of knoppen.

```
  button:focus  { outline: thick solid black }
```

7.5 Lijsten opmaken

7.5.1 list-style-type

De eigenschap `list-style-type` bepaalt hoe de markering van een lijst (een `ul` of `ol` element) of van een lijst-item (een `li` element) wordt weergegeven.

Mogelijke waarden zijn :

`disc (beginwaarde)`

`circle`

`square`

`decimal`

`lower-roman`

`upper-roman`

`lower-alpha`

`upper-alpha`

`none (geen markering)`

Voeg bijvoorbeeld onderstaande code toe aan je CSS:

```
ul {
  list-style-type: square;
}
```

Als je dan het resultaat bekijkt in je browser zie je dat de bolletjes vervangen werden door blokjes. Als je `list-style-type: none` gebruikt, wordt er geen markering weergegeven.

7.5.2 list-style-image

De eigenschap `list-style-image` bepaalt met welke afbeelding een lijst (`ul` of `ol` element) of een lijst-item (een `li` element) wordt weergegeven.

Mogelijke waarden zijn :

`none` (de standaardinstelling)

een `url` naar de afbeelding (met de functie `url()`)

Indien de afbeelding niet wordt gevonden, wordt de instelling van `list-style-type` gebruikt.

Vervang de vorige CSS specificatie voor het `ul` element door:

```
ul {
  list-style-image: url(img/check.png);
}
```

Je moet uiteraard nakijken of het pad van deze afbeelding wel degelijk juist is (gerekend vanuit het stylesheet)!

Als je dit opnieuw bekijkt zie je dat het blokje nu vervangen is door de afbeelding die je zelf opgaf. Dat nog niet alles mooi op de juiste plaats staat leren we later in deze cursus oplossen.



7.5.3 list-style-position

De eigenschap `list-style-position` bepaalt **hoe** de afbeelding moet geplaatst worden t.o.v. de tekst van het lijst-item.

Mogelijke waarden zijn :

`outside`, de standaardwaarde: afbeelding staat buiten het tekstblok

`inside`: de afbeelding staat binnen het tekstblok

7.5.4 list-style

De eigenschap `list-style` is de verkorte notatie voor combinaties van de voorgaande eigenschappen.

Bijvoorbeeld:

```
list-style-image : url(images\ check.png);
list-style-type : circle;
list-style-position : outside;
```

kan ook genoteerd worden als:

```
list-style : url(images\ check.png) circle outside;
```

8 MEER SELECTOREN…

Zoals we eerder zagen is elke stijl gekoppeld aan een selector. Deze kan in eerste plaats een element zijn (bv. a of h1), en we zagen ook al dat je een opsomming van selectoren kon maken door ze met een komma van elkaar te scheiden, maar er zijn nog verschillende andere mogelijkheden en combinaties, waarmee je een meer verfijnde selectie kan maken.

We bespreken hier alle selectors van de standaarden CSS2.2 en CSS3. Een nieuw level CSS4 is momenteel in aanmaak maar nog niet actief.

Je kan alles nog eens nalezen op de **CSS3 selectors module** op

http://www.w3.org/TR/2011/REC-css3-selectors-20110929/

Wat gebeurt er als een browser een selector niet ondersteunt?

☞ **dan negeert hij de volledige declaratie**!

Veronderstel dat een browser *attribuut selectoren* niet begrijpt, dan zal hij de volledige declaratie hieronder negeren en dus ook de selector `button` niet uitvoeren.

```
button, input[type=submit], input[type=reset], input[type=button] {
  padding:0.5em 1em;
  font-size:1.1em;
  color:#CC0033;
}
```

Als je vreest dat dat kan gebeuren, groepeer dan je selectoren niet, zet ze apart.

8.1 universal selector

De universal selector * selecteert **elk element**. Alle elementen in een document worden daarmee aangesproken. De universal selector gebruiken is niet hetzelfde als vertrouwen op *inheritance*: veronderstel een document structuur:

```
<body>
<div>
  <p>deze tekst moet in Verdana</p>
  <div>en deze ook</div>
</div>
<p>Copyright vdab in Times</p>
</body>
```

Veronderstel dat je enkel de copyright in *Times New Roman* wil en de rest in *Verdana*, dan geven de style rules

```
body { font-family: "Times New Roman", Times, serif}
div { font-family: Verdana, Arial, Helvetica, sans-serif }
```

en

```
* { font-family: "Times New Roman", Times, serif}
div { font-family: Verdana, Arial, Helvetica, sans-serif }
```

niet hetzelfde resultaat.

Enkel het eerste geval zal correct zijn want daar zullen ook alle *child-elementen* van de `div` door *inheritance* (want ze krijgen anders geen waarde) het lettertype Verdana krijgen. In het tweede geval specificeert de `*` selector expliciet alle elementen en zet ze in Times, enkel `div` elementen zullen Verdana zijn.

Als de `*` selector gebruikt wordt in context, kan hij ook worden weggelaten.
De twee onderstaande style rules zijn identiek:

```
*.warning {font-weight: 14px;}
.warning {font-weight: 14px;}
```

8.2 Gebruik van class en id

Eén manier om elementen vorm te geven is gebruik maken van de HTML *core attributen* `class` of `id`. Een **core attribuut** is een HTML-attribuut dat van toepassing is op **eender welk HTML element**: `class`, `id`, `style` en `title`.

Aan een element of groep elementen kan een opmaak toegekend worden door middel van het `class` attribuut. Dit laat je toe die groep te onderscheiden van normale elementen.

Een element kan ook meer dan één `class` toegekend worden, meerdere classen worden gescheiden door een spatie.

In het eerste vb krijgt een `p` element een `class` *nota* toegewezen, in het tweede vb krijgt een `div` de classes *nota* en *zwarteRand* toegewezen.

```
<p class="nota">inhoud</p>
<div class="nota zwarteRand">inhoud</div>
```

Style rules kunnen dan op deze elementen toegepast worden via een **class selector**:

```
.nota {font-family: italic 0.8em "Courier New", Courier, monospace
}
```

Een class selector begint met een punt.

Een element kan geïdentificeerd worden als een **uniek element** door middel van het `id` attribuut. Er mag dus in hetzelfde document geen tweede element voorkomen met hetzelfde `id`.

Zowel `class` als `id` waarden mogen niet met een getal beginnen en zijn hoofdlettergevoelig.

```
<div id="logo">vdab/div>
```

Style opmaak kan op dit element toegepast worden via een id selector.

```
#logo {color: red; font-size: 14px;… }
```

8.2.1 class versus id:

class	id
toegekend aan meerdere elementen	toegekend aan slechts één element
element kan meerdere classen krijgen	element kan slechts één id krijgen
specificiteit lager dan id	specificiteit hoger dan class
	gebruikt als identifier

Als beginner is het je niet altijd duidelijk of je een `class` of een `id` moet gebruiken.

Neem bijvoorbeeld een `div` die "inhoud" zal bevatten: je wil die een beperkte breedte geven. Moet dat `<div class="inhoud">` of `<div id="inhoud">` worden?

In principe zijn ze allebei goed want je kan net zo goed opmaak vastmaken aan `.inhoud` als aan `#inhoud`. Als er echter zelf maar een kleine kans is dat je dezelfde opmaak op een ander element kunt toepassen of dat je een tweede `div` "inhoud" zou kunnen hebben, gebruik dan eerder de `class` dan de `id`: wij verkiezen `<div class="inhoud">.`

Gebruik `id` zo weinig mogelijk voor opmaak maar eerder voor identificatie doeleinden (die gebruikt zullen worden door Javascripts en formulieren): serienummers, koopcodes, etc.. en als je echt niet anders kan, ook voor unieke elementen waarvan de opmaak speciaal is: `#logo`,…

8.3 class selector

We hadden het daarnet over de verschillen tussen `class` en `id`.

Een **class selector** schrijf je als een punt gevolgd door de *className*:

```
.mijnClass { … opmaak  …}
```

Meestal wordt gebruik gemaakt van **generische classen**: dat zijn classen die op **alle** HTML elementen toepasbaar zijn:

```
.klein          { font-size: small }
.groen          { background-color: green }
.zwarterand     { border: solid 2px black }
```

Je kan echter ook een class **koppelen aan een bepaald element** zoals:

```
p.nota          { background-color: yellow }
div.nota        { background-color: green }
div.zwarterand  { border: solid 2px black }
```

Let erop dat in bovenstaand voorbeeld er twee verschillende classen nota zijn: ze zijn geassocieerd met een bepaald element.

Elementen kunnen zonder problemen **meerdere classen** toegewezen krijgen door ze te scheiden met spaties:

```
<div class=”groen zwarterand klein”>blablabla</div>
```

De **volgorde** van die classen speelt daarin geen rol zolang de declaraties ervan niet dezelfde eigenschappen beschrijven.

Gebruik je een *classname* die niet terug te vinden is in het stylesheet dan vormt dat ook geen probleem.

Combineren van meerdere classen met verschillende eigenschappen is een overzichtelijke manier om opmaak toe te passen.

Toch kunnen ook class selectoren gecombineerd worden:

```
.t1 {background:red}
.t2 {color:red}
div.t1.t2 {background: magenta; color:white}
```

en toegepast

```
<p class="t1">deze tekst heeft een rode achtergrond</p>
<div class="t2">deze tekst heeft rode tekstkleur</div>
<div class="t1 t2">deze tekst moet wit op magenta zijn</div>
```

8.4 id selector

Zoals we al zeiden, gebruik je bij voorkeur een `class` om opmaak toe te passen, enkel bij echt unieke elementen gebruik je een `id` selector:

Syntax:

```
#mijnId { opmaak }
```

Een **id selector** is een **#** gevolgd door het `id`:

```
#logo            { color: red }
#hoofdmenu ul li { color: green }
```

8.5 selectoren combineren

Eén van de sterke punten van CSS is de mogelijkheid om gebruik te maken van de relatie die elementen met elkaar hebben, door selectoren te gaan combineren.

Men gebruikt wel eens de term "contextuele CSS": het laat ons toe de opmaak van eenzelfde soort element te variëren volgens de plaats (*parent*) waarin het voorkomt.

8.5.1 Descendant combinator

Descendant combinators duiden een element aan dat een *child* is van een ander element.

Dit geneste element komt dus ergens voor in de *descendant tree* op eender welk niveau. Het hoeft dus niet persé een *immediate* child te zijn.

De descendant wordt van de parent gescheiden door een spatie.

Algemene syntax:

```
element1 element2 { stijldeclaratie }
```

Waarbij element2 *child* is van element1

Voorbeelden:

```
div p {font-size:0.9em}
```

Elke `p` die zich binnen een `div` bevindt krijgt een kleinere lettergrootte.

In onderstaand voorbeeld worden alle `ul` elementen die genest zijn in een `ol` of in een andere `ul`, van een vierkante bullet voorzien:

```
ol ul, ul ul    {list-style-type: square }
```

De selectoren hoeven niet allemaal elementen te zijn; er kan ook een combinatie van element-selectoren en attribuut-selectoren worden gebruikt:

```
div .opgelet {
  background-color: #cc33cc;
}
```

Deze stijl zal enkel worden toegepast op een element met `class`="opgelet" indien die binnen een `div` element voorkomt.

8.5.2 child combinator

Een *child combinator* bepaalt de opmaak van een element dat een **onmiddellijk kind** is van een ander element.

De syntax is:

```
parent > child.
```

In onderstaand voorbeeld wordt een kop `h2` anders opgemaakt als hij een child is van een `div`.

```
div > h2 {
  … styles
}
```

Dit zal dus van toepassing zijn op de eerste titel, niet op de tweede:

```
<div>
  <p>introtekst</p>
  <h2>eerste titel</h2>
  <article>
         <h2>tweede titel</h2>
  </article>
</div>
```

8.5.3 adjacent combinator

Een adjacent combinator betreft een element dat onmiddelijk op een ander **volgt** en dezelfde parent heeft, een **sibling** dus. De elementen worden gecombineerd door een + teken.

De syntax is:

`child + child`.

In onderstaand voorbeeld wordt een kop h2 anders opgemaakt als hij onmiddellijk volgt op een h1.

```
h1 + h2  {
  margin-top: -5px
}
```

Dit zal dus van toepassing zijn op de tweede titel, niet op de eerste en derde:

```
<div>
  <h1>eerste titel</h1>
  <h2>tweede titel</h2>
  <p>tekst</p>
  <h2>derde titel</h2>
</div>
```

8.6 attribuutselectoren

Een **attribute selector** laat toe elementen op te maken die een bepaald attribuut hebben of waarvan een attribuut een bepaalde waarde heeft.
Het attribuut wordt in vierkante haakjes geschreven.

Een element heeft een title attribuut (ongeacht wat er in staat):

```
h1[title]       {color: blue }
```

Een element heeft een attribuut met een bepaalde waarde:

```
a[href="http://www.vdab.be"]   {font-weight: 800 }
```

Een element heeft een attribuut waarin een bepaalde term of enumeratie voorkomt tussen andere termen:

```
a[class~="opmerking"] { background-color : lime }
```

Een element heeft een attribuut met een waarde die in een lijst van met-koppeltekens-van-elkaar-gescheiden-waarden voorkomt:

```
span[lang|="fr"] { background-color : red }
```

Een element heeft een attribuut waarvan de inhoud (foobar) begint met de substring:

```
p[title^="foo"] { background-color : lime }
```

Een element heeft een attribuut waarvan de inhoud eindigt met de substring:

```
p[title$="bar"] { background-color : lime }
```

Een element heeft een attribuut waarvan de inhoud de substring bevat:

```
p[title*="ooba"] { background-color : lime }
```

Ondersteund door FF1.5, O9, IE7. Niet door IE6.

8.7 Pseudo-classes

Met pseudo-class selectoren kan je elementen selecteren die zich in **een bepaalde staat** bevinden, iets dat niet uit de document tree af te leiden valt.

Bijvoorbeeld een veld waar de focus op ligt, of een hyperlink tijdens de klik.

8.7.1 De dynamische pseudo-classes

Het `a` element heeft tot nu toe de exclusieve rol van hyperlink vervuld. CSS op een `a` element laat de styling van 4 dynamische pseudo-classes toe:

```
a               { font:90% monospace; text-decoration:none; }
a:link          { color: red}
a:visited       { color: blue}
a:hover         { color: yellow; text-decoration:underline; }
a:active        { color: lime }
```

Let erop dat de `:hover` pseudoclass identifier altijd na die van `:link` en `:visited` moet staan omdat anders door aangeklikte links geen zichtbaar hover effect meer zouden krijgen

Ook `:active` moet op dezelfde manier volgen op `:hover`

Door eerst het `a` element zonder pseudoclasses te stylen, brengen we enkele algemene eigenschappen via *inheritance* over.

Ook andere elementen kunnen deze pseudo-classes hebben:

```
ul.menu li:hover {background-color:#999999}
ul.menu li:active {background-color:#00FF00}
```

In dit voorbeeld creëren we een roll-over effect op een bolletjeslijst van de `class` *menu*. Ook als er op geklikt wordt wijzigt de kleur.

8.7.2 De :focus pseudo-class

Een HTML element krijgt de focus als er op geklikt wordt of als het via een toets geselecteerd wordt. Je kan het dan een speciale opmaak geven met `:focus`.

Vooral van toepassing op formulierelementen zoals inputvelden en knoppen:

```
input:focus {background-color:salmon}
```

8.7.3 De :target pseudo-class

Sommige hyperlinks verwijzen naar een bepaalde plaats binnen een bron, een "bookmark/anker" of een *fragment identifier* :

```
http://www.vdab.be/inlichtingen.php#werknemers
```

In dit voorbeeld wordt verwezen naar een element met de `id` "werknemers".

Met de `:target` pseudo-class kan je verwijzen naar zo'n target element.

als de URI geen fragment identifier heeft, is er ook geen target.

Met deze methode plaats je opmaak op het target:

```
*:target { color : red }
```

Nu wordt de tekst van alle targets rood als je erop arriveert via een URI met anker.

8.7.4 De :lang() pseudo-class

Als een document of een deel ervan in een bepaalde taal is aangegeven kan je die inhoud met deze selector aanduiden.

De `:lang()` pseudo-class selecteert een element waarvan de taal is aangegeven in de haakjes:

```
html:lang(fr) { background:yellow }
```

Hier wordt de achtergrond van een pagina waarvan de `html` tag het attribuut `language` frans heeft, geel gekleurd.

8.7.5 structurele pseudo-classes

Dit type selector baseert zich enkel op de **plaats van het element in de document tree**. We herhalen even enkele DOM begrippen:

child:	kindNode van een element
sibling:	zusterNode van een element (zelfde parent)

Hieronder sommen we ze op voor een hypothetisch element E.

`E:root`	root van het document

`E:nth-child(an+b)`	het n-th child van de parent
`E:nth-last-child(an+b)`	het n-th child van de parent, omgekeerd tellend
`E:nth-of-type(n)`	de n-th sibling van het type
`E:nth-last-of-type(n)`	de n-th sibling van het type, omgekeerd tellend
`E:first-child`	eerste child van zijn parent
`E:last-child`	laatste child van zijn parent
`E:first-of-type`	eerste sibling van dit type
`E:last-of-type`	laatste sibling van dit type
`E:only-child`	enige child van de parent
`E:only-of-type`	enige sibling van dit type
`E:empty`	leeg element (geen children inclusief text nodes)

Het is duidelijk dat je hiermee zowat elk child element uit de DOM-tree kunt selecteren. Tot en met CSS3 bestaat er jammer genoeg geen `E:parent` selector waarmee we de *parent* van een element zou kunnen aanduiden. CSS4 (in ontwikkeling) zou die wel voorzien.

In het argument `an+b`, zijn `a` en `b` integers. Ze kunnen ook vervangen worden door de sleutelwoorden `odd` en `even` (zie voorbeelden).
Als `a=0` dan slaat de opmaak slechts op het `b`-de element

De `of-type` pseudoclasses hebben betrekking op het soort element waarop ze toegepast worden.

Enkele losse voorbeelden:

```
div > p:first-child     /* de eerste alinea die een child is van een div */
tr:nth-child(2n+1)      /* elke oneven rij van een HTML table */
tr:nth-child(odd)       /* idem */
tr:nth-child(2n)        /* elke even rij van een HTML table */
tr:nth-child(even)      /* idem */
```

Alternerende alinea kleuren

```
p:nth-child(4n+1) { color: navy; }
p:nth-child(4n+2) { color: green; }
p:nth-child(4n+3) { color: maroon; }
p:nth-child(4n+4) { color: purple; }
```

In een pagina met meerdere inline figuren worden deze alternerend links en rechts gefloat:

```
img:nth-of-type(2n+1) { float: right; }
img:nth-of-type(2n) { float: left; }
```

In een pagina met verschillende `pre` elementen wordt elk tweede `pre` element van zijn *parent container* een rode gestipte rand gegeven:

```
pre:nth-of-type(2) { border:3px dashed red;}
```

Alle moderne browsers ondersteunen deze pseudo-classes.

8.7.6 De UI States pseudo-classes :enabled :focus :disabled :checked :intermediate

Deze pseudo-classes van een *User Interface* element – een formulier element – hebben betrekking op de staat waarin het zich bevindt.

Een element is `:disabled` als de gebruiker niet in staat is het te activeren of er de `:focus` naar toe te verplaatsen.

Radiobuttons en checkboxes kunnen aangevinkt worden: dan zijn ze `:checked`. Ze kunnen echter ook in een neutrale staat zijn: niet aan- of afgevinkt. In dat geval is de `:intermediate` pseudoclass van toepassing.

8.7.7 De :not negatie pseudo-class

De `:not()` negatie pseudo-class neemt een andere selector als argument (met uitzondering van zichzelf en de pseudo-elementen) en selecteert deze niet.

```
*:not(div)             /* alle elementen met uitzondering van de divs */
button:not([DISABLED]) /* alle knoppen tenzij ze disabled zijn */
```

8.8 Pseudo-elements

Pseudo-elementen laten ons toe opmaak te voorzien voor elementen die niet bestaan in de DOM. Het zijn dikwijls delen van elementen (bv, `::first-line`) of inhoud die op het moment zelf aangemaakt wordt (bv, `::after`) (Ja je hoort het goed: CSS kan inhoud aanmaken zonder Javascript!).

Om ze te onderscheiden van pseudo-classes schrijven we ze met een dubbele dubbelpunt gevolgd door het pseudo-element:

```
::pseudo
```

De oude schrijfwijze had maar één dubbelpunt.

8.8.1 Het ::first-line pseudo-element

Het `::first-line` pseudo-element selecteert de eerste lijn van een block-level element of een tabel-cel.

```
p::first-line { font-style: italic }
```

Opmerking:

- als je dit zou willen gebruiken om de eerste lijn te doen inspringen, gebruik dan liever de `text-indent` property die automatisch de eerste lijn van een blok element doet inspringen

8.8.2 Het ::first-letter pseudo-element

Het `::first-letter` pseudo-element selecteert de eerste letter van een element en dient om *drop-caps* te maken

```
p{
        font-size: 1emt;
        line-height: 1.2em;
 }
p::first-letter {
        font-size: 200%;
        font-style: italic;
        font-weight: bold;
        float: left;
 }
```

8.8.3 De ::before en ::after pseudo-elementen

De `::before` en `::after` pseudo-elementen worden gebruikt om inhoud aan te maken en in te voegen voor of achter een element. Zie verder bij de topic "Generated Content"

```
h1::before {content: counter(chapno, upper-roman) ". "}
```

9 INHERITANCE EN DE CASCADE

Een HTML-element kan van opmaak voorzien worden door inline, embedded en externe stylesheets.

Sommige stylesheets werden geschreven door maker van een website (Author stylesheets).Daarnaast zijn er ook nog de lay-outs van de browsers (User Agent stylesheet) en zelf surfers kunnen een eigen css aan de browser koppelen.

Hoe bepaalt de browser dan de uiteindelijke opmaak?

Dat wordt geregeld door de principes van Inheritance en de Cascade.

9.1.1 Oorsprong van stylesheets

Er bestaan 3 soorten stylesheets:

- **Default stylesheet van de User agent:**

  De browser heeft een ingebouwd stylesheet dat een aantal standaardwaarde bevat voor de opmaak van de elementen: Dat is de reden waarom een `h1` groter is dan een `h2` zonder dat jij iets doet. De rules van dit stylesheet worden eerst toegepast, maar worden meestal overschreven door de rules van het Author stylesheet

- **Author stylesheet:**

  Dit is een *intern* of een *extern* stylesheet gekoppeld aan de pagina en dus meestal gemaakt door de ontwikkelaar van de site.

  Heeft voorrang op het User stylesheet.

- **User stylesheet:**

  De gebruiker kan besluiten **zelf een stylesheet te koppelen** aan het doc. Dat kan enkel via het menu van de browser.
  Zelden gebruikt tenzij door mensen met gezichtsproblemen om hoog contrast en zeer grote letters te gebruiken

9.2 Hoe wordt CSS toegepast door de browser?

Als een browser een HTML of XML document ontvangt, interpreteert (*parst*) hij dit document en wordt er een **document tree** opgesteld.

Daarna leest hij alle stijlinformatie die hij kan vinden en berekent voor elk element in de document tree de waarde voor elke CSS eigenschap aan de hand van enkele regels:

1. De browser zoekt **alle stijlregels** in alle stylesheets en inline styles.
2. Hij splitst die informatie op per **medium** (voor screen, mobiel, print, ...)
3. Hij maakt per medium een **Visual Formatting Model** rekening houdend met alle informatie die hij heeft en met de voorrangsregels.

Daarna toont hij het document op het medium (scherm, printer,…)

9.3 De Cascade



Waarom werkt mijn selector niet???
Een veelgeslaakte kreet...
heeft altijd te maken met oorsprong, inheritance, specificiteit en volgorde
Wie CSS wil begrijpen moet de Cascade snappen...

De cascade is het mechanisme waarbij een browser alle declaraties die op een element betrekking hebben sorteert en de uiteindelijke waarde toekent voor een eigenschap:

- naar **oorsprong**: uit welk stylesheet komen ze?
- naar **specificiteit**: als meerdere selectoren van toepassing zijn, welke is dan meest specifiek?
- naar **volgorde**: als twee regels dezelfde specificiteit hebben, wint de laatste in volgorde

9.3.1 Volgorde

In een CSS bestand staat het volgende:

```
p { color:red; }
p { color:blue; }
```

Wat zal de tekstkleur zijn van alle paragrafen? Blauw.

De **volgorde** van stijlregels is van belang: de browser begint bovenaan te lezen; bij regels **met dezelfde specificiteit** overschrijft de tweede regel de vorige.

9.3.2 Specificiteit

In een CSS bestand staat het volgende:

```
div p    { color:red; }
p        { color:blue; }
```

Wat zal de tekstkleur zijn van alle paragrafen? Blauw? Nee, de paragrafen die in een `div` staan zullen rood zijn omdat die eerste stijlregel een **hogere specificiteit** heeft.

De eerste regel heeft een specificiteit van **0,0,0,2**, de tweede van **0,0,0,1**.



Elementen krijgen dikwijls meerdere style declaraties op zich toegepast. Welke is er uiteindelijk van toepassing? **de selector met de hogere specificiteit zal winnen**.

Een ander voorbeeld: een `h1` element heeft een `id` *speciaal*: als we het stylesheet overlopen zijn een aantal selectoren op dit element van toepassing:

```
h1              {color:red; background:yellow;}
body h1         {color:blue; background:pink;}
#speciaal       {color:green;}
```

```
…
```

In eerste instantie is de parser verplicht alle declaraties op te splitsen *per eigenschap*, dus bovenstaande wordt:

```
h1 {color:red;}
h1 {background:yellow;}
body h1 {color:blue;}
body h1 {background:pink;}
#speciaal {color:green;}
```

Welke style rules zullen winnen? Het antwoord hangt weer af van de specificiteit van de stijlregel. In dit voorbeeld zullen alle `h1` elementen blauw met roze achtergrond zijn, maar het `h1` element met de `id` speciaal, zal groen op roze zijn.

9.3.3 berekenen van de specificiteit

De specificiteit bestaat uit vier getallen startend op nul: **0,0,0,0**
De getallen worden **van links naar rechts** toegekend volgens de volgende regels:

1. Tel 1 als de declaratie inline is, dus via het HTML attribuut `style`, zoniet 0
2. Tel het aantal `id` attributen in de selector
3. Tel het aantal `class`, attribute of pseudo-class selectoren in de selector
4. Tel het aantal elementen of pseudo-elementen

Voorbeeld:

```
div p    { color:red; }
p        { color:blue;}
p.opm    { color:green;}
div p.opm { color:cyan;}
#speciaal { color:black;}
<p style="color: purple" class="opm">tekst</p>
```

In volgorde:

1. 0,0,0,2 = 2
2. 0,0,0,1 = 1
3. 0,0,1,1 = 11
4. 0,0,1,2 = 12
5. 0,1,0,0 = 100
6. 1,0,1,0 = 1010

Nog enkele berekende voorbeelden:

selector	inline style	id	class, attribute selector of pseudo	elementen of pseudo-elementen	specificiteit

			class		
`*`	0	0	0	0	0,0,0,0
`p`	0	0	0	1	0,0,0,1
`ul ul`	0	0	0	2	0,0,0,2
`div>p`	0	0	0	2	0,0,0,2
`h1+p`	0	0	0	2	0,0,0,2
`#footer`	0	1	0	0	0,1,0,0
`div#footer`	0	1	0	1	0,1,0,1
`.nieuw`	0	0	1	0	0,0,1,0
`p.nieuw`	0	0	1	1	0,0,1,1
`a:active`	0	0	1	1	0,0,1,1
`p:first-letter`	0	0	0	1	0,0,0,1
`#container div p.nieuw`	0	1	1	2	0,1,1,2
`div#navbalk *[href]`	0	1	1	1	0,1,1,1
`tr[id=”special”]`	0	0	1	1	0,0,1,1
`tr#special`	0	1	0	1	0,1,0,1
`<p style=”color:red;”>`	1	0	0	0	1,0,0,0

Enkele observaties:

- de *universal selector* heeft een specificity van 0. Een reden om hem zo weinig mogelijk te gebruiken
- de getallen zijn niet *base10*, bv:
  zo heeft een class selector **altijd** een hogere spec dan eender welk aantal elementen:
  `.actief`: 0,0,1,0
  `html body section article section div section div p span a`: 0,0,0,11
- gebruik best een korte selector, dus liever de class selector

- een id selector is hoger dan een attribuut id selector:
  `#speciaal`: 0,1,0,0
  `*[id=speciaal]`: 0,0,1,0
- hyperlinks zijn speciaal
- *inline styles* winnen altijd van stylesheet rules!
  daarom worden inline styles gebruikt bij Javascripts: ze overheersen een stylesheet rule altijd
- *!important* wint echter van *inline style*

9.3.4 Specificiteit van hyperlinks

De opmaak van hyperlinks wordt bepaald met pseudo-classes:

```
a:link {color:blue;}
a:visited {color:gray;}
a:hover {color:green;}
a:active {color:red;}
```

Hun specificiteit is steeds gelijk: `0,0,1,1`. Meerdere kunnen tegelijkertijd toepasselijk zijn op dezelfde hyperlink: zo kan een visited link ge-hoverd worden en active zijn op hetzelfde moment.

Om die reden is hun **volgorde** belangrijk: als de specificiteit gelijk is, wint de laatste.

Moest `:active` vóór `:hover` staan, dan zou deze nooit geactiveerd worden. Daarom is `:link :visited :hover :active` de **enige juiste volgorde**.

Ook belangrijk te vermelden is dat hyperlinks de eigenschap `color` niet automatisch erven van hun parent. Om de kleur van een hyperlink in te stellen zal je dus een selector voor het `a` element moeten maken en een kleur toekennen of de waarde `inherit` geven.

9.3.5 !important

De belangrijkheid van een regel kan verhoogd worden door er een `!important` statement achter te plaatsen (voor de punt-komma).

Een `!important` declaratie **heeft steeds de overhand op andere declaraties** voor een element, wat hun specificiteit ook moge zijn.

```
p { text-indent: 1em !important;}
```

`!important` declaraties **zijn enkel ontworpen om de declaraties in een User stylesheet de overhand te doen krijgen op die van een Author stylesheet**. Op die manier kunnen gebruikers met toegankelijkheidsproblemen hun eigen stylesheets koppelen aan webpaginas.



HELAAS hebben programmeurs - *die niet veel snappen van specificiteit* - de neiging hun CSS problemen op te lossen door `!important` statements toe te passen...

Als je `!important` gebruikt om een selector te doen winnen BEN JE SLECHT BEZIG!

Een voorbeeld:

gegeven deze HTML

```
<div id="voorbeeld">
   <p>tekst</p>
   <p id="groen">tekst</p>
 </div>
```

met de CSS

```
#voorbeeld p    {color:blue;}
#groen          {color:green;}
```

Beide teksten zijn blauw: dit is normaal want `#voorbeeld p` heeft een hogere spec dan `#groen.`

De foutieve oplossing:

```
#voorbeeld p    {color:blue;}
#groen          {color:green !important;}
```

De juiste oplossing:

```
#voorbeeld #groen       {color:green;}
```

of

```
p#groen                 {color:green;}
```

9.3.6 Inheritance

Elementen kunnen een waarde voor een bepaalde eigenschap overerven (*inherit*) van hun *parent* (*container*) in de document tree. De CSS specificatie bepaalt voor alle CSS eigenschappen of ze overerven of niet en van welke property ze erven. Bijvoorbeeld, de eigenschap `font-size` is *inherited*, terwijl `border` dat niet is.
Zo ook erft `font-size` altijd van de `font-size` van de *parent container*, maar `margin` en `padding` uitgedrukt in `em`, erven deze eenheid van de **eigen** `font-size`.

Als een element via stijlregels geen waarde toegekend krijgt, kijkt de browser of dit element een eigenschap kan **erven**, dat is ook meestal het geval. Is dat niet zo dan wordt een **default waarde** toegepast.

Deze default waarden staan ingesteld in het *browser stylesheet*, een ingebouwd stylesheet dat je nooit te zien krijgt. Bijvoorbeeld, zo is de default `font-size` in de meeste browsers 16px.

Een voorbeeld:

```
<head>
<style>
  body   { font-size: 16px; }
  h1     { font-size: 200%;
           margin: 1em;
         }
  small  { font-size: 0.9em;
           padding: 0.5em;
          }
</style>
</head>
<body>
<section>
  <h1>Een <em>dikke</em> kop</h1>
  <p>dit is normale tekst <small>dit is kleiner</small></p>
</section>
</body>
```

In dit voorbeeld zal de `font-size` property van

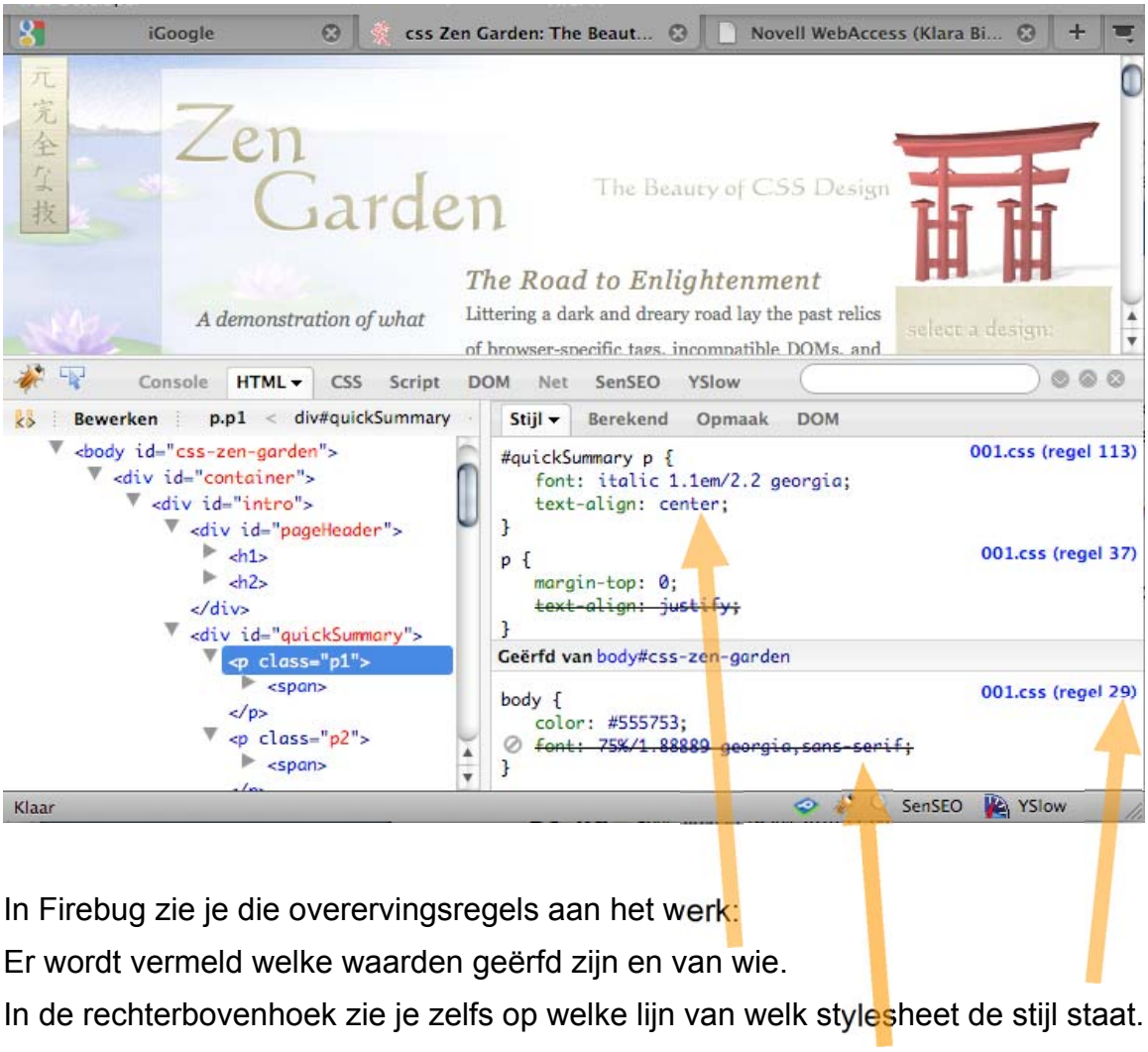
- het `p` element ook `16px` zijn want het heeft het geërfd van `section` die het erfde van `body`. Dit betekent ook dat de relatieve eenheid `em` daaraan gelijkgesteld wordt:
  `1em = 16px`

- het `h1` element zal de berekende waarde
  `200% X 16px = 32px` krijgen. Daardoor wordt `1em` in dit element `32px` en dus wordt de `margin` hier 32px.

- het `small` element zal als `font-size` de berekende waarde
  `16px X 0.9em = 14.4px` krijgen want het erft van `p`.

  Het zal ook een berekende `padding` van
  `14.4px X 0.5em =  7.2px` krijgen want `padding` erft zijn eenheid van de **eigen** `font-size.`

Om *inheritance* aan het werk te zien hoef je enkel een website te openen en te bekijken met een *webdevelopers tool* zoals Firebug: bij het *style panel* wordt daar visueel aangegeven welke regels er op een element van toepassing zijn en welke overschreven worden door **inheritance**, **specificity** en **volgorde**.



In Firebug zie je die overervingsregels aan het werk:

Er wordt vermeld welke waarden geërfd zijn en van wie.

In de rechterbovenhoek zie je zelfs op welke lijn van welk stylesheet de stijl staat.

Een stijlregel die door een andere overschreven is, wordt doorstreept.

Als een element geen stijl heeft meegekregen zie je de default stijl, dat begint met moz- (van Mozilla)

9.3.7 De waarde inherit

Elke eigenschap kan echter de waarde `inherit` toegekend krijgen, en zal dus expliciet de waarde van zijn *parent* overerven. Dit kan je dus toepassen voor eigenschappen die normaal niet overgeërfd worden.

In onderstaand voorbeeld erft een `table` element de `background-color` eigenschappen van zijn container.

```
div#menu        { background-color:#FFCCCC;}
div#menu table  { background-color: inherit; }
```

We vermeldden al eerder dat hyperlinks op die manier de kleur van hun parent kunnen overnemen.

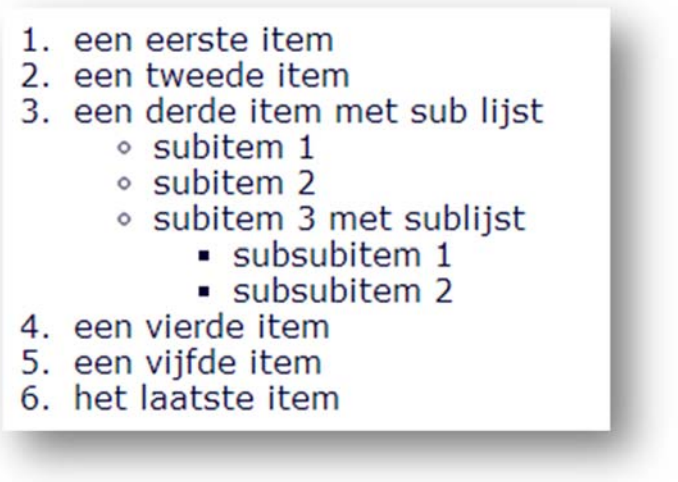
9.3.8 Nog enkele voorbeelden

Lijsten

gegeven deze html

```
<ol>
    <li>een eerste item</li>
    <li>een tweede item</li>
    <li> een derde item met subLijst
      <ul>
        <li>subitem 1</li>
        <li>subitem 2</li>
        <li>subitem 3 met sublijst
          <ul>
            <li>subsubitem 1</li>
            <li>subsubitem 2</li>
          </ul>
        </li>
      </ul>
    </li>
    <li>een vierde item</li>
    <li>een vijfde item</li>
    <li>het laatste item</li>
  </ol>
```

zonder extra CSS – dus default stylesheet - ziet dit er zo uit:

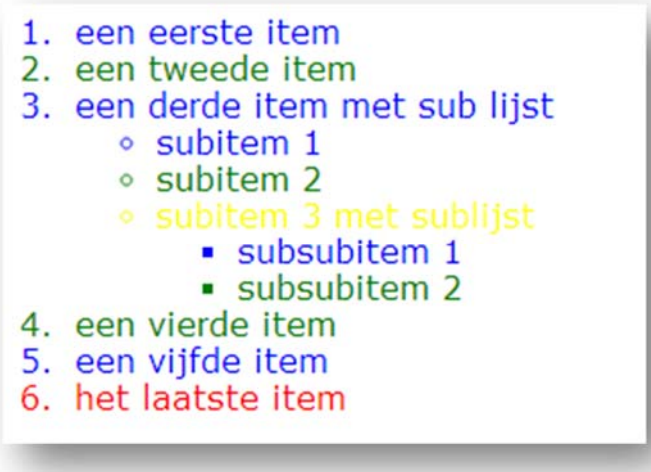


Passen we nu volgende CSS toe

```
ol, ul                {color:blue;}
li:last-child         {color:yellow;}
ol li:nth-child(2n+0) {color:green;}
ol > li:last-child    {color:red;}
```

dan wordt de kleur

- **blauw**: alles uitgezonderd wat volgt.
  `ol, ul`: twee selectors met elk **0001**
- **geel**: "subitem 3 met sublijst"
  `li:lastchild`: **0011**
  selecteert alle `li` elementen die de laatste child zijn van hun parent.
  Er zijn er hier 3 maar twee worden overruled door de hogere specificiteit van wat volgt
- **groen**: "een tweede item", "subitem 2", "subsubitem 2", "een vierde item"
  `ol li:nth-child(2n+0)`: **0012**
  selecteert alle even children onder de `ol`, dus ook deze van de `ul`
- **rood**:"het laatste item"
  `ol > li:last-child`: **0012**
  de vorige rule is ook op dit element van toepassing want even. Deze rule heeft ook dezelfde spec als de vorige, maar staat erna dus volgorde telt



Een menu

Jan heeft problemen: in een menu werken sommige van zijn selectors schijnbaar niet…

De HTML

```
<ul id="menu">
    <li class="actief"><a href="#">home</a></li>
    <li><a href="#">what</a></li>
    <li><a href="#">contact</a></li>
    <li><a href="#">about</a></li>
</ul>
```

de CSS

```
#menu {
  list-style-type:none;
}
#menu li {
  display:block;
  float:left;
  padding:1em;
```

```
  background-color:yellow;
  color:black;
}
#menu li:hover {
  background-color:red;
  color:white;
}
#menu li a {
  display:inline;
  text-decoration:none;
}
.actief {
  background:orange;
  font-weight:bold;
}
```

Resultaat:



Jan's problemen:

1. Waarom werkt de `class` actief niet? de achtergrond moet oranje zijn
2. Waarom is de tekst van de hyperlinks wel vet maar niet zwart?

