

design your future

Web Development 1 – Deel 2: CSS3

Christophe De Waele



















handelswetenschappen en bedrijfskunde

bachelor in de toegepaste informatica

campus Kortrijk

academiejaar 2021-2022

Legende van de gebruikte iconen

	Denkvraag		Studeeraanwijzingen
	Leerdoelen		Tijdsinschatting
	Formule		Toledo
	Extra informatie		Beeld-/geluidsfragment
	Niet vergeten		Voorbeeld
	Opdracht/Oefening		Tools/Apps
	Presentatie (PowerPoint)		Website
	Rekenblad (Excel)		Zelfstudie
	Samenwerking		(Zelf)toets

Inhoudsopgave

Legende van de gebruikte iconen	2
Inhoudsopgave.....	3
1 Inleiding.....	6
1.1 Wat is CSS?.....	8
1.2 Wat zijn de voordelen van CSS?	8
1.3 Stylesheets verbinden met HTML5.....	9
1.3.1 <i>Inline stijl</i>	9
1.3.2 <i>Stijlblok</i>	9
1.3.3 <i>Extern stijlblad</i>	10
1.3.4 <i>Media-query's</i>	11
1.4 Het document object model.....	11
2 Selectors	14
2.1 Introductie selectors.....	14
2.2 Element-selector.....	14
2.2.1 <i>Enkelvoudige-selector</i>	14
2.2.2 <i>Meervoudige-selector</i>	15
2.2.3 <i>Descendent-selector</i>	15
2.2.4 <i>Child-selector</i>	15
2.2.5 <i>Adjacent sibling selector</i>	16
2.3 Class en id selector	16
2.4 Pseudo-element selectors	19
2.5 Pseudo-class selector.....	21
3 Overerfbare eigenschappen (inheritance)	26
3.1 Volgorde van uitvoeren	29
3.2 Standaardwaarden en default stylesheet.....	29
3.3 Stylesheet van gebruiker	29
3.4 Stylesheet van de auteur (bouwer)	30
3.4.1 <i>Stijlregels op één plaats</i>	30

3.4.2	<i>Meerdere externe stijlbestanden</i>	30
3.4.3	<i>Stylesheet, stijlblok en inline</i>	31
3.4.4	<i>Geïmporteerde stylesheets</i>	31
3.4.5	<i>Specificiteit (specificity)</i>	31
3.5	<i>!important regels</i>	34
4	Visuele effecten	37
4.1	Werking van prefixes	37
4.2	Transformations	38
4.2.1	<i>transition-property</i>	39
4.2.2	<i>transition-duration</i>	39
4.2.3	<i>transition-timing-function</i>	39
4.2.4	<i>transition-delay</i>	40
4.2.5	<i>transition</i>	40
4.2.6	<i>Voorbeeld</i>	41
4.3	Borders (randen)	41
4.3.1	<i>Afgeronde hoeken</i>	41
4.3.2	<i>border-image</i>	44
4.3.3	<i>Box-schaduw</i>	46
5	Werken met tekst	47
5.1	Tekst	47
5.1.1	<i>Text-shaduw (vanaf CSS3)</i>	47
5.1.2	<i>Lettertype</i>	48
5.1.3	<i>Kleur en achtergrond</i>	51
5.1.4	<i>Uitbreiding: background instellen browser</i>	52
6	Navigatie	54
6.1	Het opmaken van een link in CSS	54
6.2	Het <i>list</i> element als structuur voor je navigatie	57
6.3	Een horizontale navigatie in een list	61
7	Positionering van elementen	62
7.1	De box	64
7.2	Visual formatting model	67

7.2.1	<i>Inline elementen</i>	67
7.2.2	<i>Block elementen</i>	68
7.3	Positioneren.....	69
7.3.1	<i>Zwevende elementen</i>	69
7.3.2	<i>Relatief positioneren</i>	71
7.3.3	<i>Absoluut positioneren</i>	73
7.4	Indelen van pagina.....	74

1 Inleiding

Wie de specificaties van HTML 2.0 bekijkt, zal merken dat er bijna geen mogelijkheden zijn om een pagina vorm te geven. Oorspronkelijk was dat de bedoeling: er kon aangegeven worden wat de titel was, wat de kop in een tekst waren en welke stukken tekst benadrukt moesten worden. Hoe de tekst uiteindelijk werd vormgegeven, viel buiten het opzet van HTML.

In de wetenschappelijke wereld, waarin HTML in eerste instantie vooral werd gebruikt, voldeed de genoemde wijze van presenteren uitstekend. Met de toegenomen verbreding in het gebruik van Internet ontstond bij webauteurs echter de behoefte naar meer mogelijkheden bij de opmaak van documenten. Op deze behoefte is door de ontwikkelaars van browsers (vooral Netscape en Microsoft) ingespeeld door het introduceren van nieuwe elementen en attributen. Bijvoorbeeld de elementen `center` en `font`, attributen `size`, `color`, `bgcolor`, `face` en `align` en browser-specifieke elementen `blink`, `multicol`, `spacer` en `marquee`.

Webauteurs zelf gebruikten hun creativiteit om met bestaande HTML-elementen meer invloed op de opmaak van hun documenten te krijgen. Voorbeelden zijn:

- Het gebruik van het `blockquote`¹ element om tekst te laten inspringen.
- Het opnemen van transparante “gifjes” om de witruimte te controleren.
- Het gebruik van afbeeldingen om tekst op een bepaalde manier weer te geven (in een specifiek lettertype, of in een grootte welke met HTML niet bereikt kan worden).
- Het toepassen van tabellen om tekst en andere inhoud van een document in een vaste lay-out te krijgen (zoals *ingesprongen*² of in kolommen).

Zeker de browser-specifieke elementen en attributen, maar ook het gebruik van de andere mogelijkheden voor de presentatie hebben vaak tot gevolg dat een document niet meer in elke browser en op elk platform goed te bekijken is. Daarnaast heeft het (oneigenlijke) gebruik van afbeeldingen geleid tot een aanzienlijke toename in het dataverkeer.