Probleem achtergrond:

- spec van `.actief` is **0010**
- spec van `#menu li` is **0101**, dus hoger

oplossing:

- wijzig selector naar `#menu .actief`

Probleem hyperlink tekst

- de `font-weight:bold` wordt door de `a` overgeërfd en toegepast
- maar `color` wordt door hyperlinks niet overgeërfd, een uitzondering op de regel.

oplossing:

- wijzig de selector `#menu li a` en plaats er `color:inherit` of `color:black` in

Contextuele CSS

Een slimme manier om aan groepen pagina's in een website verschillende "stijlen" (=kleurenschema's) toe te kennen is gebruik maken van een `class` in de body tag.

Veronderstel dat je in je website pagina's hebt die tot de groep "koken" of "tuin" of "personeel", "admin", etc…

Dan kunnen we een dergelijke pagina een class meegeven in de body tag:

```
<body class="koken">
```

En op die manier contextuele CSS schrijven:

```
/* algemene CSS */
body      { color: black; background:white }
h1, h2    { color: blue }

/* koken */
.koken    { color: purple; background: snow}
.koken h1 { color: purple;}
.koken h2 { color: darkgreen;}
```

10 HET BOX-MODEL

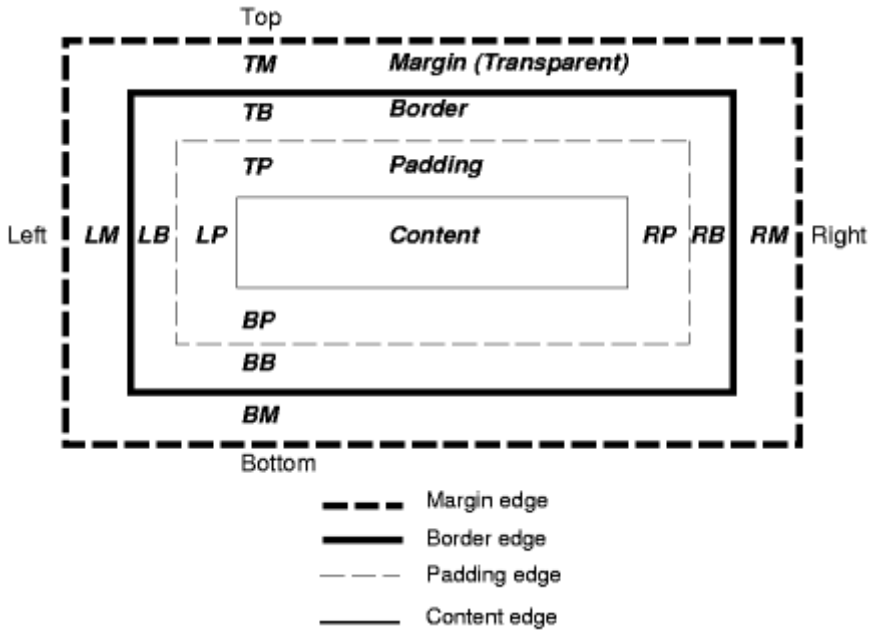
De CSS specificatie stelt dat elk HTML element in de document tree een box voorstelt.

Die box wordt opgemaakt volgens het *Visual Formatting Model* (VFM).

Elke **box** bestaat respectievelijk - van binnen naar buiten - uit

- een **content area**, de eigenlijke inhoud dus (tekst, figuur of meer elementen), omringd aan vier zijden door
- de **padding** zone (binnenruimte), die op zijn beurt omringd is door
- de **border** (rand), die zelf vervat is in
- de **margin** zone (marges).

Onderstaande figuur (ex W3C) stelt het **CSS2 box model** voor:



Elk element - dus ook een inline elementen zoals een `span` - heeft zo zijn eigen persoonlijke ruimte.

De totale dimensies die deze box inneemt kunnen dus berekend worden door alle maten op te tellen:

totale breedte =

```
margin-left + border-left + padding-left
+ width
+ padding-right + border-right + margin-right
```

totale hoogte =

```
margin-top + border-top + padding-top
+ height
+ padding-bottom + border- bottom + margin- bottom
```

Om een lay-out te maken moet je dus dikwijls berekeningen maken, ofwel in exacte maten zoals pixels, ofwel in relatieve eenheden zoals percentages of ems.

Een voorbeeld:

> Veronderstellen we dus een content area met een breedte van 100px, een hoogte van 50px, een algemene padding van 10px, een border van 2px en een algemene marge van 20px dan heeft die box een totale breedte van

```
20px + 2px + 10px + 100px + 10px + 2px + 20px = 164px
```

> en een totale hoogte van

```
20px + 2px + 10px + 50px + 10px + 2px + 20px = 114px
```

10.1.1 width berekenen

Een box zit bijna altijd in een andere container, je hoeft slechts de DOM structuur van een pagina bekijken om te zien welk element in een ander element zit. Zo is de buitenste container het `html` element dat zelf de `body` bevat. In de `body` zitten dan bv. een `header` en een `article` en een `footer` die zelf nog andere boxen kan bevatten. Al deze boxes volgen het box-model.

Meestal is dit geen probleem: een `div` in een andere `div` neemt de **maximale ruimte** in die hij kan krijgen. Dit komt omdat de standaardinstelling van `width:auto` is:

```
<div id="outer">
     <div id="inner">
     </div>
  </div>
```

met de volgende css:

```
#outer { margin:0; padding:0; border:none; width:300px}
```

dan heeft `#inner` een totale breedte van 300px beschikbaar en hij zal die dan ook **volledig innemen**, want `#inner` heeft geen – en dus de standaard instelling.

Een berekening zal nodig zijn als je bijvoorbeeld een aantal boxes naast elkaar in de beschikbare ruimte wil plaatsen.

Een voorbeeld schept meer duidelijkheid:

10.1.2 voorbeeld met absolute eenheden:

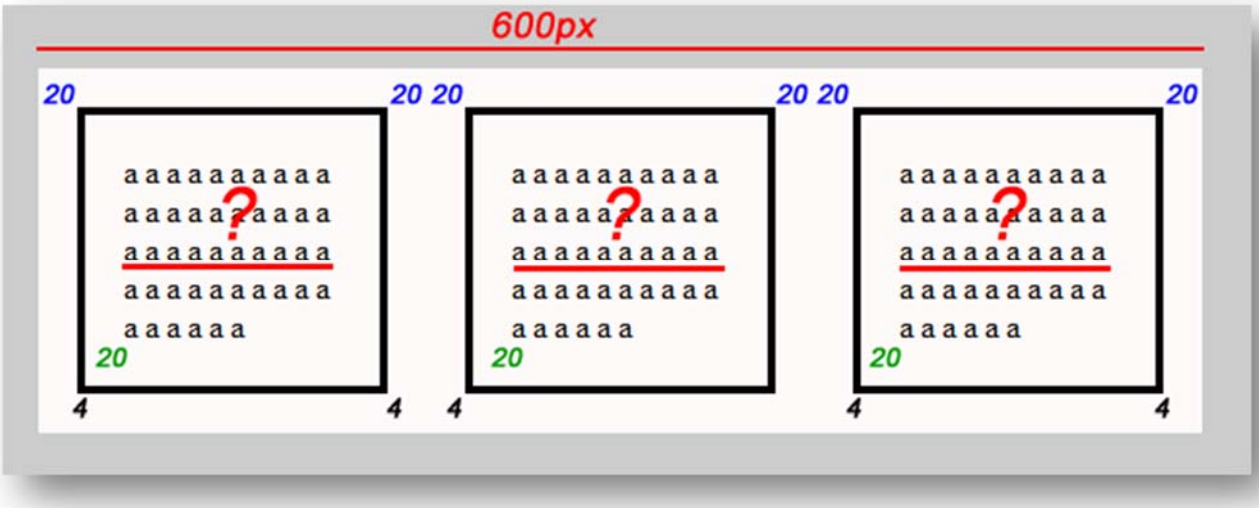
Veronderstel: je wil drie boxes **naast elkaar** inpassen in een andere container. Die 3 zijn *floats(zie verder)*: dat betekent dat ze opeen volgen en een volgende lijn nemen als er niet voldoende plaats is.

```
#outer {
```

```
  margin: 10px;
  padding: 0;
  width: 600px;
}
.vlot {
  float: left;
  margin: 20px;
  padding: 20px;
  border: 4px solid black;
  width: ?;
  height: 100px;
}
```

In ons voorbeeld is de `width` van de hoofdcontainer 600px.

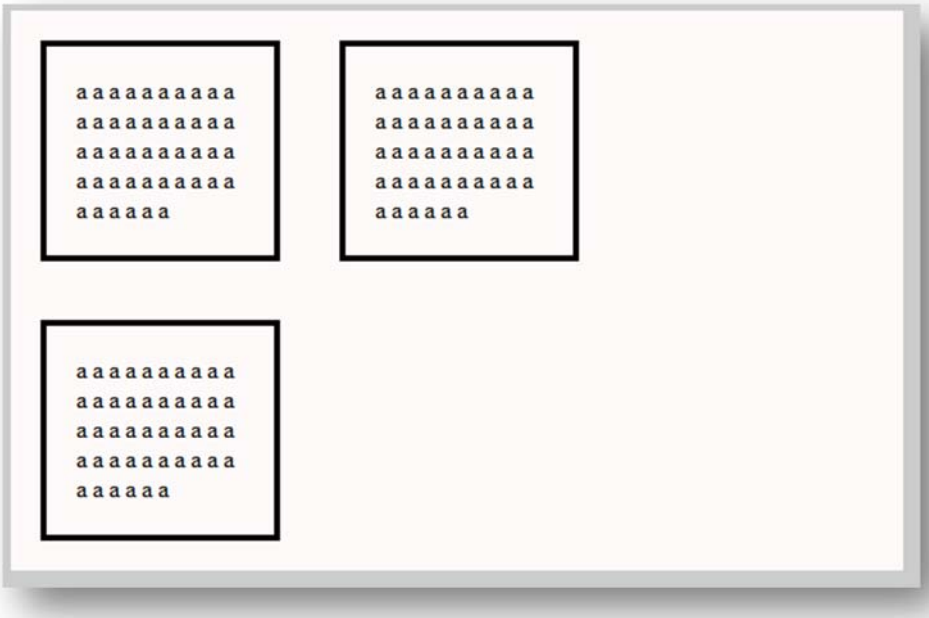
De *floats* (met class `. vlot`) moeten een `margin` van 20px hebben, een `border` van 4px en een `padding` van 20px. Hoeveel moet de `width` dan zijn om net binnen de hoofdcontainer naast elkaar te te passen?



Je moet er dus voor zorgen dat de totale dimensie maximaal 600px is:

```
width = (tot width/aantal) – (2*margin) – (2*border) – (2*padding)
width = (600px / 3) – (2 * 20px) – (2 * 4px) – (2 * 20px) = 112px
```

De `width` van een `.vlot` mag dus max 112px zijn om ze alle drie op een rij te krijgen. Komt het totaal meer uit dan krijg je dit:

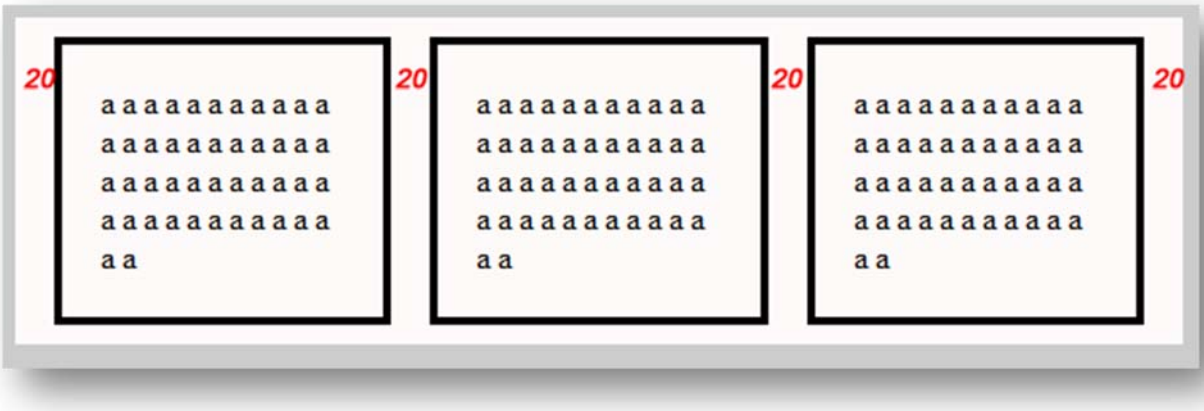


Bemerk ook dat de ruimte tussen twee *floats* uit een dubbele marge bestaat. Een webdesigner zal misschien willen dat de witruimte **tussen** de *floats* en die **links** en **rechts** identiek is: een zogenaamde gelijke *gutter*. Dat kunnen we o.a. oplossen door de margin te halveren voor alle floats en de eerste *float* een dubbele linkermarge te geven en de laatste een dubbele rechtermarge, bv:

Dan wordt de berekening wat ingewikkelder en kan je een rekenmachine gebruiken:

```
width = (tot width – ((dubbele linker en rechter marges) – (rest normale marges)) –(alle borders) – (alle padding))/aantal
```

```
width = (600px – ((2 * 20px) – (4 * 10px)) –(6 * 4px) – (6 * 20px))/aantal = 125.33333px
```




```
.vlot {
  float:left;
  margin: 10px;
  padding: 20px;
```

```
  border: 4px solid black;
  width: 125.3333px;
  height: 100px;
}
.vlot:first-child {margin-left:20px;}
.vlot:last-child {margin-right:20px;}
```

Nu hebben we onze egale *gutter*.

Laat je niet afschrikken door het decimale getal, halve pixels zijn inderdaad onmogelijk maar de browser zal afronden. Controleer wel of dat goed gebeurt in alle browsers, anders moet je bv 125px gebruiken.

10.1.3 voorbeeld met relatieve eenheden:

Gebruik je relatieve eenheden, `%` of `em`, dan kan de berekening problemen opleveren. En dat is zeker het geval als je absolute en relatieve eenheden mengt , iets wat veel voorkomt.

Veronderstel dat we de drie *floats* willen verdelen op een 1/3 basis in een *fluid* container (de container heeft zelf een relatieve breedte), maar met een absolute `border` (1px) dan kunnen we niet exact uitrekenen wat de `width` zal zijn, we kunnen enkel benaderen.

```
#outer{ width:60%; }
.vlot {
  float:left;
  margin: 2%;
  padding: 2%;
  border: 1px solid black;
  width: ? %;
  height: 100px;
}
```

Voor één *float*::

`1/3 = 33.33333% - margins - padding – border`

`33.3333% - (2 * 2%)– (2 * 2%) = 25.3333% - (2 * 1px) ≅ 25.1% ?`

De vraag rijst: hoeveel percent vertegenwoordigt 1 pixel? dat hangt natuurlijk af van de breedte van de hoofdcontainer en die kennen we ook niet met zekerheid.

Omdat de browsers de relatieve maten omzetten in pixels hangt het slagen af van a) de exacte breedte van de hoofdcontainer, b) welke browser er gebruikt wordt.

Als je dit uitprobeert op een gemaximaliseerd venster dan lukt het, verklein je het venster, dan plots niet meer. Een kleine veiligheidsmarge (25.0%) inbouwen is hier de boodschap.

11 HET VISUAL FORMATTING MODEL

Het ***Visual Formatting Model*** is het deel van CSS dat de document tree *lay-out* en daarna *toont* op een medium.
Daarbij moet het voor elk element heel wat beslissingen nemen:

- Het **type** box: is het een lijn of een blokje?
- De **looks**: letters, kleurtjes, randen etc…
- De **plaats**: waar moet het staan?
- De **laag**: indien ze overlappen, wie staat vooraan?
- De **zichtbaarheid**: is het te zien of niet? En zo ja, hoeveel ervan zie je?

Daar komen heel wat CSS eigenschappen bij te pas.

De standaard kan je vinden op het W3C als het basic box model.

11.1.1 Block elementen en inline elementen

Elk element wordt in een **box** geplaatst. Voor elke box moet er beslist worden of het

- een **block box** is die zich als een *doos* gedraagt en stapelbaar is
- een **inline box** die zich als *lijn* gedraagt

Dit hangt meestal af van het soort element, bv. een `p`, `section`, `div, p` en `li` element zijn per definitie **block** level elementen, terwijl een `span, code, b, i` en `em` **inline** zijn

Een block element kan andere block elementen of inline elementen bevatten.

Je kan echter via CSS het type wijzigen, en ook de normale plaats die dit element zou innemen in de **Flow**.

De **Flow** is de **natuurlijke volgorde** van de elementen zoals de parser ze tegenkomt in de HTML code: elke element heeft een plaatsje en neemt dit in volgens de html-code.

11.1.2 display

Met de `display` eigenschap kan je het soort box van een element bepalen.

Property	waarden
`display`	none | inline | block | inline-block | list-item | run-in | compact | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | ruby | ruby-base | ruby-base-group | ruby-text | ruby-text-group

Waarom zou je van een `div` - die normaal een block is, iets anders maken?

Goede vraag en er is ook zeer weinig reden om dat te doen, maar soms kan het handig zijn: om bijvoorbeeld een horizontaal menu te maken op basis van een lijst zullen we het type van de `li` en `a` elementen aanpassen (zie voorbeeld verderop).

Zoals je merkt aan het aantal waarden hierboven zijn er heel wat **types** boxen.
De voornaamste zijn:

- `none`: deze waarde zorgt ervoor dat het element **niet zichtbaar** is en dat het ook **geen plaats inneemt**
- `inline`: een gewoon inline element zoals een stukje tekst
- `block`: een element die zich gedraagt als een doos. Meerdere dozen stapelen van nature boven elkaar
- `inline-block`: een blok element die niet stapelt maar zich als een inline element naast een andere element plaatst.
  Er zijn van nature geen *inline-block* elementen, maar een block die een `float` krijgt wordt een inline-block genoemd
- `list-item`: een item van een `ul` of `ol`: een `li` element dus
- `table`: een tabel

De andere zullen we zelden nodig hebben.

Een voorbeeld: een `ul` lijst is ideaal als basis voor een menu, horizontaal of verticaal, omdat je er zo gemakkelijk een item kunt aan toevoegen of uit wegnemen:

de HTML:

```
<ul id="menu">
    <li><a href="home.html">home</a></li>
    <li><a href="wat.html">wat</a></li>
    <li><a href="contact.html">contact</a></li>
    <li><a href="about.html">about</a></li>
 </ul>
```

de CSS:

```
#menu {
  list-style-type:none;
  overflow:auto;
}
#menu li {
  display:inline-block;
  padding:1em;
  background-color:yellow;
  }
#menu li:hover {
  background-color:red;
  color:white;
}
#menu li a {
  display:inline;
  text-decoration:none;
}
```

Let vooral op

- het wegnemen van de bolletjes met `list-style-type`
- `display: inline-block` op de `li` elementen

11.1.3 background

Met background kan je een achtergrondkleur of beeld instellen

Property	waarden
`background-color`	kleurwaarde | transparent
`background-image`	URI | none
`background-repeat`	`repeat | repeat-x | repeat-y | no-repeat`
`background-attachment`	`scroll | fixed`
`background-position`	% % | lengte lengte | sleutelwoord of -combinaties van `top, center, left, right en bottom`
`background`	verkorte notatie

Vb: eenvoudige achtergrondkleur

```
#vb1 {
  margin:1em;
  padding:1em;
  background:#CCFF33;
}
```

Hier werd de verkorte versie gebruikt: `background-color:#CCFF33` is identiek

Vb: achtergrondfiguur, linksboven, niet herhaald.

```
#vb2 {
  margin:1em;
  padding:1em;
  width:400px;
  height:120px;
  background:transparent url(../images/pattern_140.gif) top left no-repeat;
}
```

ook hier werd de verkorte versie gebruikt. Onverkort wordt dat:

```
background-color:       transparent;
background-image:       url(../images/pattern_140.gif);
background-position:    top left;
background-repeat:      no-repeat;
```

```
}
```

11.1.4 Margin

Property	waarden
margin-top, margin-right, margin-bottom, margin-left	lengte | % | auto
margin	lengte | %| auto

De marges van een box kunnen apart of verkort ingegeven worden. Indien ze met `margin` verkort gegeven worden, kan dat als 1, 2, 3 of 4 waarden:

```
div {margin: 4px 6px 4px 8px;}
div {margin: 1em 2em 3em;}
div {margin: 0 auto;}
div {margin: 5%;}
```

4 waarden: wijzerzin vanaf top, right, bottom, left (**trbl**)

3 waarden: top, left=right en bottom

2 waarden: top=bottom en left=right

1 waarde: top=bottom=left=right

Horizontaal centreren:

Om een container met een ingestelde breedte**, horizontaal te centreren** binnen zijn parent, stel dan de linker- en rechtermarge in op `auto`

```
.center {
 width:500px;
 margin:10px auto;
 text-align:center;
}
```

Vertikaal centreren op deze manier is onmogelijk.

De `text-align:center` property in dit vb centreert de inhoud van de container, niet de container zelf

11.1.5 Margin-collapse

Het box-model werkt echter niet altijd als een optelling van `width, padding, border` en `margin`, soms gebeurt iets eigenaardig.

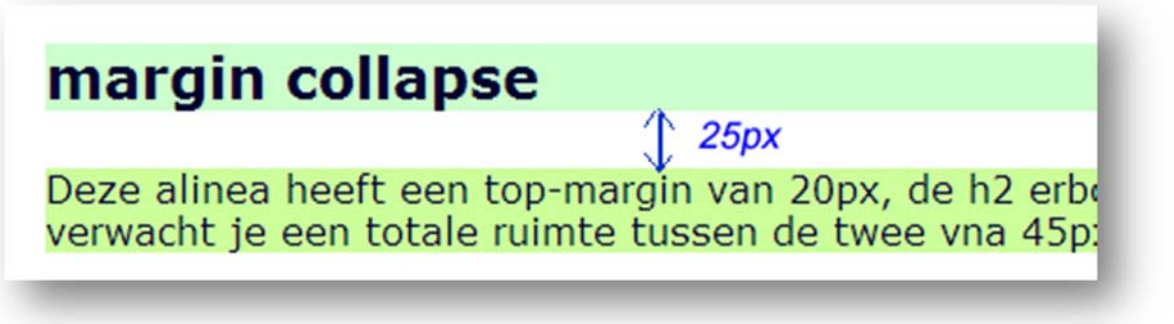
Veronderstel twee buurelementen: een `h2` en een `p` die niet leeg zijn en geen `border`, `padding` of `clear` tussen zich hebben:

```
<h2>margin collapse</h2>
<p>Deze alinea heeft een top-margin van 20px, de h2 erboven heeft een margin-bottom van
  25px, dan verwacht je een totale ruimte tussen de twee van 45px. Dit is niet zo: er is
  enkel 25px ruimte.</p>
```

met de CSS:

```
h2 {
  margin 25px 0 25px 0;
  background: #cfc;
}
p {
  margin: 20px 0 0 0;
  background: #cf9;
}
```

Theoretisch verwacht je (volgens het box-model) 45 px ruimte tussen de twee elementen. Dat is niet het geval: er is 25px ruimte tussen de twee.



Dit noemen we ***Collapsing margins***. Het gebeurt als **twee marges elkaar raken** zonder dat er een `padding` of `border` tussenzit.

Het wordt beschreven in de CSS standaard (en is dus geen bug):
http://www.w3.org/TR/CSS21/box.html#collapsing-margins



"De vertikale marges van twee buurelementen kunnen gecombineerd worden tot één *collapsed margin* die de hoogte heeft van de grootste van de twee waarden"

Als één van de twee margins negatief is dan worden ze wiskundig opgeteld, dus -5px en 20px wordt 15px.

In de meeste gevallen is *margin collapse* wat je wil. Het wordt bijvoorbeeld steeds toegepast tussen `li` elementen in een lijst en is ook wat de meeste auteurs verwachten tussen een titel en een volgende alinea.

Margin collapse zal NIET gebeuren in deze gevallen:

- in combinatie met het *root* element
- met *float* elementen
- met absoluut gepositioneerde elementen
- met `inline-block` elementen
- als de `overflow` property ingesteld op iets anders dan `visible` (de standaardwaarde)
- als de `clear` property ingesteld is op iets anders dan `none`
- horizontale marges combineren NOOIT

Margin collapse tussen parent en child

Maar soms gebeurt het waar je niet wil dat het gebeurt, bijvoorbeeld in het volgende, veel voorkomende "mysterie":

Veronderstel:

```
<p>tekst.</p>
<div>
  <h2>margin-collapse tussen parent en child</h2>
  <p>Biscuit candy marzipan sweet roll cotton candy lollipop. Jujubes liquorice sweet
  soufflé sweet carrot cake marzipan gummies ice cream. </p>
</div>
```

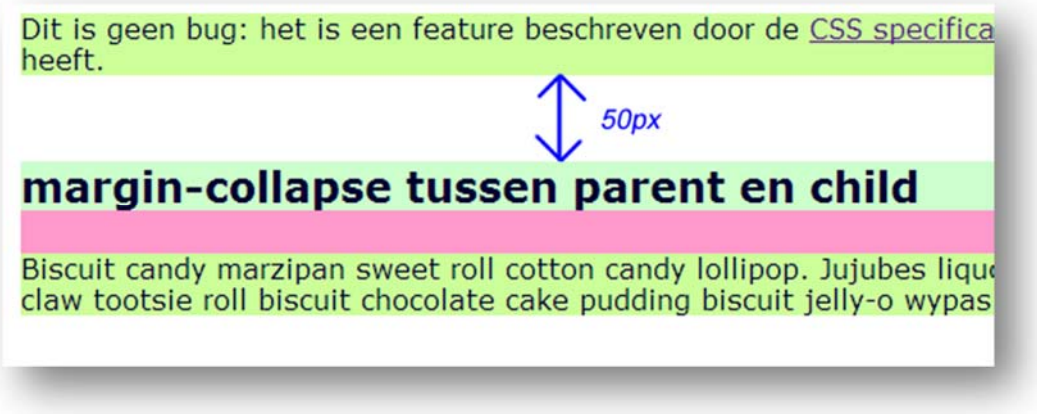
De `h2` is dus een *child* van de `div`

de CSS:

```
div {    background:#F9C; }
div p {
         margin: 20px 0 0 0;
         background: #cf9;
}
div h2 {
         margin: 50px 0 25px 0;
         background: #cfc;
}
```

Bemerk de margin-top van 50px op de `h2`.

Wat zou je verwachten? je zou een marge van 50px verwachten tussen de bovenrand van de `h2` en de rand van de `div`. Wat blijkt is dat de `div` 50px afstand neemt van de alinea ervoor:



De roze achtergrond van de `div` is niet te zien boven de `h2`.

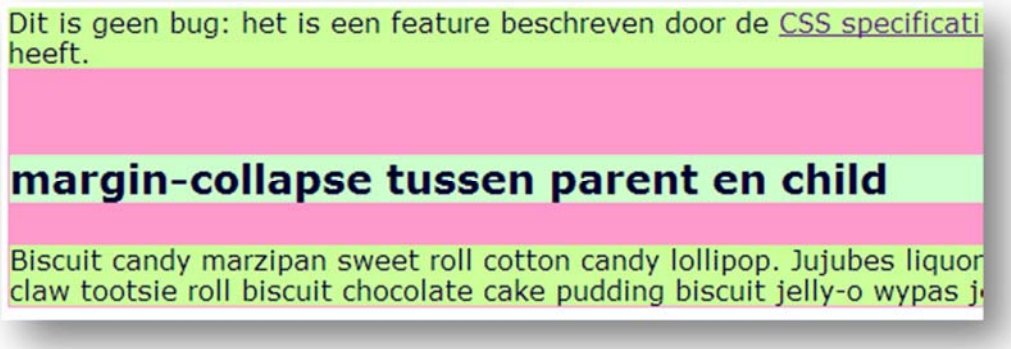
Dit is opnieuw een toepassing van een *collapsed margin*, maar wordt dikwijls gezien als een probleem want de ontwikkelaar wil eigenlijk ruimte scheppen tussen `h2` en de bovenrand van de `div`.

margin collapse vermijden

Er zijn twee eenvoudige "oplossingen" om dit te vermijden: plaats ofwel een `border` ofwel een `padding` op de *parent container*.

Als je de regels hierboven naleest merk je dat er door positionering nog meer oplossigen zijn, maar die kunnen niet altijd gebruikt worden.

```
div {
  background:#F9C;
  padding:1px;
}
div p {
  margin: 20px 0 0 0;
  background: #cf9;
}
div h2 {
  margin: 50px 0 25px 0;
  background: #cfc;
}
```



11.1.6 Border

Property	waarden
border-width	lengte | thin | medium | thick
border-style	none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset
border-color	kleurwaarde | transparent
border	verkorte notatie

Elk van deze eigenschappen kan gespecificeerd worden voor een aparte rand van de box, dus zijn ook border-top-width en border-right-color geldige properties.

Elk van deze eigenschappen kan ook geschreven worden met 1, 2, 3 of 4 waarden, volgens hetzelfde principe als margin.

```
p {border: thin dotted magenta}
div {
  border-width: 5px;
  border-style: groove dotted;
  border-color: red green blue;}
```

11.1.7 Padding

Property	waarden
padding-top, padding-right, padding-bottom, padding-left	lengte | %

padding	lengte | %

De instelling van de binnenruimte van een box is vergelijkbaar met die van de marges.

11.1.8 Content-area

De ***content-area*** van een box is de **echte inhoud** zonder enige padding, border of margin. De breedte ervan kan je instellen met width, de hoogte met height.

11.1.9 width, height

Bepalen de dimensies van de content-area van een box. width en height zijn niet van toepassing op inline elementen.

Property	waarden
width	lengte | % | auto
height	lengte | % | auto
min-width, min-height	lengte | %
max-width, max-height	lengte | % | none

De standaardwaarde van beiden is voor de meeste elementen auto, dat wil zeggen dat de inhoud de breedte/hoogte bepaalt. Ze kunnen dus een waarde 0 hebben als er niets inzit. Zo worden de dimensies van een img element bepaalt door de dimensies van het beeld die er in zit.

Voor andere elementen is de standaardwaarde van width 100%: een p element zal altijd de maximale breedte proberen in te nemen. Zijn height daarentegen is auto en zal bepaald worden door de hoeveelheid lijnen en de line-height.

Als een **percentage** gebruikt wordt is dat **altijd in verhouding tot de width van de parent container**. Indien de parent container de body is, is dat in verhouding tot het venster.

Dus veronderstel dat de div#outer een vaste breedte heeft van 600px:

```
<div id="outer">
  <div id="inner"></div>
</div>
```

en heeft div#inner de volgende css:

```
#outer {width:600px;}
#inner {width:50%; height:20%;}
```

dan is de breedte van div#inner 300px en de hoogte 120px.

11.1.10 min-width, min-height, max-width, max-height

Bepalen de minimale en maximale dimensies van de content-area van een box. Als een element een relatieve dimensie krijgt, zoals een percentage, of als door interactiviteit met de gebruiker zijn dimensies wijzigen, kan je een minimale breedte en hoogte vragen met `min-width` en `min-height`:

```
select {
  width: 60%;
  min-width: 200px;
}
```

`min-width` en `min-height` hebben altijd voorrang op repectievelijk `max-width, width` en `max-height, height.`

11.1.11 margin of padding gebruiken?

Wat gekozen, `margin` of `padding`? dat hangt een beetje af wat je wil. Veronderstel deze HTML:

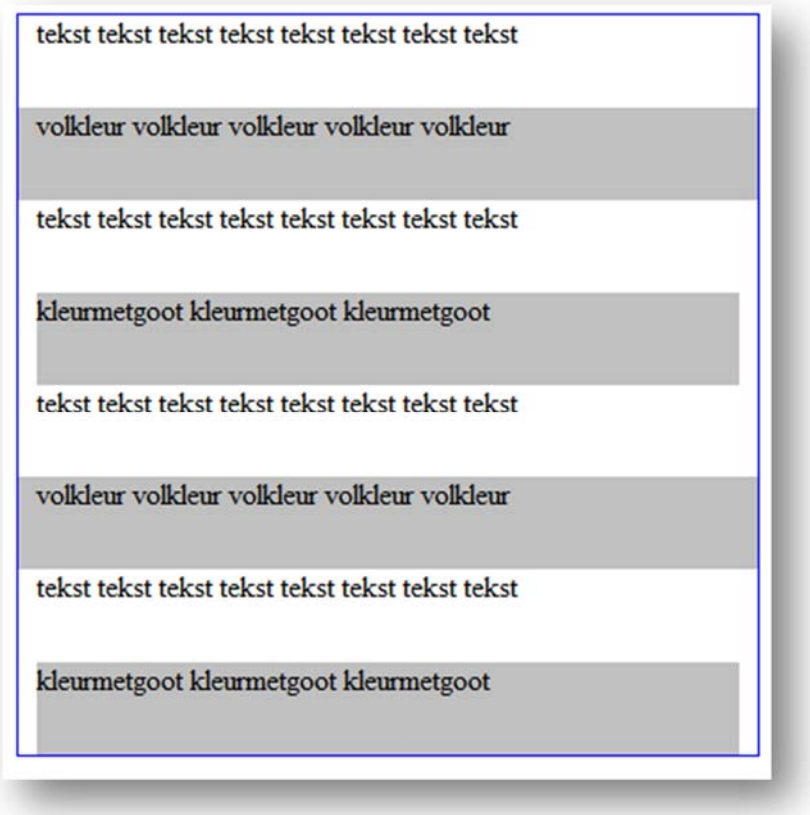
```
<div class=buiten>
  <div class="binnen tekst">tekst tekst tekst tekst tekst tekst tekst tekst</div>
  <div class="binnen volkleur">volkleur volkleur volkleur volkleur volkleur</div>
  <div class="binnen tekst">tekst tekst tekst tekst tekst tekst tekst tekst</div>
  <div class="binnen kleurmetgoot">kleurmetgoot kleurmetgoot kleurmetgoot </div>
  <div class="binnen tekst">tekst tekst tekst tekst tekst tekst tekst tekst</div>
  <div class="binnen volkleur">volkleur volkleur volkleur volkleur volkleur</div>
  <div class="binnen tekst">tekst tekst tekst tekst tekst tekst tekst tekst</div>
  <div class="binnen kleurmetgoot">kleurmetgoot kleurmetgoot kleurmetgoot </div>
</div>
```

met deze CSS:

```
div {
  margin:0;
  padding:0;
}
.buiten {
  border:1px solid blue;
  width: 400px;
  overflow:auto;
}
.binnen {
  width:auto;
  height:50px;
}
.tekst {
  padding:  0 10px;
}
.volkleur {
  background: silver;
```

```
  padding: 0 10px;
}
.kleurmetgoot {
  background: silver;
  margin: 0 10px;
}
```

dan zie je dit:



- we houden hier enkel rekening met de horizontale dimensie
- het klassieke box-model zegt dat de achtergrondkleur de content `(width)` + `padding` ruimte bedekt, een achtergrondkleur kan dus nooit de `margin` ruimte bedekken
- De `div.buiten` mag je geen `padding` geven als je wil dat een `div.binnen.volkleur` aansluit aan zijn `border`
- voor `div.binnen.tekst` kan je kiezen: `margin` of `padding` je ziet geen verschil
- voor `div.binnen.kleurmetgoot`, die niet mag aansluiten, gebruiken we een `margin`

11.2 Een box plaatsen

CSS plaatst een element op drie mogelijke manieren op een scherm of in een afdruk:

- in de **Normal Flow**
- met **Absolute Positionering**
- als een **Float**

11.2.1 de Normal Flow

Is wat je standaard krijgt: in de *Normal Flow* nemen boxes hun **logische** plaats in:

- volgt *el2* in de HTML code op *el1*, dan staat *el2* ook visueel na *el1op het medium (scherm, print,…)*.
  Met andere woorden, de voorstelling van de elementen volgt de **hiërarchie van de document tree**.
- HTML en CSS bepaalt of de box een **block** ofwel een **inline** box is en plaatst hem navenant

11.2.1.1 voorbeeld Normal Flow

Zoals je nu wel snapt hoef je daarvoor niets te doen, een block of inline box gaat uit zichzelf in de *Normal Flow* staan.

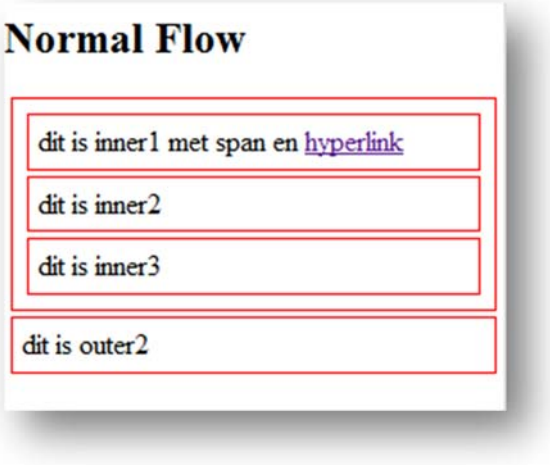
Nemen we als voorbeeld deze document tree:

```
<section>
<h2>Normal Flow</h2>
<div id="outer1">
<div id="inner1"> dit is inner1 met <span>span</span> en
                        <a href="http://www.vdab.be">hyperlink</a></div>
<div id="inner2"> dit is inner2 </div>
<div id="inner3"> dit is inner3 </div>
</div>
<div id="outer2">dit is outer2</div>
</section>
```

Met deze CSS

```
div {
  outline:       1px solid red;
  margin:        0.3em;
  padding:       0.3em;
}
```

ziet dat er zo uit in de browser



de `div` elementen zijn **block** elementen – nemen dus een nieuwe "lijn", de `span` en de `a` **inline** elementen.

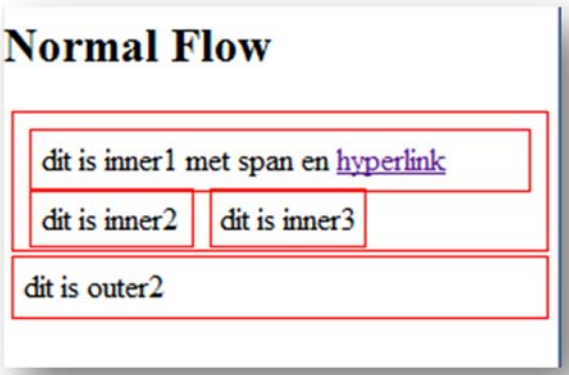
Een box die je na één of andere stylewijziging terug **expliciet normaal** wil zetten krijgt `position:static`. Dus dit is hetzelfde als niets zetten.

11.2.1.2 Block, inline of inline-block

Je kan ook het type van een block omzetten met de `display` property. De normale waarde van `display` voor een `div` is `block`, maar dat kunnen we wijzigen:

```
#inner2, #inner3 { display:inline; }
```

dan krijgen we

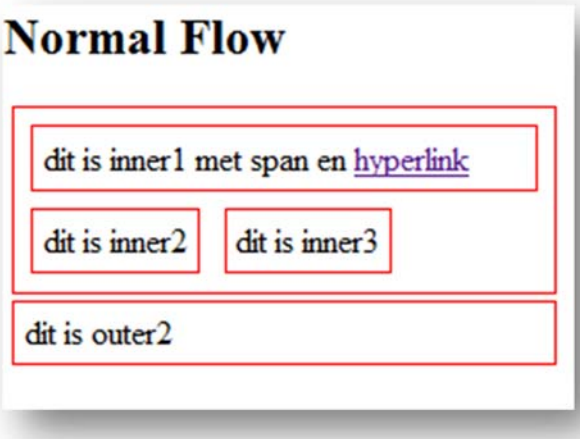


Nu gedragen `#inner2` en `#inner3` zich alsof ze stukjes tekst zijn binnen `#inner1`: ze nemen geen nieuwe lijn meer en ze volgen de `line-height` qua ruimte. Ze bevinden zich echter nog steeds in de *Normal flow*.