Als oplossing voor deze problemen zijn stylesheets geïntroduceerd. De achterliggende gedachte is de **scheiding tussen structuur en presentatie**: HTML moet zoals oorspronkelijk bedoeld zorgen voor de structuur van het document terwijl de presentatie wordt bepaald met behulp van stylesheets. Daarbij

¹ Tekst in een HTML-document kan op verschillende manieren gestructureerd worden. De meest algemene vorm is het indelen van tekst in paragrafen. Hiervoor kan gebruik gemaakt worden van het `p` element. Daarnaast zijn er elementen, welke gebruikt kunnen worden, als een specifieke betekenis van een blok tekst benadrukt moet worden. Het `blockquote` element is bedoeld voor tekst welke wordt geciteerd en geeft deze ingesprongen weer.

² Als alternatief voor het oneigenlijke gebruik van het `blockquote` element, wordt vaak gebruik gemaakt van een tabel. Je kunt dan zelf bepalen aan welke kant en hoeveel er wordt ingesprongen. Wil je bijvoorbeeld alleen aan de linkerkant inspringen, dan maak je een tabel met één rij en twee kolommen. Het `width` attribuut van het `td` element waarmee je de eerste cel definieert, bepaalt de mate van inspringen. De tekst plaats je in de tweede cel.

```
<table cellspacing="0" cellpadding="0" border="0">
<tr>
<td width="25"></td>
<td valign="top">Tekst van de “eerste” cel.<br/></td>
</tr>
</table>
```

is de idee dat stylesheets niet alleen gebruikt worden als vervanging van huidige opmaakmogelijkheden, maar veel meer gaan bieden. Dat moet dan bovendien op een flexibele en gemakkelijk beheersbare manier.

De ontwikkeling van Cascading Style Sheets (CSS) gaat terug naar 1996. Het World Wide Web Consortium (W3C), de toezichthouder op het web, publiceerde toen de specificaties van *CSS level 1*. Een nieuwe denkrichting voor het vormgeven van HTML-documenten was geboren: geen HTML-opmaakregels meer in de HTML-code zelf, maar CSS-instructies in een apart stijlblok of extern stijlblad. De vormgeving en de inhoud van een webpagina werden uit elkaar getrokken. De aanbeveling van CSS2 verscheen in 1998 en is inmiddels grondig herzien (CSS 2.1 – 2002). CSS 2.1 is momenteel de officiële standaard. Momenteel bevindt CSS3 (in 2005 opgestart) zich in het ontwikkelingsstadium en is het net zoals HTML5 nog niet af.

Eén van de belangrijkste voordelen van stylesheets was dat het backwards compatible is. Oudere browsers en eenvoudige browsers op bijvoorbeeld handcomputers, die geen CSS ondersteunen, laten alleen de eenvoudige HTML zien en nieuwere browsers laten mooiere en betere pagina's zien.

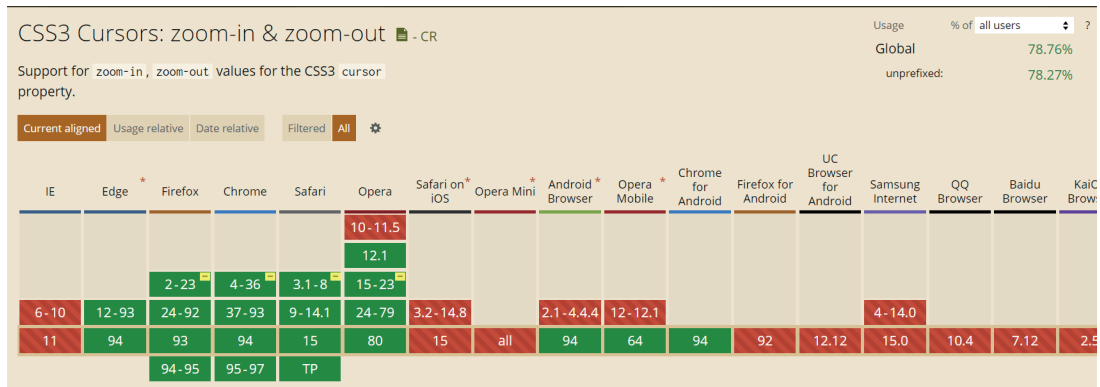
CSS heeft al een behoorlijke evolutie doorgemaakt:

- **CSS level 1** bevat mogelijkheden voor lettertypes, kleuren, marges, ... die bijna elke CSS-pagina nodig heeft.
- **CSS level 2 revision 1** bevat alles van CSS 1 met daarbij uitbreidingen voor, e.g., absolute positionering, automatische nummering en pagina-eindes.
- **CSS level 3** splitst de standaard op in modules die elk afzonderlijk en met verschillende snelheden ontwikkeld worden. Het meest in het oog springend zijn: transformaties, niet rechthoekige boxes, slagschaduw en media query's.

Ons onmiddellijk verdiepen in CSS3 zou niet verstandig zijn. Er is namelijk nog geen enkele browser die alle functionaliteiten van CSS3 ondersteunt. Op de site "*Can I Use?*"³ kan je testen welke browser welke functionaliteit ondersteunt (zie Figuur 1). Op de site "*The CSS3 Test*"⁴ kan je je eigen browser testen op CSS3 ondersteuning.

³ <https://caniuse.com/>

⁴ <https://css3test.com/>



Figuur 1 Screenshot van de "Can I Use?" website.

1.1 Wat is CSS?

CSS is een afkorting voor *Cascading Style sheets*. Deze stylesheets dienen om de opmaak van webpagina's in te stellen en ze bieden meer opmaakmogelijkheden dan HTML.

Wanneer je een website maakt, wil je al je pagina's dezelfde opmaak meegeven. Dit lukt natuurlijk door elke pagina afzonderlijk op te maken maar daar kruipt veel tijd in. Met CSS maak je een soort sjabloon met opmaakprofielen die je op verschillende webpagina's kunt toepassen.

1.2 Wat zijn de voordelen van CSS?

Het werken met CSS voor de opmaak van HTML5-pagina's op een website biedt een aantal voordelen:

1. Met CSS kan je in hoge mate de weergave van je pagina bepalen en kan je effecten bereiken die door gebruik van *enkel* HTML-elementen niet mogelijk zijn. Zo kun je bijvoorbeeld bepalen dat alles wat tussen `<h3>` tags staat, 18 punten groot moet zijn in de kleur rood en met het lettertype Arial. Het is dus veel flexibeler dan HTML5 wat de vormgeving betreft.
2. CSS stelt je in staat om alle stijlelementen van een website in een document onder te brengen. Dat wil zeggen dat je maar één document hoeft te veranderen om alle pagina's van je site aan te passen. Wil je bijvoorbeeld de kleur van al je *headings* (h1, h2, ...) veranderen of het lettertype dat je in je paragrafen gebruikt? Dan verander je het CSS document waarna alle pagina's deze stijl overnemen. Met HTML5 zou je deze wijzigingen in alle pagina's apart moeten aanbrengen, wat veel meer werk is.
3. De pagina's van je site worden sneller doordat je veel minder code hoeft te gebruiken. Hierdoor wordt je site dus sneller geladen.
4. De broncode is beter te indexeren voor zoekmachines.

In dit hoofdstuk komt vooral aan de orde op welke wijze je stylesheets met HTML kunt verbinden. Achtereenvolgens komen aan de orde: inline stijlen, het stijlblok en het extern stijlblad. Daarna wordt het gebruik van de attributen `class` en `id` en de elementen `div` en `span` toegelicht. Tenslotte wordt ingegaan op het begrip *cascading* en worden voorbeelden gegeven van het gebruik van alternatieve stijlbladen.

1.3 Stylesheets verbinden met HTML5

Een stylesheet is een verzameling stijlregels die bepaalt hoe elementen in een document door de browser (of andere apparaten, zoals de printer) weergegeven moeten worden. De stijlregels kunnen bijvoorbeeld betrekking hebben op het lettertype, de lettergrootte, de kleur van de tekst, de achtergrondkleur en de uitlijning, maar ook op het inspringen en de plaats in het browservenster.

Stylesheets kunnen op verschillende manieren aan de elementen in een HTML5-document gekoppeld worden: via een inline stijl, een stijlblok of via een extern stijlblad.

1.3.1 Inline stijl

Wanneer een stijl slechts een enkele keer gebruikt wordt in een document, dan kan door het toevoegen van het `style` attribuut aan een element, een inline stijl vastgelegd worden

Je doet dat door het `style` attribuut toe te voegen aan het element, waarvan je de opmaak van de ingesloten inhoud wilt bepalen. Als waarde van het `style` attribuut neem je de stijldeclaratie op, waarin de gewenste stijl is vastgelegd:

```
<element style="stijldeclaratie">
```

In het volgende voorbeeld wordt met een inline stijl vastgelegd dat de tekst, ingesloten door het betreffende `<h1>`-element, in rood moet worden weergegeven.

```
<h1 style="color: red; background-color: white;"> </h1>
```

1.3.2 Stijlblok

Wanneer een stijl vaker gebruikt wordt voor meer elementen, maar slechts binnen één document, dan kunnen de stijlregels met het `style` element in een ingesloten *stijlblok* in het head van het document geplaatst worden. Een stijlblok heeft de volgende opbouw:

```
<head>
  <style type="text/css">
    voeg hier je stijlregels toe
  </style>
</head>
```

In het volgende voorbeeld bevat een stijlblok twee stijlregels.

```
<style type="text/css">
  body { font-family: Arial, Helvetica, sans-serif;}
  h1 { color: #FFFFFF; background-color: #336699; }
</style>
```

Dit maakt het voor de auteur een stuk eenvoudiger om wijzigingen aan te brengen in een reeds vastgelegde stijl, of om nieuwe stijlen te definiëren. In plaats van telkens een element in een document gebruikt wordt een inline stijl te moeten wijzigen of toe te voegen, hoeft de aanpassing slechts één keer in het stijlblok te worden aangebracht.

1.3.3 Extern stijlblad

Wanneer in meerdere documenten voor één of meer elementen dezelfde stijlregels toegepast moeten worden, dan kunnen deze het beste in een apart document worden opgenomen: een extern stijlblad. In de HTML-documenten volstaat met behulp van het `link` element een eenvoudige verwijzing naar het stijlblad. Aan het `link` element worden *in ieder geval* de attributen `href`, `rel` en `type` toegevoegd. Het `href` attribuut specificeert welk stijlblad geopend moet worden. Bij Cascading Style Sheets gaat het om een bestand, waarvan de bestandsnaam de extensie `".css"` heeft. Het `rel` attribuut geeft aan dat het bij het gerelateerde bestand gaat om een stylesheet en heeft dan ook als waarde `"stylesheet"`. Het `type` attribuut definieert het Internet Media (MIME) type van het bestand waarnaar verwezen wordt. Voor Cascading Style Sheets is dat `"text/css"`. In html5 moet voor het `link` element geen `type` attribuut opgegeven worden.

Het `link` element mag een onbeperkt aantal malen opgenomen worden in een document.

In het volgende voorbeeld wordt een extern stijlblad aan een HTML-document gekoppeld. Het stijlblad heeft de naam `"basis.css"` en bevindt zich in een subdirectory met de naam `"stijl"`.

```
<head>
  <link href="stijl/basis.css" rel="stylesheet" type="text/css" />
</head>
```

Wanneer de auteur iets aan een gebruikte stijl wil wijzigen, hoeft de wijziging niet in elk document afzonderlijk, maar slechts op één plaats in het stijlblad te worden aangebracht.

Een stijlblad bevat geen HTML5-code, maar uitsluitend stijlregels. Bij Cascading Style Sheets zijn deze stijlregels opgebouwd volgens het selector-mechanisme, dat beschreven wordt in Sectie 2.