Een andere mogelijkheid is `display:inline-block`:

```
#inner2, #inner3 { display: inline-block; }
```

dan krijgen we



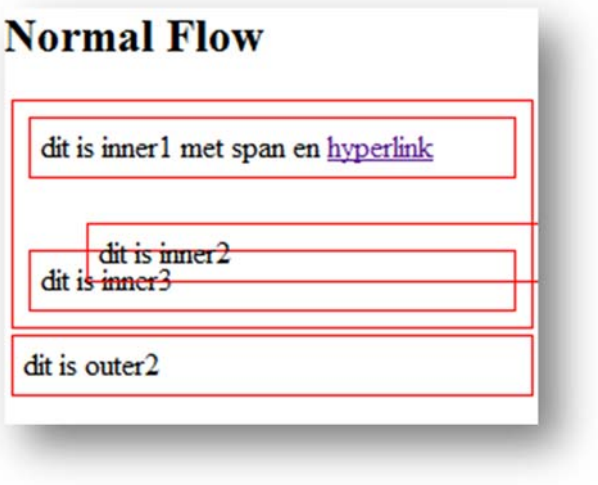
Ook hier gedragen de twee zich als een inline-element, met één verschil, ze nemen nu wel ruimte in als blokje, maar nemen geen nieuwe lijn.

11.2.2 Een relatief geplaatste box

Door `position:relative` in te stellen en één of meerdere van de eigenschappen `top`, `left`, `bottom` of `right` te gebruiken **verplaats je de box ten opzichte van zijn normale plek in het document**.

```
#inner2 {
  position: relative;
  top:20px;
  left:30px;
}
```

Deze CSS declaratie verplaatst de `div#inner2` ten opzichte van zijn normale plaats 20 pixels naar onder en 30 pixels naar rechts.



De box overlapt dus de inhoud eronder. **Let er echter op dat de ruimte die hij normaal inneemt niet gesloten wordt**.

De `div#inner2` bevindt zich dus nog steeds in de *Normal flow* en zal dus mee verhuizen als er meer inhoud ingevoegd wordt of als zijn `div#outer1` container verplaatst wordt.

Nog wat opmerkingen:

- Negatieve waarden voor `top`, `left`, `bottom` en `right` zijn toegelaten.
  `top:-20px` verschuift de box naar boven
  `left:-30px` verschuift de box naar links
- Het is ook perfect mogelijk een box zó ver buiten het scherm te plaatsen dat je het niet meer ziet. Dat kan gebruikt worden voor "*image replacement*"
- Gebruik relatieve positionering niet voor een in- of uitsprong, daarvoor is er `margin`

- `position:relative` **zonder** `top`, `left`, `bottom` of `right` wordt gebruikt om er een **containing block** van te maken: dit betekent dat hij als anker zal dienen voor een absoluut gepositioneerd child

11.2.3 Een absoluut geplaatste box

Met `position:absolute` licht je de box uit de *Normal Flow* en plaatst hem ten opzichte van zijn **containing block**.

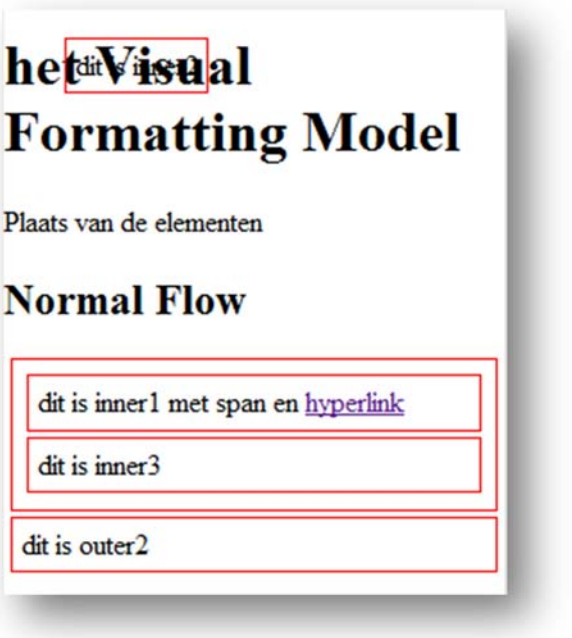
Wat is een *containing block*? dat is ofwel

- het **eerste parent element** dat zelf een `position` heeft
- de `body` (als er geen gepositioneerde parents zijn)

In ons voorbeeld, als we deze css toepassen:

```
#inner2 {
  position: absolute;
  top:20px;
  left:30px;
}
```

dan zien we

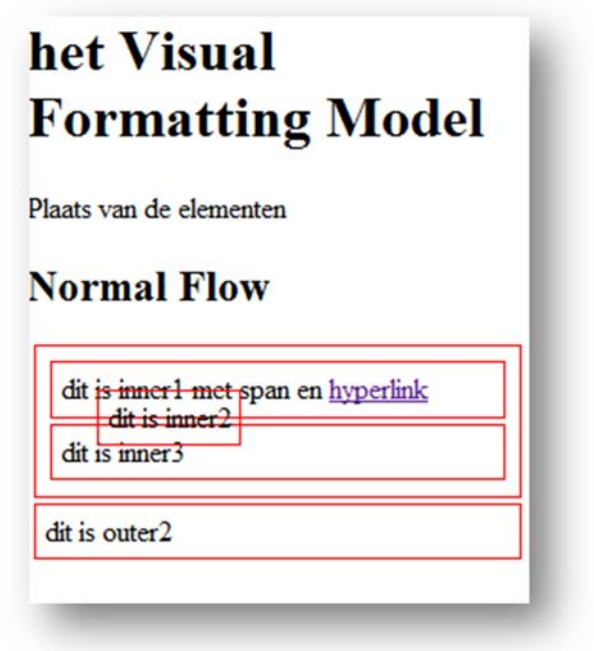


`#inner2` werd *uit de Normal Flow* gelicht (zijn oorspronkelijk ingenomen ruimte is verdwenen) en heeft zich geplaatst ten opzichte van de bovenrand van het **scherm** (de `body`).

Als we nu `#outer1` ook een `position` geven (eender welk type):

```
#outer1{position:relative;}
```

dan zien we dit:



`#inner2` werd *uit de Normal Flow* gelicht en heeft zich geplaatst ten opzichte van de bovenrand van zijn **gepositioneerde parent** `#outer1`.

Nog enkele bemerkingen:

- Een absoluut geplaatste box plaatst zich **bovenop** de rest van de document tree en oefent er verder geen invloed op uit: tekst vloeit er niet omheen
- De box bevindt zich niet meer in de Flow: zijn oorspronkelijke ruimte wordt gesloten
- Negatieve waarden zijn toegelaten
- een absoluut geplaatste box wordt automatisch zelf *containing block* voor zijn children
- met de eigenschap `z-index` kan je de **stacking-order** (volgorde van lagen) van meerdere gepositioneerde boxes bepalen
- grafische programma's noemen absoluut gepositioneerde boxes ***layers***

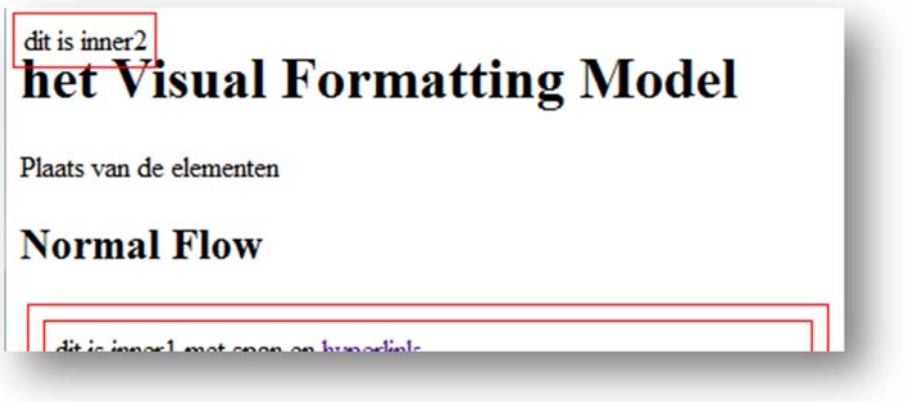
11.2.4 Een fixed geplaatste box

Met `position:fixed` licht je de box uit de *Normal Flow* en plaatst hem **ten opzichte van de viewport** (het venster zonder de toolbars).

Fixed positioning is in feite een subcategorie van absolute positioning waarin de **containing block steeds het browservenster** is. De rest van de inhoud *scrollt* eronderdoor.

```
#inner2 {
  position: fixed;
  top:0;
  left:0;
}
```

Geeft dit:



Wordt soms toegepast om menubalken continue op het scherm te houden.

```
.menubalk{
  position:fixed;
  width:100%;
  height:80px;
  top:0px;
  left:0px;
  }
```

11.2.5 Floats

De `float` **eigenschap** plaatst een **box links of rechts van de lijn waarop het zich bevindt**. Andere content vloeit er omheen. De box wordt uit de Normal Flow gelicht en neemt er dus ook geen plaats meer in in, maar is nog steeds gekoppeld aan zijn plaats in de code.

Een "*float*" gedraagt zich in feite net als een `img` element met zijn `align` attribuut ingesteld op `left/right`.

Floats worden zeer veel toegepast omdat ze erg handig zijn voor lay-out.

Een float verschilt van een absoluut of relatief gepositioneerd element op volgende vlakken:

- hij blijft gekoppeld aan zijn oorspronkelijke plek en aan zijn volgorde met andere elementen
- hij kan zich enkel links of rechts uitlijnen op de rand van zijn container, nergens anders.
- Meerdere floats creëren hun eigen flow.
  Bijvoorbeeld, 3 boxes in dezelfde parent die links ge-float worden, zullen elkaar netjes van links naar rechts volgen en de rest van de inhoud errond laten vloeien.
  ander voorbeeld: 10 boxes die ge-float worden volgen elkaar op in een continue slang in de beschikbare ruimte.

Mogelijke waarden zijn:

`left`	de Float lijnt links uit op de rand van zijn container
`right`	de Float lijnt rechts uit op de rand van zijn container

none	de box is geen Float

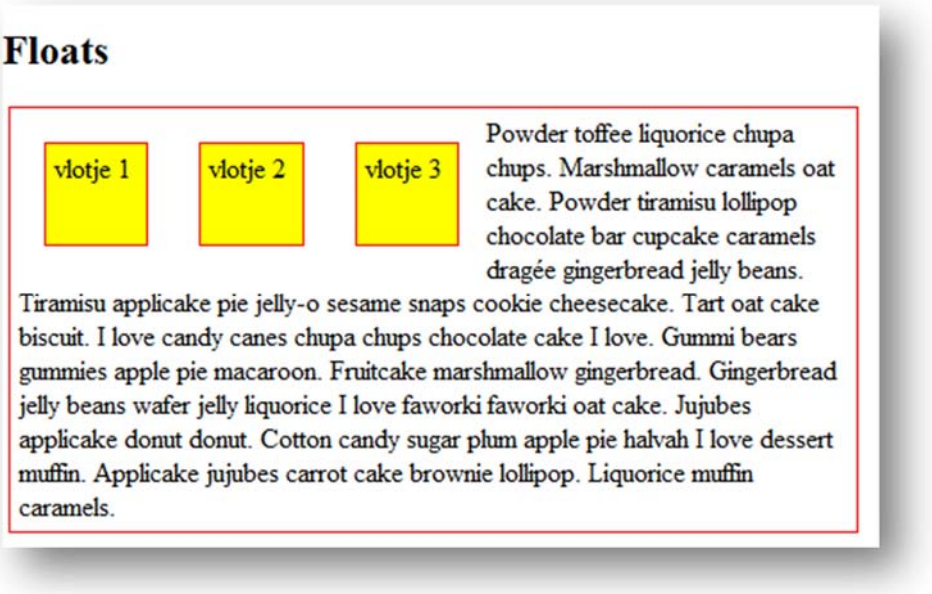
Voorbeeld1, de HTML:

```
<div id="outer1">
    <div class="vlot"> vlotje 1</div>
    <div class="vlot"> vlotje 2</div>
    <div class="vlot"> vlotje 3</div>
    Powder toffee liquorice chupa chups. Marshmallow caramels oat cake. Powder tiramisu
  lollipop chocolate bar cupcake caramels dragée gingerbread jelly beans. Tiramisu
  applicake pie jelly-o sesame snaps cookie cheesecake. Tart oat cake biscuit. I love
  candy canes chupa chups chocolate cake I love. Gummi bears gummies apple pie macaroon.
  Fruitcake marshmallow gingerbread. Gingerbread jelly beans wafer jelly liquorice I love
  faworki faworki oat cake. Jujubes applicake donut donut. Cotton candy sugar plum apple
  pie halvah I love dessert muffin. Applicake jujubes carrot cake brownie lollipop.
  Liquorice muffin caramels.
  </div>
```

met de volgende CSS:

```
#outer1 { width: 80%; }
.vlot {
  float:left;
  width:50px;
  height:50px;
  margin:1em;
  background:yellow;
}
```

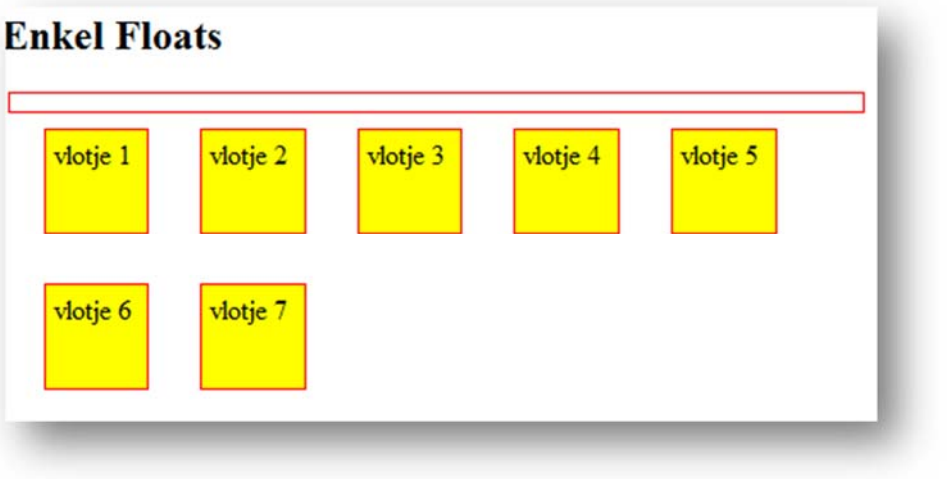
geeft:



Hier leven floats en andere inhoud samen in dezelfde container: je bemerkt de outline die volledig rond gaat: de `height` van `#outer1` is bepaald.

11.2.6 lay-out problemen met floats

In onderstaand voorbeeld zien we een container die **enkel** *floats* bevat:



Je ziet dat *floats* hun eigen *flow* maken: ze volgen elkaar netjes op en vloeien binnen de breedte van hun container.

Maar deze screenshot toont ook een fenomeen dat veel beginnende ontwerpers als een probleem ervaren: een ***collapsing parent***.

Omdat de `div#outer1` geen `height` meer heeft (want floats zitten niet in de *Normal Flow*) is zijn hoogte in principe 0! dat betekent dat elke rand of achtergrondkleur verdwijnt of enkel te zien is als een soort streep bovenaan… In osn voorbeeld hebben we het een beetje zichtbaar gemaakt.

Dit "probleem" heeft ook andere namen: "*clear space probleem*", het "*wasdraad fenomeen*", etc…

Het is belangrijk te beseffen dat dit eigenlijk een gewenst gedrag is in infeite geen "probleem"…

Als je toch wil dat de parentcontainer zich rond zijn inhoud "vouwt" (en dus zijn rand of achtergrond toont), zijn er een aantal snelle oplossingen:

- *float* de parentcontainer ook: plaats een `float:left` op de container
- plaats een `overflow:auto` op de container

Vooral deze laatste oplossing is goed voor het merendeel van de gevallen, maar er zijn uitzonderingen waar je het niet kan gebruiken.

Een 100% sluitende oplossing bestaat er uit een **clearFix hack** toe te passen: dat is een stuk CSS die dmv generated content een tabelcel genereert na de parent.

De **clearFix** (dank zij Nicholas Galagher op http://nicolasgallagher.com/micro-clearfix-hack)

```
/**
 * For modern browsers
 * 1. The space content is one way to avoid an Opera bug when the
 *    contenteditable attribute is included anywhere else in the document.
 *    Otherwise it causes space to appear at the top and bottom of elements
 *    that are clearfixed.
```

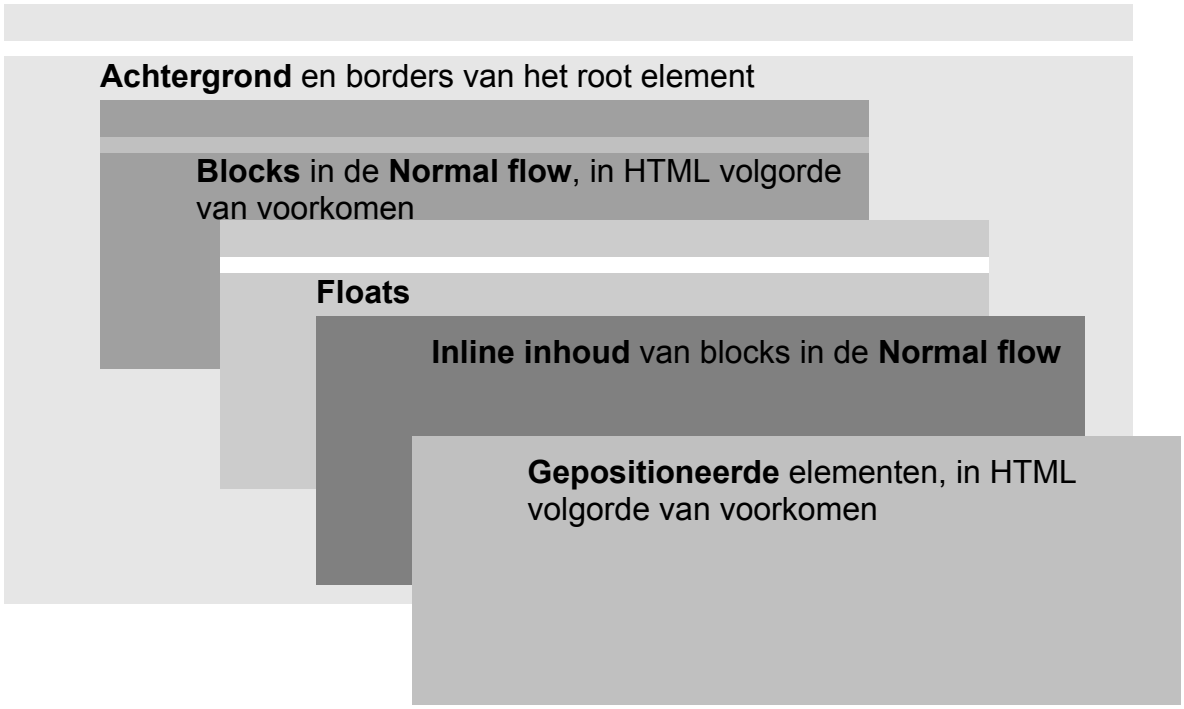
```
 * 2. The use of `table` rather than `block` is only necessary if using
 *    `:before` to contain the top-margins of child elements.
 */
.cf:before,
.cf:after {
    content: " "; /* 1 */
    display: table; /* 2 */
}

.cf:after {
    clear: both;
}

/**
 * For IE 6/7 only
 * Include this rule to trigger hasLayout and contain floats.
 */
.cf {
    *zoom: 1;
}
```

Om die nu te gebruiken pas je de `class` *cf* toe op de parent container.

In het project "Werken met floats" bespreken we alle mogelijkheden, problemen en oplossingen.

11.2.7 volgorde van lagen

De natuurlijke *stacking order* van lagen is als volgt:



Dat betekent dus dat floats **wel de inhoud** van een niet-gepositioneerd element beïnvloeden, **maar niet zijn achtergrond en randen**. Gepositioneerde boxes drijven boven alles en volgen hun eigen volgorde.

Voor **gepositioneerde lagen die elkaar overlappen** kan je die natuurlijke volgorde wijzigen met de eigenschap z-index.

z-index kan de waarden auto, inherit of een integer hebben. Het normale stacking niveau komt overeen met de integerwaarde 0. Een **hogere waarde** brengt de box **naar boven** - dus dichter bij de gebruiker.

Een containing block die een z-index toegewezen krijgt, creëert voor zijn child elements zijn eigen stacking order.

Enkele problemen:

- In IE6 vormen form controls, zoals een select element, een specifiek probleem omdat ze z-index volledig negeren en steeds boven drijven
- In FF beïnvloed opacity de natuurlijke stacking order met enkele rare effecten tot gevolg

11.3 Zichtbaarheid van de box

De eigenschappen `visibility`, `display`, `overflow` en `clip` bepalen hoeveel je van een box ziet.

11.3.1 Visibility, tonen of niet tonen

De `visibility` eigenschap bepaalt of een box zichtbaar is.

Mogelijke waarden zijn:

`visible`	de box is zichtbaar
`hidden`	de box is volledig transparant, dus onzichtbaar, maar neemt zijn rechtmatige plaats in
`collapse`	enkel toepasbaar op tabelkolommen en rijen, waar het de kolom/rij verbergt. Zie eigenschappen voor tabellen

Wordt vooral gebruikt in scripts, om boxes aan/uit te zetten.

Om een box **onzichtbaar** te maken kan je zowel `visibility:hidden` gebruiken als `display:none`. Er is wel een groot verschil:

- Met `visibility:hidden` wordt de box onzichtbaar, maar neemt hij nog steeds zijn normale ruimte in, met andere woorden, je ziet een lege ruimte.
- Bij `display:none` wordt de box uit de Flow gehaald en neemt dus geen ruimte in beslag, met andere woorden zijn plaats wordt ingenomen door het volgende element. Je zou kunnen stellen dat de HTML elementen in de head van het document zich zo gedragen.

De verschillen:

`visibility:hidden`	`display:none`
onzichtbaar	onzichtbaar
in Normal Flow	niet in Normal Flow
neemt ruimte in beslag	neemt geen ruimte in beslag
tegenpoolwaarde:	tegenpoolwaarde:
visible	meerdere mogelijkheden, zoals block, inline, list-item,…

```
<ol>
  <li>een list item</li>
  <li>onder dit item volgt een item met <code>display:none</code></li>
  <li style="display:none">een display:none list item</li>
  <li>onder dit item volgt een item met <code>visibility:hidden</code></li>
  <li style="visibility:hidden">een hidden list item</li>
  <li>een list item</li>
</ol>
```

Geeft het volgende resultaat:

1. een list item
2. onder dit item volgt een item met `display:none`
3. onder dit item volgt een item met `visibilty:hidden`

5. een list item

Beide eigenschappen worden intens gebruikt in JavaScript.

11.3.2 Overflow: kleine box, grote inhoud

De `overflow` eigenschap bepaalt hoe een te grote inhoud van een box behandeld wordt.

Mogelijke waarden zijn:

`visible`	de inhoud van de box wordt niet afgeknipt en kan dus buiten de box vloeien
`hidden`	de inhoud van de box wordt afgeknipt en er worden geen scrollbars getoond
`scroll`	de inhoud van de box wordt afgeknipt en de UA voorziet altijd scrollbars, ook al is de inhoud niet te groot
`auto`	de inhoud van de box wordt afgeknipt en de UA voorziet altijd scrollbars, ook al is de inhoud niet te groot

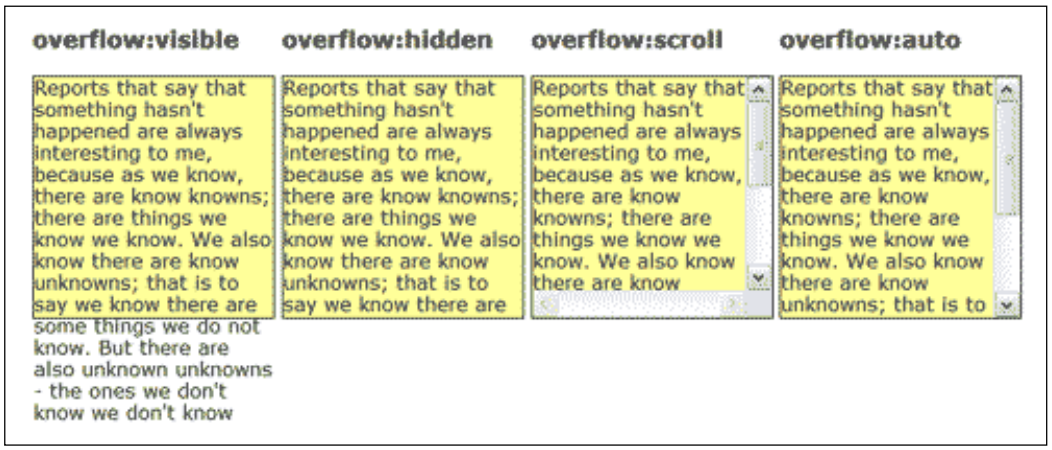
Wat gebeurt er als de box te klein is voor zijn inhoud?

Wat er met uitpuilende inhoud gebeurt, bepaal je met de eigenschap `overflow`.

Onderstaand voorbeeld illustreert de mogelijke waarden van overflow voor vier boxes waarvan de dimensies eigenlijk te krap zijn voor hun gelijke inhoud:

```
div {width:200px; height:200px; border:1px solid #000;}
div#overflow1 {overflow:visible;}
div#overflow2 {overflow:hidden;}
div#overflow3 {overflow:scroll;}
div#overflow4 {overflow:auto;}
…
<div id="overflow1">Reports that say that something hasnt happened are always
  interesting to me, because as we know, there are known knowns; there are things we know
  we know. We also know there are known unknowns; that is to say we know there are some
  things we do not know. But there are also unknown unknowns - the ones we dont know we
  dont know </div>
<div id="overflow2">…</div>
<div id="overflow3">…</div>
<div id="overflow4">…</div>
```

Het resultaat kan er als volgt uitzien:



De eerste box met `overflow:visible` **is de standaardsituatie**: de inhoud vloeit er uit. Met `overflow:hidden` verberg je alle uitpuilende inhoud zonder scrollbars te voorzien.

Het verschil tussen de waarden `scroll` en `auto` is te vinden in de scrollbars: bij de eerste waarde worden ze steeds getoond, bij auto enkel indien nodig.

11.3.3 Clip, een stukje box afknippen

Je kan ook besluiten slechts een deel van de zichtbare inhoud te tonen met `clip`.



Clipping is enkel toepasbaar op **absoluut** gepositioneerde boxes!

clip laat slechts twee mogelijke waarden toe (naast inherit uiteraard):

`auto`	**de inhoud wordt niet ge-clipped**
`rect()` functie	een rechthoekige regio bepaald door deze fucntie is zichtbaar, alle andere inhoud is onzichtbaar.

De syntax van de `rect()` functie:

```
clip: rect(top, right, bottom, left)
```

Waarbij deze vier argumenten een lengte of ook de waarde auto mogen hebben:

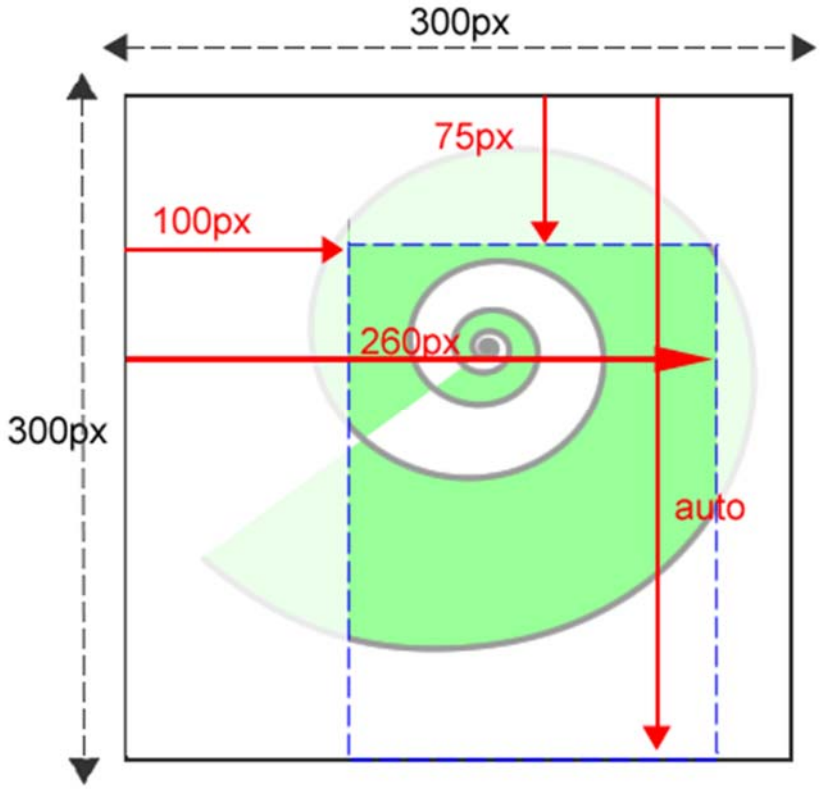
- `top` en `bottom` zijn offsets gerekend vanaf de bovenzijde van de bovenborder van de box
- `right` en `left` zijn offsets gerekend vanaf de linkerzijde van de linkerborder van de box
- de waarde `auto` voor één van de vier betekent dat de cliprand zal samenvallen met de werkelijke rand van de box
- negatieve waarden zijn toegelaten en hebben het effect van een margin

Eén voorbeeld is beter dan duizend woorden:

```
div#knip{
```

```
position:absolute;top:0;left:0;
height:300px;width:300px;
border:2px solid black; background:white url(spiral.png) no-repeat;
clip:rect(75px,260px,auto,100px);
}
```

Zal de `div#knip` op de volgende manier clippen:



Ondersteuning is OK in FF en O, maar IE6 en IE7 verkiezen een syntax zonder komma's: rect(75px 260px auto 100px) In het hoodstuk over hacks leer je hoe je een alternatieve stijlregel voor IE kan gebruiken.

Clipping wordt soms gebruikt in combinatie met JavaScript om een zoom-effect te krijgen.

12 CSS VOOR LIJSTEN

Geordende en ongeordende lijsten kunnen volledig opgemaakt worden met CSS, zelfs in die mate dat ze er uit gaan zien als tabellen of menus. Een navigatiebalk op basis van een lijst is zeer makkelijk aan te passen.

12.1 De geordende of ongeordende lijst

Een ongeordende of ogeordende ljst bestaat respectievelijk uit een `ul` of `ol` element en minstens één `li` element. Een `li` element kan een ander `ul` of `ol` element bevatten:

```
<ul>
 <li>Natuurverenigingen
  <ul>
  <li>Vogelbescherming Vlaanderen</li>
  <li>Bond beter Leefmilieu</li>
  <li>Natuurpunt</li>
  <li>WWF Vlaanderen</li>
  </ul>
 </li>
 <li>Vlaamse overheid
  <ul>
  <li>www.vdab.be</li>
  <li>www.vlaanderen.be</li>
  </ul>
 </li>
 <li>Federale overheid</li>
</ul>
```

Valideer altijd de structuur van geneste lijsten vooraleer je ze opmaakt!

Zowel `ul`, `ol` als `li` zijn blokelementen en beantwoorden aan het **box-model**, dus hebben ze een **padding, margin en border**.

- `margin` en `padding` van de `ul` controleren de ruimte rond en in dit element.
- `margin` en `padding` van de `li` elementen bepalen de ruimte rond elk list-item.
- `padding-left` van de `li` controleert de afstand tussen bullet en tekst

De standaardwaarden voor `margin` en `padding` verschillen tussen browsers. Een normalize stylesheet kan hier van pas komen.

Als we bijvoorbeeld de volgende CSS toepassen op bovenstaand voorbeeld,

```
ul {margin-top:20px;margin-bottom:20px;}
```

dan krijgen we meer ruimte boven en onder de geneste blokken:

- Natuurverenigingen
  - Vogelbescherming Vlaanderen
  - Bond beter Leefmilieu
  - Natuurpunt
  - WWF Vlaanderen
- Vlaamse overheid
  - www.vdab.be
  - www.vlaanderen.be
- Federale overheid

Ook met de list-items zelf kan je wat spelen. Vervang het vorige door:

```
li {margin: 15px 0 15px 0; padding-left:30px;}
```

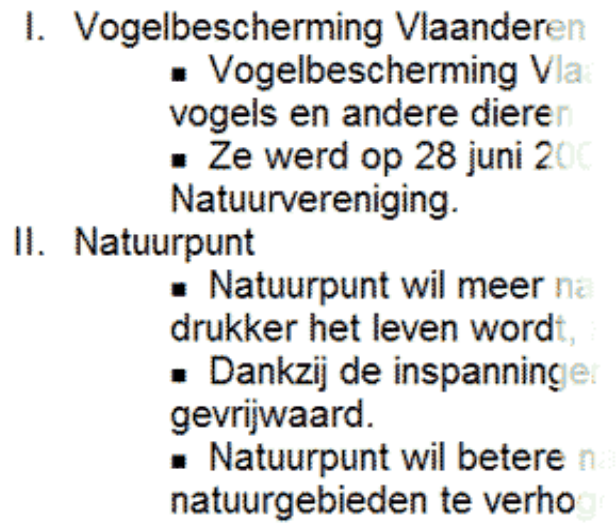
De margin van de `li` veroorzaakt nu meer ruimte tussen elk item. De `padding-left` vergroot de ruimte tussen bullet en tekst:

- Natuurverenigingen
  - Vogelbescherming Vlaanderen
  - Bond beter Leefmilieu
  - Natuurpunt
  - WWF Vlaanderen
- Vlaamse overheid
  - www.vdab.be
  - www.vlaanderen.be
- Federale overheid

De gekende `list-style` eigenschappen laten je toe het type bullet/nummering te bepalen en de plaats waar het staat t.o.v. het list-item block zelf:

```
ol {list-style-type:upper-roman}
ol ul {list-style:square inside}
```

Het voorbeeld vergt wat meer tekst:



12.2 Definition list

Eén van de meest ondergewaarderde HTML structuren, de definition list, bestaat uit een `dl` element die zelf `dt` en `dd` elementen bevat.

De definition list voorziet op die manier een eenvoudige, alternerende structuur, ideaal voor verklarende woordenlijsten e.d.

```
<dl>
 <dt>Vogelbescherming Vlaanderen</dt>
 <dd>Vogelbescherming Vlaanderen is een vereniging zonder winstgevend doel die zich
  inzet voor alle in het wild levende vogels en andere dieren…</dd>
 <dt>Bond Beter Leefmilieu</dt>
 <dd>Bond Beter Leefmilieu is de federatie van 120 natuur- en milieuorganisaties in
  Vlaanderen. BBL overkoepelt grote nationale verenigingen, regionale organisaties en
  tientallen lokale natuur- en milieuverenigingen …</dd>
 <dt>Natuurpunt</dt>
 <dd>Dit is sinds 1 januari 2002 een nieuwe natuurvereniging in Vlaanderen, geboren uit
  de fusie tussen twee bekende organisaties die zich al verschillende decennia inzetten
  voor het natuurbehoud in Vlaanderen, namelijk Natuurreservaten en De Wielewaal…</dd>
 <dt>WWF Vlaanderen</dt>
 <dd>Iedereen kent WWF (World Wide Fund For Nature) via zijn logo: de reuzenpanda. Maar
  WWF is zo veel meer ! Vandaag de dag is WWF de belangrijkste en meest ervaren
  onafhankelijke organisatie voor natuurbescherming ter wereld. …</dd>
</dl>
```

Het is niet noodzakelijk elk `dt` element te laten volgen door een `dd`: de spec zegt daarover niets en dus kan je meerdere `dt's` of `dd's` na elkaar plaatsen

Er zijn geen specifieke eigenschappen voor dit type lijst, maar je kan ze perfect stylen met box en tekst eigenschappen:

```
dt {
  margin:0; padding:18px 0 18px 60px;
  font-size: 1.2em;
```

```
  font-weight: bold;
  color: white;
  background-color: #CCCC66;}
dd {
  margin:0;
  padding:6px 0 6px 60px;
  font-style:italic;
  background-color:#BCC5DA;
  }
```

Het resultaat:

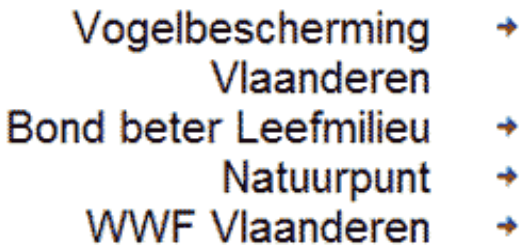


12.3 Background-image i.p.v. bullets

Elke box kan een `background-image` hebben, dus ook een `li` element.
Als die de normale list-style-type vervangt, krijg je wat meer flexibiliteit en kan je bijvoorbeeld rechtsuitgelijnde bullets maken. De exacte plaatsing van de achtergrond bepaal je dan met `background-position`:

```
ul {width:300px; text-align:right; list-style-type:none;}
li{
 background-image:url(externalblue.gif);
 background-repeat:no-repeat;
 background-position:100% 0.3em;
 padding:0 2em 0 0;
}
```

In dit voorbeeld werd de breedte van de lijst wat beperkt om meerdere lijnen te veroorzaken:



In een tweede voorbeeld voegen we een ikoontje toe aan de `dl` list van hierboven:

```
dt {
  margin:0; padding:18px 0 18px 60px;
  font-size:1.2em; font-weight:bold; color:white;
  background-color:#CCCC66;
  background-image:url(blad.gif);
  background-repeat:no-repeat;
  background-position:4px center;
}
dd {
  margin:0; padding:6px 0 6px 60px;
  font-style:italic;
  background-color:#BCC5DA;}
```



12.4 Lijsten worden menu's

Een menu wordt best gemaakt als een (unordered) list met hyperlinks. Vroeger werden die in een `table` of `div` structuur geplaatst om de nodige uitlijning te krijgen.

We vertrekken nu van een eenvoudige structuur met list-items die een hyperlink bevatten. In dit voorbeeld wordt voor het gemak de lijst in een nav element geplaatst:

```
<nav id=”menu”>
  <ul>
  <li><a href="#">Vogelbescherming Vlaanderen</a></li>
  <li><a href="#">Bond beter Leefmilieu</a></li>
  <li><a href="#">Natuurpunt</a></li>
  <li><a href="#">WWF Vlaanderen</a></li>
  </ul>
</nav>
```

Om een lijst om te vormen naar een vertikaal navigatiebalkje, volgen we de volgende strategie:

1. De lijst wordt beperkt in breedte en hoogte, afhankelijk van zijn horizontale of vertikale oriëntatie
2. De list-items worden ingesteld als *inline* element
3. Indien vertikaal, nemen de hyperlinks de rol van *block* element over

Alle verdere CSS heeft te maken met opmaak:

```
#menu{
  width:265px;
  margin:0; padding:0;
  background-color:#fff;
}
#menu ul{
  margin:0; padding:0;
}
#menu li{
  display:inline;
  list-style-type:none;
  margin:0; padding:0;
}
#menu a{
  display:block;
  text-align:right;
  text-decoration:none; font-size:small; font-variant:small-caps;
  background-color:#99CC33; color:#660033;
  padding:2px 8px; margin:2px;
}
#menu a:hover{
  background-color:#339933; color:white;
}
```

Produceert in IE6:



Opmerkingen:

- De lijst in een containing block (`nav` of `div` of iets anders) plaatsen, heeft enkele voordelen, maar is niet echt essentiëel, je kan evengoed gebruik maken van het `ul`, `ol` of `dl` element zelf.
- Het instellen van de hyperlinks als block doen we uiteraard om het menu vertikaal te maken. Dat heeft ook als gevolg dat *het volledige block aanklikbaar wordt!*
- De `padding:0` van de `ul` dient om de ruimte aan de linkerkant - voorzien voor de bullet van `li`'s - te verwijderen en de achtergrond de volledige interne ruimte te geven

- Andere padding en margin instellingen hebben te maken met ruimte in en rond de hyperlinks. Dat doe je volgens eigen smaak, maar we raden je aan ze steeds expliciet in te stellen, ook al zijn ze nul.

Hetzelfde voorbeeld in een horizontale orientatie:

```
#menu{
  width:100%;
  margin:0; padding:0;
  background-color:#fff;
  overflow:hidden;
}
#menu ul{
  float:right;
  margin:0; padding:0;
  background-color:#99CC33;
}
#menu li{
  display:inline;
  list-style-type:none;
  margin:0; padding:0;
}
#menu a{
  display:inline;
  text-align:right;
  text-decoration:none; font-size:small; font-variant:small-caps;
  background-color:#99CC33; color:#660033;
  padding:2px 8px; margin:0;
}
#menu a:hover{
  background-color:#339933; color:white;
}
```

In FF:



Opmerkingen:

- Het containing block, de `nav`, wordt nu op de volledige breedte van de pagina ingesteld
- Het `ul` element wordt hier rechts ge-float in de `nav`.
  Een gewone `text-align:right` in de container zou ook kunnen
- de `overflow:hidden` in de container heeft te maken met het gebruik van float: dit voorkomt het toeklappen van de container (*collapsing parent*).
- Voor een horizontale oriëntatie moet het `a` element weer `display:inline` hebben. Maar omdat het van nature inline is, kan je die lijn ook weglaten

13 CSS VOOR TABELLEN



Gebruik tabellen voor gegevens, niet voor lay-out.

Alhoewel een formulier een overzicht van gegevens is, kan je die veel beter opmaken zonder een tabel te gebruiken (zie CSS voor formulieren).

De web standaard (zowel XHTML als HTML5) laat nog steeds een aantal HTML attributen toe die dienen voor opmaak, *nota bene* het `border` attribuut (dat soms handig kan zijn om tijdens de ontwikeling de `table` zichtbaar te maken), toch maken wij een tabel volledig op met CSS.

Veronderstel dit voorbeeld:

```
<table id="vb1">
  <caption>Verkoopcijfers van Nadine's online shop</caption>
  <colgroup id="titels" />
  <colgroup span="2" id="midden"  />
  <colgroup id="totaal" />
  <thead>
    <tr>
      <th colspan="4">Nadine's vrolijke fruitmandje</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td colspan="3">Totaal</td>
      <td>651.5</td>
    </tr>
  </tfoot>
  <tbody id="subkop">
    <tr>
      <th>Produkt</th>
      <th>Prijs</th>
      <th>Aantal</th>
      <th>Totaal</th>
    </tr>
  </tbody>
  <tbody>
    <tr>
      <th>Appels</th>
      <td>1,25</td>
      <td>2</td>
      <td>2,50</td>
    </tr>
    <tr>
      <th>Banaan</th>
      <td>2,25</td>
```

```
      <td>4</td>
      <td>9</td>
    </tr>
    <tr>
      <th>Mango</th>
      <td>10,40</td>
      <td>100</td>
      <td></td>
    </tr>
  </tbody>
</table>
```

13.1 Randen, kleuren en achtergronden

Voorbeeld1:

een dikke, rode, volle lijn rond de tabel, blauwe stippellijntjes rond de cellen en horizontale lijnen op de `tbody`, `thead`, `tfoot`:

```
table {
  border-collapse: collapse;
  empty-cells:show;
  border: 2px double red;
}
th, td { border:1px dotted blue;}
thead, tfoot, tbody {
  border-top: 2px solid gray;
  border-bottom: 2px solid gray;
}
```

Verkoopcijfers van Nadine's online shop

Nadine's vrolijke fruitmandje			
Produkt	**Prijs**	**Aantal**	**Totaal**
Appels	1,25	2	2,50
Banaan	2,25	4	9
Mango	10,40	100	
Totaal			651.5

De eigenschap `border-collapse:collapse` van `table` versmelt de randen van de cellen, die anders als individuele hokjes te zien zouden zijn.

Indien je `border-collapse:separate` gebruikt, krijg je dus gescheiden borders en kan je met `border-spacing` de ruimte tussen de randen bepalen.

De eigenschap `empty-cells:show` van `table` zorgt ervoor dat de randen van lege cellen te zien zijn. Sommige oudere browsers doen dat niet, wat vroeger opgevangen werd door er een ` ` karakter in te zetten. Dat is niet langer nodig.

Voorbeeld2:

groen kader met groene lijnen tussen `thead`, `tfoot, tbody` en `colgroup`'s

```
table {
  border-collapse: collapse;
  empty-cells:show;
  background-color:#FFC;
}
colgroup, tbody, thead, tfoot { border:1px solid green;}
```

Verkoopcijfers van Nadine's online shop

Nadine's vrolijke fruitmandje			
Produkt	**Prijs**	**Aantal**	**Totaal**
Appels	1,25	2	2,50
Banaan	2,25	4	9
Mango	10,40	100	
Totaal			651.5

Voorbeeld3:

```
table {
  border-collapse: separate;
  border-spacing: 0.2em 0.3em;
  empty-cells:show;
}
tbody tr:nth-child(even)      { background-color: #8D7DF5;}
tbody tr:nth-child(odd)       { background-color:#CDC7F5;}
thead, tfoot                  { background-color:#DB0058;color:white;}
tbody#subkop tr               { background-color:#FBFE00;}
```

In dit voorbeeld geen borders, maar door de `border-collapse:separate` en de eigfenschap `border-spacing` wordt er ruimte tussen de cellen gemaakt, waardoor het lijkt of er witte lijnen zijn.

de selectors `tbody tr:nth-child(even)` en `tbody tr:nth-child(odd)` creeren alternerende kleuren op de rijen.

In dit screenshot hebben we wat extra rijen gemaakt om te demonstreren:



Verkoopcijfers van Nadine's online shop

Nadine's vrolijke fruitmandje			
Produkt	**Prijs**	**Aantal**	**Totaal**
Appels	1,25	2	2,50
Banaan	2,25	4	9
Mango	10,40	100	
Appels	1,25	2	2,50
Banaan	2,25	4	9
Mango	10,40	100	
Totaal			651.5

13.2 Uitlijning en ruimte

Het vroegere HTML attribuut cell-padding vervangen we in CSS eenvoudig door padding op de cel-elementen td en th:

```
td, th { padding:2px 5px;}
```

Je kan een tabel, net als andere blok elementen, centreren in zijn containing block door linker-en rechtermarge op auto in te stellen:

```
table{
  border-collapse:collapse;
  empty-cells:show;
  margin: 0 auto;
}
```

De inhoud van de cellen kan horizontaal uitgelijnd worden met `text-align` en vertikaal met `vertical-align`:

```
tr {
  text-align:right;
  vertical-align:top;
}
```

Tabel-elementen zijn de enige containers waar vertikale uitlijning mogelijk is!

De `vertical-align` property bepaalt de vertikale plaats van een `table`-element.

waarde	beschrijving
`baseline`	uitlijning op de baseline van de inline box
`middle`	uitlijning op het vertikale middenpunt van de box
`sub`	uitlijning lager dan de baseline van de inline box. Heeft geen effect op de font-size
`super`	uitlijning hoger dan de baseline van de inline box. Heeft geen effect op de font-size
`text-top`	uitlijning langs de bovenrand van de tekstregel
`text-bottom`	uitlijning langs de onderrand van de tekstregel
`%`	uitlijning hoger (positieve waarde) of lager (negatieve waarde) van zijn normale baseline met de percentage waarde berekend op line-height waarde . 0% = baseline
`lengte-eenheid`	uitlijning hoger (positieve waarde) of lager (negatieve waarde) van zijn normale baseline met de gespecificeerde waarde
`top`	uitlijning langs de bovenrand van de box
`bottom`	uitlijning langs de onderrand van de box

13.2.1 Tabel-breedte

De properties `table-layout` en `width` bepalen de breedte van de tabel.

De mogelijke waarden voor `table-layout` zijn:

`auto`	(standaard instelling) de breedte van de tabel afhankelijk is van de berekende breedte van alle kolommen, dus van de inhoud.
`fixed`	de breedte van de tabel is vast, dus onafhankelijk van de inhoud.

Veronderstel dat de tabel een `width:300px` heeft en er wordt een figuur met een breedte van 400px ingeladen in een kolom, dan gebeurt het volgende in de twee gevallen:

- `table-layout:auto`
  de tabel wordt breder dan 300px. Hij past zich aan aan de inhoud.

  Nadeel: de `table` wordt gelay-out tijdens het laden van de inhoud: dat kan lang duren en het scherm verspringt voortdurend. Dit is één van de redenen waarom je geen tabellen voor lay-out moet gebruiken.

  Voordeel: de tabelbreedte is automatisch aangepast.

- `table-layout:fixed`
  de tabel blijft 300px breed. De figuur steekt ofwel uit of wordt ge-clipped.

  Voordeel: de tabel moet niet wachten op zijn inhoud: lay-out gebeurt zeer snel.

  Nadeel: de tabelbreedte is niet automatisch.



Een `table` met een `width:120px` en `table-layout:fixed`

13.2.2 Kolom-breedte met col en colgroup

Maak gebruik van `col` of `colgroup` elementen met de `width` property:
In ons voorbeeld kunnen we de breedte van de kolommen bepalen met:

```
table{
  table-lay-out:auto;
  border-collapse:collapse;
  empty-cells:show;
}
tr                              {text-align:right; vertical-align:top;}
td, th                          {padding:5px 10px;}
colgroup, tbody, thead, tfoot   {border:1px solid green;}
thead, tfoot                    {background-color:#66CC66;}
colgroup#titels                 {width:200px;}
colgroup#midden                 {width:100px;}
colgroup#totaal                 {width:50px;}
```

Verkoopcijfers van Nadine's online shop

Nadine's vrolijke fruitmandje			
Produkt	**Prijs**	**Aantal**	**Totaal**
Appels	1,25	2	2,50
Banaan	2,25	4	9
Mango	10,40	100	
		Totaal	651.5

13.2.3 De caption

Het `caption` element geeft wat meer **uitleg** over de inhoud van de tabel. Gebruik dit element of (als je de uitleg niet wil tonen) het `summary` attribuut van `table` om anders-validen vooraf duidelijk te maken wat de tabel inhoudt.

```
caption{
  width: auto;
  text-align: center;
  margin:inherit;
  color:#66CC66;
  font-style:italic;
}
```

Nadine's vrolijke fruitmandje			
Produkt	**Prijs**	**Aantal**	**Totaal**
Appels	1,25	2	2,50
Banaan	2,25	4	9
Mango	10,40	100	
		Totaal	651.5

Verkoopcijfers van Nadine's online shop

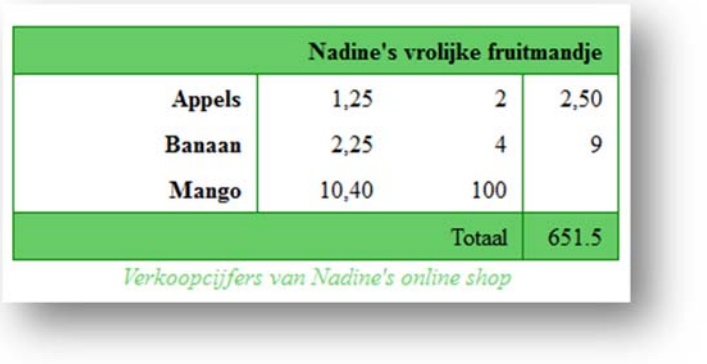
De eigenschap `caption-side` kan de waarden, `top`, `right`, `bottom`, `left` krijgen om de `caption` daar te plaatsen.

Bemerk de `margin:inherit` die werd bijgevoegd omdat de caption in FF een gecentreerde tabel anders niet volgt.

13.2.4 Dynamische effecten op rij en kolom

De CSS eigenschap `visibility` kan de waarde `collapse` krijgen voor rij of kolom elementen. Deze waarde zorgt ervoor dat de volledige rij of kolom uit het zicht verwijderd wordt en dat de ruimte ingenomen wordt door de buur-elementen:

```
tbody#subkop { visibility:collapse;}
```



	Nadine's vrolijke fruitmandje		
Appels	1,25	2	2,50
Banaan	2,25	4	9
Mango	10,40	100	
		Totaal	651.5

Verkoopcijfers van Nadine's online shop

In statische HTML ben je hier niet veel mee: het wordt vooral gebruikt in scripting om rijen dynamisch te tonen/verbergen.

14 CSS VOOR FORMULIEREN

Heldere en duidelijke formulieren zijn een breekpunt voor een interactieve website: de gebruiker moet weten wat hij kiest of invult, en dat moet ook gelden voor gebruikers met toegankelijkheidsproblemen.

14.1 Formulier en Formulier elementen

Het `form` element zelf heeft naast zijn interactieve functies ook de rol van *container* voor zijn controls. Dat betekent dat je het dimensies *kunt* geven en *kunt* zichtbaar maken met een achtergrond, randen, marges en padding. Meestal echter wordt de ruimte die het form krijgt overgelaten aan een andere container en neemt de form 100% breedte daarvan in. Dat laat ons toe de CSS voor forms onafhankelijk te houden.

Een `form` element hoeft dus niet noodzakelijk in een *containing block* zoals een `table` of een `div` te zitten.

Voor de *form controls* is de situatie soms onduidelijk: net zoals bij tabellen overlappen HTML attributen en CSS elkaar.

Er zijn nog steeds enkele HTML attributen die hoogte en breedte bepalen:

- `cols` en `rows` voor `textarea` zijn verplichte attributen

Hou er ook rekening mee dat voor `input`, die via zijn `type` attribuut zowel een invulveld, een radiobutton, een checkbox, een submit en een reset knop produceert, algemene CSS niet altijd kan. Attribuut selectoren laten je toe hier een nette scheiding te maken – op voorwaarde natuurlijk dat de browser attribuut selectoren ondersteunt:

```
input[type=submit], input[type=reset], input[type=button], button{
  padding:0.5em 1em;
  font-size:1.1em;
  color:#CC0033;
}
```

14.2 Form lay-out

Formulieren worden best gelay-out als een reeks containers die dienst doen als ***rij***, met daarin één of meerdere labels en de *controls*. `label` en *control* vormen dan een 'kolom'. Een voorbeeld structuur:

```
<form>
  <div><label>naam</label><input …/></div>
  <div><label>adres</label><input …/></div>
…
</form>
```

Je kunt daar veel variaties op maken, bijvoorbeeld door een een derde' kolom' aan toe te voegen voor foutboodschappen:

```
<form>
  <div><label>naam</label><input …/><label class="fout">verplicht veld!</label></div>
  <div><label>postnr</label><input …/><label class="fout">enkel nummers</label></div>
…
</form>
```

14.2.1 Het label element

Een `label` geeft een verklarende tekst voor een formulierelement zoals een tekstveld, radiobutton of checkbox , of kan ook foutberichten bevatten.

Het kan op twee manieren gekoppeld worden aan het element:

1. *impliciet*, door het formulierelement **in** het `label` te zetten
2. *expliciet*, door het met zijn `for` attribuut te koppelen aan een `id` van het formulierelement

Een voorbeeld:

```
<div>
  <label><input name="partnersexe" type="radio" value="m" />mannen</label>
  <label><input name="partnersexe" type="radio" value="v" />vrouwen</label>
  <label><input name="partnersexe" type="radio" value="b" />beide</label>
</div>
<div>
  <label for="voornaam">voornaam</label>
  <input type="text" name="voornaam" id="voornaam"/>
</div>
```

`label` elementen zijn in principe *inline elementen* (`display:inline`) en vormen dus geen box. Je kan dat natuurlijk wijzigen met `display`.

14.2.2 het form element zelf

Het `form` element zit meestal in een andere container die zijn breedte bepaalt; we stellen dus liever geen `width` op een `form` in, maar laten dat aan zijn parentcontainer over.

In ons voorbeeld veronderstellen we dat de breedte van het form beperkt wordt door zijn container (vb `40em`)

Buiten wat algemene styles geven we weinig mee:

```
form {
  font-size:1em;
  color: #333;
  margin:0;
  padding:0;
  outline:1px dashed red;
}
```

De outline dient enkel om het form zichtbaar te maken.

Box-sizing

om de lay-out sluitend te maken, gebruiken we het nieuwe `box-sizing:border-box` model (zie verder). Dat stellen we in voor alle elementen van het form:

```
form *{
  -webkit-box-sizing:border-box;
  -moz-box-sizing:border-box;
  box-sizing:border-box;
  }
```

14.2.3 de container-rijen

Zoals eerder gezegd gebruiken we een container als 'rij': dat kan een `div,` een `p` of nog iets anders zijn.

Daarin plaatsen we een `label` element dat geassocieerd is aan de formcontrol ernaast: `input`, `select`, `textarea`, etc…

```
…
<div>
   <label for="vnaam" class="verplicht" title="verplicht">naam:</label>
   <input type="text" id="naam" name="naam" placeholder="uw naam" title="vul hier uw
  naam in " required>
   <label class="fieldval" id="val_naam">vul uw naam in</label>
</div>
…
```

dan gebruiken we volgende CSS om dit als rij te lay-outen:

```
form div {
  position:      relative;
  clear:         left;
  display:       block;
  zoom:          1;
  margin:        0.2em 0;
  padding:       0;
}
/******* velden, controls *******/
input[type="email"],
input[type="password"],
input[type="text"],
input[type="url"],
input[type="search"],
input[type="number"],
input[type="date"],
input[type="tel"],
textarea {
  width:         19.7em;
  min-height:    1em;
  padding:       0.4em 0.5em;
  margin:        0;
```

```
  font-size:     inherit;
}

/****** labels en berichten *****/
label{
  /* algemeen, losse labels */
  float:         none;
  display:       inline;
  min-height:    0.875em;
  padding:       0.3em 0 ;
  margin:0;
  font-size:     1em;
  text-align:    left;
}

form div label {
  /* labels in div */
  float:         left;
  font-size:     1em;
  display:       inline-block;
  width:         10em;
  margin:        0;
  padding-right:1em;
  text-align:    right;

}
form div label.fieldval {
  /* veld gebonden berichten */
  display:       inline-block;
  float:         right;
  width:         10em;
  color:         red;
  font-style:    italic;
  text-align:    left;
  padding-left:  1em;
  }
```

Dit geeft:



In deze lay-out wordt de ruimte ongeveer ¼, ½, ¼ verdeeld `label` – `input` – `label`: 10em – 20em - 10em. Dit is mogelijk dank zij het `border-box` model.

Maar Chrome heeft een probleem en totaliseert niet correct, daarom zijn we verplicht slechts 19.7em te gebruiken.

Voor een kleiner input veld kunnen we een `class` maken:

```
<div>
 <label for="postnr" class="verplicht" >Postnummer:</label>
 <input type="number" id="postnr" name="postnr"  class="kort" title="het postnummer van
  uw gemeente"  required>
 <label class="fieldval" id="val_postnr"></label>
</div>
```

de CSS:

```
input.kort { width:7em;}
```

14.2.4 radiobuttons en checkboxes

Om radio buttons te lay-outen die naast elkaar staan gebruiken we de volgende html:

```
<div>
     <label>Geslacht:</label>
     <input  type="radio" id="vrouw" value="v" name="sexe" title="uw geslacht" />
     <label class="labelRadio" for="vrouw" >Vrouw</label>
     <input type="radio" id="man" value="m"  name="sexe"  />
     <label class="labelRadio" for="man" >Man</label>
     <span class="fieldval" id="val_sexe"></span>
</div>
```

De CSS voor de radiobuttons en checkboxes:

```
input[type="checkbox"], input[type="radio"] {
  padding: 0;
  margin: 0.4em 0.5em;
  }
```

Omdat we hier dikwijls tekst tussen de radiobuttons nodig hebben gebruiken we een speciale `class` *labelRadio* voor die labels:

```
form label.labelCheckbox, label.labelRadio {
  float: none;
  display: inline;
  zoom: 1;
  padding: 0;
  text-align: left;
}
```

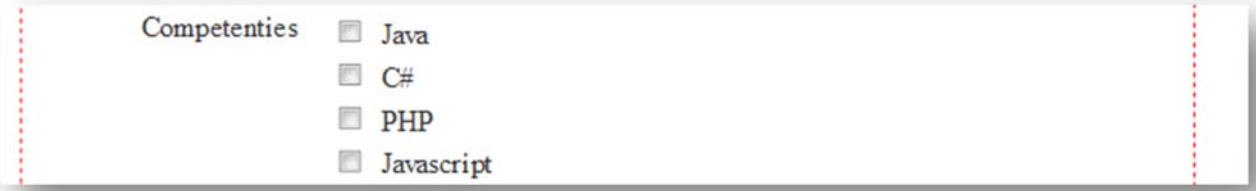
Deze `class` maakt de algemene *float* ongedaan en toont deze labels *inline*.

Het resultaat:



Deze "inline" lay-out kan je evengoed gebruiken voor checkboxes.

Soms wil je echter het volgende:



Dat maken we met de volgende html:

```
<div>
     <label>Competenties</label>
     <div class="controlbox">
          <input type="checkbox" id="java" value="java"  name="java"  />
          <label class="labelCheckbox" for="java" > Java </label><br />
          <input type="checkbox" id="C#" value="C#"  name="C#"  />
          <label class="labelCheckbox" for="C#" >C#</label><br />
          <input type="checkbox" id="php" value="php" name="php"  />
          <label class="labelCheckbox" for="php" >PHP</label><br />
          <input type="checkbox" id="js" value="js"  name="js"  />
          <label class="labelCheckbox" for="js" >Javascript</label>
     </div>
</div>
```

Hier gebruiken we een extra container met `class` *controlbox* om een goede lay-out te krijgen:

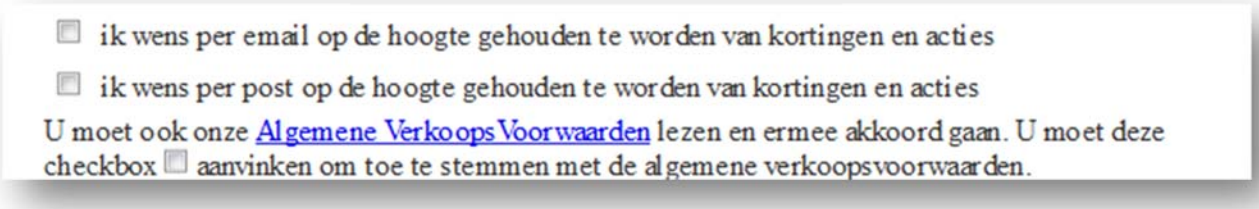
```
div.controlbox {
  width: 19.7em;
  display: inline-block;
}
```

Het is ook mogelijk dat je checkboxes wil hebben die het twee/drie-kolommen systeem niet gebruiken. Bijvoorbeeld:

```
<div>
      <label class="labelCheckbox">
        <input type="checkbox" id="actiesmail" value name="kortingen"  />
        ik wens per email op de hoogte gehouden te worden van kortingen en acties
         </label>
</div>
```

```
<div>
  <label class="labelCheckbox">
   <input type="checkbox" id="actiespost" value name="actiespost"  />
        ik wens per post op de hoogte gehouden te worden van kortingen en acties
  </label>
</div>
<div>U moet ook onze <a href="#">Algemene VerkoopsVoorwaarden</a> lezen en ermee akkoord
  gaan. U moet deze checkbox <input type="checkbox" class="inline" id="akkoord" value
  name="akkoord"  /> aanvinken om toe te stemmen met de algemene verkoopsvoorwaarden.
</div>
```

Dit geeft het volgende resultaat:



De `class` *inline* voor de checkbox neemt de `margin` weg:

```
input[type="checkbox"].inline, input[type="radio"].inline { margin:0; }
```

14.2.5 textarea

Een `textarea` verschilt van een `input` element in dat het een inhoud heeft: er *kan* tekst tussen de tags geplaatst worden. Hou er dan wel rekening mee dat de gebruiker die moet verwijderen, of als hij dat niet doet, dat je altijd die standaardtekst mee doorgestuurd krijgt. Daarom is een `placeholder` attribuut beter.

Een tweede verschil is dat je kan gebruik maken van de `rows` en `cols` attributen om hoogte en breedte te bepalen. Voor de breedte prefereren we CSS, voor de hoogte gebruiken we `rows`:

```
<div>
    <label for="vragen" >Hebt u nog vragen?</label>
    <textarea rows="8" id="vragen" name="vragen" placeholder="vragen?"
                                    title="Heeft u nog vragen?" ></textarea>
    <span class="fieldval" id="val_geboren"></span>
</div>
```

14.2.6 Knoppen

Er zijn in principe 3 soorten knoppen: `input type="submit"`, `input type="reset"` en `button` (met `type`'s *`submit`*, *`reset`* en *`button`*).

Knoppen hebben altijd een ingebouwde opmaak van de browser: een grijs kleurverloop en meestal ronde hoekjes.

In eerste instantie kunnen we de grootte wat aanpassen door bijvoorbeeld meer padding en een groter lettertype te gebruiken:

```
button, input[type="button"],input[type="submit"], input[type="reset"]{
  padding: 0.3em 1.3em;
  font-size: 1.1em;
}
```

Als we echter CSS3 properties gebruiken kunnen we er hele mooie dingen van maken, hier een voorbeeld overgenomen uit de CSS3 generator css3button.net:

```
.css3button {
  color: #ffffff;
  padding: 10px 20px;
  background: -moz-linear-gradient(  top,  #fff3db 0%,  #ffc821 25%,  #ff3c00);
  background: -webkit-gradient(  linear, left top, left bottom,  from(#fff3db),
                                          color-stop(0.25, #ffc821),  to(#ff3c00));
  -moz-border-radius: 6px;
  -webkit-border-radius: 6px;
  border-radius: 6px;
  border: 1px solid #b85f00;
  -moz-box-shadow: 0px 1px 3px rgba(000,000,000,0.5),
                       inset 0px -1px 0px rgba(255,255,255,0.7);
  -webkit-box-shadow: 0px 1px 3px rgba(000,000,000,0.5),
                       inset 0px -1px 0px rgba(255,255,255,0.7);
  box-shadow: 0px 1px 3px rgba(000,000,000,0.5),
                       inset 0px -1px 0px rgba(255,255,255,0.7);
  text-shadow: 0px -1px 1px rgba(181,050,181,0.2),
                       0px 1px 0px rgba(255,255,255,0.3);
}
```



14.2.7 Verplicht label

Soms moet je aanduiden dat een veld verplicht is. Een manier om dat te doen is opnieuw gebruik te maken van een `class` voor het label: *.verplicht*.

```
label.verplicht {
  position:relative;
  padding-right:1.2em;
  }
```

```
label.verplicht::before {
  content: "\2764";
  display: inline-block;
  position: absolute;
  top: 0.1em;
  right: 0em;
  color: magenta;
  padding-right: 0.1em;
}
```

Hier maken we dus gebruik van *generated content* om een hartje absoluut te plaatsen rechts van het label. Daarom moet het `label.verplicht` zelf relatief geplaatst zijn.

Het resultaat:



14.2.8 Gebruik de :focus

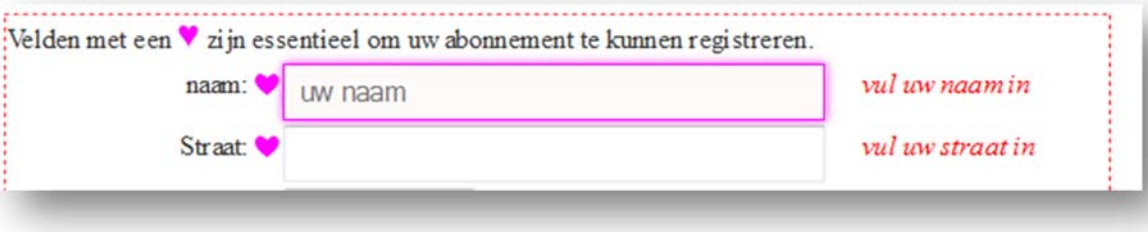
Je kan duidelijk de plaats aanduiden waar een gebruiker bezig is door de `:focus` pseudo-class te gebruiken:

```
input:focus,textarea:focus,select:focus  {
  border:1px solid magenta;
  background:snow;
}
```

of nog beter met wat "glow" erbij

```
input:focus,textarea:focus,select:focus  {
  border:1px solid magenta;
  background:snow;
  -webkit-box-shadow: 0 0 10px rgb(255,44,255);
  -moz-box-shadow: 0 0 5px rgb(255,44,255);
```

```
box-shadow: 0 0 5px rgb(255,44,255);
}
```



15 CSS VOOR PAGED MEDIA

Styles en stylesheets kunnen bedoeld zijn voor een bepaald medium met de `@media` rule of via het `media` attribuut.

Paged media (“print”, “projection”), zijn alle media waarvan het beeld onderbroken is in pagina’s, dus zowel papieren afdrukken als slides en pagina’s die op een scherm getoond worden.

15.1 Afdrukken

Een stylesheet voor het `media` type “`all`” of “`screen`” wordt in principe ook gebruikt voor de afdruk zolang er geen specifiek stylesheet voor “`print`” of “`projection`” geassocieerd is.

Denk er aan dat je ook een `@media` print rule in je gewone stylesheet kan zetten, een apart stylesheet is geen must:

```
<style type="text/CSS">
@media screen, print {
  h2 {
        padding:8px 6px;
  }
  table {
        table-lay-out:fixed;
        border-collapse:collapse;
        width:100%;
  }
  table, td {
        border:1px solid red;
  }
  div {
        border:1px dotted black;
        margin:10px;
        padding:19px;
  }
}
@media print {
  @page{
        margin:2cm 6cm;
  }
  body {
        background-color:white;
        color:black;
        font: 9px/15px Georgia, "Times New Roman", Times, serif;
        }
  h2 {
        background-color:black;
        color:white;
        border:1px solid gray;
        page-break-before:always;
```

```
  }
  h1+h2 {page-break-before:avoid;}
  div {
        border-color:gray;
        margin-left:30px;
  }
  table{
        page-break-inside:avoid;
  }
}
@media screen {
  body {
        background-color:#CCCCFF;
        color:#660066;
        font: 12px/19px Verdana, Arial, Helvetica, sans-serif;
  }
  h2 {
        background-color:white;
        border-bottom:2px ridge #660000;
  }
}
</style>
```

Je moet er echter wel mee rekening houden dat CSS een aantal zaken **niet** via het stylesheet kan instellen, deze kunnen enkel via de browser ingesteld worden:

- Het pagina-formaat
- Portrait/landscape oriëntatie
- Kop- en voettekst
- Wel/niet afdrukken van de achtergrondfiguren & kleuren
- Automatische compressie

Meestal kan je dat in een browser via het Page Setup menu instellen, soms moet je wat dieper in de opties zoeken.

De browser laat wel toe automatisch de title en het URL van de webpagina te gebruiken.

En opnieuw het gekende liedje: browsersupport is sporadisch en onbetrouwbaar…

15.1.1 De Page box

De af te drukken pagina wordt voorgesteld door de *Page Box*.

De *Page box* zelf bevat twee zones, nl.

- de page area en
- de margin area

De eigenschappen van een *Page box* worden ingesteld door de `@page` rule.

```
@page    { size: auto; margin: 2cm }
```

Je kan via de pseudo-classes `:left`, `:right` en `:first` andere waarden ingeven voor deze respectieve paginas

```
@page:left      { margin-left: 2cm ; margin-right: 4cm; }
@page:right     { margin-left: 4cm ; margin-right: 2cm; }
```

Bemerkingen:

- Ondersteuning voor het instellen van marges is zeer zwak
- De grootte van een Page box kan in CSS2.1 niet ingesteld worden

15.1.2 Page breaks

De eigenschappen `page-break-before`, `page-break-after` en `page-break-inside` bepalen paginasprongen in HTML elementen.

Ondersteuning door de browsers is matig: `page-break-inside` werkt niet.

Samengevat

Een aantal eigenschappen specifiek voor Paged Media: afdrukken, slides…

Property	waarden
`page-break-before`	auto | always | avoid | left | right
`page-break-after`	auto | always | avoid | left | right
`page-break-inside`	avoid | auto
`orphans`	integer
`widows`	integer

De waarden voor deze eigenschappen hebben de volgende betekenis:

waarde	beschrijving
`auto`	geen paginasprong plaatsen of vermijden. Standaardinstelling, dus net als in continuous media
`always`	altijd een paginasprong voor de box
`avoid`	vermijd een paginasprong voor de box
`left`	zet één of twee paginasprongen voor/na de box zodat deze op de linkerpagina staat
`right`	zet één of twee paginasprongen voor/na de box zodat deze op de rechterpagina staat

15.1.3 Orphans

De eigenschap `orphans` bepaalt het minimum aantal lijnen van een alinea dat aan de onderkant van de pagina aanwezig moet blijven. De standaardinstelling is 2.

15.1.4 Widows

Deze eigenschap bepaalt het minimum aantal lijnen van een alinea dat aan de bovenzijde van de pagina aanwezig moet blijven. De standaardinstelling is 2.

16 CSS GENERATED CONTENT

Met ***Generated Content*** bedoelen we het invoegen van inhoud die niet in de document tree staat - voor of na een element.



Ja, je hoort het goed: **CSS is in staat inhoud aan te maken!** beperkt weliswaar.

Een voorbeeld hiervan vind je ook in tekstverwerking als je hoofdstukken automatisch nummert.

Om een en ander mogelijk te maken hebben we minstens twee zaken nodig:

- een **pseudo-element selector** die bepaalt **waar** er ingevoegd moet worden en
- een **eigenschap** die bepaalt **wat** er ingevoegd moet worden.

Alle moderne browsers ondersteunen Generated content.

16.1 de ::before en ::after pseudoelementen

De `::before` en `::after` **pseudoelementen** specificeren **waar** er inhoud ingevoegd wordt. Ze zijn toepasbaar op alle andere selectoren:

```
p.note::before {
  content: "Opmerking: \A";
  }
```

Bemerk de syntax met **twee dubbelpunten** `::` (vroeger werd een enkel dubbelpunt gebruikt).

In dit voorbeeld wordt het woord "Opmerking: " op een nieuwe regel vóór de `p.note` ingevoegd.

16.2 content

De CSS eigenschap `content` is enkel toepasbaar op `::before` en `::after` en kan de volgende waarden toegewezen krijgen:

content: waarde	beschrijving
`normal`	er wordt geen content gegenereerd
`<string>`	een tekstuele inhoud
`<counter>`	een teller
`attr(X)`	geeft de waarde van het attribuut X die eventueel aanwezig is in de selector
`open-quote | close-quote | no-open-quote | no-close-quote ]`	voegen de respectieve aanhalingstekens in die gespecificeerd werden in de quotes property
inherit	erft van parent

Bemerkingen:

- Je kan **geen HTML elementen invoegen** met *Generated Content*, dus geen “`<p>yes</p>`”
- Je kan de selector wel **stylen** zodat de ingevoegde inhoud zich gedraagt als een HTML-element: zo kan je `display:block` gebruiken en marges, randen, padding, achtergrond, etc toevoegen
- *Generated Content* bevindt zich **niet in de document tree**. Je kan het dus niet aanspreken met DOM
- *Generated Content* heeft wel invloed op de lay-out van de pagina. Zo kan het gebruikt worden als hack om sommige lay-out problemen op te lossen (clearFix hack)
- In een `string` waarde kan je de **escape sequences** “\A” gebruiken om een nieuw regel te veroorzaken
- De functie `attr( )` kan gebruikt worden om de waarde van een **attribuut** van het geselecteerde element te tonen:

```
p.taal::before {
      content: "de taal van deze alinea is " attr(lang);
      display: block;
}
```

16.3 De eigenschap quotes

De `quotes` property laat je toe **twee sets tekens** te specificeren (gescheiden door spaties) die gebruikt kunnen worden in `content`.

Het eerste paar tekens dient voor de buitenste niveau van quotering, het tweede paar voor een verder binnenniveau. Deze tekens kunnen taal-afhankelijk zijn.

Hieronder zie je twee alternatieve sets van aanhalingstekens voor een element `q` in de engelse en nederlandse taal.

```
q:lang(en) { quotes: " " "" "" }
q:lang(nl) { quotes: "«" "»" " " }
```

deze sets kunnen dan toegepast worden als volgt:

```
q:before { content: open-quote }
q:after { content: close-quote }
```

en resulteren in een output zoals

“The crazy fox jumps over the lazy dog”

«De gekke koe springt over het luie konijn»

16.4 Automatische nummering

De properties `counter-increment` en `counter-reset` laten toe elementen **automatisch te nummeren**.

De waarde voor `counter-increment` moet een *identifier* zijn die voorkomt in `content`, eventueel gevolgd (en gescheiden door een spatie) door een *integer* die de stap voorstelt. Achterwege laten van deze integer veroorzaakt een verhoging van 1.

In onderstaand voorbeeld wordt voor het `h1` element een tekst “Hoofdstuk “ ingevoegd gevolgd door een teller die telkens met 1 zal verhogen. Terzelfdertijd wordt de identifier sectie terug op nul gezet.

Bij de elementen `h2` en `h3` gebeurt iets gelijkaardigs.

```
h1::before {
  content: "Hoofdstuk " counter(hoofdstuk) ". ";
  counter-increment: hoofdstuk;
  counter-reset: sectie;
  counter-reset: onderdeel;
}
h2::before {
  content: counter(hoofdstuk) "." counter(sectie) " ";
  counter-increment: sectie;
  counter-reset: onderdeel;
}
h3::before {
  content: counter(hoofdstuk) "." counter(sectie) "." counter(onderdeel) " ";
  counter-increment: onderdeel;
}
```

Elementen die genest worden veroorzaken een automatische reset. De volgende style-rules

```
ol { counter-reset: item }
li { display: block }
li::before {
  content: counter(item, ".");
  counter-increment: item
}
```

moeten een geneste nummering weergeven.

17 CSS3 FEATURES

17.1 browser ondersteuning

Net zoals met HTML5 moeten we met CSS3 features oppassen of de browser ze wel ondersteunt en een eventuele opvang voorzien.

Wat kan je doen?

- gebruik een **HTML5 shiv** of een boilerplate die de shiv bevat om afwezigheid van HTML5 elementen op te vangen
- gebruik **browserprefixes** voor CSS3 features
- gebruik een testing library zoals Modernizr om aan **feature detecting** te doen

Je kan dus zoveel CSS3 features gebruiken als je wil, **zorg er wel altijd voor dat je design ook werkt zonder CSS3!**

Browser-prefixes

Tijdens de ontwikkelingsfase van CSS3 zijn de verschillende bedrijven niet stil bijven zitten en hebben de nieuwe properties al getest. Ze deden dat door *eigen properties* te maken, met de bedoeling die bij de definitieve fase te laten vervallen. Die *test-properties* zijn samengesteld uit de naam van de **standaard property** met een **prefix** die de browser voorstelt:

```
-moz-     /* Firefox en andere browsers met Mozilla's engine */
-webkit-  /* Safari, Chrome en browsers die Webkit gebruiken */
-o-       /* Opera */
-ms-      /* Internet Explorer  */
```

Bijvoorbeeld voor de W3C property `transform` kan je dus schrijven:

```
-moz-transform:      rotate(45deg);
-webkit-transform:   rotate(45deg);
-o-transform:        rotate(45deg);
transform:           rotate(45deg);
```

- je weet dat een browser eigenschappen negeert die hij niet herkent, dus gaat hij alle lijnen af tot hij er een tegenkomt die hij kan toepassen.
- de laatste lijn is de standaard eigenschap, plaats die altijd laatst.
- meestal – maar niet altijd – is de waarde van de eigenschap gelijk voor elke browserprefix. Ga dit echter na, vooral in het geval van IE

Prefixes zijn natuurlijk een extra last: voor 1 property mag je 4 lijnen schrijven…

Prefixes zijn dus **tijdelijk**. Als blijkt – en in de komende jaren zal dat zo zijn – dat de browser-prefix niet langer nodig is, mag je die laten vallen en enkel de standaard property overhouden.

Dat is bijvoorbeeld reeds het geval met `border-radius` die door alle browser ondersteund wordt zonder prefix (of helemaal niet: IE7-8).

De website caniuse.com toont in zijn overzichten of een prefix nog nodig is of niet.

17.2 box-sizing

Het **CSS2.1 box model** stelt dat de `width` en `height` van een box de *content-area* bepaalt. Alle `padding`, `border` en `margin` moeten daarbij opgeteld worden:

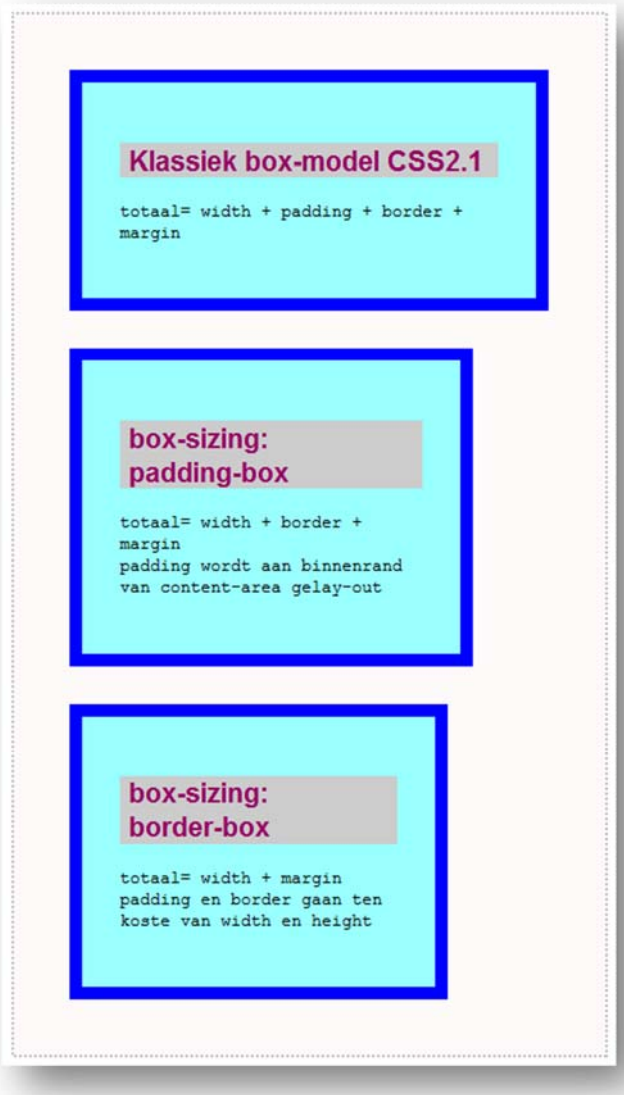
```
totale breedte = margin+border+padding+width+padding+border+margin
```

Alle huidige browsers passen dit model toe.