Het `media` attribuut kan aan het `link` element toegevoegd worden, om aan te geven dat de stijlregels betrekking hebben op de weergave van het document door een bepaald apparaat. Het `media` attribuut kan een aantal waarden krijgen, waaronder `"screen"` (voor de weergave zonder pagina-indeling op een computerscherm), `"print"` (voor de weergave in pagina-opmaak bij het afdrukken of als print-preview) en `"aural"` (voor de weergave door een spraaksynthesizer). Bij de waarde `"all"` worden de stijlregels toegepast bij de weergave door alle apparaten. De standaardwaarde van het `media` attribuut is `"screen"`. Dat betekent dat de browser bij het ontbreken van het `media` attribuut de stijlen alleen op het scherm moet weergeven. De praktijk is echter dat de meeste browsers de stijlen in dat geval ook weergeven bij het afdrukken.

De verschillende waarden van het attribuut `media` kan je vinden in Tabel 1.

Value	Description
screen	Computer screens (this is default)
tty	Teletypes and similar media using a fixed-pitch character grid
tv	Television type devices (low resolution, limited scroll ability)
projection	Projectors
handheld	Handheld devices (small screen, limited bandwidth)
print	Print preview mode/printed pages
braille	Braille feedback devices
aural	Speech synthesizers
all	Suitable for all devices

Tabel 1 Mogelijke waarden voor het attribuut "media".

In het volgende voorbeeld worden de stijlregels in het stijlblad alleen toegepast bij het afdrukken:

```
<link href="stijl/afdrukken.css" rel="stylesheet" type="text/css" media="print" />
```

Wanneer je verschillende stijlbladen hebt voor bijvoorbeeld *weergave op het scherm* en *afdrukken*, dan neem je het link element eenvoudig meerdere keren op.

```
<link href="stijl/scherm.css" rel="stylesheet" type="text/css" media="screen" />  
<link href="stijl/afdrukken.css" rel="stylesheet" type="text/css" media="print" />
```

1.3.4 Media-query's

Media-query's zijn een uitbreiding in CSS3 t.o.v. CSS2. Met media-query's kan CSS als het ware de browser ondervragen en afhankelijk van de uitkomst een bepaalde stylesheet laden. Je kan bijvoorbeeld stylesheets maken voor browsers op mobiele apparaten, waarvan de schermbreedte minder is dan 480 pixels. In mobiele stylesheets kunnen bijvoorbeeld kolommen verborgen of verplaatst worden. Knoppen en menu's kunnen groter worden weergegeven zodat ze beter geschikt zijn voor touchscreens. Deze topics wordt verder behandeld in de cursus Mobile devices.

1.4 Het document object model.

Het **Document Object Model (DOM)** is een verzameling regels en afspraken over hoe een browser een webpagina interpreteert. Het DOM zit ingebakken in een browser en je kunt er niets aan wijzigen. Maar als je weet volgens welke basisregels het DOM werkt, kun je er wel gebruik van maken.

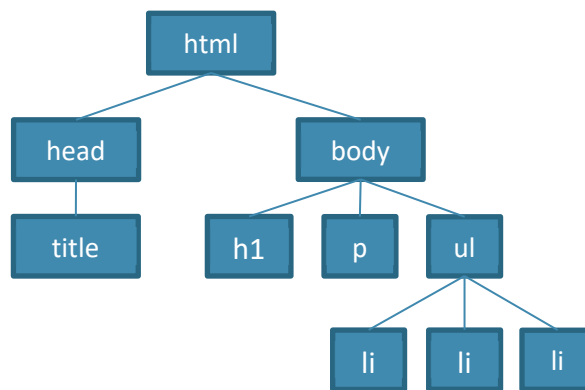
Het DOM gaat uit van een hiërarchische structuur van een HTML-pagina en dat komt goed uit, want daar kunnen wij mooi gebruik van maken. Sinds de introductie van het DOM wordt een HTML-pagina niet langer gezien als een lange serie tekens, maar als een verzameling op zichzelf staande elementen

(objecten) die onafhankelijk van elkaar kunnen worden benaderd en gemanipuleerd. We noemen het DOM *object georiënteerd*.

De hiërarchische structuur van een HTML-document wordt omschreven in de vorm van *parent*-elementen en *child*-elementen. Er bestaat dus een ouder/kind relatie tussen de elementen.

Bovenaan in deze structuur staat het `<html>` element. Zeg maar de *Padre de Familias*, the boss of all bosses. Het `<html>` element heeft twee directe 'children': `<head>` en `<body>`. Vervolgens is bijvoorbeeld `<title>` weer een 'child' van `<head>`. Je begrijpt dat `<body>` niet stil heeft gezeten en stikt van de 'children'.

Wanneer je het ene element in het andere plaatst wordt dit geneste element automatisch het 'child' van het bovenliggende element.



Figuur 2 Illustratie HTML DOM boomstructuur.

Uit de boomstructuur (geïllustreerd in Figuur 2) kan je afleiden dat bepaalde elementen omvat worden door andere elementen en dat bepaalde elementen zich op hetzelfde niveau bevinden als andere elementen.

We kunnen over de pagina uit Figuur 2 het volgende zeggen:

- De `li`-elementen zijn **children** van het `ul`-element.
- Het `ul`-element is **parent** van de `li`-elementen.
- Het `body`-element is een **ancestor** van het `h1`-element, het `p`-element, het `ul`-element en de `li`-elementen.
- `p`, `h1`, `ul` en `li` en `body` zijn **descendants** van het `html`-element.
- De `li`-elementen zijn **sibling**-elementen.

```
<div><p>Eigenlijk is de structuur van een HTML-document een  
grote<strong><em>family</em> business</strong></p></div>
```

In bovenstaand voorbeeld is de `<p>` het 'child' van de `<div>`, terwijl `<p>` weer de 'parent' is van ``. `` is op zijn beurt weer het 'child' van ``. Het is vooral het object-georiënteerde JavaScript dat voortdurend en handig gebruik maakt van het DOM.