Het was enkel <IE6 in "quirks mode" die zich niet aan het box model hield en de `border` en `padding` **aftrok** van de `width` en de `height`. Sommige ontwikkelaars vonden dit echter logischer omdat je dan de **visuele** dimensie van een box bepaalt met `width` - margins zijn nooit zichtbaar.

CSS3 maakt het nu mogelijk de berekening van het box-model te wijzigen met `box-sizing`:

```
box-sizing: content-box | padding-box | border-box | inherit
```



- De standaardwaarde is `content-box`, m.a.w. het klassieke CSS2.1 box model: de `width` en `height` bepalen enkel de ***content-area***.
- `padding-box`: de `width` en `height` bepalen de ***padding-box*** van het element: de content-area wordt berekend door de `padding` af te trekken van `width` en `height`.
- `border-box`: de `width` en `height` bepalen de ***border-box*** van het element: de content-area wordt berekend door `border` en `padding` af te trekken van `width` en `height`. Dit komt overeen met het vroegere *IE6-quirks mode* model.
- `inherit` erft natuurlijk de waarde van de parent

De optie `box-sizing:border-box` maakt nu echter een serieuze come-back met responsive web design omdat het klassiek box-model het moeilijk maakt (en soms onmogelijk) om *pixel-perfect fitting* te bereiken als je met verschillende eenheden werkt.

Bijvoorbeeld:

```
<div class="container">
  <div class="binnen">linkerhelft</div>
  <div class=" binnen ">rechterhelft</div>
</div>
```

CSS:

```
div.container {
  width:80%;
  border:1em solid black;
}
div.binnen {
  width:48%;
  border:1em silver ridge;
  float:left;
}
```

De bedoeling is de twee binnen-`div'` s de container-`div` te laten verdelen in twee helften.

Perfectie is niet te bereiken omdat je niet weet wat de bereking zal uitkomen:

`(1em + 48% + 1em) * 2  ~= 100% ???????`

daar komt nog bij dat de `container.div` zelf een relatieve breedte heeft (80%) die misschien flexibel moet zijn want op een mobile site. Het resultaat *crashed* gemakkelijk:



Het probleem kan eenvoudigweg opgelost worden door de `box-sizing` van de binnen-`div's` anders te zetten:

```
div.container {
  width:80%;
  border:1em solid black;
}
div.binnen {
  box-sizing: border-box;
  width:50%;
  border:1em silver ridge;
  float:left;
}
```

Je gebruikt best browser-prefixes:

```
div.binnen {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
  width:50%;
  border:1em silver ridge;
  float:left;
}
```



De ondersteuning is nog niet helemaal in orde: Chrome en IE hebben een probleem met de waarde `padding-box.`

Firefox <16 had problemen in combinatie met `min-` en `max-width` en `-height.`

17.3 Gradients

Met een CSS3 gradient kan je een achtergrond een **kleurverloop** tussen twee of meer kleuren geven. Beeldbestanden (degradés) als achtergrond zijn hierdoor overbodig geworden.

De CSS Image Values and Replaced Content Module Level 3 beschrijft de standaard.

Er zijn twee soorten gradients: **lineair** en **radiaal**.

De standaard syntax is eenvoudig:

```
background: linear-gradient(hoek-in-deg | to hoek-of-zijde, kleurstop, kleurstop [,kleurstop]);
```

Alle gangbare CSS kleurspecificaties zijn toegelaten.

Hier heb je bijvoorbeeld een vertikale overgang van geel naar rood in verschillende mogelijkheden :

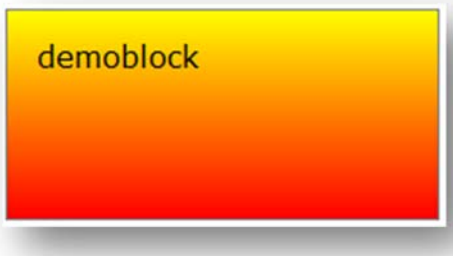
```
background: linear-gradient(yellow, red);
background: linear-gradient(to bottom, yellow, red);
background: linear-gradient(to top, red, yellow);
background: linear-gradient(180deg, yellow, red);
```

Je kan hierbij dus keywords `gebruiken to top, to bottom, to right, to left`, maar ook hoeken zoals `to top-right`,etc..

Vandaag wordt deze syntax door ongeveer de helft van de browsers ondersteund (Opera 12.1, IE 10, Fx 16), daarom is een prefix nodig voor *Webkit*:

```
#d1{
```

```
  background: -webkit-linear-gradient(top, yellow, red);
  background: linear-gradient(yellow, red);
  }
```



IE9< ondersteunen gradients helemaal niet.

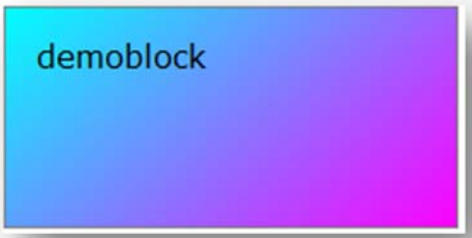
Een horizontaal kleurverloop:

```
#d2{
  background: -webkit-linear-gradient(left, yellow, red);
  background: linear-gradient(to right, yellow, red);
  }
```



Een kleurverloop met een hoek:

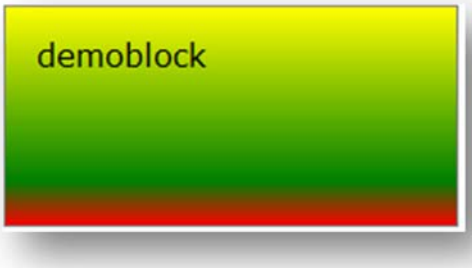
```
#d3{
  background: -webkit-linear-gradient(140deg, magenta,cyan);
  background: linear-gradient(140deg,cyan,magenta);
  }
```



Een kleurverloop met drie kleuren:

```
#d4{
```

```
  background: -webkit-linear-gradient( yellow, green 80%, red);
  background: linear-gradient(yellow, green 80%, red);
  }
```



Waarbij de tweede kleur een **kleurstop** heeft: dat betekent dat op dat punt de kleur volledig moet zijn, de andere punten worden automatisch berekend.

Bemerk dat in vele gevallen de syntax voor de webkit compleet het tegenovergestelde is van de standaard.

Radial gradients starten vanuit een middenpunt en strekken zich cirkelvormig uit.

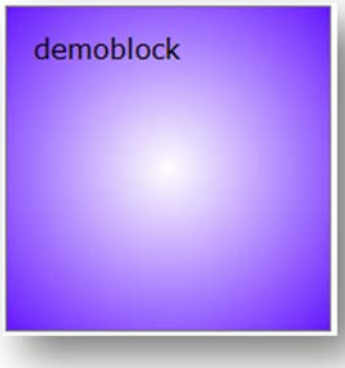
```
background: radial-gradient( [ [ vorm || grootte ] [ at <positie> ]?,
        | at <positie>, ]?  kleurstop [, kleurstop])
```

`vorm` kan `circle` of `ellipse` zijn. `grootte` kan `closest-side, farthest-side, closest-corner`, of `cfarthest-corner` zijn.

Enkele voorbeelden illustreren het principe.

Een eenvoudige cirkel:

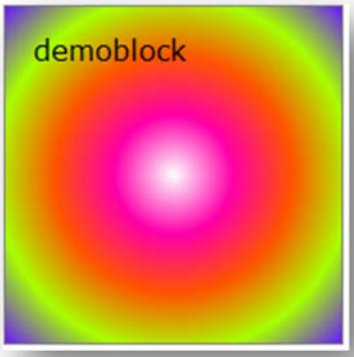
```
#d5{
  background: -webkit-radial-gradient( white, #6619FF);
  background: radial-gradient(white,#6619FF);
  }
```



Meerder kleuren waarvan één met kleurstop:

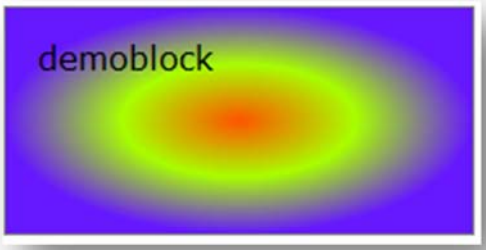
```
#d6{
  background: -webkit-radial-gradient( white,#FF00A9,#ff5500,#A9FF00 50%, #6619FF);
```

```
  background: radial-gradient(white,#FF00A9,#ff5500,#A9FF00,#6619FF);
  }
```



Een ellipse:

```
#d7{
  background: -webkit-radial-gradient(ellipse closest-side, #ff5500,#A9FF00, #6619FF);
  background: radial-gradient(ellipse closest-side, #ff5500,#A9FF00,#6619FF);
  }
```



Op het internet vind je verschillende *gradient generators*.

17.4 Transparantie

In moderne browsers kan je ook op twee manieren een kleur of een image transparant maken:

- met `opacity`
- door kleurtransparantie `RGBA` of `HSLA` te gebruiken

De eigenschap `opacity` bepaalt de doorzichtigheid van de achtergrond van een box. De waarde is een decimaal getal tussen 0 (volledig transparant) en 1 (volledig ondoorzichtig). De standaardwaarde is 1.

Dit gebeurt nadat andere tekst- en achtergrondeigenschappen zoals `color` , `background-color` en `background-image` toegepast zijn.

```
#t2{
```

```
border:6px solid red;
background-image:url(../img/cactus1.jpg);
opacity:0.4;
}
```



De `opacity` werkt op de volledige box: de achtergrond, de tekst en de border worden transparant.

Met kleurtransparantie kan je transparantie direct meegeven in de kleur. Er zijn twee mogelijkheden:

- de kleurruimte `RGBA`
- de kleurruimte `HSLA`

RGBA

Staat voor *Red Green Blue Alpha* en is identiek aan RGB maar met een extra decimale parameter "alpha" die de transparantie bepaalt.

Een RGBA kleur kan niet vertaald worden naar een Hexadecimale waarde.

```
background-color: rgba(178,24,255,0.6);
```

HSLA

Staat voor *Hue-Saturation-Lightness-Alpha* en is identiek aan HSL met een extra decimale parameter "alpha" die de transparantie bepaalt.

```
color: hsla(240, 100%, 50%, 0.5)
```

Met RGBA en HSLA kan je uiteraard aparte transparante kleuren zetten in tegenstelling met opacity.

Gradients ondersteunen RGBA en HSLA kleuren.

17.5 Backgrounds

De CSS Backgrounds and Borders Module Level 3 bepaalt de standaard.

Nieuw in CSS3:

- meerdere achtergrondbeelden mogelijk op hetzelfde element
- `background-origin` laat je bepalen waar je de achtergrond laat starten, niet enkel meer op de *padding-zone*
- `background-size` laat je de grootte van een background bepalen
- `background-clip` laat je een achtergrond bijknippen
- `background-repeat` heeft een aantal nieuwe waarden (maar geen nieuwe mogelijkheden)
- `background-attachment` heeft een nieuwe waarde

Multiple backgrounds

Eén van de beste verbeteringen is de mogelijkheid om meerdere achtergrondfiguren te plaatsen op hetzelfde element. Hoeveel maal heb je extra containers moeten toevoegen puur en alleen om in staat te zijn meerdere beelden op een achtergrond te zetten? Gedaan daarmee!

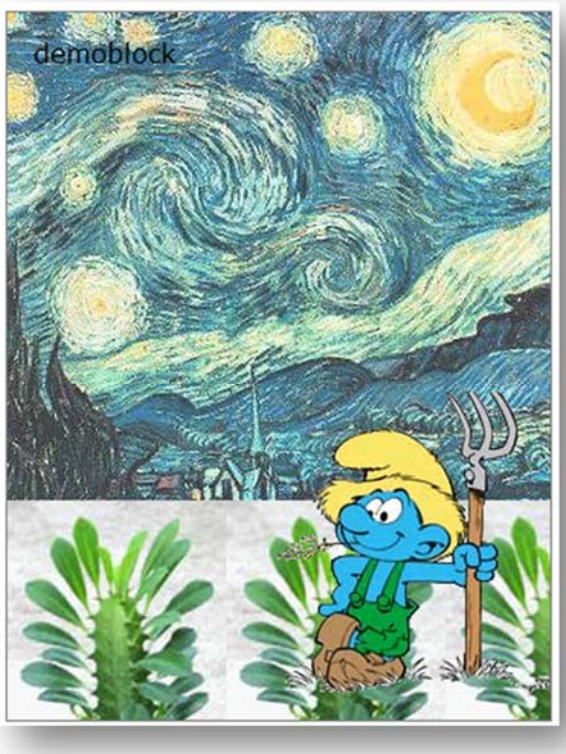
De syntax is verbazingwekkend eenvoudig: herhaal de syntax van een eenvoudige achtergrond, gescheiden door komma's:

```
#d1 {
  background-image: url(../img/boerensmurf.gif), url(../img/euphorbia.jpg), ↵
                                                  url(../img/van_gogh.jpg);
  background-position: bottom right, bottom right, top right;
  background-repeat: no-repeat,repeat-x,no-repeat;
}
```

is equivalent aan:

```
#d2 {
  background:           url(../img/boerensmurf.gif) bottom right no-repeat,
                        url(../img/euphorbia.jpg) bottom left repeat-x,
                        url(../img/van_gogh.jpg)top right no-repeat;
}
```

met dit resultaat:



Multiple backgrounds worden door bijna alle browsers ondersteund zonder prefixes.

background-origin

In CSS2 wordt een *background-image* altijd geplaatst in relatie met de hoeken van de padding-zone. Je kan het uitlijnen vanuit elke hoek (met `background-position`) ervan maar nooit de border of de margin gebruiken als punt van origine.

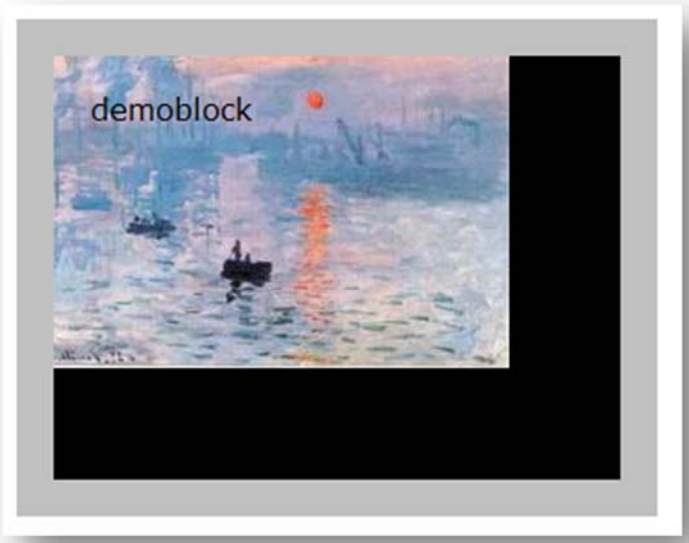
De eigenschap `background-origin` laat je toe dat wel te doen. Je kan er de waarden `padding-box` (standaard), `border-box, content-box` aan geven.

De volgende voorbeelden hebben allemaal dit gemeen :

```
.demoblock2 {
  width:                266px;
  height:               189px;
  padding:              20px;
  margin:               10px;
  border:               20px solid silver;
  background-color:     black;
  background-image:     url(../img/sunrise.jpg);
  background-repeat:    no-repeat;
}
```

en nu plaatsen we er de volgende CSS bij:

```
#d4 {
  background-origin: border-box;
  background-position:top left;
  }
```



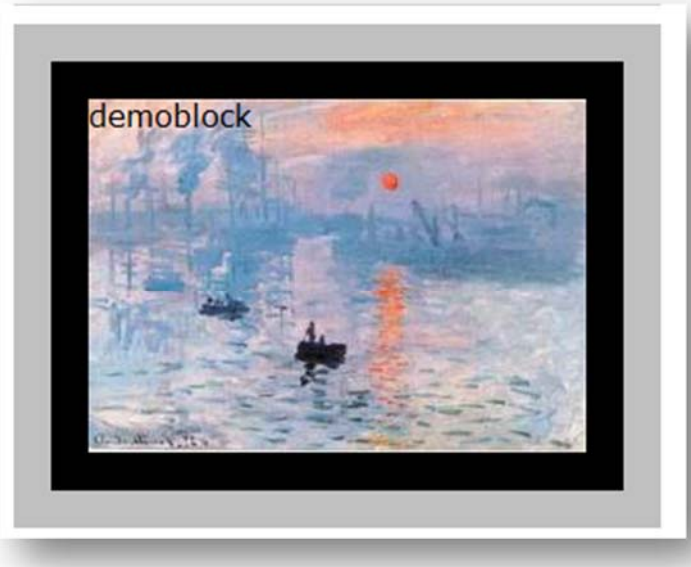
Hier bemerk je dat de image afgeknipt is: de oorsprong zit in de linkerbovenhoek van de border

```
#d5 {
  background-origin: padding-box;
  background-position:top left;
}
```



Dit is de situatie zoals ze ook in CSS2 is: de padding-box is de standaard positie.

```
#d6 {
  background-origin: content-box;
  background-position:top left;
  }
```



Ook deze situatie is nieuw: het beeld heeft de linkerbovenhoek van de content-box als oorsprong.

Enkele opmerkingen:

- de border plaatst zich **altijd** bovenop het achtergrondbeeld, maar omdat deze een transparante kleur kan hebben of zelf uit een image kan bestaan, zijn mooie effecten mogelijk
- `background-clip` heeft een invloed op `background-origin`

Voorbeeld van combinatie van transparante border met `background-origin: border-box`:



background-clip

De nieuwe CSS property `background-clip` bepaalt hoe een achtergrondbeeld "afgeknipt" wordt, via één van de volgende waarden : `padding-box, border-box` of `content-box`. De standaardwaarde is `padding-box`.

Deze eigenschap bepaalt infeite hoever de achtergrondfiguur/kleur getekend wordt: normaal is dat de `padding-box`, maar hij kan zich ook achter de border uitstrekken of kleiner: enkel op de inhoud liggen. Om dit te demonstreren hebben we een tweede achtergrond nodig en een semi-transparante border.

Deze voorbeelden hebben allemaal:

```
width:260px;
height:260px;
border:20px solid rgba(8,8,8,0.3);
margin:20px;
padding:20px;
background-image:url(../img/sorolla2.jpg);
background-repeat:no-repeat;
background-origin:border-box;
```

En zitten zelf in een andere container met streepjes. Ze hebben dan ook in volgorde: `background-clip:content-box, padding-box` en `border-box`.



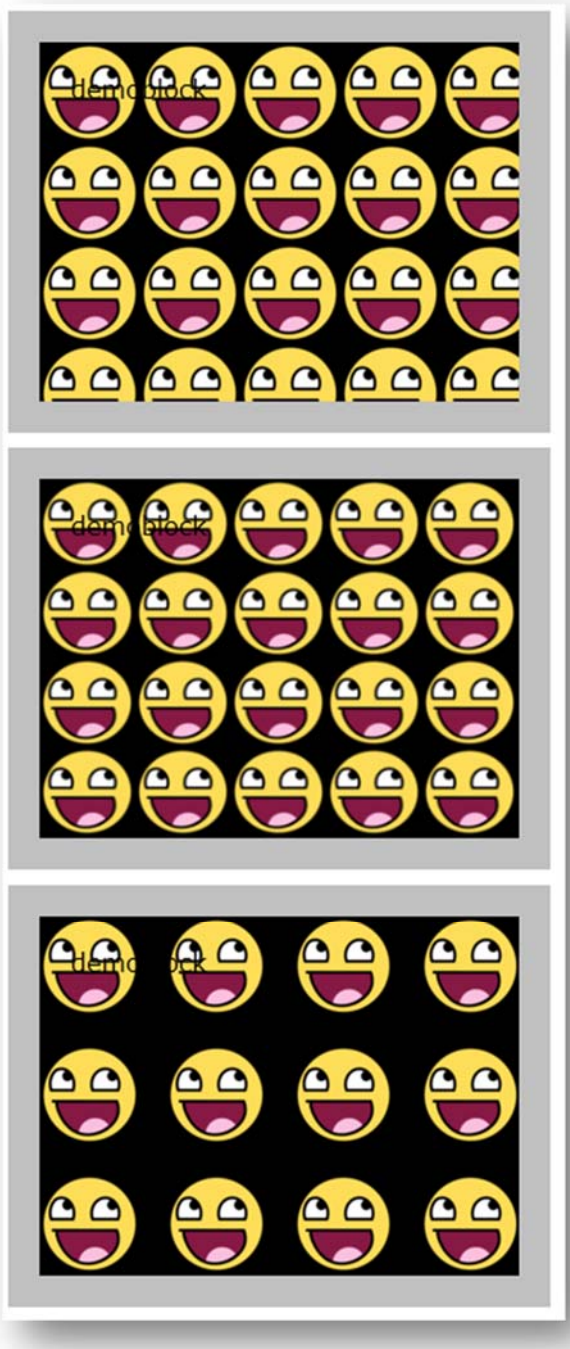
background-repeat

De CSS2 property `background-repeat` heeft een aantal nieuwe waarden gekregen.

background-repeat waarde	geschreven als twee waarden	beschrijving
`repeat-x`	`repeat no-repeat`	herhaal de achtergrondfiguur enkel horizontaal. Clipping kan gebeuren
`repeat-y`	`no-repeat repeat`	herhaal de achtergrondfiguur enkel vertikaal. Clipping kan gebeuren
`repeat`	`repeat repeat`	herhaal de achtergrondfiguur zowel horizontaal als vertikaal. Clipping kan gebeuren
`space`	`space space`	de achtergrondfiguur wordt zoveel keer herhaald als een geheel aantal keer zal passen en dan wordt de ruimte (space) tussen de figuren verdeeld om de *background-positioning area* te vullen. De eerste en laatste figuur raken de randen van de *background-positioning area*.
`round`	`round round`	de achtergrondfiguur wordt zoveel keer herhaald als een geheel aantal keer zal passen. De figuur zelf wordt gestrekt of gekrompen om de *background-positioning area* te vullen. De eerste en laatste figuur raken de randen van de *background-positioning area*.
`no-repeat`	`no-repeat no-repeat`	geen herhaling

De nieuwe waarden `round` en `space` worden momenteel enkele door IE9+ en O12 ondersteund.

De volgende voorbeelden in IE9 hebben in volgorde de waarden `repeat`, `round` en `space` voor `background-repeat`.



background-size

De nieuwe CSS property `background-size` bepaalt de grootte van een achtergrondbeeld in absolute of relatieve termen. Mogelijke waarden zijn:

```
[ % | px | auto ] | {1,2}, cover, contain
```

De containers in deze voorbeelden hebben de dimensie 266 X 189. De achtegrondfiguur heeft de werkelijke dimensies 420 X 317, groter dus.

Voorbeelden in volgorde links -> rechts, boven->onder, screenshot IE9:

- geen `background-size`
  de normale instelling waaruit blijkt dat de figuur groter is dan de container
- `background-size: contain`
  de figuur wordt volledig getoond en behoudt zijn hoogte-breedte verhouding. Een deel witruimte is rechts te zien.
- `background-size: cover`
  de figuur bedekt de container volledig, behoudt zijn hoogte-breedte verhouding tot ofwel breedte ofwel hoogte de container bedekt. Hier is dat de breedte, dus je ziet een deel van de hoogte niet
- `background-size:200px 100px`
  de figuur wordt gedwongen 200px breed en 100px hoog te zijn. De eerste waarde is altijd de `width`, de tweede de `height`
- `background-size:100%`
  de figuur wordt zowel in de hoogte als in de breedte 100% van de background-positioning area, bv hier de `padding-box`
- `background-size:30%; background-repeat:repeat`
  de figuur wordt herhaald en heeft 30% `width`. Dus is een deel van een vierde herhaling zichtbaar
- `background-size:30%; background-repeat:round`
  de figuur wordt herhaald en de waarde 30% wordt afgerond zodat er een geheel aantal keer herhaald wordt.



background-attachment

De `background-attachment` property heeft een nieuwe waarde. De mogelijkheden nu zijn `fixed`, `local` en `scroll`

background-attachment waarde	beschrijving
`fixed`	de achtergrondfiguur staat **vast** t.o.v. de *viewport* (het scherm in de meeste gevallen) en beweegt niet mee met het scrollen.
`scroll`	de achtergrondfiguur beweegt mee met het scrollen van de *viewport* en de inhoud.
`local`	de achtergrondfiguur beweegt enkel mee met het scrollen van de inhoud van een element (vb. in een overflow element)

background shorthand notation

De shorthand notatie die deze nieuwe properties bevat is dus:

```
background:  <bg-image> || <bg-position> [ / <bg-size> ]? ||
 <repeat-style> || <attachment> || <box>{1,2},]* || <background-color>
```

Een voorbeeld:

```
#d13 {
  width:          300px;
  height:         400px;
  overflow:       scroll;
  background:     url(../img/boerensmurf.gif) bottom right no-repeat local,
                  url(../img/nerd_64.png) content-box top left space,
                  orange;
}
```

ziet er zo uit in Opera:



De smurf scrollt mee met de inhoud.

17.6 Borders

Ook bij de randen zijn er nieuwigheden:

- afgeronde hoeken met `border-radius`
- beelden gebruiken als rand
- `box-shadow`

De standaard is ook na te lezen op de CSS Backgrounds and Borders Module Level 3

afgeronde hoeken

De nieuwe CSS property `border-radius` bepaalt de **mate** van afronding van een hoek.



border-radius heeft geen browserprefixes meer nodig.

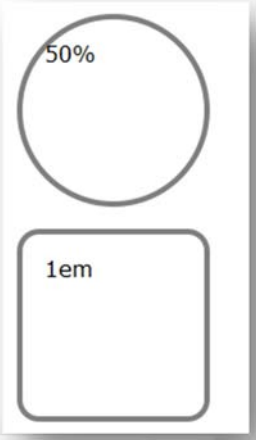
Een waarde `0` geeft een rechte hoek, dus geen afronding.

De standaard syntax voor **symmetrische hoeken**:

- 4 gelijke hoeken: `lengte | %`

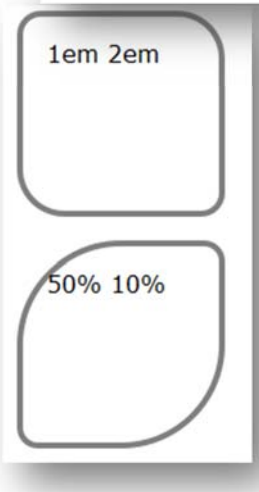
```
border-radius: 50%
border-radius: 1em
```

Een percentage is berekend op de lengte van de border: dus 50% is de helft van de border, vormt dus een cirkel



- *top-left* + *bottom-right* en *top-right* + *bottom-left*:

```
 lengte|%   lengte|%

border-radius: 1em 2em
border-radius: 50% 10%
```

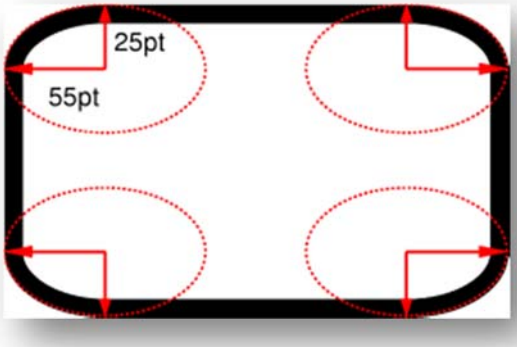


- 4 verschillende hoeken

```
lengte|%  lengte|%   lengte|%   lengte|%
```

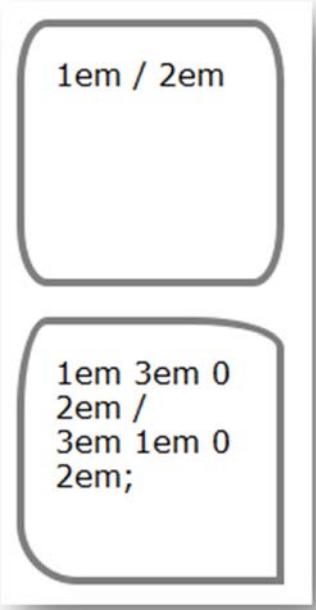


Voor **niet-symmetrische hoeken** moet je telkens twee waarden per hoek specifiëren gescheiden door een slash `/`




```
lengte|%   /  lengte|%
```

waar de eerste waarde de horizontale straal is van de ellips, de tweede de vertikale.



Twee voorbeelden:

```
border-radius:1em / 2em
```

```
border-radius:1em 3em 0 2em / 3em 1em 0 2em
```

Er kan uiteraard ook gecombineerd worden met achtergronden, gradients en box shadow, maar IE9 heeft een probleem in combinatie met gradients.

box-shadow

Met de CSS3 property `box-shadow` plaats je één of meer schaduwen op een box.

Een schaduw specificatie bestaat uit een kleur, 2 of 4 lengtes pixels/procent, en eventueel het woord `inset`. Zowel kleur als `inset` zijn optioneel, ook hun plaats is niet belangrijk.

De syntax:

```
box-shadow:  inset? lengte {2,4} kleur, [herhaal]
```

- **2 lengtes**

  bepalen de **offset** van de schaduw:
  - de horizontale offset (negatieve waarde: links van de box),
  - de tweede vertikaal (negatieve waarde: boven de box).

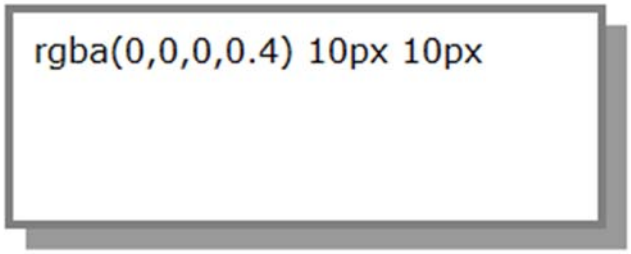
- **4 lengtes**

  - offset als hierboven,
  - de derde lengte is de **blur radius**. Bij 0 is de rand scherp. Geen negatieve waarde
  - de vierde lengte is de **spread distance**.
    Positieve waarden doen de schaduw verspreiden in alle richtingen, negatieve doen de schaduw samentrekken.

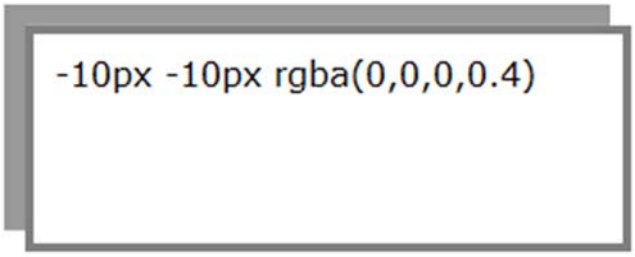
Deze syntax kan herhaald worden voor meerdere schaduwen, bv een inset en een outset.

Enkele voorbeelden

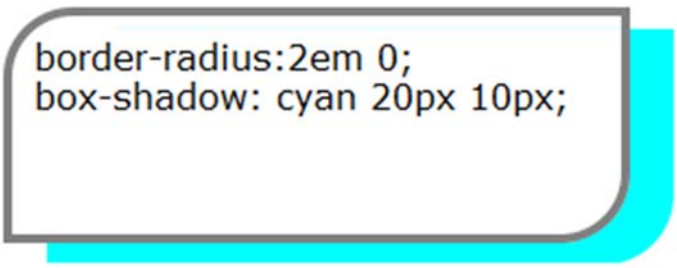
```
box-shadow: rgba(0,0,0,0.4) 10px 10px;
```



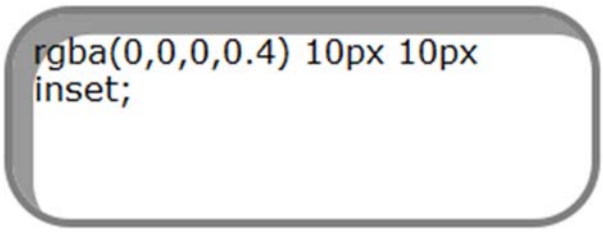

```
box-shadow:-10px -10px rgba(0,0,0,0.4)
```



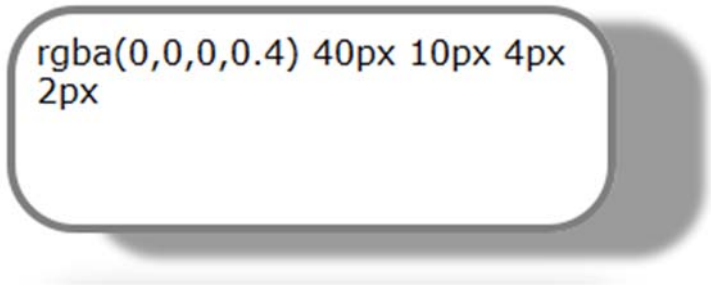

```
border-radius:2em 0;
box-shadow: cyan 20px 10px;
```



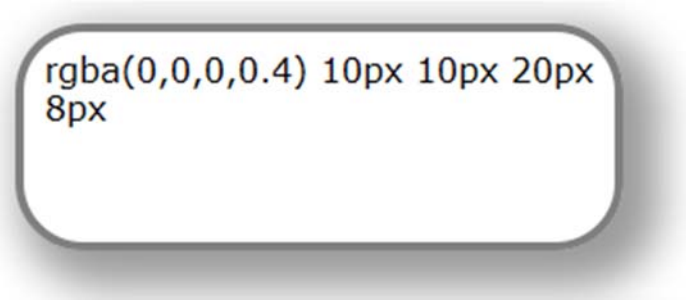

```
border-radius:2em;
box-shadow: rgba(0,0,0,0.4) 10px 10px inset;
```



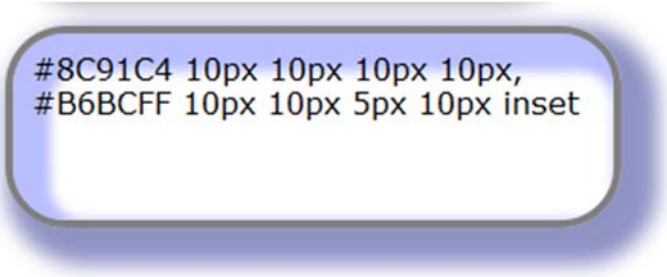

```
border-radius:2em;
rgba(0,0,0,0.4) 40px 10px 4px 2px
```




```
border-radius:2em;
box-shadow: rgba(0,0,0,0.4) 10px 10px 0px 8px;
```




```
border-radius:2em;
box-shadow:  #8C91C4 10px 10px 10px 10px , ↵
               #B6BCFF  10px 10px 5px 10px inset ;
```



figuren als border

De nieuwe CSS property `border-image` laat nu toe een figuur te gebruiken ipv een `border-style`. Deze figuur heeft geen enkele invloed op de layout van de box.

Een voorbeeld: de figuur hierronder is 81px vierkant en wordt gebruikt als border-image voor de container rechts.





```
border: 10px solid black;
border-image: url(../img/border.png) 27  round round;
```

Belangrijk: om een border-image te zetten **moet** de border ingesteld zijn. Hoe dikker de border hoe dikker de border-image!

`border-image` is de shorthand versie van 3 subproperties:

- `border-image-source`: het bronbestand
- `border-image-slice`: 27px, zie hieronder
- `border-image-repeat`: met round worden de centrale delen van de slice een geheel aantal keer herhaald

17.7 Text shadow

De Text Decoration Module level 3 bepaalt een nieuwe property `text-shadow` die erg gelijkt op `box-shadow`.

De syntax is een komma gescheiden tekst van "schaduwen" zoals hieronder:

```
text-shadow:  lengte {2,3} kleur, [herhaal]
```

- **2 lengtes**

  bepalen de **offset** van de schaduw:
  - de **horizontale** offset (negatieve waarde: links van de box),
  - de tweede **vertikaal** (negatieve waarde: boven de box).

- **3 lengtes**

  - offset als hierboven,
  - de derde lengte is de **blur radius**. Bij 0 is de rand scherp. Geen negatieve waarde

Enkele voorbeelden:

```
#t1 {text-shadow:  #B6BCFF 10px 10px; }
```




```
#t2 {text-shadow:  #B6BCFF 20px 10px 10px;}
```




```
#t3 {text-shadow:       #B6BCFF 4px 4px 2px,
                        #E184EA -40px -50px 6px,
                        #8DEA84 60px 60px 10px;
}
```

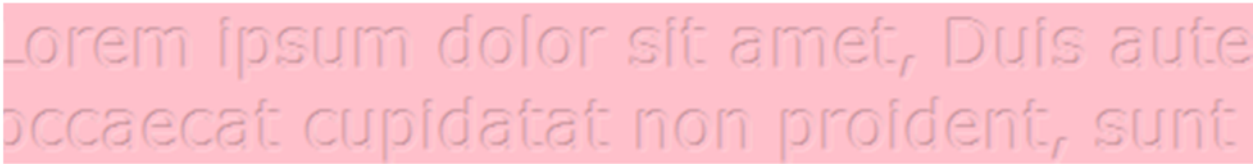



```
#t4 {
  font-size: 1.8em;
  color:#A7ADEB;
  text-shadow:  #6E75C4 20px 10px 4px;
}
#t4::first-letter {
```

```
  float: left;
  color: #FF228D;
  font-size: 75px;
  line-height: 60px;
  padding-top: 4px;
  padding-right: 4px;
  padding-left: 3px;
  font-family: Georgia;
  text-shadow:  #BD6EC4 20px 10px 10px;
}
```




```
#t5 {
  text-shadow:          rgba(0,0,0,0.1) -1px 0, ⏎
                        rgba(0,0,0,0.1) 0 -1px, ⏎
                        rgba(255,255,255,0.1) 1px 0, ⏎
                        rgba(255,255,255,0.1) 0 1px, ⏎
                        rgba(0,0,0,0.1) -1px -1px, ⏎
                        rgba(255,255,255,0.1) 1px 1px;
  color:pink;
  background: pink;
}
```



17.8 Web fonts

De CSS3 Fonts module op het W3C bevat naast de gekende CSS2.1 properties (`font-family, font-weight`, ...) ook enkele nieuwe zaken, o.a.. **webfonts**: lettertypes die op het web beschikbaar zijn en dus niet op de PC van de gebruiker.
Deze lettertypes kunnen op dezelfde website zitten of op een andere gemakkelijk bereikbare site.

17.8.1 Lettertype bestanden

Een lettertype komt in een aantal bestandstypes:

extensie	font type	beschrijving	support
`TTF`	True Type Font	allomtegenwoordig	FF, Chrome, O, Saf, IE9, Android
`EOT`	Embedded Open Type	Microsoft. Wrapper voor TTF's met digitale rechten bescherming.	enkel IE
`OTF`	Open Type	ook een True Type font	alle, niet IE en niet Saf iOs, Android
`SVG`	Scalable Vector Graphics	SVG = XML formaat. SVG kan lettertypes genereren	Saf, O, Chrome
`WOFF`	Web Open Font Format	Nieuw. Wrapper voor TTF en OTF. Waarschijnlijk wordt dit het W3C officiële recommendatie	Cross-browser: FF3.6, Chrome6, O11, IE9,

Safari op iOs (iPhone, iPad,...) begrijpt enkel SVG.

Waar haal je die lettertypes? Er zijn meerdere websites waar je gratis of betalend lettertypebestanden kunt downloaden of ernaartoe koppelen, hier slechts enkele:

- Fontsquirrel (gratis)
- Google webfonts (gratis)
- Typekit by Adobe (betalend)

Opmerking: webfont services zoals Typekit en Google Web fonts gebruiken dikwijls een speciale methode om de lettertypes toe te passen, niet de standaard methode hieronder beschreven. Lees de documentatie goed na.

17.8.2 @font-face

De klassieke manier om een speciaal lettertype te gebruiken, was een grafisch beeld te maken met Photoshop en dit dan via *image replacement* te plaatsen op de tekst, een kop bijvoorbeeld.

Met `@font-face` kan je het speciale lettertype direct toepassen op de tekst, *image replacement* is niet meer nodig.
`@font-face` laat je toe een **bron** te specificeren voor een lettertype die niet op het

toestel van de gebruiker aanwezig is, en dus gedownload wordt naar zijn toestel.
De enige voorwaarde is dat de browser `@font-face` ondersteunt.

De `@font-face` rule geeft een **naam** aan het lettertypebestand(en) die je beschikbaar stelt. Het werkt als volgt:

```
@font-face {
        font-family: Michroma;
        src: url(../fonts/Michroma.ttf);
  }
```

In dit voorbeeld wordt de naam "Michroma" **gegeven** via de `font-family` property. Je gebruikt uiteraard best de echte naam van het lettertype, maar in principe kan je eender welke naam gebruiken.
De `src` property bevat het url van het bestand, hier in een submap van de eigen website.

Eenmaal de `@font-face` rule ingesteld kunnen we de fontnaam gaan **gebruiken**, zoals in deze voorbeelden:

```
#logo {  font-family: Michroma, Georgia, "Times New Roman", Times, serif;}

h2 {     font-family: UbuntuMono,"Courier New", Courier, monospace;
         font-size:1.4em;
         font-weight:bold;
}
```

Het is ook best de gebruikelijke "web-safe" font collectie **na** je webfont te zetten, als fall-back.
Je kan uiteraard meerdere `@font-face` rules in je stylesheet zetten.