Wij zullen onszelf meer concentreren op de manier waarop je via CSS de verschillende elementen direct kunt benaderen en manipuleren.

Uit de voorgaande voorbeelden komen we tot volgende afspraken:

Relatie	Beschrijving
parent (ouder)	Bovenliggend element in de boomstructuur: elk element heeft exact één parent, behalve het root-object (in het geval een HTML-document is dat <code><html></code>) dat er geen heeft.
child (kind)	A is een child van B <i>als en alleen als</i> B de parent is van A.
descendant (afstammeling)	A is een descendant van B <i>als</i> B de parent is van A <i>of</i> A is een child van C dat een descendant is van B.
ancestor (voorouder)	A is een ancestor van B <i>als en alleen als</i> B een descendant is van A.
sibling (gelijke)	A is een sibling van B <i>als en alleen als</i> A en B hetzelfde parent-element hebben. <ul style="list-style-type: none">• preceding sibling: komt <i>voor</i> een element uit de boomstructuur• following sibling: komt <i>na</i> een element uit de boomstructuur

2 Selectors

2.1 Introductie selectors

Stijlen kunnen op verschillende manieren aan HTML gekoppeld worden: via een inline stijl, met behulp van een stijlblok in het *head* van het document of met een extern stijlblad.

In een stijlblok en stijlblad worden de stijlen vastgelegd met stijlregels. Een stijlregel bestaat uit twee delen: een selector en een blok met één of meer stijldeclaraties, dat wordt begrensd door accolades (gekrulde haken).

Er bestaan verschillende soorten selectors:

- *element-selectors*
 - *type selectors*
 - *descendant selectors*
 - *child selectors*
 - *adjacent sibling selector*
 - *universele selector* (wordt niet besproken)
- *attribuut-selectors* (wordt niet besproken)
- *class selectors*
- *id selectors*
- *pseudo-element selectors*
- *pseudo-class selectors*

2.2 Element-selector

Er zijn meerdere *element-selectors*, waarvan de enkelvoudige de gemakkelijkste variant is. Deze bekijken we eerst.

2.2.1 Enkelvoudige-selector

Een style sheet bevat een aantal style rules (stijlregels). Elke style rule bepaalt de opmaak van een bepaald deel uit de webpagina. Een stijlregel is als volgt opgebouwd:

- De **selector** definieert de HTML-elementen (beschouw het als een filter) waarvoor je een bepaalde opmaak wil bepalen. Zo kan je voor iedere paragraaf een opmaak maken en bijgevolg gebruik je dan het element `p`.
- De **declaration** is de definitie van de opmaak die je aan de selector wil toekennen. De declaration staat tussen accolades: `{ }`.
- **Property** en **value**. Elke declaration kent aan één of meer eigenschappen een bepaalde waarde toe. Hier wordt aan de eigenschap `color` (bepaalt de tekstkleur) de waarde `orange` toegekend. Elke heading `h1` zal in het oranje weergegeven worden.

Natuurlijk stel je meestal niet één eigenschap in maar meerdere. Stel dat je in titel h1 de tekst in het rood wenst en een groene achtergrondkleur wil, dan moet je gebruik maken van 2 declarations, i.e., color en background-color. Deze twee scheid je door een puntkomma en dan ziet dit er zo uit:

```
h1 {color: red; background-color: #33F933}
```

2.2.2 Meervoudige-selector

Je kunt 1 stijl ook aan meerdere elementen koppelen. Dat doe je bijvoorbeeld met de volgende CSS regels:

```
h2, li {  
  font-size: 80%;  
  color: #FFFFFF;  
}
```

De twee stijlregels (font-size en color) zijn nu van toepassing op de volgende elementen:

- *Alle* level two headers.
- *Alle* list items.

2.2.3 Descendent-selector

De term descendant staat voor afstammeling. Bij een descendant element selector wordt de opmaak toegepast op een element dat afstamt van een voorgaand element. Is dat niet het geval dan blijft de nakomeling verstoken van de stijl.



De volgende CSS regel:

```
p em {  
  color:red  
}
```

kan toegepast worden op de volgende HTML code:

```
<h1>De <em>schuine tekst</em> is niet rood</h1>  
<p>Deze <em>schuine tekst</em> is wel rood</p>
```

De browser kleurt de benadrukte (em) tekst in de kop niet rood, want in dit geval stamt het element em niet af van de alineatag (p). In de tweede regel zal dit wel het geval zijn.

2.2.4 Child-selector

De stijl wordt alleen toegepast als het element *direct* afstamt van de parent (het is er dus een kind van). Deze methode is herkenbaar aan het groter-dan-teken.

Opmerking: Child selectors worden niet ondersteund in IE6.



De volgende CSS regel

```
p > em {color:red}
```

kan toegepast worden op de volgende HTML code:

```
<h1>De <em>schuine tekst</em> is niet rood</h1>
<p>Deze <em>schuine tekst</em> is wel rood</p>
<p><strong>Deze vette en <em>schuine tekst</em> is NIET
rood</strong></p>
```

In dit geval wordt enkel de tweede lijn tekst rood gekleurd, aangezien enkel daar em een rechtstreeks kind van een p tag.

2.2.5 Adjacent sibling selector

Een *adjacent sibling* element-selector past de gedefinieerde vormgeving toe op elementen op hetzelfde niveau. Er hoeft geen sprake te zijn van child elements. De elementen moeten elkaar direct opvolgen.



De volgende CSS regel

```
h1 + h2 {margin-top:-5mm}
```

kan toegepast worden op de volgende HTML code:

```
<h1>Hoofding 1</h1>
<p>Alinea</p>
<h2>Hoofding 2 na alinea</h2>
<h1>Hoofding 1</h1>
<h2>Hoofding 2 na hoofding 1</h2>
```

Een h2 die onmiddellijk na een h1 komt in de boomstructuur van de pagina zal 5mm minder boven-marge hebben dan een gewone h2.

2.3 Class en id selector

Een class- en id-selector wordt toegevoegd aan een element om de koppeling met een stijl te realiseren. Het verschil tussen de twee selectors is, dat de class-selector een onbeperkt aantal keren met een bepaalde class-naam in een document gebruikt mag worden, terwijl het id attribuut slechts één keer met een bepaalde id-waarde in een document mag staan.

Het `class` attribuut wordt toegepast, wanneer een element niet elke keer in dezelfde stijl moet worden uitgevoerd (en dus geen stijl voor het element gedefinieerd kan worden), of als dezelfde stijl voor verschillende elementen gebruikt moet kunnen worden.

Het verbinden van de stijl met een element gebeurt bij gebruik van het `class` attribuut via een class-naam.

```
<element class="class-naam">
```

De stijlregels in een stijlblad of een stijlblok voor een class-naam hebben de volgende opbouw:

```
.class-naam {  
    stijldeclaratie  
}
```

In het volgende voorbeeld is in een stijlblok een stijlregel opgenomen voor de class-naam “speciaal”.

```
<style type="text/css">  
    .speciaal { color: red; background-color: white }  
</style>
```

Van elk element, waaraan het `class` attribuut met de class-naam “speciaal” is toegevoegd, is de kleur van de tekst rood. De volgende twee HTML elementen voldoen hier allebei aan:

```
<h1 class="speciaal">rode tekst</h1>  
<p class="speciaal">rode tekst</p>
```

In Sectie 1.4 (DOM) hebben we het gehad over de hiërarchische structuur van een HTML document en het feit dat er sprake is van *parent*-elementen en *child*-elementen. Dat komt nu mooi uit, want we kunnen in onze CSS regels deze children direct aanspreken zonder dat we telkens een aparte *class* hoeven aan te maken. Stel dat je de volgende HTML hebt:

```
<p class="voorbeeld">De stijldeclaratie met de naam 'voorbeeld' zal <span>van  
toepassing zijn</span> op deze paragraaf.</p>
```

In dit stukje HTML is dus de `` het child van de paragraaf met de class ‘voorbeeld’. Je gebruikt in de rest van je pagina’s (of elders op dezelfde pagina) ook nog wel `span`’s. Maar je wilt graag een aparte opmaak voor deze éne `span`, die niet terugkomt bij andere `span`’s. Je zou een aparte class voor deze `span` kunnen aanmaken, maar je kunt ook gebruik maken van het ‘*parent-child*’ concept. Dat doe je met de volgende CSS regel:

```
p.voorbeeld span{  
    background-color: #999999;  
}
```



Deze CSS regels:

```
.voorbeeld{
  font-family: Verdana, Arial, Helvetica, sans-serif;
  color: #FF6600;
  font-size: 90%;
}
```

toepassen op deze HTML code:

```
<p class="voorbeeld">De stijldeclaratie met selector '.voorbeeld'
zal van toepassing zijn op deze paragraaf.</p>
<p>De stijldeclaratie met selector '.voorbeeld' zal niet van
toepassing zijn op deze paragraaf.</p>
```

resulteert in het volgende gedrag: de CSS regels worden enkel toegepast op de eerste alinea.

De class uit het bovenstaande voorbeeld kun je aan ieder element hangen. Er is ook een manier om een class te beschrijven zodat deze *enkel* van toepassing kan zijn op 1 soort element:

```
ul.voorbeeld{
  font-family: verdana, helvetica, arial, sans-serif;
  color: #FF6600;
  font-size: 90%;
}
```

De class uit dit voorbeeld werkt *enkel* bij ul's. Als je deze class nu aan een <p> zou hangen dan doet hij dus niets.

Een id werkt op dezelfde manier als een class, maar wordt in de CSS niet voorafgegaan door een dot (.) maar door een hashtag (#). In de HTML-code schrijf je:

```
<p id="naam_van_id">
```

Verder is er nog 1 groot verschil tussen id's en classes. Een class mag je per HTML-pagina zo vaak gebruiken als je maar wilt. Er kunnen dus op 1 pagina een heleboel paragrafen zijn met de class 'voorbeeld'. Een id mag je per pagina maar 1 keer gebruiken. Een id is dus altijd uniek.

Stijlregels waarin gebruik gemaakt wordt van een id-selector kunnen er als volgt uitzien:

```
#id-waarde {
  stijldeclaratie
}
```

```
element#id-waarde {  
  stijldeclaratie  
}
```

De eerste stijlregel kan aan elk element gekoppeld worden door het opnemen van het id attribuut met als waarde de id-waarde. De tweede stijlregel kan gebruikt worden in situaties, waarin de stijl slechts op één bepaald element type betrekking heeft.

Je mag dus de naam voor een class of een id zelf verzinnen. Let er wel op dat deze namen nooit met een cijfer mogen beginnen en dat je geen spaties mag gebruiken.

2.4 Pseudo-element selectors

In CSS worden stijlen normaal gedefinieerd voor een element. Soms is het echter wenselijk effecten te bereiken, die niet mogelijk zijn als je alleen beschikt over *element*- of *attribuut*-selectors. Bijvoorbeeld: het in een bepaalde opmaak weergeven van de eerste letter of de eerste regel van de inhoud van een element. Om dat soort effecten mogelijk te maken, zijn pseudo-elementen geïntroduceerd. Een pseudo-element kun je zien als een denkbeeldig element, dat weliswaar niet in het document voorkomt, maar waarvoor je wel een stijl kunt definiëren.

Een stijlregel met een pseudo-element ziet er als volgt uit (merk het dubbel-punt op):

```
element:pseudo-element{  
  declaratie  
}
```

Er zijn vier verschillende pseudo-elementen, hieronder beschreven en geïllustreerd in Voorbeeld 1 (zie Code 1 en Figuur 3).

:first-line Met dit pseudo-element kan je de eerste regel van een tekst apart opmaken. We hebben het dus niet over de eerste *zin*, maar over de eerste *regel*.