17.8.3 cross-browser @ff syntax

Om cross-browser te werken gebruiken we Paul Irish's Bulletproof @ff syntax die ook toegepast wordt door Font-squirrel's @font-face kits:

```
@font-face {
      font-family: 'BitstreamVeraSansMonoRoman';
      src: url('VeraMono-webfont.eot');
      src:       url('VeraMono-webfont.eot?#iefix') format('embedded-opentype'),
                 url('VeraMono-webfont.woff') format('woff'),
                 url('VeraMono-webfont.ttf') format('truetype'),
                 url('VeraMono-webfont.svg#BitstreamVeraSansMonoRoman') format('svg');
      font-weight: normal;
      font-style: normal;
  }
```

Deze syntax voorziet voor alle mogelijke browsers en situaties. Alle verschillende

fonttypes moeten wel op je webserver aanwezig zijn.
Hou er rekening mee dat voor elke font een nieuw HTTP request naar de server gedaan wordt.
Heb je zelf een bepaald font dat je wil voorzien, dan kan je via Font-squirrel's @font-face Kit Generator je eigen "kits" aanmaken.

17.8.4 FOUT

In sommige browsers zie je een korte tijd het standaard lettertype vooraleer het webfont toegepast wordt. Dat komt omdat er gewacht wordt met het downloaden van het font, waarna de pagina opnieuw hertekend wordt.
Deze *flash* wordt een ***FOUT*** genoemd: een ***"Flash Of Unstyled Text"*** (in navolging van een *FOUC*, "*Flash Of Unstyled Content*").
IE is daarin beter dan andere browsers want hij downloadt het font onmiddellijk na de `@font-face` rule. De andere browsers wachten tot ze HTML tegenkomen waarop de CSS matched, dat veroorzaakt zeker een FOUT.
Wat kan je eraan doen?

- beperk je fonts enkel tot wat je nodig hebt
- zet de `@font-face` rules bovenaan je stylesheet
- gebruik een script die de pagina verbergt tot de font geladen is
- zet een `Expires` header in de toekomst in je *.htaccess* file op je webserver, zodat browsers het font cachen

17.8.5 Webfonts en mobile

Op mobiele toestellen kunnen web fonts problemen opleveren:

- sommige mobiele browsers ondersteunen `@font-face` niet
- ze verhogen de downloadtijd aanzienlijk door hun bestandgrootte en soms door de nood aan extra Javascript libraries
- sommige fonts worden slecht weergegeven of helemaal niet!
- sommige toestellen wijzigen je lettertype sowieso in een standaardlettertype dat beschikbaar is op het toestel (bv. Droid voor Android)

Terzijde: er moet ook opgemerkt worden dat het gebruik van images via *image replacement* al evenmin een goed idee is bij mobile: ze zoomen/schalen zeer slecht…

Je moet dus zeker controleren of een lettertype die je ook op mobieltjes wil gaan gebruiken "Mobile safe" is. Google claimt dat zijn webfonts dat allemaal zijn.

Hoe kan je zeker spelen? bouw een veiligheid in door een "generic font family" achteraan in te sluiten in je `font-family` property:

```
font-family: "Foo Regular", "Bar Sans", sans-serif;
```

17.8.6 Google Web Fonts API

De Google Web Fonts API maakt geen gebruik van `@font-face`, maar van een `link` element dat doorwijst naar de Google Font Directory, zoals hier:

```
<link href='http://fonts.googleapis.com/css?family=Droid+Sans|Open+Sans+Condensed:300'
  rel='stylesheet' type='text/css'>
```

In dit voorbeeld worden twee fonts beschikbaar gesteld: "*Open Sans Condensed*" en "*Droid Sans*".
Gebruiken gebeurt op dezelfde manier:

```
#t6 {
  font-family: 'Open Sans Condensed', sans-serif;
  font-size:3em;
}
#t7 {
  font-family: 'Droid Sans', sans-serif;
}
```

De Google Web Fonts pagina laat je gemakkelijk kiezen uit een grote selectie van lettertypes en laat je de nodige code kopiëren en plakken. Ze duidt ook de impact aan van het gebruik van die fonts op de laadtijd van de pagina.

De meeste van deze lettertypes werken goed op alle desktopbrowsers en op de meerderheid van de huidge mobile devices.

17.9 CSS Columns

Met de nieuwe Multi-column Lay-Out Module brengt CSS3 de mogelijkheid om tekst automatisch in meerdere kolommen te laten vloeien.

Maar net als met andere CSS3 features rijst de vraag of de browser van de gebruiker deze module wel ondersteunt?

Daarnaast moet je ook best gebruik maken van browser-prefixes vóór de standaard eigenschap:

```
-moz-columns: 3;
-webkit-columns: 3;
-o-columns: 3;
columns: 3;
```

Een multi-column element

is een container die kolommen bevat. Je kan gebruik maken van één van de volgende twee eigenschappen:

- `column-width` : stelt de **breedte** van de kolommen in
- `column-count`: stelt het **aantal** kolommen in
- `columns`: is een *shorthand property* en kan één van de twee voorgaande waarden bevatten

Zowel `column-count` als `column-width` kunnen de waarde `auto` hebben wat betekent dat ze geen kolommen maken.

In een eerste voorbeeld bevat een `div` met de `class`es *mc* en *vb1* een aantal andere elementen: `h2`, `p`, `img`:

```
<div class="mc vb1">
        <h2>Scrum</h2>
        <p><img src="../img/marie.jpg">Bacon ipsum dolor sit amet nostrud salami minim
  swine prosciutto irure beef est dolore ut in tempor. Jerky tri-tip meatball, tempor
  flank corned beef ut, sunt jerky ut esse. </p>
…
 <p><img src="../img/rose.jpg">Bresaola anim doner minim tongue aliqua duis biltong
  bacon laborum incididunt reprehenderit. Aliqua pariatur voluptate irure, ribeye …
        <p>Does your lorem ipsum text long for something a little meatier? Give our
  generator a try… it’s tasty! </p>
      </div>
```

In ons voorbeeld is de container *.mc* gestyled met een `background` en `outline` om hem beter zichtbaar te maken. ook de `img` elementen werden gealterneerd gefloat.

We kunnen nu gebruik maken van `column-count`:

```
.vb1 {
  -moz-column-count:    3;
  -webkit-column-count: 3;
  -o-column-count:      3;
  column-count:         3;
}
```

Met dit resultaat:



Het aantal kolommen is hier bepaald, de breedte ervan wordt dus berekend en is fluid.

of van `column-width`:

```
.vb2 {
  -moz-column-width:    14em;
  -webkit-column-width: 14em;
  -o-column-width:      14em;
  column-width:         14em;
}
```

met dit resultaat (voor een breed scherm):



Hier is de breedte van de kolommen bepaald, dus is hun aantal variabel (volgens de schermbreedte). Het is bij deze instelling dus perfect mogelijk dat er aan de rechterzijde een lege ruimte ontstaat.

Men spreekt van "*column-boxes*" in een "*row*". Omdat de column-boxes automatisch berekend worden door de browser heb je er geen invloed op.

Probeer eens het venster te verbreden/versmallen.

Beide voorbeelden kan je ook bereiken met, respectievelijk `columns: 3` of `columns: 14em`.

Ruimte en lijnen tussen de kolommen

De ruimte tussen de kolommen kan bepaald worden met de property `column-gap`

```
column-gap: 20px
```

Je kan ook een verticale lijn vragen met `column-rule`.

```
column-rule: 1px solid black
```

Dit is de shorthand property voor `column-rule-style`, `column-rule-color` en `column-rule-width`. Dus dit is het equivalent, afzonderlijk geschreven:

```
column-rule-style:      solid;
column-rule-color:      black;
column-rule-width:      1px;
```

In ons voorbeeld met prefixed properties:

```
.vb4 {
  -moz-column-count:    3;
  -webkit-column-count: 3;
  -o-column-count:      3;
  column-count:         3;

  -moz-column-gap:      4em;
  -webkit-column-gap:   4em;
  -o-column-gap:        4em;
  column-gap:           4em;

  -moz-column-rule:     1px solid black;
  -webkit-column-rule:  1px solid black;
  -o-column-rule:       1px solid black;
  column-rule:          1px solid black;
}
```

ziet er zo uit:



Een nieuwe kolom beginnen

In een multi-column element vloeien de kolomen automatisch. Met de properties `break-before`, `break-after` en `break-inside` kunnen we **kolom-sprongen** zetten en vermijden. Het zijn dezelfde eigenschappen als deze gebruikt voor paginasprongen in een printstylesheet. Je kan deze properties gebruiken op een element binnen een *multi-column* element.

`break-before` en `break-after` kunnen de volgende waarden hebben (waarvan enkele nieuwe):

```
break-before/break-after:   auto | always | avoid | left | right |
                            page | column | avoid-page | avoid-column
```

De property `break-inside` kan enkel de volgende waarden hebben:

```
break-inside:               auto | avoid | avoid-page | avoid-column;
```

Ze betekenen het volgende:

waarde	beschrijving
auto	automatische verdeling, de standaardwaarde
always	begin altijd een nieuwe pagina voor/achter dit element
avoid	vermijd een nieuwe pagina/kolom voor/achter dit element
left, right	zet één of twee nieuwe pagina's zodat het element zich op een linkse/rechtse pagina bevindt
page	begin altijd een nieuwe pagina voor/achter dit element
column	begin altijd een nieuwe kolom voor/achter dit element
avoid-page	vermijd een nieuwe pagina achter/binnen dit element
avoid-column	vermijd een nieuwe kolom achter/binnen dit element

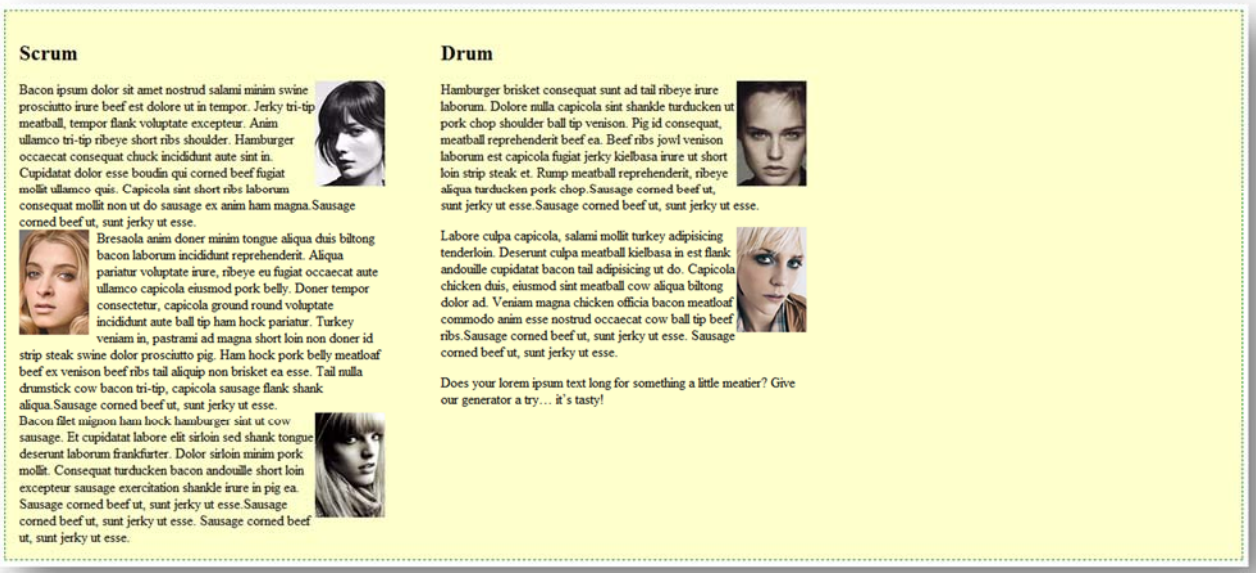
Je bemerkt dat er voor kolommen slechts enkele waarden van toepassing zijn.

als we bijvoorbeeld op een van de vorige voorbeelden verder werken met deze CSS:

```
.vb2 p {
  -moz-break-inside:    avoid-column;
  -webkit-break-inside: avoid-column;
  -o-break-inside:      avoid-column;
  break-inside:         avoid-column;
}
.vb2 h2 {
  -moz-break-before:    column;
```

```
  -webkit-break-before: column;
  -o-break-before:      column;
  break-before:         column;
}
```

dan krijg je in Opera12:



Momenteel is de ondersteuning van deze property zwak.

Kolommen samenvoegen

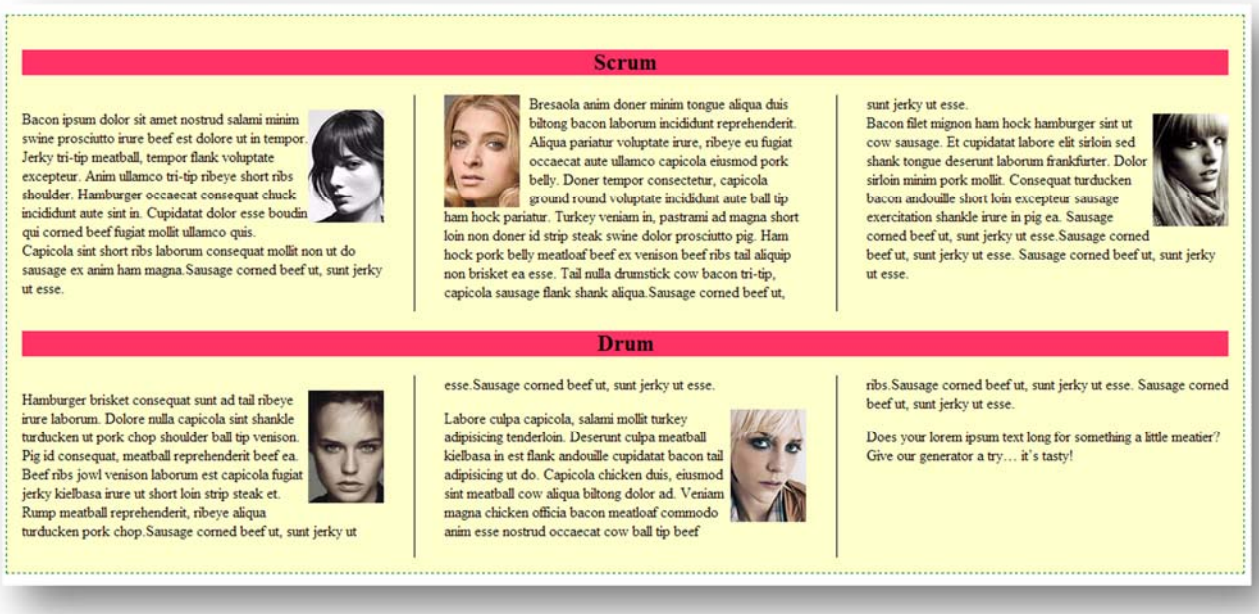
Met de property column-span is het mogelijk meerdere kolommen te overspannen. Helaas zijn er maar twee mogelijke waarden: `none` en `all` .

Veronderstel:

```
.vb6 {
  -moz-column-count:    3;
  -webkit-column-count: 3;
  -ie-column-count:     3;
  -o-column-count:      3;
  column-count:         3;

  -moz-column-gap:      4em;
  -webkit-column-gap:   4em;
  -o-column-gap:        4em;
  column-gap:           4em;

  -moz-column-rule:     1px solid black;
  -webkit-column-rule:  1px solid black;
```

```
  -o-column-rule:       1px solid black;
  column-rule:          1px solid black;
}

.vb6 h2 {
  background:           #FF3366;
  text-align:           center;
  -moz-column-span:     all;
  -webkit-column-span:  all;
  -o-column-span:       all;
  column-span:          all;
}
```

Dan krijgen we in Chrome:



Uitvullen

Alle voorbeelden die we tot nu toe maakten, vertonen aan het einde wat lege ruimte omdat er onvoldoende inhoud is om de kolom volledig op te vullen: ze zijn niet uitgevuld. met de eigenschap `column-fill` wordt die mogelijkheid geboden.Er zijn twee mogelijke waarden: `auto` (niet uitgevuld, standaard) en `balance` (uitgevuld).

We proberen:

```
.vb7{
  -moz-column-count:3;
  -webkit-column-count:3;
  -o-column-count:3;
  column-count:3;
```

```
  -moz-column-fill:balance;
  -webkit-column-fill:balance;
  -o-column-fill:balance;
  column-fill:balance;
}
```

Het resultaat is onduidelijk: voor de meeste browsers is er momenteel (2012) geen enkele ondersteuning en in Firefox is er weinig verschil te merken tussen `auto` en `balance`..

17.10 Transforms

Met de CSS3 2D Transforms Module Level 3 kan je boxes verplaatsen, roteren, verkleinen, vergroten en vervormen. Deze Module voorziet enkel de praktische uitwerking van 2-dimensionele transforms, een 3D module is voorzien voor de toekomst.

In het klassieke box-model kennen we reeds het systeem van coördinaten die vertrekken vanuit een hoek, denk maar aan het plaatsen van een achtergrond `top left`, etc…

De nieuwe `transform` property kan je één of meerdere **transformfuncties** toewijzen die een vervorming van de box toelaten, bv:

```
div { transform: translate(100px, 100px) scale(1.5, 1.5) rotate(45deg)  }
```

Hier wordt een `div` element schuin naar onder verplaatst, verkleind en gedraaid.

De `transform-origin` property zet het **punt van oorsprong** bij een transformatie. Het bestaat uit twee coördinaten elk met één van de volgende waarden :

`<percentage> | <length> | left | center | right`

De standaardwaarde van `transform-origin` is `center center`. Als je dus een box roteert zal die standaard rond zijn middelpunt roteren.

Andere voorbeelden:

```
tranform-origin: 0 0 ; /* zelfde als top left */
tranform-origin: 100% 100% ; /* zelfde als bottom right */
```



Opmerking: deze eigenschappen hebben *momenteel* nood aan ***browserprefixes*** om ze te doen werken.

Verplaatsen:

Met `translate(tx[,ty])` wordt de vorm verplaatst volgens de *vector (tx ty)* vanuit de `transform-origin`.
De afgeleide functies `translateX` en `translateY` doen dat respectievelijk enkel horizontaal en vertikaal.

```
#t0 { transform: translateX(100px); }
#t1 { transform: translate(-50px,-50px); }
#t2 { transform: translate(200%); }
```

In het eerste voorbeeld wordt de box 100px horizontaal naar rechts verplaatst. In het tweede gaat de box 50px naar links én naar boven t.o.v. zijn *origin*. Het derde voorbeeld verplaatst de box naar rechts over een afstand van tweemaal zijn breedte.

Deze functie alleen gebruiken komt op hetzelfde neer als een relatieve of absolute positionering uitvoeren.

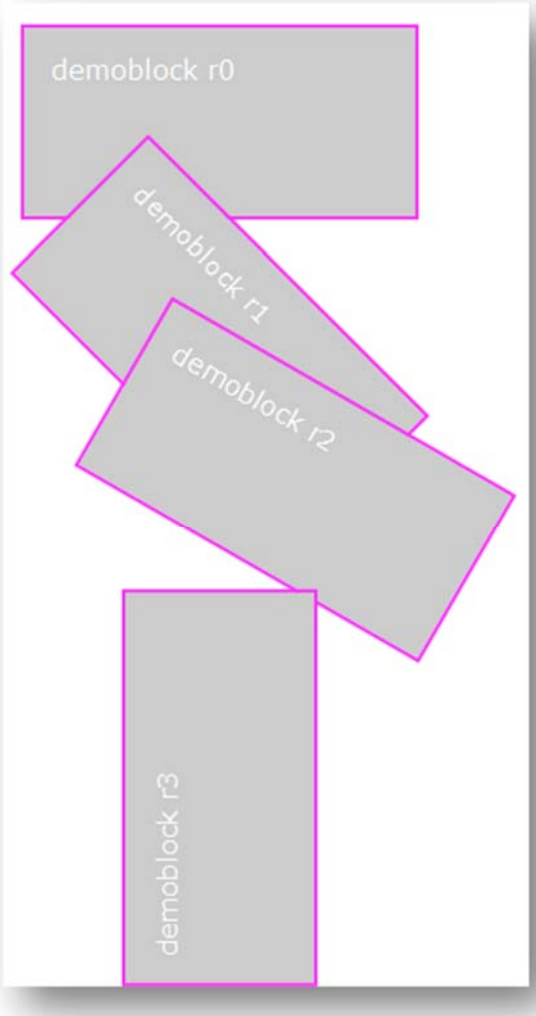
Draaien:

Roteren met `rotate(d)` gebeurt in wijzerzin volgens een hoek uitgedrukt in "`deg`" rond de `transform-origin` . De eenheid "`deg`" is uitgedrukt in graden.

Voorbeelden:

```
#r1 {    transform: rotate(45deg); }
#r2 {    transform: rotate(30deg);
         transform-origin: bottom right);
  }
#r3 {    transform: rotate(270deg); }
```

Geeft dit resultaat t.o.v. een niet gedraaide box `r0`:



Je bemerkt dat de eerste box `r1` roteert rond zijn middelpunt, de tweede `r2` rond zijn rechterbenedenhoek.

`r3` staat vertikaal.

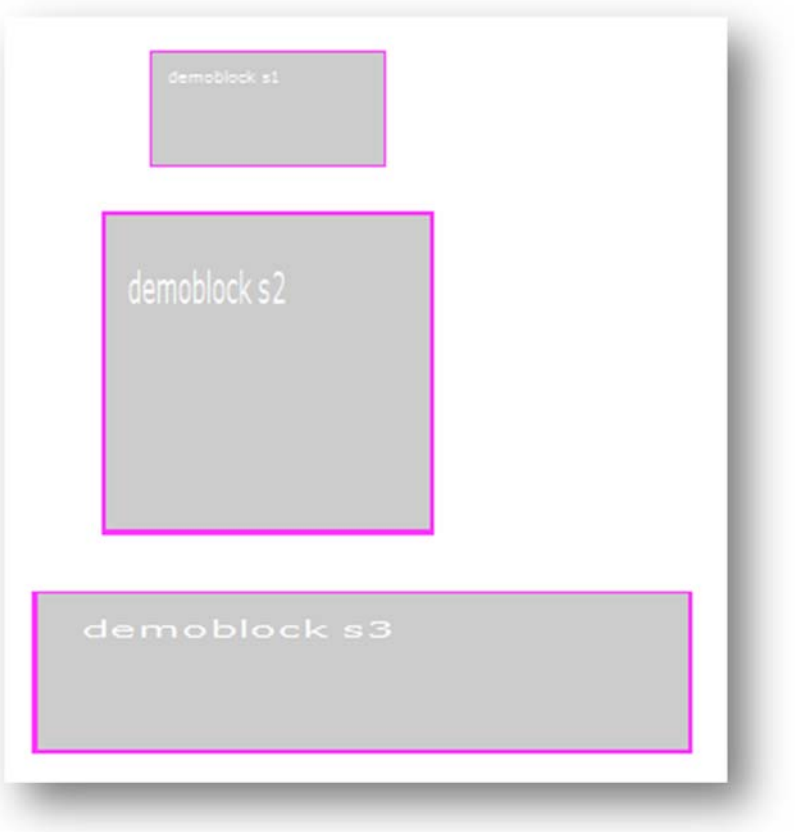
Vergroten/verkleinen:

Met `scale(sx[,sy])` vergroot/verklein je een object. Gebruik je één argument, dan wordt zowel vertikaal als horizontaal gewijzigd, met twee argumenten is het eerst horizontaal, dan vertikaal. De argumenten `sx` en `sy` zijn decimale getallen waarbij 1 100% is.

De afgeleide functies `scaleX` en `scaleY` doen dat respectievelijk enkel horizontaal en vertikaal.

```
#s1 {    transform: scale(0.5); }
#s2 {    transform: scale(0.7,1.4); }
#s3 {    transform: scale(1.4,0.7);
         transform-origin:bottom left;
  }
```

geeft dit:



Bij `s3` zorgt de `transform-origin` er enkel voor dat een stukje van het blok niet van het scherm verdwijnt.

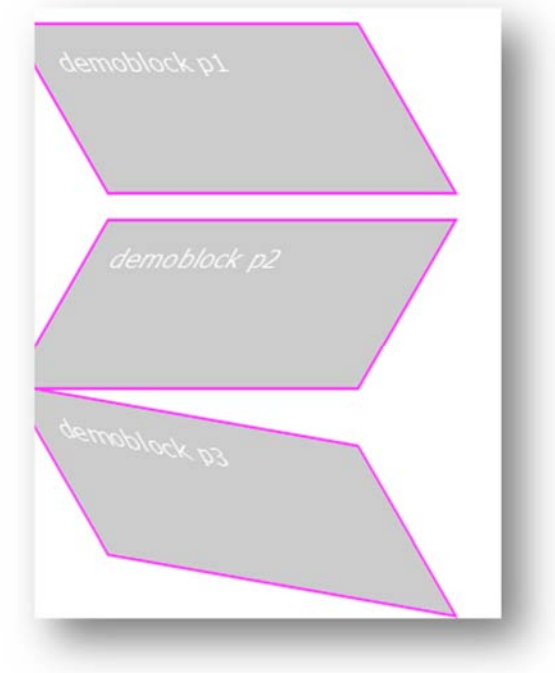
Parallelogram:

Met `skew(ax[,ay])` trek je een box scheef, dus in een parallellogram. Gebruik je slechts één argument, dan worden enkel de vertikale zijden gekanteld, met twee argumenten is het tweede argument voor de horizontale zijden.

De afgeleide functies `skewX` en `skewY` doen dat respectievelijk enkel horizontaal en vertikaal.

```
#p1 { transform:skew(30deg); }
#p2 { transform:skew(-30deg); }
#p3 { transform:skew(30deg, 10deg); }
```

geeft:



matrix

De functie `matrix` combineert alle bovenstaande methodes in één. De syntax is

```
transform:  matrix(a, c, b, d, tx, ty)
```

Waarbij `a, b, c, d` de transformation matrix vormen

```
┌     ┐
│ a b │
│ c d │
└     ┘
```

en `tx`, `ty` de *translate* waarden zijn

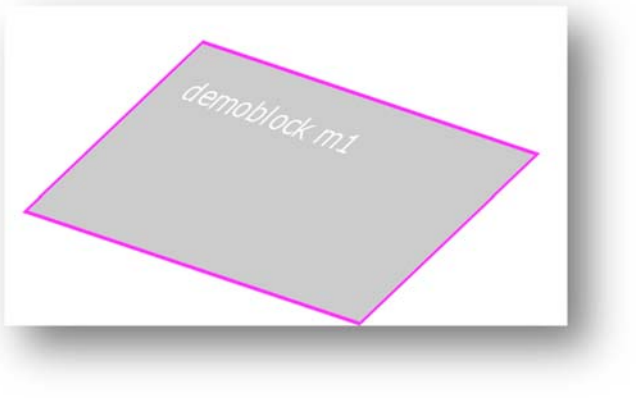
Merk ook op dat de volgorde van de matrix argumenten `a,c,b,d` is.
Voor een gedetailleerde uitleg zie

[The css3-matrix-transform-for-the-mathematically-challenged]

Een voorbeeld:

```
m1 { transform: matrix(1.5, 0.500, -1.6, 1.5, 419, 57); }
```

resulteert in:



17.11 Transitions

Met CSS3 Transitions en Animations kan je **visuele effecten** maken: overgangen, beweging, enz… Dat kan zonder daarbij Javascript te gebruiken, enkel met CSS dus. Javascript kán er natuurlijk bij betrokken worden.
De *CSS Transitions Module Level 3* op het W3C bepaalt de standaard

Op het eerste gezicht lijken Transitions en Animations erg op elkaar:

- je stelt de CSS properties in die moeten wijzigen
- je zet er een *easing* functie op om het verloop te bepalen
- je stelt een duurtijd in
- Javascript hoeft er niet aan te pas te komen, maar je kan het gebruiken, o.a. met *transition events*

Waar verschillen ze dan in?

- *Transitions* reageren op een verandering in een CSS property, bv. een `:hover` event of een `class` wissel via Javascript
- *Animations* vergen geen expliciete trigger: ze starten zodra je ze gedefinieerd hebt

De ondersteuning is momenteel nogal verscheiden:

- `-webkit-` prefix nodig
- ondersteuning op mobile devices is zwak
- <IE9 geen ondersteuning

Een transition is een overgang zoals een hover-effect maar minder plotseling, met een effect gespreid over een bepaalde tijd.

Je kan het instellen met de `transition` property (verkorte schrijfwijze) of apart met de properties `transition-property`, `transition-duration`, `transition-timing-function` en `transition-delay`.

Bij de `transition-property` property zeg je **welke CSS eigenschap** of lijst van komma-gescheiden eigenschappen, gebruikt zal worden voor de *transition*:

```
div { transition-property: opacity;  }
div { transition-property: opacity, background-color, left, top;  }
```

In het eerste voorbeeld zal de `opacity` eigenschap een *transitie* ondergaan, in het tweede de `opacity`, `background-color`, `left` en `top` eigenschappen. De meeste CSS properties zijn geschikt voor transitions

De `transition-duration` property geeft de **tijdspanne** van de transitie. Zijn er meerdere transition-properties gezet, dan kan je ook hier een lijst van komma-gescheiden tijden instellen. De tijd wordt uitgdrukt in '`s`' (seconden) of `'ms'` (milliseconden). De waarde `0` betekent "onmiddellijk":

```
div {
  transition-property: opacity;
   transition-duration: 3ms;
  }

div {
  transition-property: opacity, background-color, left, top;
  transition-duration: 3ms,2ms,1ms,0;
   }
```

De `transition-timing-function` property gebruikt een ***easing* functie** om de overgang te maken:

functie	beschrijving
`linear`	rechtdoor
`ease`	eerst traag, daarna sneller
`ease-in`	eerst traag, daarna sneller
`ease-out`	eerst snel, daarna trager
`ease-in-out`	eerst traag, daarna sneller, om opnieuw traag te stoppen

functie	beschrijving
`cubic-bezier(d,d,d,d)`	eigen instelling

De standaard waarde is `ease`.

Alle functies zijn afgeleiden van `cubic-bezier`. De getalwaarden van de `cubic-bezier` argumenten moet tussen `0` en `1` liggen.

Voorbeeld:

```
div {
  transition-property: opacity;
   transition-duration: 3ms;
  transition-timing-function: cubic-bezier(0.62, 0, 0.38, 1.0)
  }
```

De `transition-delay` property zet het tijdstip na de activering wanneer de transitie zal starten, een uitstel dus. De waarde `0` betekent "onmiddellijk":

Voorbeeld:

```
div {
  transition-property: opacity;
  transition-duration: 3ms;
  transition-timing-function: cubic-bezier(0.62, 0, 0.38, 1.0);
  transition-delay: 1ms;
  }
```

De property `transition` is de korte schrijfwijze van all sub-properties hierboven.
Syntax:

```
transition: transition-property || transition-duration ||
              transition-timing-function || transition-delay
            [, volgende transition ]*
```

Voorbeeld identiek aan hierboven:

```
div { transition: opacity 3ms cubic-bezier(0.62, 0, 0.38, 1.0) 1ms; }
```

Bij het volgende uitgewerkte voorbeeld hebben we de browser-prefixes weggelaten om overzichtelijk te zijn.

Voorbeeld : een transition op de `hover` state van een `button` element.

```
button {
```

```
        padding: 1em;
        border: 1px solid #A30F94;
        color: #A30F94;
        font-weight: bold;
        box-shadow: rgba(0,0,0,0.4) 6px 6px 2px 2px;
        border-radius: 23px;
        background: linear-gradient(top, ⏎
                    rgba(252,236,252,1) 0%, ⏎
                    rgba(251,166,225,1) 50%,⏎
                    rgba(253,137,215,1) 51%,⏎
                    rgba(255,124,216,1) 100%);

        transition: box-shadow 0.2s ease-out, ⏎
                    background 0.5s ease-in, ⏎
                    border 0.3s ease-in, ⏎
                    color 0.3s ease-in;
}

button:hover{
        border: 1px solid white;
        box-shadow: magenta 8px 8px 8px 8px;
        color:white;
        background: linear-gradient(top, ⏎
                    rgba(249,247,249,1) 0%,⏎
                    rgba(251,166,225,1) 31%,⏎
                    rgba(253,137,215,1) 33%,⏎
                    rgba(255,5,180,1) 72%);
 }
```



De transition werkt goed in Firefox, Opera en Chrome, helemaal niet in IE.

Op *mobile devices* geeft het sowieso problemen want hover state bestaat daar niet!

17.12 Animations

Met CSS *defined animations* kan je gewone HTML elementen bewegen en vervormen zonder Javascript. Een animatie hoeft niet opgestart te worden door een actie van de gebruiker, maar kan. De effecten kunnen ook zeer gedetailleerd ingesteld worden met behulp van *keyframes*.

Keyframes

Met een CSS `@keyframes` rule bepaal je de *keyframes* van de animatie. Een *keyframe* is **een punt waarop de animatie een bepaalde stand bereikt** moet hebben. De browser zorgt voor de vloeiende overgangen tussin (*tweening*).

De `@keyframes` rule wordt gevolgd door een ***Identifier*** = de naam van de animatie en bevat een aantal percentages met waarden voor de te animeren CSS properties. Een voorbeeld:

```
@keyframes zwalpen {
      0% {left: 100px;}
     40% {left: 150px;}
     60% {left: 75px;}
     100% {left: 100px;}
   }
```



Opgelet! gebruik geen aanhalingstekens rond het keyword, Webkit herkent die, maar de andere browsers niet, dus

NIET `@keyframes 'zwalpen'`
WEL `@keyframes zwalpen`

Hier worden 4 keyframes bepaald op 0%, 40%, 60% en 100% van de bewegingsduur waarbij de `left` property telkens een andere waarde krijgt.
Dat kunnen ook meerdere CSS properties zijn.

Je kan ook de sleutelwoorden `FROM` en `TO` gebruiken:

```
@keyframes zwalpen {
     FROM {left: 100px;}
     40% {left: 150px;}
     60% {left: 75px;}
     TO {left: 100px;}
   }
```

Voor elk keyframe kan je ook de `animation-timing-function` bepalen:

```
@keyframes zwalpen {
     FROM {left: 100px;animation-timing-function: ease-out;}
     40% {left: 150px;animation-timing-function: ease-in;}
     60% {left: 75px;animation-timing-function: ease-out;}
     TO {left: 100px;}
   }
```

De `animation-timing-function`'s zijn dezelfde als de `transition-timing-function`'s voor *transitions*.

Prefixes

De standaard properties en rules worden niet door alle browsers geaccepteerd, je moet dus soms gebruik maken van browser prefixes. Momenteel hebben FF19.2, Opera 12.4 en IE10 geen prefixes meer nodig, enkel *Webkit* browsers wel.

Doe dit niet alleen in de properties maar ook voor de `@keyframes` rule. Een volledige animatie voor Chrome ziet er dus zo uit:

```
h1 {
        -webkit-animation-duration: 3s;
        -webkit-animation-name: slidein;
      }

@- webkit-keyframes slidein {
        from {
          margin-left:100%;
          width:300%
        }

        to {
          margin-left:0%;
          width:100%;
        }
      }
```

Animaties properties

Als je voor een selector een *animation* instelt, kan je gebruik maken van deze properties:

`animation-name`

De `animation-name` property zet de 'naam' van de animatie: die moet overeenkomen met een IDENTifier van een `@keyframes` rule . Wordt die niet gevonden dan is er geen animatie. Het keyword `none` heeft hetzelfde effect: geen animatie

`animation-duration`

De `animation-duration` property is een geldige tijd: in '`s`' seconden of `'ms'` milliseconden. De waarde 0 betekent geen animatie.

`animation-timing function`

De `animation-timing-function`'s zijn dezelfde als de `transition-timing-function`'s voor transitions. Is de functie ook gebruikt in de *keyframes*, dan hebben die voorrang op de property in de selector.

`animation-iteration-count`

De `animation-iteration-count` property kan als waarde een geheel getal hebben, of de waarde `infinite` . De standaard instelling is `1`. Het bepaalt hoeveel keer de animatie herhaald wordt. Met `infinite` blijft de animatie oneindig doorlopen.

`animation-direction`

De `animation-direction` property bepaalt of de animatie bij elke onpare lus in omgekeerde richting moet spelen. Deze eigenschap kan de volgende waarden hebben: `normal | alternate`. De standaardwaarde is `normal`.

`animation-play-state`

De `animation-play-state` property geeft aan of de animatie momenteel bezig is. Deze eigenschap kan de volgende waarden hebben: `running | paused`.

`animation-delay`

De `animation-delay` property zegt wanneer de animatie zal starten. Als de waarde `0` is, start ze onmiddellijk. Als de waarde een tijd is, zal ze pas na die tijd opstarten. Als de waarde een negatieve tijd is, zal ze onmiddellijk starten maar het zal net zijn of die tijd al verstreken is in het verloop van de beweging.

`animation`

`animation` is de verkorte versie om een animatie in te stellen. De syntax is:

```
animation:      animation-name || animation-duration || ↵
                animation-timing-function || animation-delay || ↵
                animation-iteration-count || animation-direction
```

Deze voorbeelden werken enkel in FF en Chrome!
Om de uitleg wat kort te houden tonen we in onderstaande code voorbeelden enkel de standaard property, de prefixes worden echter wel gebruikt in de code. Ververs eventueel de pagina om de animation te zien.

Je kan deze voorbeelden in actie zien op de ondersteunende website.

resize

Een eenvoudige box wordt groter en kleiner. De code:

```
@keyframes bounce {
                0% {transform: scale(1.0); }
                25% {transform: scaleX(1.4); }
                50% {transform: scale(1); }
                75% {transform: scaleY(1.4); }
                100% {transform: scale(1.0); }
}
```

Je bemerkt dat de box tweemaal vergroot en tweemaal krimpt volgens de 5 keyframes.

bounce

Een bal kaatst binnen een container.

Hier is de animatie complexer: er is een horizontale beweging, een vertikale en een vervorming. De horizontale beweging is gescheiden van de rest door die beweging te plaatsen op een extra `div#doos`, die de "bal" bevat. Belangrijk is ook dat de "vaste" plaats van de bal deze is waar de beweging eindigt: daarom de negatieve waarden in de `translate`.

```
#doos {
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
        animation: ball-x 2.5s cubic-bezier(0, 0, 0.35, 1);
        }
#bal{
        width:50px;
```

```
        height:50px;
        position: absolute;
        bottom: 25px;
        left: 225px;
        border:2px solid red;
        border-radius:50%;
        background-color:#FF9900;

        transform-origin: 50% 50%;
        animation: ball-y 2s;
        }

@keyframes ball-x {
         /* horizontale beweging */
        0% { transform: translateX(-275px); }
        100% { transform: translateX(0px); }
}

@keyframes ball-y {
         /* vertikale bounce */
        0% { transform: translateY(-205px); animation-timing-function: ease-in; }
        40% { transform: translateY(-100px); animation-timing-function: ease-in; }
        65% { transform: translateY(-52px); animation-timing-function: ease-in; }
        82% { transform: translateY(-25px); animation-timing-function: ease-in; }
        92% { transform: translateY(-12px); animation-timing-function: ease-in; }

        25%, 55%, 75%, 87%, 97%, 100% { transform: translateY(0px); animation-timing-
  function: ease-out; }

         /* bounce vervorming */
        27%, 57%,77%,89% {transform: scale(1.2,0.8); }
        25%, 29%, 55%,59%, 75% ,79%, 87%,91%, 97%, 100% {transform: scale(1); }

}
```

Een animation (opnieuw) opstarten vanuit Javascript

De eenvoudigste manier om een animatie via een 'trigger' op te starten is de animatie in een `class` te stoppen en die `class` bij de trigger eerst te verwijderen en nadien opnieuw toe te voegen, hier via jQuery:

Verwijderen en nadien toevoegen van de `class` lijken voldoende maar zijn het niet: de timer moet terug op 0 gezet worden

```
$("#knop").click(function(e){
        $('#anim2').removeClass('anim-resize').animate({'nothing':null}, 1, function ()
  {$(this).addClass('anim-resize')});
  });
```

17.13Flex box

De CSS3 Flexible Box Layout Module , de "Flexbox" module, is een **volledig nieuwe manier om lay-outs te maken** (zonder tables, extra divs, margin collapse en floats).

Dit is goed want:

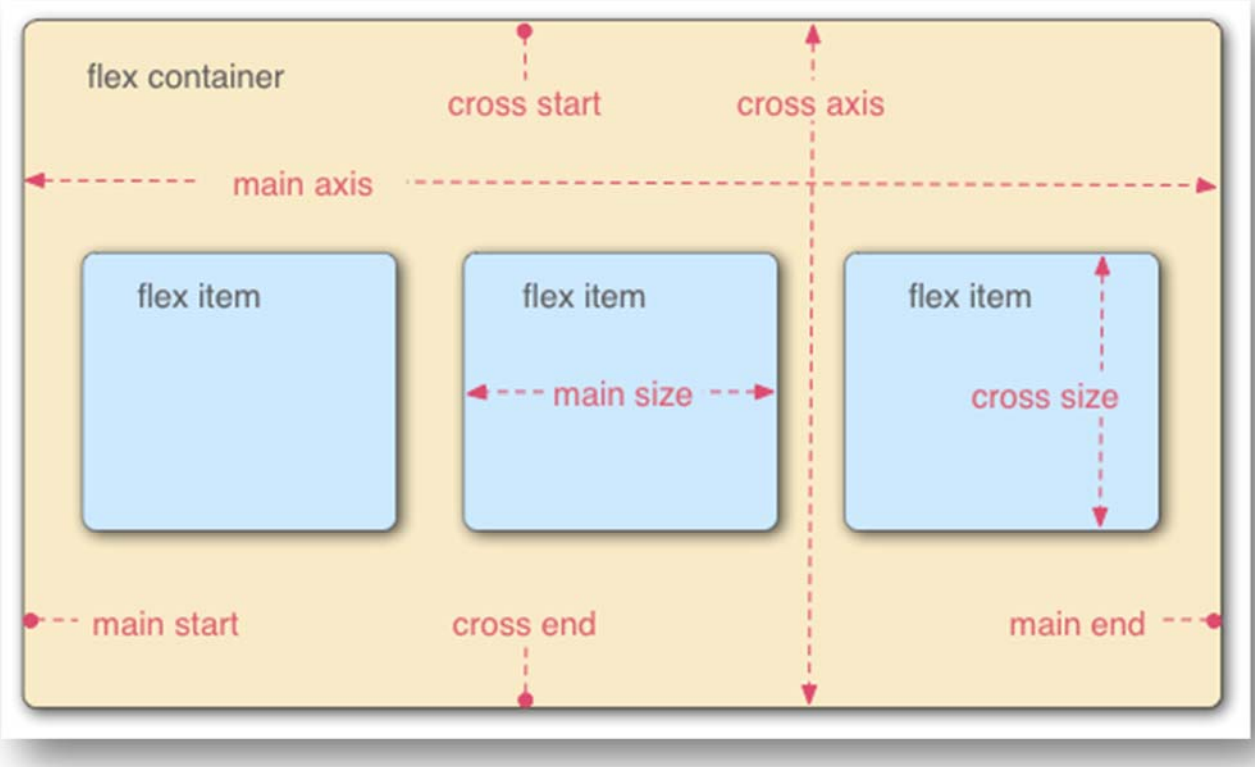
- het neemt complexiteit uit de HTML weg en brengt ze over naar de CSS: meer opmaak gescheiden van inhoud
- het biedt een oplossing voor lay-out op meerdere toestellen



De *Flexbox module* heeft de laatste jaren erg veel veranderingen ondergaan en browsers zijn doorheen een aantal testfases gegaan.
Er is sprake van een "oud" (`display:box, display:flexbox`) en een "nieuw" flexbox model (`display:flex`).
Wij gebruiken hier enkel het **nieuwe model** zoals het op de W3 specificatie staat.
De ondersteuning door de # browsers is zeer verscheiden: Opera12 & O Mobile, Chrome21>, BlackBerry10>. IE9, Safari6 en Firefox17 niet...

Modernizr test ondersteuning via de classes `flexbox` en `flexbox-legacy` (oud).

Terminologie



Flexbox (*flex container*)

is een element waarvan de *children* op een flexibele manier geordend worden.

Met de gekende property `display` maak je een flexbox:

```
display: flexbox | inline-flexbox
```

Voorlopig echter moet je ook een aantal prefix-waarden toepassen om het te doen werken in alle browsers:

```
.flex {
  display: -webkit-flexbox;
  display: -moz-flexbox;
  display: -ms-flexbox;
  display: flexbox;
}
```

Flex items

Alle onmiddellijke children van een *flex box* zijn automatisch *flex items*.
Een flex Item is onderhevig aan de lay-out gedicteerd vanuit de *flex box*.
Ook tekst die direct in een flexbox zit wordt automatisch een *anonymous flex item*

Assen

Het flexbox model is noch vertikaal (block elementen) noch horizontaal (inline elementen) georiënteerd, maar volgt de gespecifiëerde axes.

Een flexbox heeft 2 axes:

- de **main axis**: de as waarlangs de items elkaar opvolgen
- de **cross axis**: de as haaks op de *main axis*

De eigenschap `flex-direction` bepaalt de richting van de *main axis*.

De eigenschap `justify-content` bepaalt **hoe** *flex items* langs die main axis gerangschikt worden.

De eigenschap `align-items` bepaalt hoe alle *flex items* standaard gelay-out worden langs de *cross axis*.

De eigenschap `align-self` bepaalt hoe een enkel *flex item* gelay-out wordt langs de *cross axis* en overschrijft daarmee de standaard gezet door `align-items`

Richting

De **main start/main end** en **cross start/cross end** punten van een *flex container* zijn respectievelijk het begin/einde van de **flow** van de *flex items*. Ze volgen de *main axis* en de *cross axis* van de *flex box* volgens de *writing-mode* (left-to-right, right-to-left, etc.).

De `order` eigenschap bindt elementen aan een groep en bepaalt welk element eerst komt.

De `flex-flow` eigenschap is de verkorte schrijfwijze voor de `flex-direction` en `flex-wrap` eigenschappen om ze te lay-outen.

Lijnen

Flex items kunnen gelay-out worden op een enkele lijn of op meerdere lijnen. Dit wordt bepaalt door de eigenschap flex-wrap, die de richting bepaalt waarin de lijnen gestapeld worden.

Dimensies

De `min-height` en `min-width` properties hebben een nieuwe waarde, `auto` die de minimum grootte van een flex item instelt.

The `flex` eigenschap is de verkorte schrijfwijze van de eigenschappen `flex-basis`, `flex-grow`, en `flex-shrink` die de 'flexibility' van de flex items bepaalt.

Men spreekt van *positieve ruimte* als er ruimte op overschot is en *negatieve ruimte* als er ruimte tekort is.

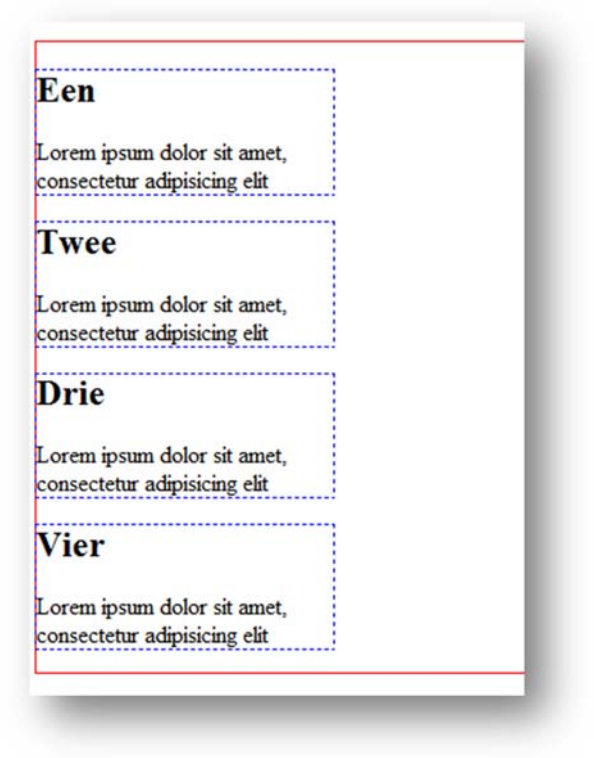
Veronderstel deze HTML:

```
<section id="fb1" class="flex">
  <div>
    <h2>Een</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit </p>
  </div>
  <div>
    <h2>Twee</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
  </div>
  <div>
    <h2>Drie</h2>
    <p> Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
  </div>
  <div>
    <h2>Vier</h2>
    <p> Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
  </div>
</section>
```

met de volgende standaards css:

```
.flex {
  border: 1px solid red;
}
.flex > div {
  width:200px;
  outline: 1px dashed blue;
}
```

dan krijgen we initiëel de volgende lay-out (in Opera):



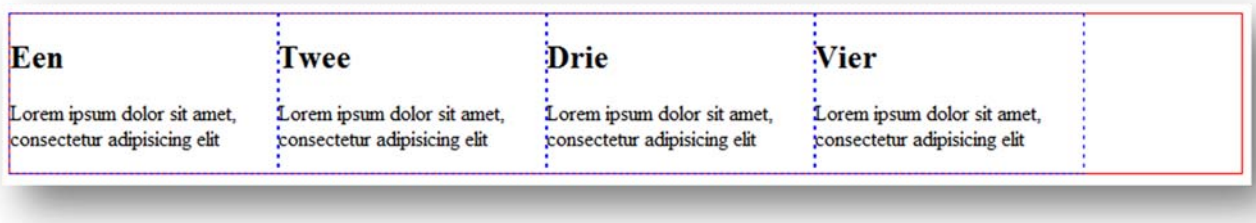
met andere woorden: het klassieke *basic box model*.

flex-direction en **flex-wrap**

Voegen we nu de volgende css toe:

```
.flex {
  display: -webkit-flex;
  display: -moz-flex;
  display: flex;

  -webkit-flex-direction: row;
  -moz-flex-direction: row;
  flex-direction: row;

  -webkit-flex-wrap: wrap;
  -moz-flex-wrap: wrap;
  flex-wrap: wrap;

  border: 1px solid red;
}
```

Resultaat:



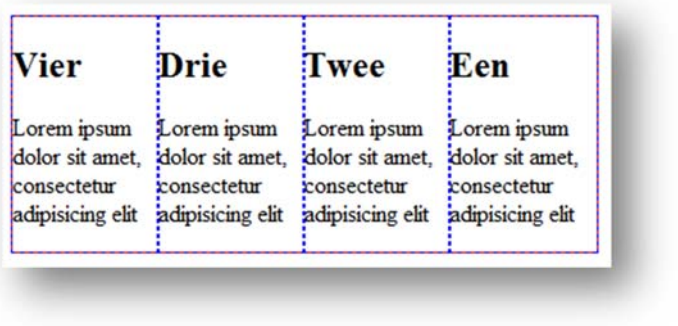
Bespreking:

- de property `display: flex` maakt van de `section` een *flexbox* en dus van alle children *flex items*
- de property `flex-direction:row` stelt de *main axis* in als een 'rij', dus horizontaal. De richting ervan is dezelfde als de schrijfrichting(hier links-rechts). De *cross axis* wordt dus vertikaal.
  Mogelijke waarden zijn `row` (standaard), `row-reverse, column, column-reverse`
- de property `flex-wrap: wrap` stelt de flexbox in als *'multi-line'*, dus als de parent container onvoldoende plaats biedt om de 4 `div`'s naast elkaar te plaatsen, gaat er eentje een volgende 'rij' innemen (probeer venster te verkleinen).
  Mogelijke waarden zijn `wrap` (standaard), `nowrap, wrap-reverse`
- je bemerkt dat we telkens de browser prefixes voor –webkit- en –moz- gebruiken. Op dit ogenblik past Opera de standaard toe en heeft geen prefix nodig, webkit werkt enkel met de prefix en Firefox ondersteun het niet maar we hopen op de toekomst…

Om dit beter te begrijpen wijzigen we eens deze laatste twee properties:

```
…
  -webkit-flex-direction:row-reverse;
  -moz-flex-direction:row-reverse;
  flex-direction:row-reverse;

  -webkit-flex-wrap:nowrap;
  -moz-flex-wrap:nowrap;
  flex-wrap:nowrap;
…
```

met dit resultaat:



De waarde `flex-direction:row-reverse` draait de volgorde van de *flex items* om t.o.v. de main axis. `flex-wrap:nowrap` zorgt ervoor dat de *flex items* niet naar een volgende lijn springen. Hoe ze zich dan gedragen: *fluid* of *fixed* hangt af van een paar andere eigenschappen

Probeer ook een `column, column-reverse`…

flex-flow

`flex-flow` Is de shorthand property voor `flex-direction` en `flex-wrap`. We keren terug naar de eerste instelling en schrijven het vorige korter als:

```
.flex {
  display: -webkit-flex;
  display: -moz-flex;
  display: flex;

  -webkit-flex-flow: row wrap;
  -moz-flex-flow: row wrap;
  flex-flow: row wrap;

  border: 1px solid red;
}
```

Opmerking:

- Chrome schijnt in sommige gevallen deze korte schrijfwijze niet te begrijpen: je moet `flex-direction` en `flex-wrap` apart schrijven

volgorde

De `order` property kan toegepast worden op een *flex item*. Normaal volgt de *order* de HTML flow: het eerste item van een flexbox heeft een standaard `order` van `0`. Een kleinere waarde zorgt ervoor dat het item **eerst gelay-out** wordt.
Een `order` heeft geen invloed op de DOM tree volgorde, het gaat enkel om lay-out.

Negatieve waarden zijn toegestaan. Bijvoorbeeld:

```
#fb1 > div:nth-child(2) {order:-1}
```

resulteert in:



Het tweede *child* van onze voorbeeld *flex box* wordt naar voor geschoven en komt in deze *direction* links te staan.

Flexibility

De hele opzet van flex boxes was de inhoud volledig flexibel (fluid) te maken zodat hun hoogte en breedte zich kan aanpassen.

Dit doen we met de eigenschap `flex` op de *flex items*.

`flex` is een shorthand property voor de properties `flex-grow`, `flex-shrink` en `flex-basis`.

De syntax is:

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

`flex-grow`:

is een getal (*flex grow factor*) dat aangeeft hoeveel een flex item 'zal groeien' t.o.v. zijn siblings als er (positieve) vrije ruimte verdeeld wordt. De standaardwaarde is 1.

`flex-shrink`:

is een getal (*flex shrink factor*) dat aangeeft hoeveel een flex item 'zal krimpen' t.o.v. zijn siblings als er (negatieve) ruimte tekort is. De standaardwaarde is 1.

`flex-basis` :

is de startwaarde voor de *main size* van het item vooraleer er vrije ruimte verdeeld of weggenomen wordt. De waarde is ofwel een lengte ofwel `auto`.

De waarde `auto` komt erop neer dat deze factor dezelfde is als de `width` van het item . Wordt deze property achterwege gelaten, dan is zijn standaardwaarde `0`

De waarde `none` voor `flex` is identiek als `0 0 auto`.

Voorbeelden waarbij we **dezelfde flex toepassen op alle flex-items**. Zo blijven ze gelijk in grootte en reageren identiek. Test uit door het venster te verkleinen/vergroten:

`flex: none` (idem aan `flex: 0 0 auto`)

```
#fb1 > div {
```

```
  width:200px;
  -webkit-flex: none;
  -moz-flex: none;
  flex: none;
}
```

resultaat: de items zijn **niet flexibel**, ze kunnen ook niet kleiner worden dan hun `width`.

`flex: initial` **(** zelfde als `flex: 0 1 auto`)

resultaat: de items zijn **niet flexibel**, maar kunnen kleiner worden dan hun `width`.

`flex: auto` **(** zelfde als `flex: 1 1 auto`)

resultaat: de items zijn **volledig flexibel**, en absorberen alle positieve/negatieve ruimte die beschikbaar is. De breedte van de items wordt berekend vanaf hun werkelijke width.

`flex: 1 1 0px`

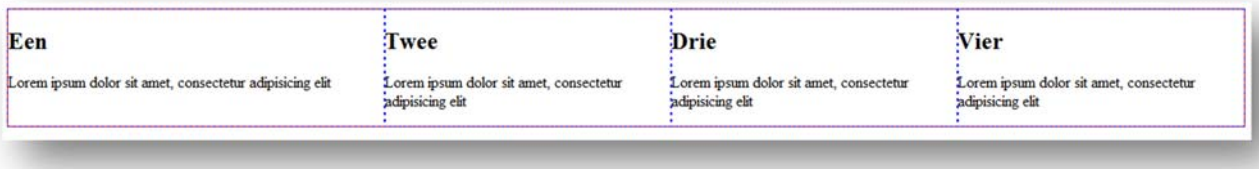
resultaat: de items zijn **volledig flexibel**, en absorberen alle positieve/negatieve ruimte die beschikbaar is, maar hun breedte en verdeling is absoluut.

Voorbeelden waarbij we **verschillende flex toepassen op de flex-items**. Ze zijn dan niet gelijk in grootte en reageren verschillend.

Het eerste item heeft een grotere flex-factor: `flex: 2 1 auto`, de andere zijn gelijk:

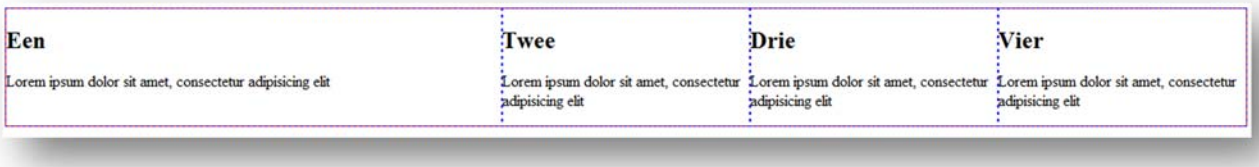
```
#fb1 > div {
  width:200px;
  -webkit-flex: 1 1 auto;
  -moz-flex: 1 1 auto;
  flex: 1 1 auto;
}

#fb1 > div:first-child {
  -webkit-flex: 2 1 auto;
  -moz-flex: 2 1 auto;
  flex: 2 1 auto;
}
```

Dit ziet er zo uit in Opera (breed venster):



De items zijn **volledig flexibel**. Bemerk dat bij een totale breedte rond de 800px, er weinig of geen verschil is tussen de items: hun startbreedte wordt bepaalt vanaf hun width. Het is pas als er extra positieve ruimte is dat het eerste item die meer absorbeert dan de andere items: hij wordt sneller breder.

Wijzigen we nu alle `flex-basis` waarden van `auto` naar `0` (`flex: 1 1 0` en `flex: 2 1 0`) dan krijgen we een heel ander beeld:



Hier is de breedteverdeling absoluut: de totale breedte wordt verdeeld in 5 delen waarvan 2/5 naar het eerste item gaan. Alle items zijn flexibel maar deze verhouding blijft bestaan.

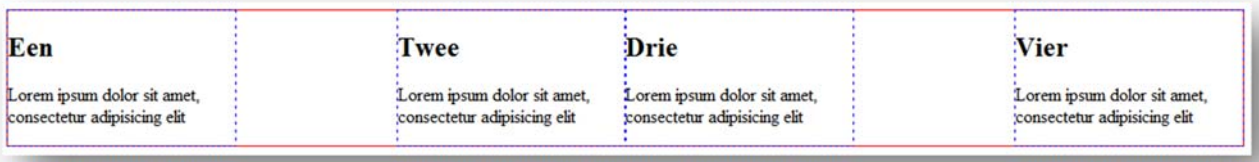
Uitlijnen met `auto` margins

De waarde `auto` van de `margins` property - die we kennen van het klassieke box model – kan in het flexbox model ook gebruikt worden om er 'groepen' items mee te maken.

Neem bijvoorbeeld een niet-flexibele instelling, vrije positieve ruimte en daarbij twee auto-margins:

```
#fb1 > div {
  width:200px;
  -webkit-flex: initial;
  -moz-flex: initial;
  flex: initial;
}
#fb1 > div:first-child { margin-right:auto; }
#fb1 > div:nth-child(3) { margin-right:auto; }
```

dan krijgen we drie groepen in de flex-items:



Uitlijnen op de main axis

De eigenschap `justify-content` van een *flex container* verdeelt de inhoud over de *main axis*. Mogelijke waarden zijn:

`flex-start | flex-end | center | space-between | space-around`.

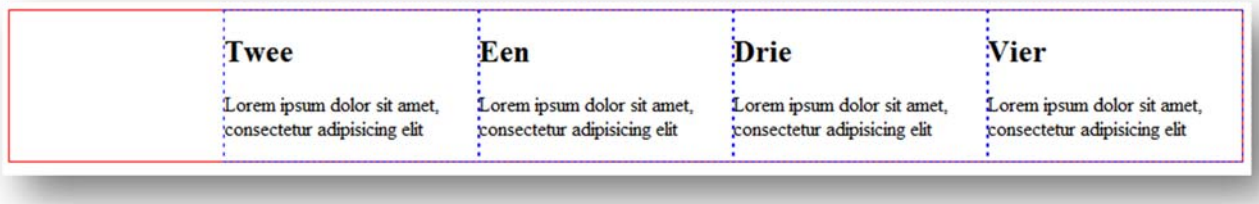
We proberen ze even met wat positieve ruimte op overschot:

```
.flex {
  …
  -webkit-justify-content: flex-start;
  -moz-justify-content: flex-start;
  justify-content: flex-start;
  …
}
```

`flex-start` (standaardinstelling)



`flex-end`



`center`



`space-between`



`space-around`



Uitlijnen op de cross axis

De utilijning op de *cross-axis* (haaks op de *main axis*) kan ingesteld worden:

- standaard met `align-items` op de ***flex box***
- individueel op de ***flex items*** met `align-self`

De eigenschap `align-items` van een *flex box* lijnt alle flex items uit over de *cross axis*. Mogelijke waarden zijn:

`flex-start | flex-end | center | baseline | stretch`

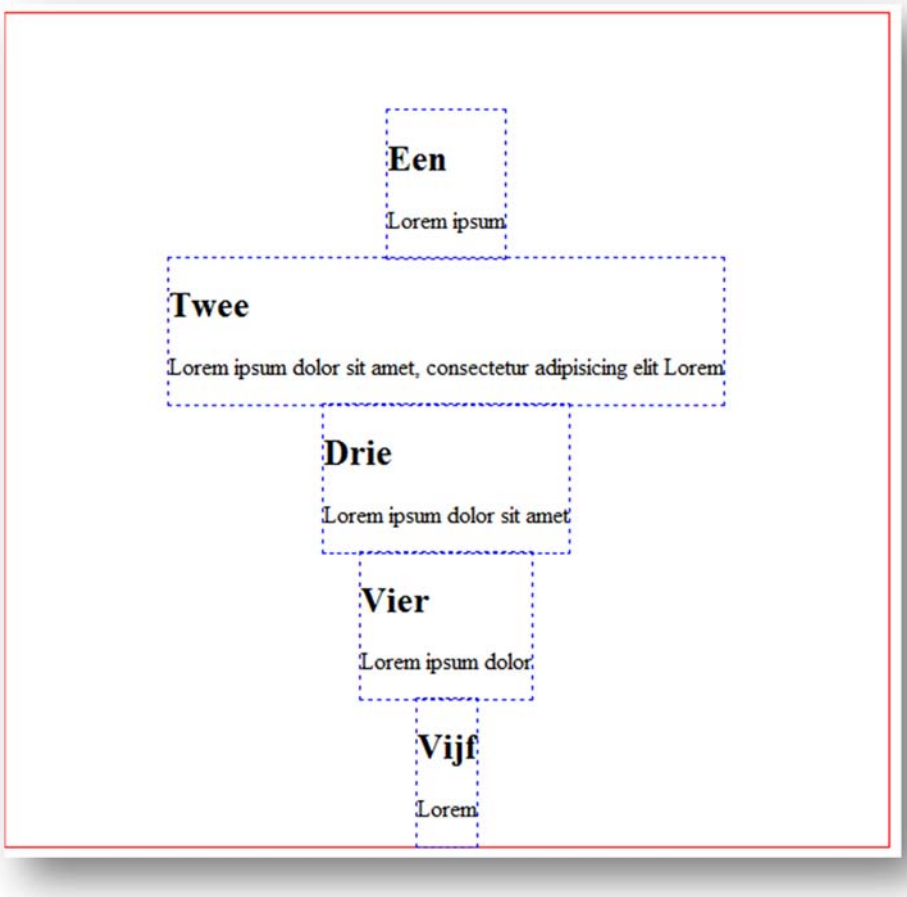
`stretch` is de standaard.

De eigenschap `align-self` van een *flex item* lijnt dit item individueel uit over de *cross axis*. Mogelijke waarden zijn:

`auto | flex-start | flex-end | center | baseline | stretch`

`auto` is de standaard en betekent dat `align-items` van de parent toegepast wordt of staat gelijk aan `stretch` als er geen waarde is.

In onderstaand voorbeeld hebben we een vertikaal gerichte flexbox die een vaste hoogte gegeven is (om demonstratieredenen) en waarvan de flex-items een `align-self` hebben:



De volledige CSS:

```
#fb2{
  display:-webkit-flex;
  display:-moz-flex;
  display:flex;

  -webkit-flex-direction:column;
  -moz-flex-direction:column;
  flex-direction:column;

  -webkit-flex-wrap: nowrap;
  -moz-flex-wrap: nowrap;
  flex-wrap: nowrap;

  -webkit-justify-content: flex-end;
  -moz-justify-content: flex-end;
  justify-content:  flex-end;

  -webkit-align-items: center;
  -moz-align-items: center;
  align-items: center;

  min-height:560px;
```

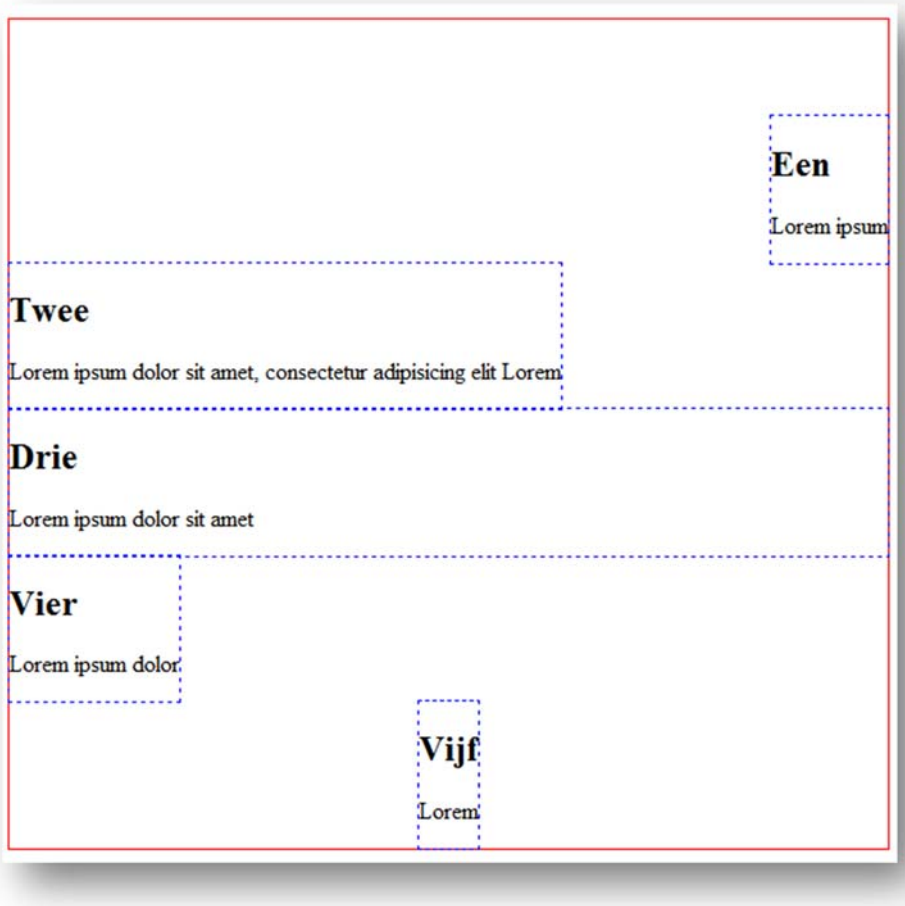
```
}
#fb2 > div{
  -webkit-flex: initial;
  -moz-flex: initial;
  flex: initial;
}
```

Je bemerkt de `flex-end` uitlijning op de *main-axis* (vertikaal) en de centrering er loodrecht op.

Als we de individuele *flex items* andere `align-self` waarden geven, dan kunnen we ze uitlijnen zoals we willen:

```
…
#fb2 > div:first-child{
  -webkit-align-self: flex-end;
  -moz-align-self: flex-end;
  align-self: flex-end;
}
#fb2 > div:nth-child(2){
  -webkit-align-self: baseline;
  -moz-align-self: baseline;
  align-self: baseline;
}
#fb2 > div:nth-child(3){
  -webkit-align-self: stretch;
  -moz-align-self: stretch;
  align-self: stretch;
}
#fb2 > div:nth-child(4){
  -webkit-align-self:  flex-start;
  -moz-align-self:  flex-start;
  align-self:  flex-start;
}
#fb2 > div:last-child{
  -webkit-align-self: center;
  -moz-align-self:center;
  align-self: center;
}
```

geeft:


Een
Lorem ipsum
Twee
Lorem ipsum dolor sit amet, consectetur adipisicing elit Lorem
Drie
Lorem ipsum dolor sit amet
Vier
Lorem ipsum dolor
Vijf
Lorem

18 MOBILE WEB DESIGN

18.1 Ontwikkelen voor mobile

Met de explosie van mobiele toestellen breekt een nieuwe periode in webontwikkeling aan. We kunnen uiteraard niet in een glazen bol kijken, maar we kunnen ons voorbereiden op de situatie nu en de toekomst. Aanpassen is de boodschap.

De uitdagingen voor de webontwikkelaar zijn veel en verscheiden:

- nieuwe operating systemen (Android, iOs, Win8, Symbian, Blackberry)
- veel nieuwe browsers
- een niet te stoppen stroom van nieuwe hardware met
    - een veelheid aan schermresoluties
    - verschillende bandbreedtes
- concurrentie van snelle Apps

Voor websites zijn de mogelijkheden:

- een **dedicated mobile domain**: *m.vdab.be* of *vdab.mobi* waar een specifieke aangepaste site staat
- een **responsive website** die alles aankan: desktop en mobiel

Het grote nadeel van een *dedicated site* is dat je dan twee websites te onderhouden hebt, een ander nadeel kan zijn dat je de dedicated site ontwikkeld hebt met het oog op één toestel: wat doe je als die van de markt verdwijnt?

Voor welke **problemen** moeten we proberen een oplossing te bieden?

- **bandbreedte**:
    - het aantal HTTP request beperken
    - de grootte van de libraries en beelden zo klein mogelijk houden
- **scherm**:
    - aangepaste lay-out voor:
        - grote verschillen in resolutie
        - schermrotatie
    - *touch* betekent nood aan grotere knoppen, lijsten, etc…
    - aangepaste navigatiesystemen
    - CSS:
        - ondersteunen de browsers je CSS?
        - sommige CSS3 features veroorzaken verhoogd stroomverbruik…
- **location**:
    - gebruik maken van de locationservices van het toestel
- **javascript**:
    - is het gebruik van een grote library nodig als je er maar een miniem deel ervan gebruikt?
    - ondersteunt de browser je script?
    - voorzien op *touch events*

Het zijn vooral de eerste twee items waar we zullen moeten mee rekening houden.

18.2 browsers

Plots ziet het browserlandschap er heel anders uit dan voorheen: naast de klassieke desktopbrowsers komen de hardwarefarikanten met een eigen aangepaste browser en andere firma's zien hun kans schoon om in de markt te raken.

we onderscheiden:

- **desktop browsers**: de klassieke client-side browsers:
  Chrome, FireFox, IE, Opera, Safari
- **proxy (cloud) browsers**: tunnelen alle verzoeken naar één proxy server en sturen een gecomprimeerd antwoord terug: *Opera mini, UCWeb, Dolphin*
  Deze browsers zijn *thin clients* die geen CSS of JS kunnen verwerken maar elke client-side wijziging doorsturen naar de proxy. Wat de "browser" ontvangt is zelf geen HTML. Er wordt gebruiksvriendelijkheid opgeofferd voor bandbreedtebesparing (tot 80%)
- **mobile browsers**: specifiek voor een OS beschikbaar:
  Android browser, Rockmelt, Opera Mobile, skyFire,…
  Ze hebben meestal alle features van een desktop browser en zijn speciaal aangepast aan een mobile omgeving. Het aantal mobile browsers is een veelvoud van de desktop browsers

Wat zijn de mogelijkheden van al deze browsers? Ondersteunen ze CSS3? Javascript?

Het huidige marktaandeel van deze browsers kan je nagaan op http://gs.statcounter.com/. Hou er rekening mee dat in sommige werelddelen de een al veel populairder is dan de andere. Je kan natuurlijk niet alles testen, maar de belangrijkste spelers moet je zeker gebruiken om te testen en wie vandaag vooraan staat is dat morgen niet zeker…

18.3 viewport

Moderne mobile browsers kunnen zonder problemen een volledige "desktop" pagina tonen zonder dat er zich enig problem voordoet: ze verkleinen de pagina zodanig dat hij perfect op het scherm past. Moderne toestellen hebben ook een hoge resolutie!

De gebruiker kan dan makkelijk inzoomen en de site slepen in alle richtingen.

Het is echter zo dat je de aanpassen wat meer onder controle wil houden: zo kan het gebeuren dat je een mooie "responsive site" gemaakt hebt maar dat de browser de volledige desktopversie gewoon "zoomt" op zijn scherm – zodat al je voorzieningen voor niets waren…

Daarvoor gebruiken we de `viewport` `meta` tag:

```
<meta name="viewport" content="width=device-width" />
```

Dit past de paginabreedte aan aan de breedte van het mobiele scherm zodat 1 paginapixel overeenkomt met 1 schermpixel.

Je kan dus als je wil exacte waarden opgeven omdat je een welbepaalde breedte wil:

```
<meta name="viewport" content="width=460px" />
```

Je kan niet enkel de `width` maar ook de `height` gaan gebruiken:

```
<meta name="viewport" content="width=device-width,height=device-height" />
```

Daarnaast kan je ook laten inzoomen of niet. De beste standaardinstelling is natuurlijk niet ingezoomd:

```
<meta name="viewport" content="width=device-width,initial-scale=1" />
```

18.4 Responsive Web Design

"Responsive web design" (**RWD**) is een term eerst gebruikt door *Ethan Marcotte* in een artikel op de *A List Apart* website. Het gaat erom dat een website zo ontworpen is dat hij zichzelf aanpast aan verschillende soorten toestellen en de gebruiker **dezelfde site** op de beste mogelijke manier voorschotelt, zonder fouten, problemen en met een minimum aan scrollen, zoomen en schuiven.

RWD stoelt op volgende technieken:

- een **fluid lay-out**
- **fluid images**
- **media queries**

RWD is een *manier van ontwerpen*, je kan het niet onmiddellijk toepassen op een bestaande website. Ook de **inhoud** en de **structuurelementen** van de site moet bekeken worden in het licht van deze methode: werken ze wel optimaal in een responsive omgeving? Dit geldt zeker voor de navigatie.

Wat we ook steeds in het oog moeten houden is dat we ***Future-proof*** proberen te werken: pin je niet vast op één populair toestel, op één techniek, morgen is de situatie anders dan nu. Onze sites moeten ook *flexible* zijn naar de toekomst toe.

Terminologie

Er wordt in de webwereld nogal wat gediscussiëerd over het *geslacht van de engelen…* Is het nu *Adaptive design* of *Responsive design???*

Hier is wat wij denken dat er mee bedoeld wordt:

- ***fluid*** of ***flexible***: betekent dat een box elastisch is en dus van breedte/hoogte verandert
- ***adaptive***: houdt een verandering in: de lay-out wordt anders/ er komt een andere figuur tevoorschijn
- ***responsive***: reageert op een wijziging: een responsive design bevat zowel flexible als adaptive elementen

18.4.1 Design principes

Omdat we te maken hebben met zoveel verschillende toestellen/OS kunnen we problemen verwachten, daarom gebruiken we best een design strategie.

Twee belangrijke algemene programmatieprincipes :

- **Graceful Degradation**:

  Graceful degradation (GD) of *fault tolerance* is een watervalsysteem dat verder kan werken bij het optreden van fouten of tekortkomingen. Je voorziet oplossingen voor situaties waar gebruikers problemen ondervinden.
  GD is een ***top-down*** benadering.
  In een *graceful degrading design* bouw je een website voor het beste maar voorzie je ook voor het mindere.
  Enkele toepassing van GD zijn het gebruik van `alt` attributen voor images, en het gebruik van hacks voor oudere browsers.

- **Progressive Enhancement**:

  Progressive enhancement (PE) is een ***bottom-up*** benadering die werkt met gelaagdheid: men start vanuit een **basis** die werkt in **alle** situaties en bouwt daarna extra functionaliteit en kwaliteit in laagjes op.
  In RWD is het principe ***Mobile First*** hiervan een toepassing: men start met een website voor kleine, eenvoudige mobieltjes en bouwt daarop verder voor tablet en smartphones en laatst voor desktops en meer.
  Ook *Unobtrusive Javascript* is een toepassing van PE.

In *RWD* is *graceful degradation* de minder goede oplossing: er zijn veel meer low-range toestellen (GSM's) dan upper-range (smartphones). Eenvoudige toestellen hebben niet de geheugencapaciteit, bandbreedte en ondersteuning die top-range toestellen hebben.

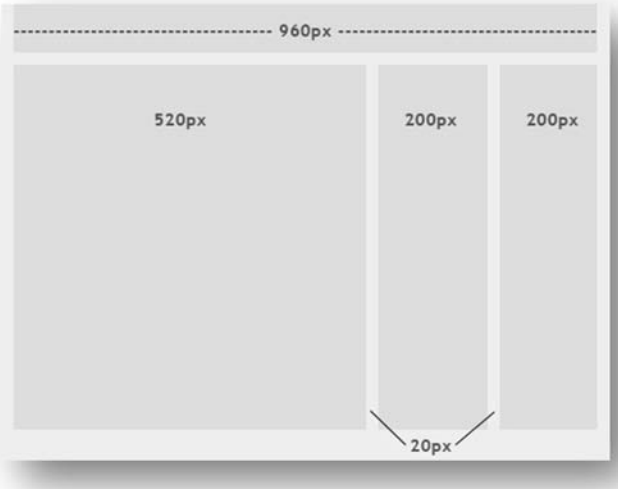
18.4.2 Fluid vs Fixed lay-outs

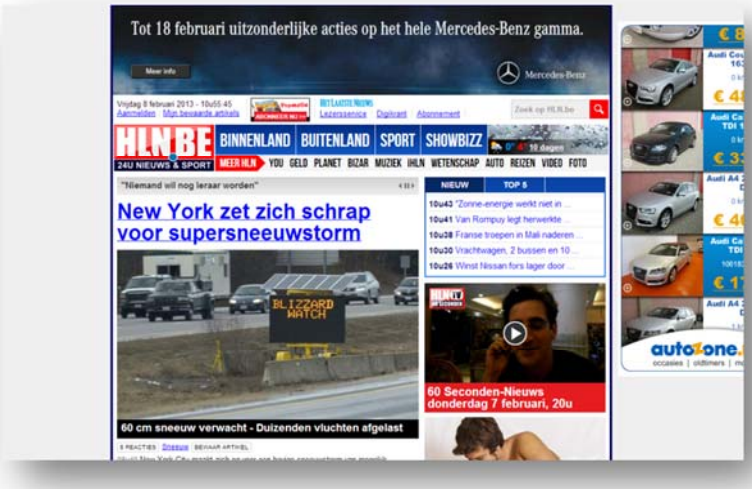


Een *fluid lay-out* is een essentiëel onderdeel van RWD.

In de projecten ben je al in aanraking gekomen met een aantal lay-out vormen. We zetten ze even op een rijtje:

Fixed lay-out: gebruikt vaste waarden in **pixels** als breedte.





Voordelen:

- lay-out is volledig onder controle en beweegt nooit
- je kan exact de padding, border en marges uitrekenen: je hebt houvast

Nadelen:

- past zich niet aan:
  - groter scherm: veel verloren ruimte
  - kleiner scherm: delen zijn niet zichtbaar: scrollen
- nieuwe inhoud (vb beelden) verplicht aangepast aan beschikbare ruimte
- problemen als de gebruiker de tekstgrootte meer dan "normal" zet, omdat de dimensies er niet kunnen in voorzien

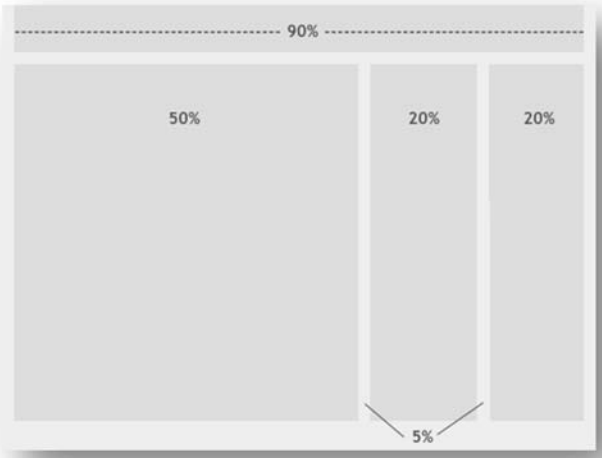
Gecentreerde fixed lay-outs werden vroeger zeer veel gebruikt om ervoor te zorgen dat gebruikers met een klein standaardscherm (bv. 800px) alles op hun scherm kregen. Op grotere schermen was er links en rechts dan veel witruimte, maar er waren nooit problemen.

Een voorbeeld: KMI.be

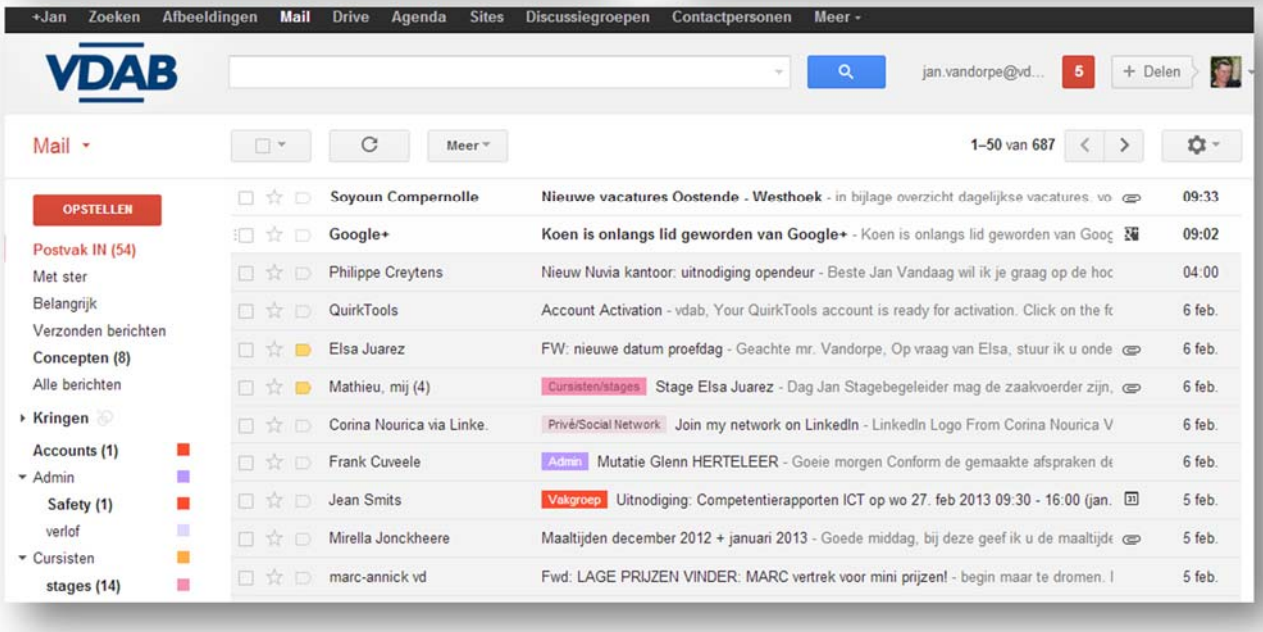
Fluid lay-out (*flexible, liquid*) gebruikt **percentages** als breedte.

In principe is een pagina **zonder breedte** volledig *Fluid*: alle inhoud past zich aan aan elke breedte: de eerste pagina's op het WWW zijn daar voorbeelden van.

Een modern voorbeeld van een fluid lay-out is Gmail.







Een fluid lay-out is afhankelijk van de breedte van zijn parent container en past zich dus ook aan aan de breedte van het scherm.

Voordelen:

- de breedte is altijd proportioneel aan de parent of het scherm
- beschikbare ruimte kan maximaal ingenomen worden
- je kunt een `max-width` instellen om ervoor te zorgen dat het niet té breed wordt

Nadelen:

- de lay-out is moeilijker te controleren:
  - groter scherm: lay-out strekt te breed

    - kleiner scherm: lay-out kan volledig verkeerd lopen (*floats*)
- oudere browsers ondersteunen `max-width` niet

Elastic lay-out gebruikt `em` of `rem` als breedte-eenheid

Een elastic lay-out vergroot/verkleint als de gebruiker de lettergrootte wijzigt.

Dit type is veel moelijker te maken maar kan zeer gebruiksvriendelijk zijn.

Een voorbeeld: *Designbit.co.uk*

Combinaties:

elke combinatie van bovenstaande is mogelijk.

Voorbeelden : *Wikipedia* en *Amazon* gebruiken een fixed kolom links en de rest fluid.

18.4.3 images in RWD

Met beelden doen zich twee problemen voor in een mobiele omgeving:

- een normaal `img` element past zich niet aan aan een *fluid lay-out*
- hoge resolutiebeelden zijn grote bestanden en nemen dus veel bandbreedte in, een probleem op kleine toestellen. Elke vermindering van downloadtijd is "winst".

Fluid image

Hoe kan je een beeld eveneens fluid maken in een fluid lay-out?

Hieronder zie je twee images in een flexible container die *verkleind* wordt:

- als gewoon `img` element
- als `background-image` (via css) op een block element

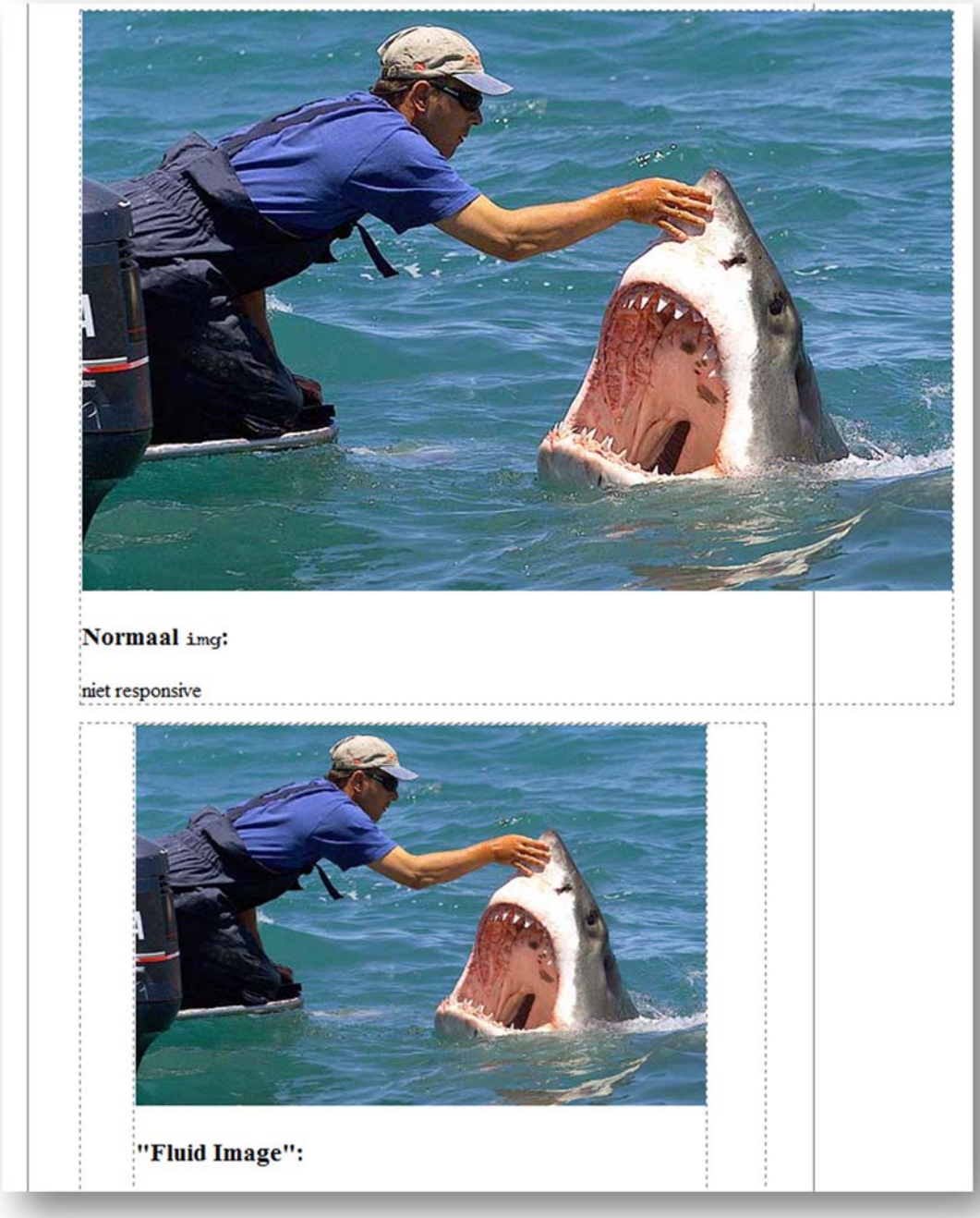


- we bemerken dat het gewone `img` element zijn container te buiten gaat; het beeld blijft even groot
- gebruikt als *background* blijft het ook zijn grootte behouden maar schuift weg achter de randen van het element. Als dat voor het ontwerp geen probleem is, kan je ook op deze manier tewerkgaan

We kunnen een image heel gemakkelijk *fluid* (flexible) maken door er de volgende css op toe te passen:

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

Het resultaat is dat het beeld zich aanpast aan de dimensies van de container:



Opmerkingen:

- het beeld kan niet *groter* worden: zijn echte formaat is het maximum. Jij moet dus een voldoende groot beeld voorzien
- de verkleining gebeurt door de browser en dat zal de kwaliteit van het beeld verminderen.

Adaptive (Responsive) images

Zou het niet beter zijn een ander beeld te laden in verschillenden situaties?

Bijvoorbeeld een hoog resolutiebeeld voor desktops en een steeds lagere res (en kleiner) beeld voor kleinere schermen.

Dat zou inderdaad het beste zijn, maar zoals de zaken er nu voorstaan kunnen we dat met HTML en CSS alleen niet oplossen.

De beste oplossing zou een herziening inhouden van het `img` element met verschillende `src` attributen.

Oplossingen maken ofwel gebruik van Javascript of van een *.htaccess* bestand en php scripts, iets wat buiten de scope van deze cursus valt.



Vergeet niet dat het ***optimaliseren*** van je beelden in een gespecialiseerd grafisch programma de grootste "winst" van allemaal kan betekenen!

Gebruik *Sprites*

Een ***Sprite*** is een samengesteld beeld, een beeldbestand dat bestaat uit tientallen/honderden aparte beeldjes. Het wordt typisch gebruikt in Frameworks (zoals jQuery) om de talloze icoontjes toe te passen.



ui-icons.png

We nemen één van de jQuery stylesheets als voorbeeld.

Hier is de sprite opgebouwd als een raster van 16px X 16px icoontjes. Om een dergelijk elementje te maken neem je een container (`button, a, span, div`) en past er via een CSS `class,` de *sprite* als background-image op toe:

```
.ui-icon {
  width: 16px;
  height: 16px;
  background-image: url(images/ui-icons_222222_256x240.png);
  }
```

daarna bepalen we via het raster welk icoontje we zullen te zien krijgen, bijvoorbeeld, het ikoontje linksboven krijgen we door er een tweede `class` op toe te passen:

```
.ui-icon-carat-1-n { background-position: 0 0; }
```

en het driehoek-icoontje tweede rij, vierde kolom krijgen we met

```
.ui-icon-triangle-1-s { background-position: -64px -16px; }
```

Je moet het infeite zien als een blad papier waar een gaatje van 16px inzit en dat je verschuift over het onderliggende beeld.



De "winst" die je maakt met het gebruik van één sprite tegenover 200 aparte icoonbestandjes is enorm!

Dergelijke sprites worden meestal gemaakt door de webdesigner, maar je kan gemakkelijk je eigen sprites samenstellen via een online tool zoals de **CSS Sprite generator** op http://nl.spritegen.website-performance.org/

Er zijn ook heel wat gratis sprites te vinden op het net.

gebruik HTML special characters

Gebruik HTML special characters voor kleine ikonen, vb `★` voor ★

gebruik data URI's

Nog een manier om bandbreedte te sparen is gebruik te maken van het data-URI protocol in een `img` element.
Het is een schema dat je toelaat beelden inline te gebruiken net alsof ze uit een extern bestand kwamen. Die gegevens zitten in een `base-64-encoded` string.

```
<img width="11" height="14" src="
lhCwAOAMQfAP////7+/vj4+Hh4eHd3d/v7+/Dw8HV1dfLy8ubm5vX19e3t7fr
6+nl5edra2nZ2dnx8fMHBwYODg/b29np6eujo6JGRkeHh4eTk5LCwsN3d3dfX
13Jycp2dnevr6////yH5BAEAAB8ALAAAAAALAA4AAAVq4NFw1DNAX/o9imAsB
tKpxKRd1+YEWUoIiUoiEWEAApIDMLGoRCyWiKThenkwDgeGMiggDLEXQkDoTh
CKNLpQDgjeAsY7MHgECgx8YR8oHwNHfwADBACGh4EDA4iGAYAEBAcQIg0Dk
gcEIQA7" alt="File Icon">
```

Geeft in een HTML pagina dit ikoontje

Data URI's worden ondersteund door de meeste browsers.

Een voorbeeld van gebruik in CSS:

```
.gestreept {
  width: 100px;
  height: 100px;
```

```
  background-image:
         url("
  HIJBQA7");
  border: 1px solid gray;
  padding: 10px;
}
```

Resulteert in een gestreepte achtergrond.

- Voordelen:
  - minder HTTP Requests
  - sneller in sommige gevallen, zeker bij HTTPS
  - gebruik in HTML en in CSS
- Nadelen:
  - herhaling van data voor gebruik dezelfde images (ikoontjes)
  - geen caching mogelijk in HTML, wel in CSS want stylesheets worden altijd gecached
  - aanmaak van de code vergt tijd en moet opnieuw gedaan worden bij elke wijziging
  - `base64` code ongeveer 33% groter dan oorspronkelijk data. Grootte beperking afhankelijk van de browser

Er zijn een aantal websites waar je een beeldbestand kan converteren naar base64 zoals dataurl.net/#dataurlmaker

18.4.4 media queries

Met CSS2 kan je het **Media Type** koppelen aan een stylesheet met het `media` attribuut:

```
<link rel="stylesheet" media="screen" href="screen.css" />
<style type="text/css" media="print"></style>
```

of een `@import` doen voor een medium:

```
@import url(sound.css) speech
```

of een `@media` rule in het stylesheet zetten:

```
@media screen{
  font:Arial
}
@media print {
  font:Times new Roman
}
```

Voor mobiel web design is dat onvoldoende, daarom werd deze spec uitgebreid in

CSS3 en kunnen we nu **meer waarden**, maar ook **expressies** gebruiken: *media queries*.

Waarden

CSS3 erkent de volgende waarden voor het `media` type:

`all, embossed, speech, braille, handheld, print, projection, screen, tty, tv`

Het media type `handheld` was al correct in CSS2 maar voldoet niet meer aan de huidige noden. Daar gebruiken we nu *media queries* voor.

Expressies

Een `media` attribuut met een *expressie* als waarde noemt men een **media query**.

In een expressie mag je de sleutelwoorden `AND`, `NOT` en `ONLY` (niet hoofdlettergevoelig) gebruiken samen met ***media features***.

Enkele voorbeelden van media queries gebruikt op verschillende manieren:

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

@import url(color.css) screen and (color);

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

@media (orientation: portrait) { … }

@media screen and (color), projection and (color){ … }
```

De waarde `all` mag weggelaten worden.

Het keyword `ONLY` wordt gebruikt om stylesheets te verbergen voor browsers die geen media queries aankunnen:

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

Gebruik zo weinig mogelijk `NOT`, meestal kan het anders:

```
@media not screen and (color), print and (color)
```

Betekent: niet voor een kleurenscherm, wel voor een kleurenprinter. Het had evengoed weggelaten kunnen worden.



Media queries laten je eerder de **mogelijkheden** van een toestel/toestand testen i.p.v. het type toestel. De combinatie van features (schermbreedte, orientatie, kleur,

etc..) noemen we verder een *toestand*.

media features

Een *media feature* heeft een waarde die je kan testen in je expressie. De expressie kan het feature ook testen zonder waarde, dus enkel op het bestaan ervan,
hier bijvoorbeeld of het een kleurenscherm betreft:

```
@media screen and (color)
```

De meeste *features* mag je ook combineren met `min-` en `max-` om zo teveel `AND` 's te vermijden.

`width`

een lengte (`em, px, pt, %,...`). Is de **breedte van de *display area***, bv. het browservenster of de printzone van een pagina voor print media.

`height`

ook een lengte. Is de **hoogte** van de *display area*, bv. het browservenster.

```
@media screen and (min-width: 400px) and (max-width: 700px) { … }
```

In bovenstaand voorbeeld filtert de media query enkel vensters uit tussen de 400 en de 700 pixels breed.

`device-width`

ook een lengte. Is de **breedte van de output device**, bv. het volledige scherm van het toestel of de volledige pagina voor print media.

`device-height`

ook een lengte. Is de **hoogte** van de *output device*, bv. het volledige scherm van het toestel of de volledige pagina voor print media.

```
<link rel="stylesheet" media="screen and (device-height: 600px)" />
```

In bovenstaand voorbeeld filtert de media query enkel schermen uit die exact 600 pixels hoog zijn.

`orientation`

Mogelijke waarden zijn `portrait | landscape`. Een medium is *portrait* als de `height` gelijk of groter is dan de `width`.

```
@media all and (orientation:landscape) { … }
```

Hier filtert de media query alle media uit die horizontaal georiënteerd zijn.

aspect-ratio

Waarde een getal. Is de verhouding tussen `width / height`.

device-aspect-ratio

Waarde een getal. Is de verhouding tussen `device-width / device-height`.

```
@media screen and (device-aspect-ratio: 1280/720) { … }
```

Hier filtert de media query alle schermen uit met een verhouding van 1280/720. Dit is natuurlijk ook gelijk aan de vereenvoudigde breuken, zoals 16/9.

color

Waarde een geheel getal. Is het aantal bits per kleur op de output device (*bit-depth*). Als het toestel geen kleurenscherm heeft, is de waarde 0. Vroegere PC schermen (VGA) hadden een bit-depth van 8, tegenwoordig gaat dat veel hoger.

```
@media all and (min-color: 24) { … }
```

Hier filtert de media query alle toestellen met de feature `color` eruit die minimum 24 bits per kleur bezitten.

color-index

Waarde een geheel getal. Is het aantal kleuren in de *colour lookup tabel* van het toestel. Bijvoorbeeld een VGA scherm heeft 256 kleuren in zijn tabel.

monochrome

Waarde een geheel getal. Is het aantal bits per pixel in een monochrome frame buffer. Als het toestel geen monochroom toestel is, is zijn waarde 0.

```
<link rel="stylesheet" media="print and (color)" href="http://…" />
<link rel="stylesheet" media="print and (monochrome)" href="http://…" />
```

Hier worden twee stylesheets gebruikt: één voor een kleurenprinter en één voor een zwart-wit printer.

resolution

Waarde een geldige resolutie `'dpi','dpcm'`.

```
@media print and (min-resolution: 118dpcm) { … }
```

Hier geldt het stylesheet enkel voor printtoestellen met een minimum resolutie van 118 dots per centimeter.

`scan`

Mogelijke waarden zijn `progressive | interlace`.
Enkel van toepassing op `tv` media. Het bevat de scan methode van zo'n toestel.

`grid`

Waarde een geheel getal. Wordt gebruikt om te weten te komen of het toestel 'grid' of 'bitmap' gebaseerd is. Een grid gebaseerd toestel, bijvoorbeeld een eenvoudig handtoestel met een fixed font, is de waarde 1, anders is de waarde altijd 0.

```
@media handheld and (grid) and (max-width: 15em) { … }
```

Voor ons zijn uiteraard de features die te maken hebben met het scherm het meest belangrijk: `width/height, device-width/device-height, orientation`,…

Toepassen

Media queries kunnen we gebruiken om *aparte stylesheets* te koppelen aan een *toestand*:

```
<link rel="stylesheet" type="text/css" media="all" href="desktop.css" />
<link rel="stylesheet" type="text/css" media="only screen and (max-device-width: 480px)"
  href="small-device.css" />
```

Hier wordt een "standaard" stylesheet geladen die "alle" media aankan en daarna een speciaal *voor-kleine toestellen-aangepast* stylesheet. Dit is perfect OK en zal goed werken, maar het heeft het nadeel dat je twee *http requests* richt naar de server: dit vertraagt het laden van de pagina.

Als we in het vorige stylesheet *desktop.css* een `@import` statement invoegen

```
@import url("small-device.css") only screen and (max-device-width: 480px);
```

bereiken we hetzelfde resultaat, maar dezelfde opmerking geldt: twee http requests.

Het is beter alle statements in één stylesheets te zetten: een stylesheet dat alle situaties voorziet.

Niet alle browsers ondersteunen mediaqueries:

- IE8<
- oudere Blackberry browsers
- Opera Mini (gedeeltelijk)

Modernizr heeft een ingebouwde test voor MQ.

Als je MF gebruikt zullen deze browsers de eenvoudige weergave van je site gebruiken, dus dat is te aanvaarden maar op een groot scherm zal dat resulteren in één kolom, een niet zo mooi resultaat dus.

18.4.5 Hoe media queries gebruiken

Je kan media queries (MQ) op een aantal *manieren* toepassen. Er is heel wat discussie op het net over wat de beste strategie is, elk heeft voor- en tegenstanders.

top-down of *Desktop First*

In ons voorbeeld bekijken we de styles voor een 'wrapper', een container die inhoud bevat:

```
body { font-family:Verdana, Geneva, sans-serif; }
.wrapper {
      width: 60%;
      float: left;
}
@media only screen and (max-width: 600px) {
    .wrapper {
             width: 100%;
             float: none;
        }
}
```

Bespreking:

- de opmaak zonder MQ is deze voor 'alle' breedtes, dus ook voor de desktop de breedte van de `.wrapper` wordt beperkt en hij wordt links ge-float
- de MQ `max-width` selecteert schermen die kleiner zijn dan 601px, dus mobiele toestellen: de breedte van de `.wrapper` wordt op 100% gezet en de `float` ongedaan gemaakt

Deze manier van werken gaat uit van opmaak gericht op grote schermen. Dat betekent dat we voor kleinere schermen deze opmaak moeten wijzigen en dikwijls expliciet de properties op hun standaard-instelling moeten terugzetten (vb. `float:none`).

Desktop First maakt vooral gebruik van `max-width` selecties.
Desktop First is een toepassing van het *Graceful Degrading* principe.

gelaagd

als een uitbreiding van het vorige vinden we ook stylesheets met gelaagde media queries, in deze aard:

```
@media screen and (min-width: 800px) {
/* desktops*/
}
@media screen and (min-width: 480px) and (max-width: 800px) {
   /* tablet landscape */
}
```

```
@media screen and (min-width: 400px) and (max-width: 480px) {
/* tablet portrait */
}


@media screen and (max-width: 400px) {
   /* phone */
}
```

Deze benadering is eveneens *desktop first*, maar probeert te voorzien in meer situaties. Hier worden `min-width` en `max-width` samen gebruikt.

Met deze methode kan je je afvragenwaar je de *breakpoints* gaat leggen? en hoeveel "lagen" voorzie je?

Deze benadering mag dan *full proof* lijken, het houdt het risico in dat je je op bepaalde toestellen gaat richten. In een sterk evoluerende markt is dat geen goede strategie.

bottom-up of *Mobile First*

```
body { font-family:Verdana, Geneva, sans-serif; }

@media only screen and (min-width: 600px) {
    .wrapper {
             width: 60%;
             float: left;
        }
}
```

Bespreking:

- de standaard CSS-instellingen worden toegepast voor alle situaties (`width:100%, float:none`). Er is geen CSS voor nodig. Een MQ is ook onnodig.
- de MQ `min-width` richt zich op de desktops. De breedte van de `.wrapper` wordt beperkt en hij wordt links ge-float

Deze manier van werken is korter en eenvoudiger. Ze maakt meer gebruik van de *cascade* en richt zich eerst op de eenvoudige situatie: de kleinste schermen, de minst complexe lay-out. Daarna wordt er opgebouwd en er moet minder ongedaan gemaakt worden.

Een *mobile-first* benadering gebruikt enkel `min-width` selecties.



"Mobile First" is een term gebruikt door *Luke Wroblewsky* in zijn boek Mobile First.

Mobile First (MF) is *progressive enhancement* toegepast voor web design:

- start met de belangrijkste inhoud en ontwerp éérst voor mobiles en
- bouw daarna op voor smartphones, tablets en desktop.

MF vangt ook browsers op die geen MQ ondersteunen.

18.4.6 breakpoints

Media Queries gebruiken **breakpoints** om veranderingen aan te brengen in de **lay-out**: van één kolom naar twee, naar drie etc…Deze breakpoints zijn meestal `width` voorwaarden zoals

```
@media screen and (min-width:460px)
```

Waar zet je deze breedte? En hoeveel media queries gebruik je?

Een blik op het huidige aanbod van schermen (gsm, smartphone, tablet, desktop) leert als snel dat **alle resoluties altijd voorkomen**. Dit heeft uiteraard te maken met de prijs die je betaalt. Het heeft dus weinig zin zich vast te pinnen op bepaalde "universele waarden". Die bestaan gewoonweg niet.

Een greep uit een aantal toestellen:

low-end smartphone: 240 X 320

medium-end smartphone: 480 X 800

high-end smartphone: 720 X 1280

kleine tablet: 768 X 1024

grote tablet: 1536 X 2048

kleine netbook PC: 600 X 1024

kleine E-reader: 480 X 800

high-end E-reader: 1024 X 758

high-end desktop scherm: 1080 X 1920

Enkele waarden zie je veel opduiken: 320, 480, 800, 1024. We kunnen deze als startpunt gebruiken, maar we mogen ons zeker niet vastpinnen op de resolutie van bepaalde populaire toestellen: geen goed idee want die toestellen veranderen snel.

Breakpoints moeten eerder gekozen worden **in functie van de lay-out**, daarbij houden we met het volgende rekening

- maak ***mock-ups*** van de verschillende lay-outs die je wil (gebruik daarbij eventueel een framework of maak een tekening met pen en papier)
- BPs koppelen we volledig los van bestaande toestellen: stel **arbitraire** breakpoints in (vb. 300px, 600px, 1000px) en maak de MQ's
- **pas** de breakpoints **aan** zodat de ene lay-out zo vlot mogelijk overgaat in de volgende.
- Voeg eventueel een BP toe

Je kan ook de MQ's verdelen volgens

- **major breakpoints**: (vb 480 – 768) zijn essentieel want een volledig andere lay-out en navigatie is vereist
- **minor breakpoints**: (vb 1024) voor minder belangrijke lay-out aanpassingen

Let ook op

- gebruik *fluid images*
- controleer wat er gebeurt als je *tekstzoom* gebruikt: op mobile wordt dit intensief toegepast

19 CSS FRAMEWORKS, SHIMS, PREPROCESSORS

Front-end ontwikkelaars maken hoe langer hoe meer gebruik van kant-en-klare sjablonen en frameworks om snel te kunnen werken. Alhoewel jullie nu voldoende HTML en CSS kennen om jullie eigen systeem op te zetten, bespaar je enorm veel tijd als je kan vertrekken van een bestaand "sjabloon".

Laat er echter geen twijfel over bestaan: om Frameworks, Boilerplates en shims te gebruiken en aan te passen aan je eigen noden, moet je een **zeer degelijke kennis CSS** hebben! Ze zijn echt niet bedoeld voor beginners…

In dit hoofdstuk geven we enkel een overzicht van wat er te verkrijgen is op dit vlak. In het projectenboek leer je er mee omgaan.

19.1 Modernizr, Shims en Polyfills

We hebben het al vele malen gezegd: niet alle browsers ondersteunen alle HTML5 en CSS3. Je hebt nood aan detectie en oplossingen.

Een "shim" in programmeren is een workaround of patch voor een compabilititsprobleem. In HTML en CSS spreekt men ook wel eens van een "polyfill": dat is een javascript dat een dergelijk probleem oplost.

De HTML5 Shim

Browsers die de HTML5 elementen niet ondersteunen, negeren die gewoon, maar daar komt bij dat ze ook geen children kunnen hebben en dat je ze niet kunt stylen met CSS.

John Resig gebruikte ooit de term "shiv" ervoor en sindsdien betekenen ze hetzelfde. De HTML5 shiv is een javascript die dit probleem oplost (het maakt de elementen aan in de DOM). Te vinden op http://code.google.com/p/html5shiv/

Plaats een conditioneel script element in de `head` van je pagina, als volgt

```
<!--[if lt IE 9]>
  <script src="html5shiv.js"></script>
<![endif]-->
```

Vanaf nu kan je zonder problemen in elke browser HTML5 elementen style.

Opmerkingen:

- wat deze shim niet doet is achterliggende functionaliteit voorzien.
  Bv: in IE8 zal je met de Shim een `canvas` element kunnen opmaken met CSS, maar het element zal niet werken dmv script.
- de *html5shiv* zit reeds verwerkt in veel boilerplates en frameworks

Modernizr

is een JS library die ons kan helpen *HTML5 en CSS3 features* te detecteren en (soms) te verhelpen. Het bevat ook een *HTML5 shim* die je toelaat HTML5 sectioning elementen in oudere browser te *stylen*. Deze shim doet echter alleen dat, het voegt geen enkele HTML5 of CSS3 functionaliteit toe! Modernizr is dus enkel een testframework.

Het bevat ook nog de javascript library *yepnope.js* (zie download opties), die gebruikt kan worden om javascripts *on demand* op te roepen.

Modernizr is soms ingesloten in andere Frameworks, zoals Bootstrap en Foundation.

Wat kan je er mee doen?

- selectief CSS selectors toepassen om toch nieuwe CSS3 toe te passen op nieuwe browsers en de problemen op te vangen in oudere browsers
- selectief javascripts opstarten die HTML5 features gebruiken (via *yepnope.js*)
- HTML5 sectioning elementen opmaken

Modernizr is te vinden op http://modernizr.com/.

Als je Modernizr aan je pagina koppelt krijg je een in de html tag een aantal classes te zien waarvan je kan gebruik maken:

```
<html class=" js flexbox flexboxlegacy canvas canvastext webgl no-touch geolocation
  postmessage websqldatabase indexeddb hashchange history draganddrop websockets rgba
  hsla multiplebgs backgroundsize borderimage borderradius boxshadow textshadow opacity
  cssanimations csscolumns cssgradients cssreflections csstransforms csstransforms3d
  csstransitions fontface generatedcontent video audio localstorage sessionstorage
  webworkers applicationcache svg inlinesvg smil svgclippaths" style="">
<head>
```

Het betreft hier alle *features* die getest werden. Als er een `no-` voorstaat betekent dit dat het *feature* niet werkt in deze browser (maar misschien wel in een andere).

Respond

Respond.js is een *polyfill* voor min- en max media queries.

IE6-8 ondersteunen geen media queries zoals

```
@media screen and (min-width: 480px){
        ...
    }
```

Respond lost dit probleem voor je op. Je kan het downloaden van https://github.com/scottjehl/Respond

De beste en eenvoudigste manier om *Respond* te gebruiken is met een voorwaardelijk commentaar:

```
<!--[if lte IE 8]>
  <script src="../js/respond.min.js"></script>
<![endif]-->
```

Opmerkingen:

- zorg ervoor dat je - tijdens het testen – je html bestand steeds bekijkt vanaf een webserver, niet direct vanaf het file system:
  Respond gebruikt namelijk Ajax en dat heeft een webserver nodig
- vroeger was *Respond* ingesloten in *Modernizr*: dat is langer het geval: installeer het zelf

19.2 Boilerplates en Frameworks

Een *boilerplate*

is de term die gebruikt wordt voor een **HTML sjabloon** dat je als **startpunt** gebruikt voor al je andere pagina's . Het bevat alle nodige HTML elementen, zoals `meta` tags en ook koppelingen naar "reset" of "normalise" stylesheets. Alle stylesheets en soms ook Javascript libraries worden bijgeleverd. Je moet er niets aan veranderen, enkele de inhoud opvullen.

Als een boilerplate sterk uitgebreid wordt met volledige CSS stylesheets, images, JS libs, etc.. spreken we van een *Framework*. De grens ertussen vervaagt snel.

De meeste gekende zijn

- de **HTML5 Boilerplate** (http://html5boilerplate.com/) voor HTML pagina's
- de **Mobile Boilerplate** (http://html5boilerplate.com/html5boilerplate.com/dist/mobile/) specifiek voor mobile apps
- de **HTML Email Boilerplate** (http://htmlemailboilerplate.com/) speciaal voor mailings via email

HTML5 Boilerplate (ver. 4) (momenteel erg populair) bevat

- het HTML5 sjabloon
- normalise.css
- modernizr
- jquery

Een CSS Framework

is een uitgebreid systeem van HTML en CSS sjablonen die je toelaten snel een website op te zetten (op front-end gebied).

De voordelen:

- **Snel**: de CSS staat kant en klaar, je moet enkel classes toepassen. Je kan een klant in korte tijd een mock-up tonen
- **Compatibel**: de boilerplate html is voorzien met de allerlaatste snufjes op gebied van compabiliteit en cross-browser ondersteuning voor HTML5 en CSS3: je moet niets zelf meer uitzoeken. Ze bevatten meestal *Modernizr* en soms een aantal shivs
- **Mooi**: heel wat frameworks voorzien ingebouwde mooie styles voor knoppen, forms etc. Je bespaart uren werk
- **Grid**: ze bevatten bijna allemaal een grid systeem om snel een lay-out te kunnen maken
- **Responsive**: de meeste hebben nu *Responsive Design* ingebouwd (niet altijd *Mobile First* weliswaar)

- **Javascript**: dikwijls krijg je een Javascript library inclusief (jquery, Prototype,…) met ingebouwde widgets

De nadelen:

- je moet het **begrijpen**: sommige zijn complex (Foundation), andere zijn zeer eenvoudig.
  Deze die werken op basis van een CSS preprocessor (Less, Sass): je moet er leren mee werken
- **beperkingen**: je moet zeker zijn dat ze voldoen aan je eisen: kan je ermee doen wat je wil?
- **mobile**: sommige zoals jQuery Mobile zijn enkel toegespitst op mobile Apps: ze dienen niet voor een gewone website
- **niet semantisch**: bijna allemaal vergen ze dat je extra `div` elementen met classes gebruikt voor je grid
- **niet volledig fluid:** problemen duiken op bij geneste kolommen

Er zijn tegenwoordig **tientallen** Frameworks… Je moet zelf je keuze maken op basis van de argumenten hierboven. We pikken er slechts enkele uit:

- **Foundation**
  http://foundation.zurb.com/

  Uitgebreid, Sass, Compass, templates, icons

- **Bootstrap**
  http://twitter.github.com/bootstrap/

  Uitgebreid met veel componenten, Responsive grid, Less

- **The semantic grid**
  http://semantic.gs/

  eenvoudig, semantisch, responsive, enkel grid, Less, SCSS, Stylus

- **320 and Up**
  http://stuffandnonsense.co.uk/projects/320andup/

  eenvoudig, Less, Sass, modernizr, responsive, Bootstrap styles

- **The responsive grid system**,
  http://www.responsivegridsystem.com/

  enkel een grid system, correct responsive

19.3 CSS preprocessors

CSS preprocessors zijn scriptsystemen die vanuit een eigen format CSS genereren (compileren). De bekendste zijn

- Less, http://lesscss.org/
- Sass, http://sass-lang.com/
- Stylus, http://learnboost.github.com/stylus/

Ze verschillen wat in syntax en in werkomgeving maar werken voor de rest zeer gelijkaardig. Een preprocessor gebruiken biedt een aantal grote voordelen:

Variabelen

Je kan variabelen gebruiken, iets dat onmogelijk is in zuivere CSS.
Een voorbeeld in Sass:

```
$mainColor: #0982c1;
$siteWidth: 1024px;
$borderStyle: dotted;
body {
  color: $mainColor;
  border: 1px $borderStyle $mainColor;
  max-width: $siteWidth;
}
```

Na compilatie krijg je dit:

```
body {
  color: #0982c1;
  border: 1px dotted #0982c1;
  max-width: 1024px;
}
```

Mixins

Mixins zijn stukjes CSS die je gebruikt in andere selectors. Ze zijn vooral handig voor veel voorkomende properties die je telkens moet kopiëren. Ze kunnen ook een **parameter** bevatten en dus als een soort functie gebruikt worden. Een voorbeeld in Less:

```
.border-radius(@radius) {
  -webkit-border-radius: @radius;
     -moz-border-radius: @radius;
          border-radius: @radius;
}
.container{
  .border-radius(1em);
}
```

Na compilatie krijgen we

```
 .container{
```

```
        -webkit-border-radius: 1em;
        -moz-border-radius: 1em;
        border-radius: 1em;
}
```

Nesting

Als je veel contextuele selectors moet schrijven zoals

```
section {  margin: 10px; }
section nav {  height: 25px; }
section nav a { color: #0982C1; }
section nav a:hover { text-decoration: underline; }
```

dan kan je dat duidelijker genest schrijven, de syntax is bij allemaal identiek:

```
section {
  margin: 10px;
  nav {
    height: 25px;
    a {
      color: #0982C1;
      &:hover {
        text-decoration: underline;
      }
    }
  }
}
```

Het `&` teken in `&:hover` heeft de betekenis van de onmiddellijke parent, dus `a:hover`.

Funties

Ze bevatten ook allemaal een reeks handige functies, zoals hier de kleurfunctie `darken()` gebruikt in Stylus:

```
color = #0982C1
h1
  background color
  border 3px solid darken(color, 50%)
```

Berekeningen

Eén van de grootste voordelen is dat je kan dynamische berekeningen maken. Een voorbeeld in Less:

```
.navbarhor(@aantal,@paddingPx: 2px) {
 ul{
  li {
```

```
    margin:0;
    padding:@paddingPx*2 @paddingPx;
    width: floor((@containerWidth - (@aantal * @paddingPx * 2) - (@aantal + 1))/@aantal);
         }
```

De `width` van het `li` element wordt berekend op basis van de parameter `@aantal` en `@paddingPx` en een variabele `@containerWidth`.

Imports

Preprocessors laten je toe **modulair** te werken: je kan de CSS scheiden in aparte delen zoals `kleuren, structuur, font, menu, etc…`

Door middel van `@import` statements verzamel je dan je verschillende onderdelen in één stylesheet.

In zuivere CSS wordt een `@import` van een ander stylesheet niet als optimaal bekeken omdat het extra *HTTP request* betekent, maar in een preprocessor is dat anders want het gebeurt tijdens de compilatie (en dus vooraf).

Frameworks die werken op basis van een preprocessor (Foundation, Bootstrap,…) werken altijd op deze manier om hun hoofdstylesheet samen te stellen.

Hier de code uit de **Foundation** master scss file:

```
@charset "UTF-8";
// Foundation Features
// =============

@import "compass/css3";
@import "modular-scale";
@import "foundation/functions/all";
@import "foundation/settings";
@import "foundation/mixins/all";
@import "foundation/common/base";
@import "foundation/components/grid", "foundation/components/modules/all";
// Media Queries Overrides
@import "foundation/components/modules/mqueries";
```

20 ACCESSIBILITY EN HTML

Accessibility of **Webtoegankelijkheid** streeft ernaar dat een website even gemakkelijk te gebruiken is voor iemand met beperkingen als voor een ander. Je leest meer over toegankelijkheid op de website van *Anysurfer*, www.anysurfer.be.

Mogelijke beperkingen kunnen zijn:

- Slecht/niet zien, kleurenblindheid
- Doof zijn
- De muis niet kunnen gebruiken
- Een touchscreen niet kunnen gebruiken

Verwar *Accessibility* niet met *Usability* - bruikbaarheid - dat gaat over het vlotte gebruik van een site: zijn de knoppen en velden groot genoeg voor je vingers, hoe snel vind je wat je zoekt, is de flow logisch, is er voldoende feedback voor je acties, is de scherm lay-out goed ontworpen. Niet hetzelfde, maar je beseft dat een slecht *usable* site nóg meer problemen voor een gehandicapte met zich meebrengt.

In zekere zin brengt een website op een mobiel toestel je in een dergelijke situatie: je hebt geen muis, het scherm is kleiner en roteert. Is de website dan nog even goed bruikbaar als op een desktop? Zowel *webtoegankelijkheid* als *bruikbaarheid* spelen hier een rol.

Dit hoofdstukje heeft enkel de bedoeling de practische toepassing van *Accessibility* in HTML te belichten:
welke zijn de problemen die zich kunnen voordoen in een pagina, welke elementen en attricbuten spelen een belangrijke rol bij webtoegankelijkheid en hoe kunnen ze worden opgelost.

20.1 Navigatie

Hyperlinks

21 BIJLAGE : INTERNET LINKS

Wat?	URL
HTML4.01 spec	http://www.w3.org/TR/html401/
XHTML1.0 spec	http://www.w3.org/TR/xhtml1/
HTML5 spec WHATWG	http://www.whatwg.org/specs/web-apps/current-work/multipage/
HTML5 spec W3C (identiek)	http://www.w3.org/TR/html5/
HTML5 Doctor	http://html5doctor.com/
HTML5 test	http://www.html5test.com/
HTML5 Mozilla reference	http://developer.mozilla.org
CSS2.1 specification	http://www.w3.org/TR/CSS2/
CSS3 W3C overzicht current status	http://www.w3.org/Style/CSS/current-work.en.html
CSS3.info overzicht alle modules	http://www.css3.info/modules/
CSS Selectors Level3	http://www.w3.org/TR/2011/REC-css3-selectors-20110929/
CSS properties index	https://developer.mozilla.org/en-US/docs/CSS/CSS_Reference
CSS Discussion List Wiki	http://css-discuss.incutio.com/wiki/Main_Page
CSS3 test	http://css3test.com/
caniuse compabiliteit overzicht	http://caniuse.com

Quirksmode CSS3 ondersteuning	http://www.quirksmode.org/css/contents.html
CSS3 test	http://css3test.com/
Less	http://lescss.org
Google Fonts API	https://developers.google.com/webfonts/

COLOFON

Domeinexpertisemanager	Rita Van Damme
Moduleverantwoordelijke	Jean Smits
Auteurs	Adinda Mattens Ilse Palmaers Jan Vandorpe
Versie	21/8/2013
Codes	Peoplesoftcode: Wettelijk depot:

Omschrijving module-inhoud

Abstract	Doelgroep	ICT cursisten
	Aanpak	zelfstudiecursus
	Doelstelling	Ontwerpen van webpagina's met html en css, theorie
Trefwoorden		Html css html5 css3 webpagina website
Bronnen/meer info		