:first-letter Ongeveer hetzelfde als :first-line, maar met :first-letter kan de eerste letter van een bepaald element anders worden opgemaakt dan de rest van het element.

:before :after Met deze twee pseudo-elementen kunnen, samen met de content eigenschap, extra tekst of afbeeldingen toegevoegd worden aan het begin (:before) of einde (:after) van het element.

::selection Wanneer de gebruiker tekst selecteert in de browser, kan je de opmaak hiermee wijzigen. Dit wordt geïllustreerd in Voorbeeld 2 (zie Code 2 en Figuur 4). **Opgepast:** slechts beschikbaar vanaf CSS3. Let op het dubbele dubbel-punt.

```
<html>  
  <head>  
    <title>voorbeeld: pseudo-element</title>  
    <style type="text/css">  
      p { width:150px; }  
    </style>  
  </head>  
</html>
```

5 Werken met tekst

De bladspiegel is het resultaat van hoe de verschillende beeld- en tekstelementen over de bladzijde zijn verdeeld. Het omschrijft de totale indeling van een pagina. De bladspiegel kan een website maken of breken. Een goede, *evenwichtige bladspiegel* zal bijdragen aan een mooie en rustige uitstraling van een website.

Een belangrijk onderdeel van die bladspiegel is de tekst. Via CSS hebben we een bijna onuitputtelijke voorraad aan variaties waarmee we onze teksten kunnen *opmaken*, *uitlijnen* en *positioneren*. Online kan je verschillende overviews vinden van de verschillende CSS properties, waaronder de websites [htmlcheatsheet](https://htmlcheatsheet.com)²¹ en [w3schools](https://www.w3schools.com/cssref)²². Het is niet de bedoeling alle properties te kennen, maar bekijk zeker de properties die in de onderstaande secties worden aangehaald.

5.1 Tekst

Property	Beschrijving
text-align	Aligneer tekst horizontal in een element.
text-decoration	Voeg <i>decoratie</i> toe aan een gegeven tekst.
text-indent	Indenteer de eerste lijn van een tekstblok.
text-transform	Controleert de capitalisatie (upper/lowercase) van tekst.

5.1.1 Text-shadow (vanaf CSS3)

Voeg schaduw effecten toe aan tekst. De syntax is als volgt:

```
/* eerste twee parameters zijn verplicht */
text-shadow: h-shadow v-shadow blur color;
```



```
h1 {
  text-shadow: 5px 5px 5px #FF0000;
}
```

Text-shadow nieuw in CSS3!

Note: Internet Explorer does not support the text-shadow property.

²¹ <https://htmlcheatsheet.com/css>

²² <https://www.w3schools.com/cssref>

5.1.2 Lettertype

Property	Beschrijving
font	Shorthand voor verschillende font-properties.
font-family	Hiermee verander je het lettertype van een element.
font-size	Dit stelt de tekstgrootte in.
font-style	Definieert de stijl van een tekst, e.g., <i>italic</i> en <i>normal</i> .
font-weight	Stelt de dikte in van tekst, e.g., <i>normal</i> en bold .

Belangrijke opmerking Er zijn een paar manieren om de afmeting van je font-size aan te geven. De meest voor de hand liggende is in *pixels*. Wanneer we onze font-grootte beschrijven in 'px' staat het de gebruikers niet meer toe om via de browser de corpgrootte te wijzigen.

We kunnen best onze corpgrootte in ons CSS niet beschrijven in 'px', maar eerder in *procenten*. De bezoekers van je website hebben op die manier alle vrijheid om zelf de corpgrootte nog aan te passen.

Een andere goede eenheid om je font in te beschrijven is de *em* unit.

5.1.2.1 Corpgrootte in procenten

Het eerste wat je moet weten is de standaard corpgrootte, dus zonder dat jij er in je CSS iets aan veranderd hebt. Dit is door W3C vastgesteld op 16px.



Met deze CSS regels:

```
p.groot{
font-family: verdana, arial, helvetica, sans-serif;
font-size: 100%;
}
p.kleiner{
font-family: verdana, arial, helvetica, sans-serif;
font-size: 80%;
}
```

En deze HTML-code:

```
<p class="groot">Dit corps is in de CSS op 100% gezet en is dus
16px.</p>
<p class="kleiner">Dit corps is in de CSS op 80% gezet en is dus 80%
van 16px. Hoeveel dat is mag je zelf uitrekenen.</p>
```

Bekom je dit resultaat:

Dit corps is in de CSS op 100% gezet en is dus 16px.

Dit corps is in de CSS op 80% gezet en is dus 80% van 16px. Hoeveel dat is mag je zelf uitrekenen.

5.1.2.2 Corpgrootte in em's

We gaan opnieuw uit van de standaard corpgrootte, 16px. De afspraak is dat 1em gelijk is aan de standaard corpgrootte.



Met deze CSS regels:

```
p.groot {  
  font-family: verdana, arial, helvetica, sans-serif;  
  font-size: 1em;  
}  
p.nog_groter {  
  font-family: verdana, arial, helvetica, sans-serif;  
  font-size: 1.2em;  
}  
p.kleiner {  
  font-family: verdana, arial, helvetica, sans-serif;  
  font-size: 0.8em;  
}
```

En deze HTML-code:

```
<p class="groot">Dit corps is in de CSS op 1em (=100%) gezet en is  
dus 16px.</p>  
<p class="nog_groter">Dit corps is in de CSS op 1.2em (=120%) gezet  
en is dus 120% van 16px.</p>  
<p class="kleiner">Dit corps is in de CSS op 0.8em (=80%) gezet en  
is dus 80% van 16px.</p>
```

Bekom je dit resultaat:

Dit corps is in de CSS op 1em (=100%) gezet en is dus 16px.

Dit corps is in de CSS op 1.2em (=120%) gezet en is dus 120% van 16px.

Dit corps is in de CSS op 0.8em (=80%) gezet en is dus 80% van 16px.

5.1.2.3 Geneste elementen

Of je nu met procenten werkt of met em's, je moet opletten met *geneste elementen* (een child-element van een bepaald parent-element noemen we een genest element). De afmetingen die je in je CSS aan het corps van de verschillende elementen geeft, worden als het ware bij elkaar opgeteld.

Bekijk het resultaat van het volgende voorbeeld maar eens.

```
p {
  font-family: verdana, arial, helvetica, sans-serif;
  font-size: 0.9em;
}
span {
  font-size: 0.9em;
  color: red;
}
```

```
<p>Geneste <span>elementen</span> kunnen voor onverwachte verrassingen zorgen.</p>
```

Geneste elementen kunnen voor onverwachte verrassingen zorgen.

Wat is hier exact gebeurd? Het `` element is een child van het `<p>` element. De corpgrootte voor `<p>` staat op `0.9em`. Omdat de `` een child is van de `<p>` staat daarmee de corpgrootte voor de `` eigenlijk ook al op `0.9em`.

Als je vervolgens in je CSS de corpgrootte voor de `` nóg een keer op `0.9em` zet, zet je hem dus eigenlijk op `0.9em` van `0.9em` (dus 90% van $90\% = 81\%$). De corpgrootte van de `` staat dus eigenlijk op `0.81em`.

Zoals we hierboven gezien hebben nemen child-elementen automatisch de corpgrootte van hun parent-element over. Dit gegeven gaan we gebruiken. Om in één klap de corpgrootte van alle elementen naar een eenvoudig werkbaar getal te zetten, gebruiken we het element `<body>`. Alle andere HTML elementen zijn immers altijd een child van het element `<body>`.

We zetten de corpgrootte van de `body` op `62.5%` (62.5% van de standaard `16px = 10px`). Alle denkbare HTML elementen die je vervolgens wilt gebruiken hebben een corpgrootte van `10px`.

Bekijk het volgende voorbeeld:

```
body {
  font-family: verdana, arial, helvetica, sans-serif;
  font-size: 62.5%; /* = 10px */
}
p.groot {
```

```
font-family: verdana, arial, helvetica, sans-serif;
font-size: 1.4em; /* = 14px */
}
p.klein {
font-family: verdana, arial, helvetica, sans-serif;
font-size: 1.2em; /* = 12px */
}
li {
font-family: verdana, arial, helvetica, sans-serif;
font-size: 1.5em; /* = 15px */
}
address {
font-family: verdana, arial, helvetica, sans-serif;
font-size: 1em; /* = 10px */
}
```

Het gebruik van bepaalde fonts kan vervelend zijn omdat je niet weet of dat font op het systeem van je bezoekers geïnstalleerd is. En dat is nodig. Als een bezoeker een font dat jij gebruikt niet heeft geïnstalleerd, kan de browser jouw font niet laten zien. Er zal voor een ander font gekozen worden.

Daarom moet je in je CSS altijd meerdere fonts aangeven. Het systeem van de bezoeker zal de fonts doorlopen. Het eerste font dat het systeem herkent zal getoond worden.

5.1.3 Kleur en achtergrond

Property	Beschrijving
background	Shorthand voor verschillende background-properties.
background-attachment	Een background-image dat niet mee scrollt met de pagina.
background-color	Dit stelt de achtergrond kleur in.
background-image	Dit stelt een achtergrond afbeelding in.
background-position	Dit positioneert het background-image.
background-repeat	Dit herhaalt het background-image (verticaal).
background-size (CSS3)	Stelt de grootte in van het background-image.
background-origin (CSS3)	Laat het background-image starten van de linkerboven hoek.

De achtergrond van een element kan een kleur, een gradient, een image als achtergrond hebben, of een combinatie van deze.

5.1.4 Uitbreiding: background instellen browser

Wanneer je een mooie achtergrond hebt is het de moeite waard dat deze qua grootte mee schaaft met de grootte van de browser. Staat deze klein ingesteld, dan is je afbeelding ook klein en is de browser groot geopend, dan groeit de afbeelding als het ware mee.

Wat je jezelf moet realiseren is dat een dergelijke afbeelding best een behoorlijk formaat moet hebben wil je deze ook in grote browserformaten ook mooi laten weergeven.

Er zijn inmiddels genoeg gebruikers die een 4k scherm hebben en daar zal je wellicht dus rekening mee willen houden.

Het laden van een dergelijke afbeelding kost wat bandbreedte. Probeer een middenweg te zoeken tussen de grootte van de afbeelding en de kwaliteit. Je wilt uiteraard geen korrelige achtergrond.

5.1.4.1 Een achtergrond instellen

Allereerst bepaal je de achtergrond met de volgende code:

```
html {  
  background-image: url(sneeuw.jpg);  
  background-repeat: no-repeat; /* geen herhaling van de achtergrond */  
  background-position: center center;  
  background-attachment: fixed;  
}
```

De eigenschap `background-attachment: fixed` is om de achtergrond vast te zetten als je scrolt. De achtergrond scrolt in dat geval niet mee als de inhoud van de pagina groter wordt. Deze instelling gebruik je dus wanneer je een achtergrond maakt voor het hele scherm.

We koppelen deze CSS aan de selector `html`: het is van toepassing op het hele document.

5.1.4.2 CSS3 eigenschap 'cover'

Nu komen we aan de nieuwe onderdelen toe waarmee de achtergrond wordt geplaatst in de hele browsergrootte. Dit doen we met de CSS3 eigenschap 'cover'.

```
background-size: cover
```

Hiermee krijg je een figuur die de container volledig bedekt. Het behoudt zijn hoogte-breedte verhouding tot ofwel breedte ofwel hoogte de container bedekt.



```
<html>  
  <head>  
    <style>  
      html {  
        background-image: url(sneeuw.jpg);  
        background-repeat: no-repeat;
```

```
background-position: center center;  
background-attachment: fixed;  
background-size: cover;  
}  
</style>  
</head>  
</html>
```

Na resizen van de browser blijft de achtergrond volledig gecentreerd:

