

Web Development 1

CSS 3

C. De Waele

PBA toegepaste informatica 1

Het digitaal cursusmateriaal is enkel bestemd voor persoonlijk gebruik door de gebruiker. Het is de gebruiker niet toegestaan om de digitale cursus geheel of gedeeltelijk te kopiëren, verkopen, verdelen, doorgeven, vertalen, verspreiden, weergeven, reproduceren, publiceren, verhuren, leasen, onderlicentiëren, vergunnen met licentie of op andere wijze over te dragen.

Web development I – CSS3

auteur: **Christophe De Waele**



studiegebied **HWB**

bachelor in het **toegepaste Informatica**

campus **Kortrijk**

academiejaar **2024-2025**

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.....	8
1.3.1 <i>Inline stijl</i>	9
1.3.2 <i>Stijlblok</i>	9
1.3.3 <i>Extern stijlblad</i>	9
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>	51
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	Lay-out	62
7.1	Normal flow	62
7.2	De box	63

7.3	Visual formatting model	66
7.4	Positioneren.....	68
7.4.1	<i>Zwevende elementen</i>	68
7.4.2	<i>Relatief positioneren</i>	70
7.4.3	<i>Absoluut positioneren</i>	71
7.5	Indelen van pagina.....	73
7.6	CSS Float	75
7.7	CSS Positioning.....	77
7.8	CSS Grid.....	79
7.9	FlexBox.....	83
7.9.1	<i>Flex container: display</i>	85
7.9.2	<i>Flex container : flex-direction</i>	85
7.9.3	<i>Flex container : flex-wrap</i>	85
7.9.4	<i>Flex container : flex-flow</i>	86
7.9.5	<i>Flex container : justify-content</i>	86
7.9.6	<i>Flex container : align-items</i>	86
7.9.7	<i>Flex container : align-content</i>	87
7.9.8	<i>Flex element: order</i>	88
7.10	Samenvatting	88
7.10.1	<i>Eerst benadering via Float</i>	89
7.10.2	<i>Tweede benadering: CSS-Grid</i>	90
7.10.3	<i>Derde benadering: FLEX</i>	90

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.

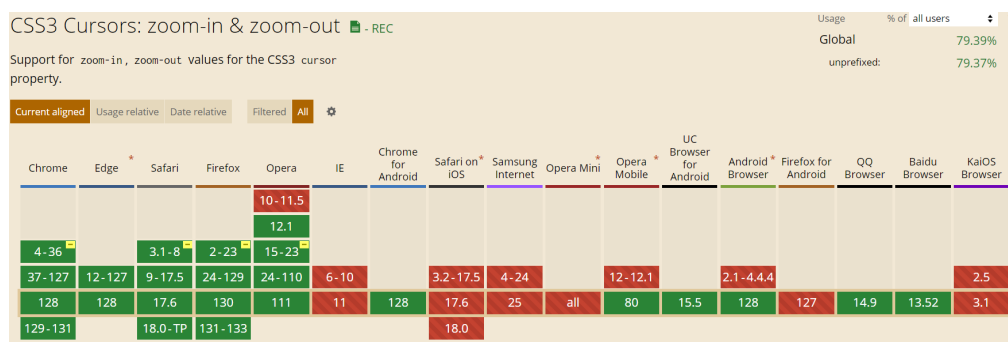
De ontwikkeling van Cascading Style Sheets (CSS) gaat terug naar 1996. Het World Wide Web Consortium (W3C), de toezichthouder op het web, publiceerde toen de specificaties van *CSS level 1*. Een nieuwe denkrichting voor het vormgeven van HTML-documenten was geboren: geen HTML-opmaakregels meer in de 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, slagschaduwen 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.



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

³ <https://caniuse.com/>

⁴ <https://css3test.com/>

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.

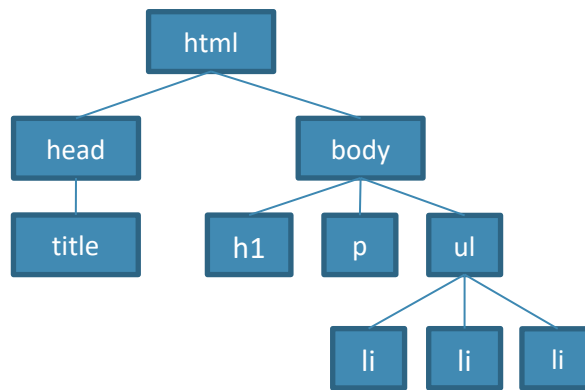
Het **Document Object Model (DOM)** is een verzameling regels en afspraken over hoe een browser een webpagina interpreteert. Het DOM zit ingebakken in een browser en je kunt er niets aan wijzigen. Maar als je weet volgens welke basisregels het DOM werkt, kun je er wel gebruik van maken.

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 <html>) 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.

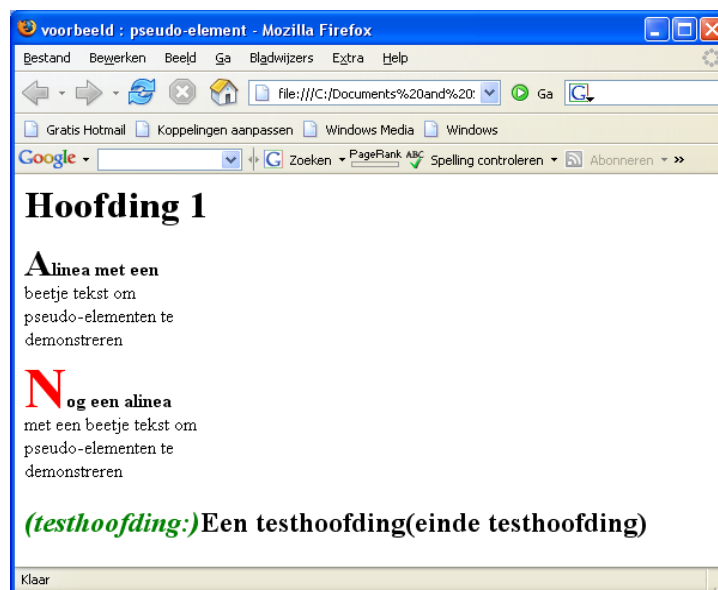
```
<html>
  <head>
    <title>voorbeeld: pseudo-element</title>
    <style type="text/css">
      p { width:150px; }
```

```

p:first-line { font-weight:bold; }
p:first-letter { font-size:24pt }
p.extra:first-letter { font-size:36pt;color:red }
h2:before {
    content: "(testhoofding)";
    color: green;
    font-style: italic
}
h2:after { content:"(einde testhoofding)" }
</style>
</head>
<body>
<h1>Hoofding 1</h1>
<p>Alinea met een beetje tekst om pseudo-elementen te demonstreren</p>
<p class="extra">Nog een alinea met een beetje tekst om pseudo-elementen te
demonstreren</p>
<h2>Een testhoofding</h2>
</body>
</html>

```

Code 1 HTML code "Voorbeeld 1".



Figuur 3 Render van "Voorbeeld 1".

```

<!DOCTYPE html>
<html>
<head>
<style>
    ::selection {
        color:red;
        background-color:blue;
    }

```

```

</style>
</head>
<body>
  <h1>Try to select some text on this page</h1>
  <p>This is a paragraph.</p>
  <div>This is some text in a div element.</div>
</body>
</html>

```

Code 2 HTML-code "Voorbeeld 2"

Try to **select some text** on this page

This is a paragraph.

This is some text in a div element.

Figuur 4 Render van "Voorbeeld 2"



Je kan zelf HTML en CSS voorbeelden uittesten op de website van w3schools⁵: plan je code in het linkse paneel van de pagina en bekijk het resultaat in het rechtse paneel door op de knop "Run" te klikken.

2.5 Pseudo-class selector

Een pseudo-class selector is een speciale constructie, die het mogelijk maakt een stijl te koppelen aan een element op basis van andere kenmerken dan de naam van het element, de attributen of de inhoud.

Je herkent een pseudo-klasse aan een dubbele punt en de naam. De syntax wordt hieronder geïllustreerd:

```

:pseudo-klasse-naam { declaratie }
element:pseudo-klasse-naam { declaratie }

```

In CSS3 is het aantal pseudo-classes sterk uitgebreid, om ingewikkeldere selecties mogelijk te maken

Notatie	Beschrijving
a:link	Selecteert niet bezochte hyperlinks.
a:visited	Selecteert bezochte hyperlinks.
a:active	Selecteert actieve hyperlinks.
:hover	Selecteert elementen waar de gebruiker over 'hangt' met zijn muisaanwijzer.

⁵ http://www.w3schools.com/html/tryit.asp?filename=tryhtml_intro

:focus	Selecteert elementen die <i>acties</i> zijn doordat ze de <i>focus</i> krijgen.
:first-child	Selecteert het eerste child van een element.
:first-of-type	Selecteert elk element dat het eerste element van het gegeven type is van zijn ouder.
:last-of-type	Hetzelfde als <i>first-of-type</i> , maar dan het <i>laatste</i> .
:only-of-type	Hetzelfde als <i>first-of-type</i> , maar dan het <i>enige</i> .
:only-child	Selecteer elk element dat het enige kind is van zijn ouder.
:nth-child(n)	Selecteer elk element dat het n ^e kind is van zijn ouder.
:nth-last-child(n)	Hetzelfde als <i>nth-child(n)</i> , maar telt vanaf het laatste ipv het eerste kind.
:nth-of-type(n)	Selecteer elk element dat het n ^e kind, van zijn type, van zijn ouder is.
:nth-last-of-type(n)	Hetzelfde als <i>nth-of-type(n)</i> , maar telt vanaf het laatste ipv het eerste kind.
:last-child	Selecteer elk element dat het laatste kind is van zijn ouder.
:enabled	Selecteer elk <i>enabled</i> element (dit is een attribuut).
:disabled	Selecteer elk <i>disabled</i> element (dit is een attribuut).
:checked	Selecteer elk <i>checked</i> element (dit is een attribuut).
:not(selector)	Selecteer elk element dat niet voldoet aan de gegeven selector.

Op de w3schools staat een testpagina⁶ waar je de verschillende pseudo-classes zelf eens kan uittesten. volgende website "", kan je bovenstaande pseudo-classes uittesten.



Voorbeeld 1

```
body {font-family: Arial, Helvetica, sans-serif; background-color: #003366;
```

⁶ http://www.w3schools.com/cssref/css_selectors.asp


```
color: #CC6600}
a:link {color: #999999; text-decoration:none}
a:visited {color: #999999; text-decoration: none }
a:hover { color: #999999; text-decoration:underline}
a:active {color: #999999; text-decoration:none }
h1 {font-size: 14px; font-style:
normal; color: #003366; background-color: #CC6600; text-align:
center}
```

In deze code merken we een buitenbeentje op, namelijk het a-element (anchor, of hyperlink). Het a-element heeft vier verschillende verschijningsvormen: standaard (link), bezocht (visited), actief (active) en muisover (hover). In het voorbeeld hierboven wordt een hyperlink *nooit onderstreept* weergegeven. Daar zorgt de 'text-decoration: none' voor.



Voorbeeld 2

```
<head>
<style>
p:first-of-type
{
background:#ff0000;
}
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>The first paragraph.</p>
<p>The second paragraph.</p>
<p>The third paragraph.</p>
<p>The fourth paragraph.</p>
</body>
```

De bovenstaande code resulteert in de volgende pagina:

This is a heading

The first paragraph.

The second paragraph.

The third paragraph.

The fourth paragraph.



Voorbeeld 3

```
<style>
p:nth-last-child(3)
{
background:#ff0000;
}
</style>
</head>
<body>
<p>The first paragraph.</p>
<p>The second paragraph.</p>
<p>The third paragraph.</p>
<p>The fourth paragraph.</p>
</body>
```

De bovenstaande code resulteert in de volgende pagina:

The first paragraph.
The second paragraph.
The third paragraph.
The fourth paragraph.

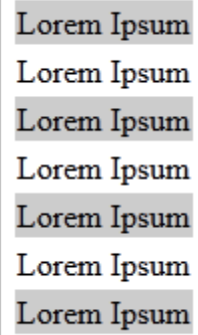


Voorbeeld 4

```
<head>
<style>
tr:nth-child(2n+1) { background-color:#ccc; }
</style>
</head>
<body>
<table>
<tr>
<td>Lorem Ipsum</td>
</tr>
<tr>
<td>Lorem Ipsum</td>
</tr>
<tr>
<td>Lorem Ipsum</td>
</tr>
<tr>
<td>Lorem Ipsum</td>
</tr>
<tr>
<td>Lorem Ipsum</td>
</tr>
<tr>
<td>Lorem Ipsum</td>
</tr>
```

```
<td>Lorem Ipsum</td>
</tr>
<tr>
  <td>Lorem Ipsum</td>
</tr>
</table>
</body>
```

De bovenstaande HTML en CSS code resulteert in de volgende pagina:



Lorem Ipsum
Lorem Ipsum
Lorem Ipsum
Lorem Ipsum
Lorem Ipsum
Lorem Ipsum
Lorem Ipsum

3 Overerfbare eigenschappen (inheritance)

Sommige eigenschappen die voor een element worden ingesteld, zijn ook van toepassing op de child-elementen van het betreffende element. Overerfbare eigenschappen zijn bijvoorbeeld `color`, `background-color` en `font`. Niet-overerfbare eigenschappen (deze moeten voor elk element opnieuw worden ingesteld) zijn `margin`, `padding`, `border`.

Overerving verschilt dus van eigenschap tot eigenschap bij CSS. Ook zonder dat je dat verder in de CSS apart aangeeft, worden sommige eigenschappen doorgegeven aan kinderen.

Zodra je zelf een bepaalde eigenschap van een element opgeeft, wint die altijd van overerving.

Elke uitgebreide handleiding over CSS zal bij elke eigenschap opgeven of die erfelijk is of niet. Een lijst van de overerfbare eigenschappen kan je vinden op de website van w3.org⁷. Een voorbeeldje kan je vinden in Figuur 5.

14.1 Foreground color: the `'color'` property

<code>'color'</code>	
<i>Value:</i>	<code><color></code> <code>inherit</code>
<i>Initial:</i>	depends on user agent
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	<code>visual</code>
<i>Computed value:</i>	as specified

Figuur 5 Voorbeeld van w3 listing voor `'color'`, dat inheritance aangeeft.

Het volgende voorbeeld illustreert hoe inheritance werkt, aan de hand van een stuk HTML-code. We beginnen met de staat die getoond wordt in voorbeeld A (zie Code 3 en

Figuur 6). Hier kan je zien dat enkel font wordt overgeërfd van de ouders, de andere eigenschappen niet.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Inherited</title>
    <style>
      div {
        font-family:serif;
        border:1px solid red;
        padding:10px;
      }
    </style>
  </head>
  <body>
```

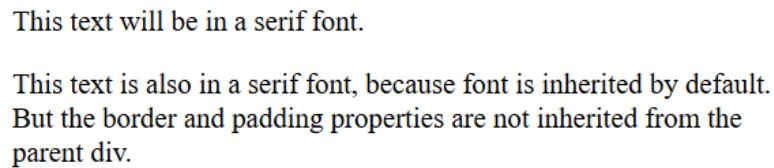
⁷ www.w3.org/TR/CSS2/cover.html

```

<div>
  This text will be in a serif font.
  <p>This text is also in a serif font, because font is inherited by default.
  But the border and padding properties are not inherited from the parent div.
  </p>
</div>
</body>
</html>

```

Code 3 Inheritance voorbeeld A



This text will be in a serif font.

This text is also in a serif font, because font is inherited by default.
But the border and padding properties are not inherited from the parent div.

Figuur 6 Visualisatie inheritance voorbeeld A

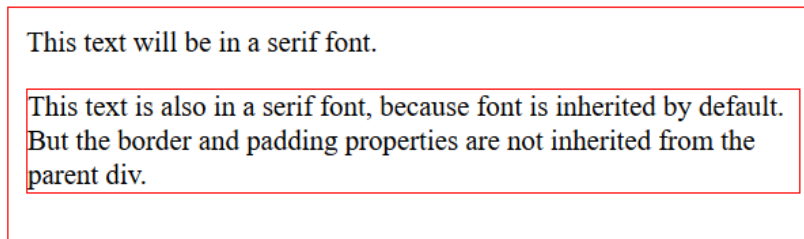
Door de waarde *inherit* op te geven bij een gegeven eigenschap, kan die eigenschap toch overgedragen worden. Dit kan je zien in voorbeeld B (zie Code 4 en Figuur 7).

```

<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Inherited</title>
    <style>
      div {
        font-family:serif;
        border:1px solid red;
        padding:10px;
      }
      p {border:inherit;}
    </style>
  </head>
  <body>
    <div>
      This text will be in a serif font.
      <p>This text is also in a serif font,
      because font is inherited by default.
      But the border and padding properties
      are not inherited from the parent div.
      </p>
    </div>
  </body>
</html>

```

Code 4 Inheritance Voorbeeld B

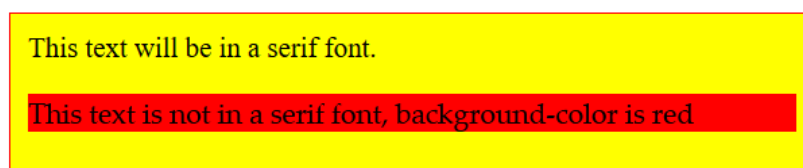


Figuur 7 Visualisatie Inheritance Voorbeeld B

Zodra je zelf voor een bepaalde eigenschap van een element een waarde opgeeft, wint die altijd van de overgeërfde waarde. Dit wordt geïllustreerd in Voorbeeld C (zie Code 5 en Figuur 8).

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Inherited</title>
    <style>
      div {
        font-family:serif;
        background-color:#FF0;
        border:1px solid red;
        padding:10px;
      }
      p {
        font-family:"Palatino Linotype", "Book Antiqua", Palatino, serif;
        background-color:red;
      }
    </style>
  </head>
  <body>
    <div>
      This text will be in a serif font.
      <p>This text is not in a serif font, background-color is red</p>
    </div>
  </body>
</html>
```

Code 5 Inheritance Voorbeeld C



Figuur 8 Visualisatie inheritance Voorbeeld C

3.1 Volgorde van uitvoeren

Stylesheets kunnen afkomstig zijn van verschillende resources en worden in de volgende *specifieke volgorde* uitgevoerd:

- 1 De browser (default look and feel).
- 2 Een style sheet gedefinieerd door de gebruiker.
- 3 Van de webpagina zelf (gedefinieerd door de auteur van de website).

Het is belangrijk deze volgorde in het achterhoofd te houden. Wanneer bepaalde regels niet lijken toegepast te worden kan het zijn dat er een ander stylesheet die regels overschrijft, later in het proces.

3.2 Standaardwaarden en default stylesheet

Veel elementen hebben standaardwaarden, die automatisch worden gebruikt. Maar zodra met behulp van CSS een andere waarde wordt opgegeven, wordt die andere waarde gebruikt. Een waarde opgegeven met behulp van CSS wint altijd van een eventuele standaardwaarde.

Overigens zitten er nogal wat verschillen tussen de diverse browsers wat betreft sommige standaardwaarden. Bijvoorbeeld de standaard-padding en -marge in een lijst (en) verschilt enorm tussen de verschillende browsers.

Die verschillen worden deels veroorzaakt, omdat elke browser een ingebouwd standaard (default) stijlbestand heeft. Een voorbeeld van zo'n default stylesheet is te vinden op de site van w3.org⁸ Een extract daarvan kan je zien in Code 6.

```
h1, h2, h3, h4, h5, h6, b, strong {
  font-weight: bolder
}
blockquote {
  margin-left: 40px;
  margin-right: 40px;
}
```

Code 6 Extract van de CSS2 default stylesheet op w3.org.

3.3 Stylesheet van gebruiker

Ook een gebruiker kan zelf een stijlbestand opgeven. Als zo'n stijlbestand aanwezig is en de bouwer van de site zelf geen CSS-regels heeft opgegeven, wordt de CSS van de gebruiker gebruikt en niet de default stylesheet. Maar zodra de bouwer CSS-regels heeft opgegeven, winnen deze.

Omdat dit een stijlbestand is dat (als het al aanwezig is) van gebruiker tot gebruiker verschilt, zal het weinig problemen opleveren. De gebruiker kent het en weet wat het doet, en de bouwer weet niet

⁸ <http://www.w3.org/TR/CSS2/sample.html>

eens dat het bestaat. Het is alleen handig om te weten dat het *kán* bestaan. Dit is bijvoorbeeld de reden dat het in de regel goed is, om te zorgen voor een voorgrondkleur én een achtergrondkleur (dus niet enkel één van beide).

Als de gebruiker, bijvoorbeeld omdat die moeite heeft met contrast, zelf kleuren opgeeft, kan het misgaan als de voorgrondkleur (waaronder de kleur van de tekst) van de bouwer en de achtergrondkleur van de gebruiker door elkaar worden gebruikt.

In principe wint de CSS van de bouwer. Daar is één uitzondering op: als de gebruiker *!important* heeft gebruikt (zie verder). Dan wint de bijbehorende stijlregel van de gebruiker *altijd*. **Ook** als de bouwer *!important* heeft gebruikt.

3.4 Stijlsheet van de auteur (bouwer)

Bij CSS worden de afzonderlijke stijlregels volgens een strikte volgorde uitgevoerd. Stel dat je een externe stylesheet gebruikt en daarnaast ook een stijlblok in de <head> van de pagina. Welke regels worden er effectief toegepast wanneer er tegenstrijdige stijlregels inzitten?

3.4.1 Stijlregels op één plaats

Normaal genomen worden stijlregels die binnen hetzelfde bestand staan, volgens de volgorde binnen dat bestand uitgevoerd: van boven naar beneden. Daarbij ‘wint’ een *latere* regel van een *vroegere*. Hetzelfde geldt voor de stijlregels die bij elkaar in een stijlblok binnen de <head> van een pagina staan. Stel dat in de CSS de volgende regels staan:

```
div { color: black; }
div { color: red; }
div { color: blue; }
```

De kleur van de div wordt nu blauw, want dat is de laatste regel in de CSS. En die overtroeft de twee eerdere regels. Er mogen hier andere regels tussen deze drie regels worden ingevoegd, dat maakt niet uit. De laatste regel met blauw wint van de twee eerdere met zwart en rood.

Hierbij is een hele belangrijke voorwaarde dat de selector (bij bovenstaande regels is dat de element selector *div*) dezelfde is, alsook dat die evenveel gewicht en *specificiteit* heeft. Als daar verschil in zit is het iets complexer om te bepalen welke regel er effectief *wint* en wordt toegepast. Dit zullen we verder in de cursus bekijken.

3.4.2 Meerdere externe stijlbestanden

Als er vanuit de pagina naar meerdere externe stylesheets wordt gelinkt, worden deze in volgorde van het linken uitgevoerd. Ze worden als het ware achter elkaar gezet. De stijlregels uit de bestanden worden dan gecombineerd toegepast. Ze vullen elkaar aan. Stel dat je in het eerste CSS-bestand de volgende eigenschap opneemt

```
h1 { font-family: courier; }
```


en dat in het tweede CSS-bestand, de volgende regel staat:

```
h1 { color: blue; }
```

In het document zullen de kopteksten <h1> in het **blauw** en `courier` worden weergegeven.

Als je in de eerste stylesheet een achtergrond rood maakt bij <div>, terwijl je die in de tweede stylesheet blauw maakt, dan wordt de achtergrond blauw. Let er bij het combineren van stylesheets altijd goed op dat de volgorde in de header van het document correct is.

3.4.3 Stylesheet, stijlblok en inline

In principe gaat een stijlblok in de <head> boven een extern stijlbestand. Het maakt daarbij niet uit of de link naar de externe stylesheet boven of onder het stijlblok staat: het stijlblok wint gewoon altijd. En een inline-style wint van een stijlblok.

Maar ook hier geldt weer dat de specificiteit en het gewicht van de selector gelijk moeten zijn. Als de regel uit het externe stijlbestand meer specificiteit heeft, wint die regel toch (zie verder).

3.4.4 Geïmporteerde stylesheets

Met behulp van `@import` kan een stijlbestand in een ander stijlbestand of stijlblok (bovenin de <head> van de pagina) worden geïmporteerd. De inhoud van een geïmporteerd stijlbestand wordt gewoon volledig tussengevoegd op de plaats waar `@import` staat. En omdat `@import` altijd helemaal bovenaan moet staan, is dat altijd helemaal bovenaan.

Het geïmporteerde bestand wordt volledig tussengevoegd, waardoor één stijlblok of stylesheet ontstaat. En daarbinnen gelden weer de gewone regels.



```
<style>
  @import url (style/styles.css);
  body {
    font-family:serif;
    background-color:#FF0;
  }
</style>
```

3.4.5 Specificiteit (specificity)

Selectors met meer specificiteit overrulen selectors met minder specificiteit. De specificiteit kan je berekenen met behulp van de volgende regels^{9,10}:

⁹ <https://www.w3.org/TR/selectors-4/#specificity-rules>

¹⁰ <https://www.w3.org/TR/CSS21/cascade.html#specificity>



De specificiteit van een selector wordt voorgesteld door een getal *s* dat bestaat uit een aantal *cijfers*: a, b, c en d, berekend uit de onderdelen (op hun beurt selectors) van die selector.

- **a** is 1 wanneer de CSS regel uit een *style attribuut* komt in plaats van een *selector*, anders is het 0.
- **b** is het aantal gebruikte id attributes in de selector.
- **c** is de som van het aantal gebruikte attributen (geen id) en pseudo-classes in de selector.
- **d** is de som van het aantal gebruikte element namen en pseudo-elementen in de selector.

Als je deze cijfers achter elkaar plaatst (**abcd**) verkrijg je *s*, de specificiteit van de beschouwde selector.

Van de bovenstaande regels worden hier een aantal voorbeelden gegeven:

Selector	Selector type	Specificiteit
*	universal selector	0000 (a = 0, b = 0, c = 0, d = 0)
li	element naam	0001 (a = 0, b = 0, c = 0, d = 1)
ul li	element naam	0002 (a = 0, b = 0, c = 0, d = 2)
div h1+p	element naam	0003 (a = 0, b = 0, c = 0, d = 3)
input[type="text"]	element naam + attribuut	0011 (a = 0, b = 0, c = 1, d = 1)
.someclass	class naam	0010 (a = 0, b = 0, c = 1, d = 0)
div.someclass	element naam + class naam	0011 (a = 0, b = 0, c = 1, d = 1)
div.someclass.someother	element naam + class naam + class naam	0021 (a = 0, b = 0, c = 2, d = 1)
#eenid	id naam	0100 (a = 0, b = 1, c = 0, d = 0)
div#eenid	element naam + id naam	0101 (a = 0, b = 1, c = 0, d = 1)

In de volgende sectie worden drie voorbeelden gegeven waarin geïllustreerd wordt wat de impact is van de specificiteit van CSS regels.



```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>De cascade</title>
    <style>
      body {
        font-size: 24px;}
      p { background-color: red; }
      p { background-color: yellow; }
    </style>
  </head>
  <body>
    <p> Deze paragraaf heeft een gele achtergrond. </p>
    <p> Deze paragraaf heeft OOK een gele achtergrond,
      omdat de laatste selector bepalend is.</p>
  </body>
</html>
```

Deze paragraaf heeft een gele achtergrond.

Deze paragraaf heeft OOK een gele achtergrond, omdat de laatste selector bepalend is.



```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>De cascade</title>
    <style>
      body { font-size: 24px; }
      p.geen { background-color: red; }
      p { background-color: yellow; }
    </style>
  </head>
  <body>
    <p> Deze paragraaf heeft een gele achtergrond. </p>
    <p class="geen">
      Deze paragraaf heeft geen gele achtergrond, omdat de
      class-selector meer specifiek is dan de element selector.
    </p>
  </body>
```

```
</html>
```

Deze paragraaf heeft een gele achtergrond.

Deze paragraaf heeft geen gele achtergrond, omdat de class-selector meer specifiek is dan de element selector.



```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>specificiteit </title>
    <style>
      body {font: 14px sans-serif;}
      span#specifiek {background: pink;}
      span {background: red;}
      span {background: yellow;}
    </style>
  </head>
  <body>
    <p>Specificiteit wordt bepaald door de specificiteit van de
      selector. <span id="specifiek"> Een meer specifieke selector
      wint</span> over een <span>meer vage (algemene)
      selector.</span></p>
    <p>De volgorde is niet van belang totdat er één of meer
      selectoren zijn met dezelfde specificiteit die allen naar
      hetzelfde element verwijzen. In dit geval <span> wint de
      laatste !!</span></p>
  </body>
</html>
```

Specificiteit wordt bepaald door de specificiteit van de selector. Een meer specifieke selector wint over een meer vage (algemene) selector.

De volgorde is niet van belang totdat er één of meer selectoren zijn met dezelfde specificiteit die allen naar hetzelfde element verwijzen. In dit geval wint de laatste !!

3.5 !important regels

Normaal gesproken werkt CSS van boven naar beneden. Als je bepaalde onderdelen in je stylesheet definieert geldt altijd de laatste gedefinieerde style.



In dit voorbeeld wordt eerst alle padding op 0px gezet, waarna enkel padding-left op 10px gezet wordt. Dit wordt effectief toegepast zoals verwacht: De paragraaf krijgt aan de linkerkant een beetje padding.

```

<head>
  <style>
    p {
      padding: 0px;
      padding-left: 10px;
    }
  </style>
</head>
<body>
  <h1>!Important</h1>
  <p>This is a paragraph</p>
  <div>This is some text in a div element</div>
</body>

```

!Important

This is a paragraph

This is some text in a div element

De idee van !important is dat de style waar je het achter plaatst, altijd voorrang heeft, ongeacht wat er na komt. Dit wordt geïllustreerd in het volgende voorbeeld.



Hier zorgt !important ervoor dat padding-left niet meer toegepast wordt. De regel padding:0px krijgt voorrang.

```

<head>
  <style>
    p {
      padding: 0px !important;
      padding-left: 10px;
    }
  </style>
</head>
<body>
  <h1>!Important</h1>
  <p>This is a paragraph.</p>
  <div>This is some text in a div element.</div>
</body>

```

!Important

This is a paragraph.

This is some text in a div element.



```
<html>
  <head>
    <title>Important regel</title>
    <style>
      body { font: 24px sans-serif; }
      p {background: pink !important; }
    </style>
  </head>
  <body>
    <p> Deze paragraaf heeft een rose achtergrond</p>
    <p style="background:lightblue;">
      Deze paragraaf heeft ook een rose achtergrond
      niettegenstaande het stijl attribuut.
    </p>
  </body>
</html>
```

Deze paragraaf heeft een rose achtergrond

Deze paragraaf heeft ook een rose achtergrond
niettegenstaande het stijl attribuut.

4 Visuele effecten

In CSS2.1 kan je niks aan een element animeren of veranderen. In CSS3 wordt het echter mogelijk om elementen te draaien, bewegen en van vorm te veranderen. Vroeger moest je hiervoor vertrouwen op Javascript¹¹.

De CSS3 specificaties worden langzaam maar zeker in meerdere webbrowsers ondersteund. In de meeste gevallen betreft dit een experimentele ondersteuning. Wanneer de specificatie wordt geschreven en gedebatteerd bij het W3C, besluiten browsers om sommige eigenschappen toch te ondersteunen en de werking op deze manier te testen in de realiteit.

Vaak wordt er bij deze experimentele ondersteuning gebruikgemaakt van een zogenaamd 'prefix'. Elke browser heeft zijn eigen prefix. Browsers negeren dus prefixes die ze niet herkennen. De prefixes voor de verschillende moderne browsers zijn als volgt:

Engine	Browsers	Prefix
WebKit	Safari, iOS	-webkit-
Blink ¹²	Chrome, Edge, Brave, Opera	-webkit-
Gecko	Firefox, Thunderbird	-moz-
KHTML / WebKit	Konqueror	-khtml-

4.1 Werking van prefixes

De werking van prefixes zullen we uitleggen aan de hand van de border-radius property. Stel dat we de hoeken van een element rond willen maken met een radius van 10 pixels. Hiervoor kunnen we de volgende CSS regels gebruiken:

```
.rounded {
  border: 2px solid;
  background-color: #ccc;
  padding: 10px;
  width: 250px;
  -webkit-border-radius: 10px;
  border-radius: 10px;
}
```

Deze CSS regels toepassen op, bijvoorbeeld een <div> met wat tekst in, resulteert in de volgende visualisatie:

¹¹ Of de Flash en Silverlight frameworks, ondertussen niet meer in gebruik.

¹² Fork van (gebaseerd op) WebKit.

The border-radius property allows you to add rounded corners to elements.

In dit voorbeeld is gebruik gemaakt van een class. De browser engine WebKit (gebruikt door o.a. Safari) ondersteunt de eigenschap d.m.v. hun eigen prefix, terwijl andere browsers de eigenschap ondersteunen *zonder* een vendor prefix.

Denk eraan om de eigenlijke CSS3 eigenschap (in dit geval 'border-radius') als laatste op te nemen in de declaratie. Op het moment dat een browser uiteindelijk de eigenschap implementeert, overschrijft de echte CSS3 eigenschap de experimentele versie (met prefix). Zo ben je er zeker van dat de uiteindelijke standaard steeds zal toegepast worden.

Gelukkig zijn er ook verschillende hulpmiddelen verschenen om je te helpen browser-specifieke CSS3 te schrijven, e.g., CSS3 Generator¹³ en CollorZilla¹⁴.



Figuur 9 Screenshot van generatie border-radius regels door CSS3 Generator.

4.2 Transformations

Hiermee wordt het mogelijk om, via CSS, gedrag toe te kennen aan elementen.

Met CSS3 *Transitions* en *Animations* kan je visuele effecten maken die de aandacht van de bezoeker trekken. Dit kan *zonder* daarbij JavaScript te gebruiken. 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.

Waar verschillen ze dan in?

- **Transitions** reageren op een verandering in een CSS property, e.g., een :hover event of een class wissel via Javascript.
- **Animations** vergen geen expliciete trigger: ze starten zodra je ze gedefinieerd hebt.

¹³ www.css3generator.com

¹⁴ www.colorzilla.com/gradient-editor

Een **transition** is een overgang zoals een hover-effect maar minder plotseling, met een effect gespreid over een bepaalde tijd. Een transition instellen doe je met enkele CSS properties die in de volgende secties worden uitgelegd.

4.2.1 transition-property

Met de transition-property zeg je welke CSS property (of een lijst van komma-gescheiden properties) 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.

4.2.2 transition-duration

De transition-duration property geeft de tijdspanne aan van de transitie. Zijn er meerdere transition-property's opgegeven, dan kan je ook hier een lijst van komma-gescheiden tijden instellen. De tijd wordt uitgedrukt in seconden (s) of milliseconden (ms). 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;
}
```

4.2.3 transition-timing-function

De transition-timing-function property gebruikt een bepaalde functie om de overgang te maken:

Functie	Beschrijving
linear	continu
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
cubic-bezier(getal,getal,getal,getal)	eigen instelling

Alle functies zijn afgeleiden van cubic-bezier. De argumenten voor deze functie moeten waarden zijn tussen 0 en 1. Ook voor deze CSS3 functies kan je online generator websites vinden om je het leven gemakkelijker te maken¹⁵. De standaard instelling is ease.



```
div {
  transition-property: opacity;
  transition-duration: 3ms;
  transition-timing-function: cubic-bezier(0.62, 0, 0.38, 1.0)
}
```

4.2.4 transition-delay

De transition-delay property stelt de start uit van de transitie nadat deze werd geactiveerd. De waarde 0 betekent “onmiddellijk”:



```
div {
  transition-property: opacity;
  transition-duration: 3ms;
  transition-timing-function: cubic-bezier(0.62, 0, 0.38, 1.0);
  transition-delay: 1ms;
}
```

4.2.5 transition

De voorgaande functies kan je combineren in een kortere schrijfwijze: transition. De syntax is als volgt:

```
transition: transition-property || transition-duration || transition-timing-
function || transition-delay [, volgende transition ]*
```



```
div { transition: opacity 3ms cubic-bezier(0.62, 0, 0.38, 1.0) 1ms;}
```

Bovenstaande CSS regel is identiek aan de hieronder volledig uitgeschreven regels.

```
div {
  transition-property: opacity;
  transition-duration: 3ms;
  transition-timing-function: cubic-bezier(0.62, 0, 0.38, 1.0);
  transition-delay: 1ms;
}
```

¹⁵ <http://cubic-bezier.com/#.17,.67,.83,.67>

4.2.6 Voorbeeld

Om de uitleg kort te houden tonen we in onderstaande code-voorbeelden enkel de standaard properties. Let erop dat in een echte applicatie de prefixes wel worden gebruikt.

```
#d1:hover, #d2:hover {
    background-color:#FF33FF;
}
#d3:hover {
    background-color:#FF33FF;border-color:#FF0;font-size:1.2em;
}
#d2 {
    transition:background-color .5s ease-in;
}
#d3 {
    transition:background-color .5s ease-in, border-color 1s ease-out,
        font-size 1s ease-in;
}
```

Dit resulteert in het volgende gedrag:

- d1: hover zonder transitie
- d2: hover met transitie
- d3: meerdere transities

Het id #d1, heeft een eenvoudige :hover ingesteld: het effect is onmiddellijk. #d2 heeft dezelfde :hover pseudo class ingesteld, maar er is *ook* een transition ingesteld die de background-color langzaam wijzigt. #d3 heeft ook een :hover pseudo class die echter meerdere *transition-properties* wijzigt.

Bijkomende info kan je vinden op w3schools¹⁶, waar je ook voorbeelden kan uittesten.

4.3 Borders (randen)

De borders module heeft enkele wijzigingen ondergaan: we kunnen nu een image als border instellen en we kunnen de hoeken afronden.

4.3.1 Afgeronde hoeken

De property border-radius bepaalt de mate van afronding van een hoek. Je geeft een waarde (als lengte of percentage) mee voor elke hoek. Je kan ook niet-symmetrische (elliptische) hoeken creëren door twee waarden op te geven met de "/" notatie. Een waarde 0 betekent een rechte hoek, dus geen afronding.

¹⁶ https://www.w3schools.com/css/css3_transitions.asp



w3c syntax afgeronde hoeken

```

voor symmetrische hoeken
/* alle hoeken */
[<length>|<percentage>]
/* top-left/bottom-right en top-right/bottom-left */
[<length>|<percentage>] [<length>|<percentage>]
/* 4 hoeken */
[<length>|<percentage>] [<length>|<percentage>]
[<length>|<percentage>] [<length>|<percentage>]
voor niet-symmetrische hoeken
/* horizontale straal / verticale straal */
[<length>|<percentage>] / [<length>|<percentage>]

```

Enkele voorbeelden:



Alle hoeken regelmatig en gelijk.

```

<style>
  div {
    border:2px solid black;
    padding:5px; /*afstand tussen rand en tekst*/
    background:#dddddd;
    width:300px;
    border-radius:10px
  }
</style>

```

The border-radius property allows you to add rounded corners to elements.



Alle hoeken regelmatig, maar top-left = bottom-right en top-right = bottom-left.

```

div {
  border:2px solid black;
  padding:5px; /*afstand tussen rand en tekst*/
  background:#dddddd;
  width:300px;
  border-radius:10px 30px
}

```

The border-radius property allows you to add rounded corners to elements.



Alle hoeken een andere radius.

```
div {
  border: 2px solid black;
  padding: 5px; /*afstand tussen rand en tekst*/
  background: #dddddd;
  width: 300px;
  border-radius: 5px 10px 15px 20px; /*TL TR BR BL */
}
```

The border-radius property allows you to add rounded corners to elements.



Alle hoeken dezelfde verhouding.

```
div {
  border: 2px solid black;
  padding: 5px; /*afstand tussen rand en tekst*/
  background: #dddddd;
  width: 300px;
  /* voor de slash horizontale radius, erna verticale */
  border-radius: 50px / 10px;
}
```

The border-radius property allows you to add rounded corners to elements.



Alle hoeken een verschillende verhouding.

```
div {
  border: 2px solid black;
  padding: 5px; /*afstand tussen rand en tekst*/
  background: #dddddd;
  width: 300px;
  border-radius: 10px 5px 10px 20px / 5px 10px 20px 5px;
}
```

De border-radius hierboven kan je trouwens identiek herschrijven als de volgende regels.

```
border-top-left-radius: 10px 5px ;
border-top-right-radius: 5px 10px;
border-bottom-right-radius: 10px 20px ;
```

```
border-bottom-left-radius:20px 5px ;
```

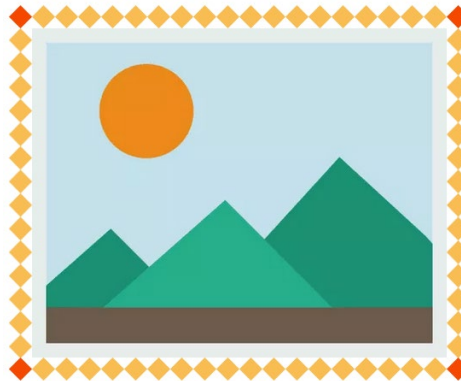
The border-radius property allows you to add rounded corners to elements.

4.3.2 border-image

De nieuwe CSS property `border-image`¹⁷ laat toe een figuur te gebruiken i.p.v. een border-style. Bekijk zeker de pagina <http://border-image.com> om hier zelf mee te spelen. Gebruik bijvoorbeeld Figuur 10 om alles uit te testen.



Figuur 10 Testafbeelding voor border-image property¹⁸.



Figuur 11 Resultaat van border-image toe te passen via border-image.com¹⁹.

De eigenschap `border-image` is de *shorthand* versie van 3 sub-properties:

- **border-image-source:** het bronbestand dat je wil gebruiken als border afbeelding.
- **border-image-slice:** definieert hoe je de figuur in 9 stukken moet delen om het als border te kunnen gebruiken.
- **border-image-repeat:** definieert hoe de afbeelding wordt herhaald, gestretched, of afgerond.

Twee voorbeelden:

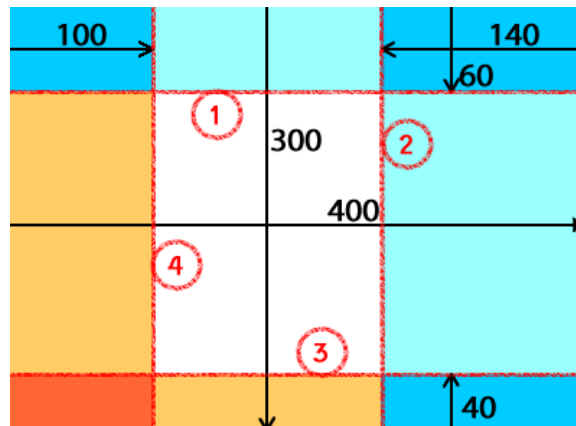
```
border-image: url(image.gif) 40 repeat;
border-image: url(image.gif) 60 140 40 100 repeat;
```

¹⁷ <https://www.w3.org/TR/css-backgrounds-3/#border-images>

¹⁸ <https://www.w3.org/TR/css-backgrounds-3/images/border.png>

¹⁹ Bekom dit resultaat op <http://border-image.com> door de 81x81 afbeelding uit Figuur 10 te gebruiken. Offset zet je op 27. Size set je op 27. Repeat zet je op 'Round'.

Veronderstel dat we het tweede voorbeeld toepassen op een figuur met dimensie 400x300. De 4 waarden na de URL zijn de border-image-slice waarden in pixels (enkel getal, geen “px” eraan schrijven), of in procenten (je moet “%” eraan schrijven). Elke “snijlijn” wordt in Figuur 12 voorgesteld als een genummerde rode lijn. Zo krijgen we dus vier “hoekslices” (blauw, oranje), vier “randmiddenslices”(lichtblauw, lichtoranje) en één centrale slice (wit). Voor het eerste voorbeeld zouden we 4 gelijke hoekvierkanten van 40 px krijgen.



Figuur 12 Illustratie werking border-image.

Hoe het image verder behandeld wordt hangt af van de border-image-repeat waarden die volgen: repeat, round of stretch.



De volgende regels kan je toepassen op een <div> om een afbeelding als border te gebruiken.

```
border-style: solid;
border-width: 27px 43px 36px 27px;
-moz-border-image:
  url(http://www.w3.org/TR/css3-background/border.png)
  27 43 36 27 repeat;
-webkit-border-image:
  url(https://www.w3.org/TR/css-backgrounds-3/images/border.png)
  27 43 36 27 repeat;
-o-border-image:
  url(https://www.w3.org/TR/css-backgrounds-3/images/border.png)
  27 43 36 27 repeat;
border-image:
  url(https://www.w3.org/TR/css-backgrounds-3/images/border.png)
  27 43 36 27 fill repeat;
```



4.3.3 Box-schaduw

Met de CSS3 property `box-shadow`²⁰ plaats je één of meer schaduwen op een box.



```
div {  
  width:300px;  
  height:100px;  
  background-color:yellow;  
  /* Firefox 3.6 and earlier */  
  -moz-box-shadow: 10px 10px 5px #888888;  
  box-shadow: 10px 10px 5px #888888;  
}
```



De syntax is als volgt, de eerste 2 parameters zijn verplicht:

```
box-shadow: h-shadow v-shadow blur spread color inset;
```

²⁰ http://www.w3schools.com/cssref/css3_pr_box-shadow.asp

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 schaal 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'.

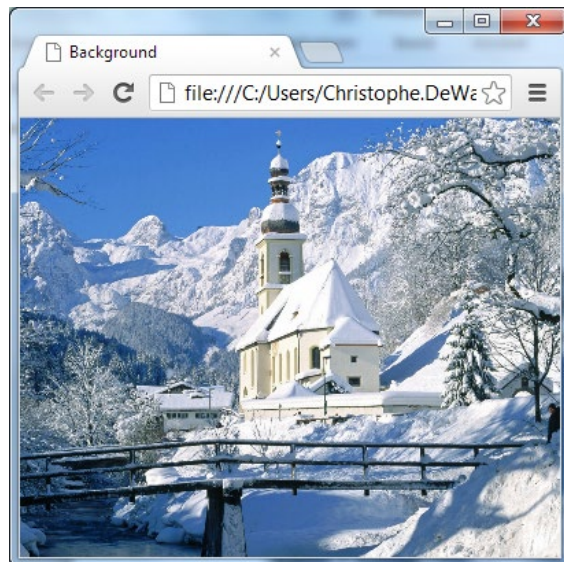
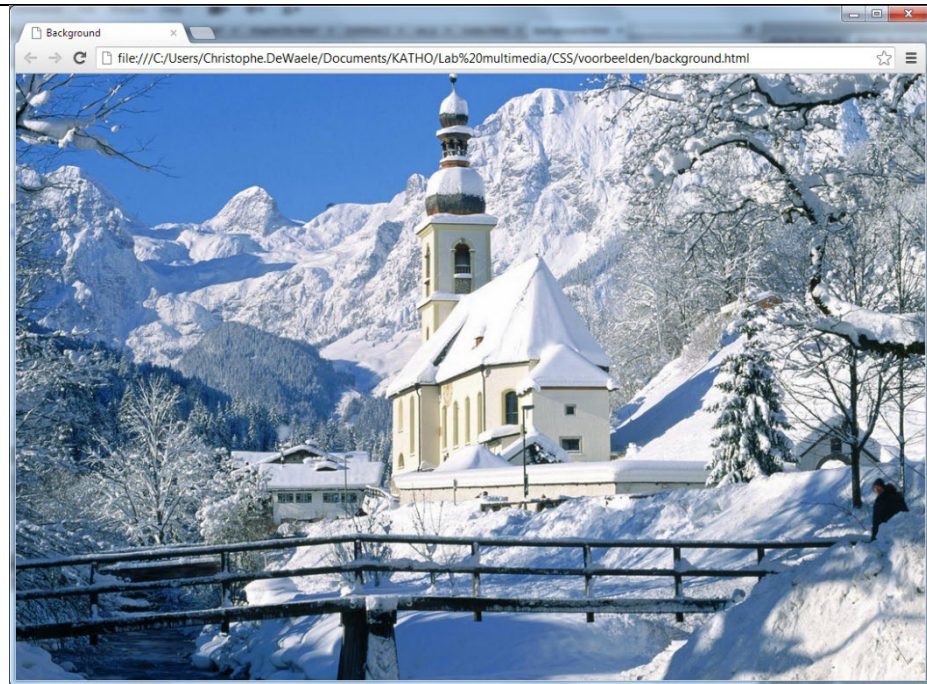
```
background-size: cover
```

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:



6 Navigatie

Het ligt voor de hand dat de **navigatiestructuur** in je website behoort tot de belangrijkste aandachtspunten²³. Met een duidelijke, logische navigatie zorg je ervoor dat de bezoekers van je website vinden wat ze zoeken.

Bij navigatiestructuren kun je denken aan concepten als:

- Hoofdnavigatie
- Subnavigatie
- Zoekfunctie
- Sitemap
- Veelgestelde vragen

Het zijn allemaal methodes om de bezoekers van je website te helpen bij het vinden wat ze zoeken.

In dit hoofdstuk behandelen we twee van de belangrijkste van de bovenstaande structuren: *hoofdnavigatie* en *subnavigatie*.

6.1 Het opmaken van een link in CSS.

Een link (in de html een ``) heeft standaard al zijn eigen opmaak. Het is underlined en heeft een blauwe kleur. Wanneer er een keer op geklikt is (de link is dan *visited*) heeft het een paarsachtige kleur.

Via CSS kan je de opmaak wijzigen (zie ook: *pseudo-klassen*). In CSS kent een link vier mogelijke verschijningsvormen:

- `a:link`, de standaard link
- `a:visited`, een link waar een keer op geklikt is
- `a:hover`, de bezoeker gaat met de cursor over de link heen
- `a:active`, het moment dat de bezoeker op de link klikt

Je kunt voor ieder van de verschijningsvormen een aparte opmaak definiëren in je CSS. Het is daarbij wel belangrijk dat je de bovenstaande volgorde aanhoudt. Dus eerst `a:link`, dan `a:visited`, dan `a:hover` en als laatste `a:active`.

Een voorbeeld wordt hieronder gegeven. Pas de volgende CSS-regels:

```
a:link {
  background-color: #FF8080;
  color: #B30700;
  text-decoration: none;
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-weight: bold;
}
```

²³ Extra info: www.css3menu.com voor het maken van menu's.


```

}
a:visited {
  background-color: transparent;
  color: #B30700;
  text-decoration: underline;
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-weight: bold;
}
a:hover {
  background-color: #B30700;
  color: #FFFFFF;
  text-decoration: none;
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-weight: bold;
}
a:active {
  background-color: transparent;
  color: #B30700;
  text-decoration: underline;
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-weight: bold;
}

```

Op de volgende HTML-code:

```
<a href="http://www.vives.be">linkje naar de VIVES-site</a>
```

Als je dit voorbeeld zelf probeert en de link bekijkt en gebruikt, kun je een paar dingen zien gebeuren.

- De standaard link heeft een eigen opmaak en is niet underlined (`text-decoration: none;`)
- Als je er met je muis overheen gaat verandert de tekstkleur en de achtergrondkleur.
- Als je op de link klikt en je muisknop ingedrukt houdt krijg je de 'active' link te zien.
- Als je op de link geklikt hebt krijg je de 'visited' link te zien. Eén van de stijl eigenschappen hiervan is bijvoorbeeld dat hij wel 'underlined' is.

Als je na het klikken op een link toch weer de standaard opmaak van de link te zien wilt krijgen is daar maar één manier voor: het verwijderen van je browsergeschiedenis. Zolang je dat niet doet blijf je de 'visited' link zien. Er is een oplossing: geef visited dezelfde opmaak als link en geef active dezelfde opmaak als hover.

Dit geeft ons meteen de mogelijkheid om de stijldeclaraties wat in te korten:

```

a:link, a:visited {
  background-color: #FF8080;
  color: #B30700;
  text-decoration: none;
  font-family: Verdana, Arial, Helvetica, sans-serif;
}

```

```

    font-weight: bold;
}
a:hover, a:active {
    background-color: #B30700;
    color: #FFFFFF;
    text-decoration: underline;
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-weight: bold;
}

```

En dat kan nog wat korter. Je ziet dat font-family en font-weight in beide gevallen hetzelfde zijn.

```

a {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-weight: bold;
}
a:link, a:visited {
    background-color: #FF8080;
    color: #B30700;
    text-decoration: none;
}
a:hover, a:active {
    background-color: #B30700;
    color: #FFFFFF;
    text-decoration: underline;
}

```

Vanzelfsprekend heb je ook de mogelijkheid om aan een link een aparte class te hangen:

```

a {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-weight: bold;
}
a:link, a:visited {
    background-color: #FF8080;
    color: #B30700;
    text-decoration: none;
}
a:hover, a:active {
    background-color: #B30700;
    color: #FFFFFF;
    text-decoration: underline;
}
a.speciaal {
    font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
    font-weight: normal;
    text-decoration: none;
}

```

```

background-color: transparent;
}
a.speciaal:link, a.speciaal:visited {
color: #24006B;
border-bottom: 1px dashed #24006B;
}
a.speciaal:hover, a.speciaal:active {
color: #AA80FF;
border-bottom: 1px dashed #AA80FF;
}

```

Test deze CSS code zelf eens uit op de volgende HTML-code:

```

<p><a href="http://www.vives.be/">link naar VIVES</a></p>
<p><a href="https://www.vives.be/campus/kortrijk" class="speciaal">link naar
campus Kortrijk</a></p>

```

6.2 Het *list* element als structuur voor je navigatie

Hyperlinks (menu's) worden vaak bij elkaar in een tabel gegroepeerd. Dit is echter niet nodig. Plaats je hyperlinks in een unordered list.

CSS

```

ul {
list-style: none;
}
a:link, a:visited {
color: #F60;
text-decoration:
none;
}
a:hover, a:active {
color: #000;
text-decoration:
none;
}

```

HTML

```

<ul>
<li><a href="#">link 1</a></li>
<li><a href="#">link 2</a></li>
<li><a href="#">link 3</a></li>
<li><a href="#">link 4</a></li>
<li><a href="#">link 5</a></li>
<li><a href="#">link 6</a></li>
</ul>

```

In het voorbeeld hierboven zorgt de opmaakstijl “list-style: none” ervoor dat opsommingtekens niet worden weergegeven. Het resultaat van deze code wordt zo gevisualiseerd:

[link 1](#)
[link 2](#)
[link 3](#)
[link 4](#)
[link 5](#)
[link 6](#)

Om een **subnavigatie** in de **hoofdnavigatie** te verwerken kunnen we een `` in een `` nesten. Om correct te zijn (conform de HTML standaard), zetten we een `` in een `` op deze manier:

CSS

```
ul {
  list-style: none;
}
a:link, a:visited {
  color: #F60;
  text-decoration:
none;
}
a:hover, a:active {
  color: #000;
  text-decoration:
none;
}
```

HTML

```
<ul>
<li><a href="#">item 1</a>
<ul>
<li><a href="#">subitem
1.1</a></li>
<li><a href="#">subitem
1.2</a></li>
<li><a href="#">subitem
1.3</a></li>
<li><a href="#">subitem
1.4</a></li>
</ul>
</li>
<li><a href="#">item 2</a></li>
<li><a href="#">item 3</a>
<ul>
<li><a href="#">subitem
3.1</a></li>
<li><a href="#">subitem
3.2</a></li>
<li><a href="#">subitem
3.3</a></li>
<li><a href="#">subitem
3.4</a></li>
</ul>
</li>
<li><a href="#">item 4</a></li>
</ul>
```

Het resultaat hiervan ziet er als volgt uit:

```

item 1
  subitem 1.1
  subitem 1.2
  subitem 1.3
  subitem 1.4
item 2
item 3
  subitem 3.1
  subitem 3.2
  subitem 3.3
  subitem 3.4
item 4

```

Je kunt het geheel nog aanpassen door te werken met margins en paddings, achtergrondkleuren, borders en van het feit dat de ene het child is van de andere . Dit wordt in het volgende code fragment getoond.

```

<!DOCTYPE html>
<html>
  <head>
    <title>menu</title>
    <style type="text/css">
      ul {
        list-style: none;
        margin: 0;
        padding: 0;
      }
      li {
        margin: 0;
        padding: 0;
        font-family: verdana;
      }
      li a {
        display: block;
        text-decoration: none;
        padding: 2px 5px 2px 5px; <!--top, right, left, bottom -->
        border-top: 1px solid #FFF;
        border-right: 1px solid #FEB660;
        border-bottom: 1px solid #FEB660;
        border-left: 1px solid #FFF;
        width: 100px;
      }
      li a:link, li a:visited {
        background: #F2C793;
        color: #E47C00;
        font-weight: bold;
      }
      li a:hover, li a:active {

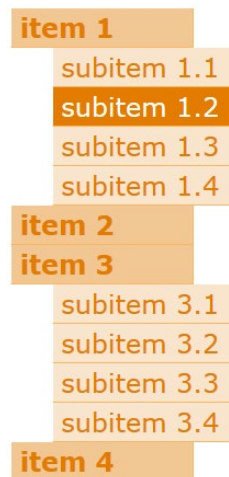
```

```

        background: #E47C00;
        color: #FFF;
        font-weight: bold;
    }
    li li a:link, li li a:visited {
        background: #F9E5CD;
        color: #E47C00;
        font-weight: normal;
    }
    li li a:hover, li li a:active {
        background: #E47C00;
        color: #FFF;
        font-weight: normal;
    }
    ul ul {
        margin: 0 0 0 25px;
    }
</style>
</head>
<body>
<ul>
    <li><a href="#">item 1</a>
        <ul>
            <li><a href="#">subitem 1.1</a></li>
            <li><a href="#">subitem 1.2</a></li>
            <li><a href="#">subitem 1.3</a></li>
            <li><a href="#">subitem 1.4</a></li>
        </ul>
    </li>
    <li><a href="#">item 2</a></li>
    <li><a href="#">item 3</a>
        <ul>
            <li><a href="#">subitem 3.1</a></li>
            <li><a href="#">subitem 3.2</a></li>
            <li><a href="#">subitem 3.3</a></li>
            <li><a href="#">subitem 3.4</a></li>
        </ul>
    </li>
    <li><a href="#">item 4</a></li>
</ul>
</body>
</html>

```

Hiermee bekom je de volgende visualisatie:



6.3 Een horizontale navigatie in een list

Tot nu toe hebben we verticaal gepositioneerde navigatie elementen behandeld. Wat als je een **horizontale navigatie** wil?

Wat je van een list (de `ul`) én van de losse list items (de `li`'s) moet weten is dat het *block elementen* zijn. Wat we eigenlijk willen is dat de `li`'s op één lijn komen te staan. We moeten dus van die block elementen *inline elementen* maken:

CSS

```
ul{
  list-style: none;
}

li{
  display: inline;
  margin: 0 10px 0 0;
}
```

HTML

```
<ul>
  <li><a href="#">link 1</a></li>
  <li><a href="#">link 2</a></li>
  <li><a href="#">link 3</a></li>
  <li><a href="#">link 4</a></li>
  <li><a href="#">link 5</a></li>
  <li><a href="#">link 6</a></li>
</ul>
```

Het bovenstaande voorbeeld resulteert in de volgende visualisatie:

[link 1](#) [link 2](#) [link 3](#) [link 4](#) [link 5](#) [link 6](#)

7 Lay-out

Het structureren van een webpagina met behulp van tabellen is **not done**. Je propt gewoon tabellen in tabellen in tabellen, net zolang tot je al je elementen gepositioneerd hebt. Tabellen zijn hier nooit voor bedoeld geweest en ze zijn ook helemaal niet nodig. Er zijn veel betere alternatieven.

In de voorgaande hoofdstukken zijn de structuur van een HTML-document en de opmaak van afzonderlijke elementen besproken. Dit hoofdstuk geeft een overzicht hoe je pagina's kan lay-outen.

Voor we inzoomen op de verschillende aspecten, bespreken we eerst nog een aantal concepten.

De **opbouw** en de **weergave** van een pagina wordt bepaald door de document normal flow, het box-model, visual formatting model en positioneringsschema. Laten we eerst de Normal flow uitleggen.

7.1 Normal flow

Realiseer je dat er zoiets is als een **natuurlijke flow** van een pagina: de manier waarop een pagina *gerendered*²⁴ wordt wanneer jij er niets aan zou veranderen, buiten elementen toevoegen aan het HTML document. Hierbij wordt de pagina van *boven naar beneden* en van *links naar rechts* opgebouwd.

Het volgende voorbeeld maakt *geen* gebruik van CSS: alle styling komt voort uit de default stylesheet (user agent stylesheet)

```
<h1>Dit is de titel van een pagina</h1>
<h2>Een subkopje</h2>
<p>De platte tekst. Zoals je ziet zijn de h1, de h2 en de p allemaal block
elementen</p>
<p>Omdat de h1 in de html bovenaan staat komt hij ook in de gerenderde pagina
bovenaan te staan, de h2 komt daar weer onder, enz, enz: de natuurlijke flow van
een pagina!</p>
```

Dit is de titel van een pagina

Een subkopje

De platte tekst. Zoals je ziet zijn de h1, de h2 en de p allemaal block elementen

Omdat de h1 in de html bovenaan staat komt hij ook in de gerenderde pagina bovenaan te staan, de h2 komt daar weer onder, enz, enz: de natuurlijke flow van een pagina!

De manier waarop dit gerenderd wordt lijkt logisch, boven naar beneden van links naar rechts.

Bij het positioneren van je elementen speelt de *flow* van een pagina een belangrijke rol en **de natuurlijke flow van de pagina is altijd ons uitgangspunt**.

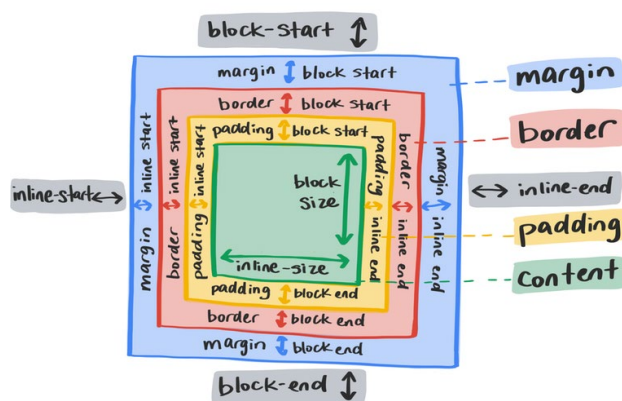
²⁴ Of: getekend.

Een webpagina is opgebouwd uit elementen: h-, p-, li-, tabel-elementen enzovoort. De opbouw van zo'n element wordt beschreven in het boxmodel. Daarover gaat het volgende subonderdeel

Hoe al die blokken op een pagina zich tot elkaar verhouden, met andere woorden: hoe met al die blokken de lay-out wordt opgebouwd, wordt beschreven in het weergavemodel (visual formatting model). Dat komt aan bod in het subonderdeel 7.3. Hierbij speelt de eigenschap `display` een hoofdrol, want die bepaalt globaal gezien de lay-out van de pagina: komen blokken naast of onder elkaar, is het een flexbox lay-out of een grid lay-out, enzovoort. Dan is er nog het positioneringsschema. Dit gaat over afwijken van de normale plaatsing met floats of absolute positionering.

7.2 De box

Wat betreft de vormgeving van een element wordt er bij *Cascading Style Sheets* van uitgegaan dat elk element wordt vormgegeven door een rechthoekig gebied, de **box**. Van binnen naar buiten bestaat de box uit de inhoud van het element, de padding, de border en de margin. Dit wordt visueel voorgesteld in onderstaande figuren.



Figuur 13 Visuele voorstelling box model.

De **margin** is de afstand van de rand van een element tot aan de rand van het parent element of de rand van een aangrenzend element. De margin is transparant en neemt dus de achtergrondkleur aan van het parent element.

De **border** is de rand van een element.

De **padding** is de afstand tussen de rand van een element en de inhoud. Indien een achtergrondkleur gebruikt wordt voor het element, dan heeft de padding dezelfde kleur.

De **grootte** van de box wordt bepaald door de som van de afmetingen van het element en de breedte van de margin, de border en de padding.

De **breedte** van de margin, de border en de padding worden vastgelegd met de eigenschappen voor de margin, de rand en de padding. De afmetingen van de inhoud van het element kunnen worden vastgelegd met de eigenschappen width en height.

Bij elementen op blokniveau is sprake van *één rechthoekige box*. Een list-item element (li) is een speciale vorm van een element op blokniveau: de box wordt voorafgegaan door een markering. De wijze waarop de markering wordt weergegeven, kan bepaald worden met behulp van de eigenschappen voor lijsten.

In verticale richting schuiven aangrenzende verticale margins in elkaar. De resterende margin krijgt de hoogte van de grootste margin. In het geval van negatieve margins, wordt de grootste negatieve margin afgetrokken van de grootste positieve margin. Wanneer er geen positieve margins zijn wordt de grootste negatieve margin aangehouden.

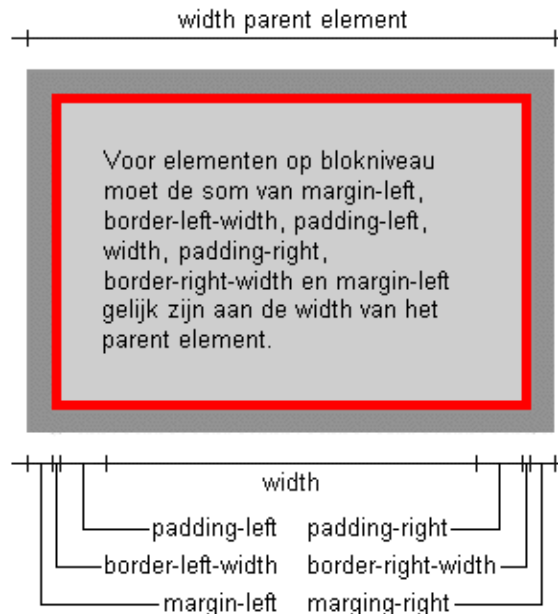
In de volgende figuur wordt het in elkaar schuiven van verticale margins gedemonstreerd. Voor een combinatie van twee h1 elementen zijn verschillende waarden voor margin-top en margin-bottom gebruikt. Bijvoorbeeld voor de combinatie linksboven:

```
<div style="border:1px solid black;">
  <h1 style="margin-bottom: 0px; background-color: silver;
    font-size: 18px; width: 200px;">margin-bottom: 0px </h1>
  <h1 style="margin-top: 0px; background-color: silver;
    font-size: 18px; width: 200px;">margin-top: 0px </h1>
</div>
```

In de volgende figuur wordt dit patroontje herhaald voor verschillende waarden voor de margins:

margin-bottom: 0px margin-top: 0px
margin-bottom: -5px margin-top: 5px
margin-bottom: 0px margin-top: 5px
margin-bottom: -5px margin-top: 10px
margin-bottom: 5px margin-top: 10px
margin-bottom: 10px margin-top: 10px

De horizontale afmeting van de box wordt bepaald door zeven eigenschappen: `margin-left`, `border-left-width`, `padding-left`, `width`, `padding-right`, `border-right-width` en `margin-right`. De *totale horizontale afmeting* van deze eigenschappen is altijd gelijk aan de *breedte van het parent element*. Dit kan je zien in **Fout! Verwijzingsbron niet gevonden..**



Figuur 14 Illustratie breedte parent element.

Om er voor te zorgen dat de totale breedte van de zeven genoemde eigenschappen gelijk is aan de breedte van het parent element, moet minstens één eigenschap de waarde **auto** hebben. Er zijn drie eigenschappen, waaraan de waarde **auto** kan worden gegeven: `margin-left`, `width` en `margin-right`.

Standaard heeft `width` de waarde **auto**. Bij een “replaced element”, een element waarvan de plaats ingenomen wordt door de inhoud waarnaar het element verwijst, wordt **auto** vervangen door de intrinsieke waarde van het element (bijvoorbeeld bij het `img` element door de breedte van de afbeelding). In dat geval moeten `margin-left` of `margin-right` ervoor zorgen dat de totale breedte van de box gelijk is aan de breedte van het parent element.

Wanneer geen enkele eigenschap de waarde **auto** heeft, dan krijgt `margin-right` automatisch de waarde **auto** toegewezen.

Wanneer meerdere eigenschappen de waarde **auto** hebben en één daarvan is `width`, dan worden `margin-left` en/of `margin-right` op “0” ingesteld en krijgt `width` de maximale breedte.

Wanneer `margin-left` en `margin-right` *beiden* de waarde **auto** hebben, dan krijgen ze een gelijke breedte.

Voor *inline elementen* kan de `width` eigenschap **niet** gebruikt worden en betekent de waarde **auto** voor `margin-left` of `margin-right` dat de margin “0” is. Bij inline elementen wordt de box opgesplitst in meerdere kleinere elementen, wanneer niet alles op een regel past.

In het volgende voorbeeld zijn voor een `p` element met een inline stijl de afmetingen vastgelegd:

```
<p style="width: 200px; height: 150px; padding: 15px; border: solid 5px red;">
  ...
</p>
```

De totale breedte van p t/m de rand moet $(200 + 2 \times 15 + 2 \times 5 =)$ 240 pixels zijn, de totale hoogte $(150 + 2 \times 15 + 2 \times 5 =)$ 190 pixels.



Ten slotte willen we het in dit onderdeel nog even hebben over *shorthand CSS*. Je zou het Engelse shorthand kunnen vertalen als *snelschrift*. Het is een verkorte versie van bepaalde stijlregels, een soort samenvatting. Als je bijvoorbeeld de border van een div wilt beschrijven kun je dat zo doen:

```
div#header{
  border-width: 1px;
  border-color: #ff6600;
  border-style: solid;
}
```

Je bereikt precies hetzelfde resultaat met deze declaratie:

```
div#header{
  border: 1px #f60 solid;
}
```

Let ook op de kleurcode in het bovenstaande voorbeeld. Je mag alle *websafe colors* op deze manier afkorten.

7.3 Visual formatting model

Het boxmodel gaat alleen over de afmetingen van elementen. Over de weergave of het gedrag gaat het visual formatting model. Het model bepaalt of deze blokken naast of onder elkaar komen te staan.

De voorgaande subtitel had betrekking op de eigenschappen van de blokken zelf. Een pagina bestaat echter uit een (grote) verzameling van blokken die op de een of andere manier op elkaar reageren. Het resultaat daarvan is de lay-out.

Alinea's komen standaard onder elkaar te staan, net als items in een lijst. Maar bijvoorbeeld hyperlinks en afbeeldingen worden naast elkaar gezet. Tenminste, tot de 'regel' vol is, want dan gaan ze op de volgende 'regel' verder. Er is een mechanisme dat HTML-elementen een bepaalde standaardweergave

geeft en dat verschil maakt tussen 'dingen naast elkaar zetten' en 'dingen onder elkaar zetten'. Verantwoordelijk daarvoor is het visual formatting model.

Elementen die de hele breedte in beslag nemen, zoals `<p>` of `<h1>`, worden **blokelementen** genoemd (block level elements). Zij genereren een block level box. Deze elementen hebben een volgende waarde voor `display`: `block`;

Een *block element* verdraagt geen tweede block element op dezelfde regel of op hetzelfde horizontale niveau. Blokelementen worden *onder elkaar* geplaatst. Je kunt een block element *wel* combineren met een inline element.

CSS	HTML
<pre>.p1{ font-weight: bold; color: #999; } .p2{ font-style: italic; color: red; }</pre>	<pre><p class="p1"> Een 'paragraph' is een voorbeeld van een block element </p> <p class="p2"> Ik kan een tweede paragraph in de html best op dezelfde regel zetten. Het resultaat in de browser zal toch zijn dat de tweede paragraph op een nieuwe regel begint. </p></pre>

De visualisatie ziet er als volgt uit:

Een 'paragraph' is een voorbeeld van een block element

Ik kan een tweede paragraph in de html best op dezelfde regel zetten. Het resultaat in de browser zal toch zijn dat de tweede paragraph op een nieuwe regel begint.

Elementen die geen gestapelde blokken vormen maar doorlopen in de regel, heten **inline** elementen (inline level elements). Deze elementen genereren een inline box en hebben een van de volgende waarden voor `display`:

- inline
- inline-block

Met een inline-block wordt een blokelement inline geplaatst.

Deze elementen worden *horizontaal* naast elkaar geplaatst. Er kunnen dus meerdere inline elementen op 1 regel staan:

CSS	HTML
-----	------

```
.groen{
  background: #269603;
  color: #FFF;
}
.rood{
  background: #EA2303;
  color: #FFF;
}
```

```
<p>
Het element span is een inline element
<span class="groen">zoals je hier
zien</span>. Je kunt eenvoudig <span
class="rood"> meerdere span's op 1
regel kwijt</span>. Ze zitten elkaar
niet in de weg.
</p>
```

De visualisatie hiervan ziet er als volgt uit:

Het element SPAN is een inline element zoals je hier kunt zien. Je kunt eenvoudig meerdere SPAN's op 1 regel kwijt. Ze zitten elkaar niet in de weg.

Hoe elementen zich gedragen in de weergave wordt bepaald door de eigenschap *display*. Daarom vult de inhoud van een `<p>`-element vanzelf de hele breedte en loopt met `` benadrukte tekst gewoon door in de tekstregel. De waarde van *display* kan echter worden veranderd om gewenst gedrag af te dwingen door de default waarde van property te overrulen.

7.4 Positioneren

Het positioneringsschema bepaalt hoe de inhoud op het scherm wordt geplaatst. Je kan elementen positioneren buiten de natuurlijke flow van de pagina. De mogelijkheden zijn *zwevende (float)* of specifieke *positionering (relatief, absoluut en fixed)*.

7.4.1 Zwevende elementen

In afwijking van hun normale plaats op het canvas kunnen elementen ook *zwevend* gemaakt worden. De box van een zwevend element wordt langs de linker- of rechterrاند van het parent element geplaatst en de boxen van andere elementen kunnen hun plek ernaast krijgen.

Met behulp van de *float* eigenschap kunnen alle elementen zwevend gemaakt worden. Wel is het nodig de breedte van het element **expliciet** vast te leggen met de **width** eigenschap.

De eigenschap *float* kent de waarden: *left*, *right* en *none*. In feite wordt een element dat de eigenschap *float* krijgt uit de natuurlijke flow van de pagina getild. De elementen die in de HTML-structuur na het 'zwevende' element komen herkennen het 'zwevende' element niet meer als onderdeel van de flow van de pagina.

Maar in dit geval hebben we daarvoor twee goede oplossingen:

- We kunnen van een element dat volgt op een 'zwevend' element ook een 'zwevend' element maken. Hiermee stellen we in feite het probleem uit, maar vaak komt dit goed van pas.
- We kunnen de eigenschap '**clear**' gebruiken. Hiermee kunnen we een 'niet-zwevend' element als het ware waarschuwen: "let op! In de HTML-structuur boven je bevinden zich floating

elements. Hou daar rekening mee en behandel ze alsof ze gewoon onderdeel uitmaken van de normale flow.” De eigenschap `clear` kent de waarden: `left`, `right`, `both` en `none`.

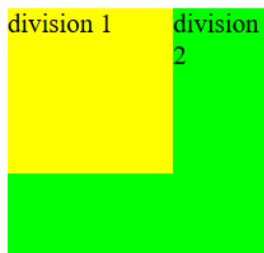
In het volgende voorbeeld is een element op blokniveau met de eigenschap `{float: left}` zwevend gemaakt. Het volgende element op blokniveau wordt, als de float eigenschap er niet voor gebruikt is, normaal in de linker bovenhoek van het parent element geplaatst en daardoor gedeeltelijk bedekt door het zwevende element. De opbouw van het document is:

```
<body>
  <div class="div1">division 1</div>
  <div class="div2">division 2</div>
</body>
```

De opbouw van het stijlblok in de head van het document is:

```
<style type="text/css">
  .div1 { float: left; width: 100px; height: 100px; background: yellow; }
  .div2 { width: 160px; height: 150px; background: lime; }
</style>
```

Deze code visualiseert als volgt:



Om de box van twee elementen op blokniveau naast elkaar te plaatsen, moet je voor beide elementen de float eigenschap opnemen. De `margin-left` eigenschap is gebruikt om wat ruimte tussen de twee boxen te creëren.

CSS

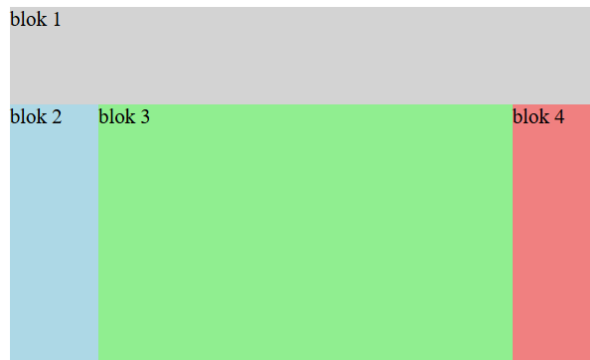
```
div#box1{
  background-color: lightgray;
  width: 100%;
  height: 75px;
}
div#box2{
  background-color: lightblue;
  float: left;
  width: 15%;
  height: 200px;
}
```

HTML

```
<div id="box1">blok 1</div>
<div id="box2">blok 2</div>
<div id="box3">blok 3</div>
<div id="box4">blok 4</div>
```

```
div#box3{
  background-color: lightgreen;
  float: left;
  width: 70%;
  height: 200px;
}
div#box4{
  background-color: lightcoral;
  float: left;
  width: 15%;
  height: 200px;
}
```

Dit rendert als volgt:



Omdat box2, -3 en -4 zwevende elementen zijn, gedragen ze zich niet meer helemaal als *block* elementen en zijn ze gedeeltelijk uit de flow van de pagina gehaald.

7.4.2 Relatief positioneren

Met de eigenschap `position: relative` wordt de plaats van het element bepaald ten opzichte van de gebruikelijke positie van het element. De ruimte die normaal gereserveerd is voor het element verplaatst niet met het element mee.

In het volgende voorbeeld bevat een element op blokniveau (`div`) een inline element (`span`). De opbouw is:

```
<body>
<div id="d1">Deze tekst vormt naast het span element <span id="s1">s1</span> de
inhoud van het div element.</div>
</body>
```

De opbouw van het stijlblok in de head van het document is:

```
<style type="text/css">
  #d1 { width: 120px; height: 150px; background: yellow; }
```



```
#s1 { font-size: 24px; background: lime; }
</style>
```

Deze tekst vormt
naast het span
element **s1** de
inhoud van het div
element.

Wanneer we het span element relatief positioneren, wordt de plaats bepaald ten opzichte van de normale positie binnen het div element. De eigenschappen `top: 80px` en `left: 20px` verplaatsen de linkerbovenhoek van het span element 80 pixels naar beneden en 20 pixels naar rechts gemeten vanaf de linkerbovenhoek van de normaal ingenomen ruimte. De normaal ingenomen ruimte verplaatst niet met het span element mee.

```
<style type="text/css">
  #d1 {
    width: 120px; height: 150px; background: yellow;
  }
  #s1 {
    position: relative; top: 80px; left: 20px; font-size: 24px; background:lime;
  }
</style>
```

Deze tekst vormt
naast het span
element de
inhoud van het div
element.

s1

7.4.3 Absoluut positioneren

Met de eigenschap `position: absolute` wordt de plaats van het element bepaald ten opzichte van de referentiebox welke gevormd wordt door het meest dichtbij gelegen (voor-)ouder element, dat de eigenschap `position` heeft met een andere waarde dan `static` (bijvoorbeeld `absolute` of `relative`). Indien zo'n element *niet* aanwezig is, wordt de plaats berekend ten opzichte van de box van het body element, waarin het gehele document zich bevindt. De ruimte die normaal gereserveerd is voor een absoluut gepositioneerd element en eventuele child elementen, verplaatst met het element mee.

Geef je een absoluut gepositioneerd element *geen coördinaten*, dan krijgt het automatisch de coördinaten: `top: 0;` en `left: 0;`

Ter illustratie gebruiken we hetzelfde voorbeeld als bij relatief positioneren.

Het div element is nu absoluut gepositioneerd. Omdat er geen parent element is met de eigenschap `position: absolute` of `position: relative`, wordt de referentiebox gevormd door het body element. De eigenschappen `top: 30px` en `left: 60px` verplaatsen de linkerbovenhoek van het div element 30 pixels naar beneden en 60 pixels naar rechts gemeten vanaf de linkerbovenhoek van het body element.

```
<style type="text/css">
  #d1 {
    width: 120px; height: 150px; position: absolute;
    top: 30px; left: 60px; background: yellow;
  }
  #s1 {
    font-size: 24px; background: lime;
  }
</style>
```

Deze tekst vormt
naast het span
element **s1** de
inhoud van het div
element.

Vervolgens wordt ook het span element absoluut gepositioneerd. De referentiebox van het span element is het div element. De eigenschappen `top: 80px` en `left: 20px` verplaatsen de linkerbovenhoek van het span element 80 pixels naar beneden en 20 pixels naar rechts gemeten vanaf de linkerbovenhoek van het div element. De normaal ingenomen ruimte verplaatst met het span element mee en de overige inhoud van het div element neemt de vrijgekomen plaats in.

```
<style type="text/css">
  #d1 {
    width: 120px; height: 150px; position: absolute;
    top: 30px; left: 60px; background: yellow;
  }
  #s1 {
    position: absolute; top: 80px; left: 20px; font-size: 24px;
    background: lime;
  }
</style>
```

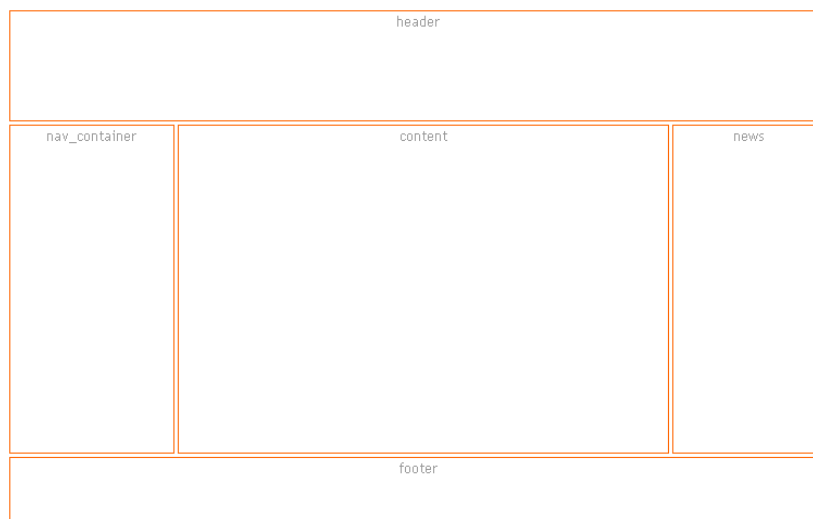
Deze tekst vormt
naast het span
element de inhoud
van het div
element.

7.5 Indelen van pagina

Nu we de belangrijkste principes doorgenomen hebben, wordt het leuk. We kunnen nu beginnen met het indelen van onze pagina, bijvoorbeeld in een *hoofding*, verschillende *kolommen* en eventueel een *footer*.

Voor de mark-up worden zowel de nieuwe HTML5-elementen gebruikt als het element `<div>`. Zoals je weet, heeft `<div>` geen betekenis zoals `<header>` en `<nav>`. Het is alleen een container om de inhoud te omsluiten, te plaatsen en op te maken met CSS. *Je gebruikt `<div>` alleen als elk ander betekenisvol element niet van toepassing is.*

HTML5 (zie cursus HTML5) heeft een aantal nieuwe elementen, die speciaal zijn bedoeld om de opbouw van een pagina aan te geven. In dit voorbeeld worden hiervan `<header>`, `<nav>`, `<article>`, `<section>` en `<footer>` gebruikt. Alle vijf gedragen zich als een gewone `<div>`, maar dan een `<div>` met een semantische betekenis.



Het uitgangspunt is het HTML-document dat alle elementen bevat. We zorgen eerst voor een ruwe indeling van de pagina. We maken gebruik van de semantische elementen en het `<div>`-element om de pagina in blokken te verdelen. Bij de `<div>`-elementen gebruik je telkens een toepasselijke 'id'. Mochten er meerdere semantische elementen met dezelfde naam voorkomen, kunnen deze ook aangesproken worden door het geven van een id-naam (vb: `<article id="news">`).

```

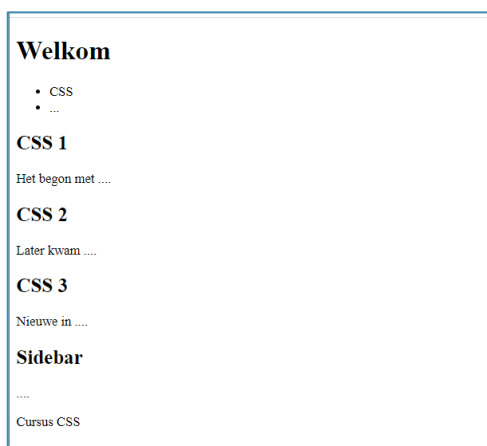
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Layout</title>
  </head>
  <body>
    <div id="container">
      <header>
        <h1>Welkom </h1>
      </header>
    </div>
  </body>
</html>
  
```

```

<nav>
  <ul>
    <li>CSS</li>
    <li>...</li>
  </ul>
</nav>
<div id="content">
  <article>
    <h1> CSS 1 </h1>
    <p>Het begon met .... </p>
  </article>
  <article>
    <h1> CSS 2 </h1>
    <p>Later kwam .... </p>
  </article>
  <article>
    <h1> CSS 3 </h1>
    <p>Nieuwe in .... </p>
  </article>
</div>
<div id="sidebar">
  <article>
    <h1>Sidebar</h1>
    <p> .... </p>
  </article>
</div>
<footer>
  <p> Cursus CSS </p>
</footer>
</div>
</body>
</html>

```

Wanneer we de bovenstaande pagina oproepen in de browser, ziet ze er als volgt uit (de natuurlijke flow van de pagina).



De verschillende blok-elementen volgen elkaar nu verticaal op. Er bestaan verschillende technieken om deze opbouw te manipuleren.

Elke techniek heeft zijn toepassingen, voordelen en nadelen, en geen enkele techniek is ontworpen om op zichzelf te worden gebruikt. Als je begrijpt waarvoor elke methode is ontworpen, zal je in staat zijn om te bepalen welke techniek de beste is om jouw doel te bereiken.

Float en Position zijn relatief oude methoden in CSS om de lay-out van pagina's aan te passen, daarom ook wel **Legacy Layout Methods** genoemd. Terwijl Flexbox en Grids erg nieuw zijn en daarom ook wel **Modern Layout Methods** worden genoemd. We hebben al wat achtergrondinformatie over Position, Float, visual formatting en met deze info kunnen we een aantal technieken bespreken.

7.6 CSS Float

De eigenschap float bepaalt of een element zweeft. (Tekst of andere elementen gaan er omheen).

Voorbeeld:

in het linker kader een rechthoek met float: left en een met float: right

In het rechter kader een rechthoek zonder float.



Opmerkingen:

- Float werkt niet bij absoluut gepositioneerde elementen.
- Vroeger werd float vaak gebruikt om verschillende blok-elementen achter elkaar op een regel te plaatsen. Meestal kan men daar gemakkelijker de nieuwe declaratie `display: inline-block` voor gebruiken.
- Een blok-element met de eigenschap float verliest enkele blokeigenschappen. Zo neemt het element niet meer de hele breedte in beslag. Je moet zelf een width opgeven!
- Float is de geëigende manier om tekst om een element heen te laten lopen.

Laten we dit even toepassen op het voorbeeld:

Het eerste “echte” element in het HTML-document is een `<div>`. Deze omsluit alle andere inhoud van het document. Zo kan voor de inhoud als een geheel een breedte worden ingesteld en ook de

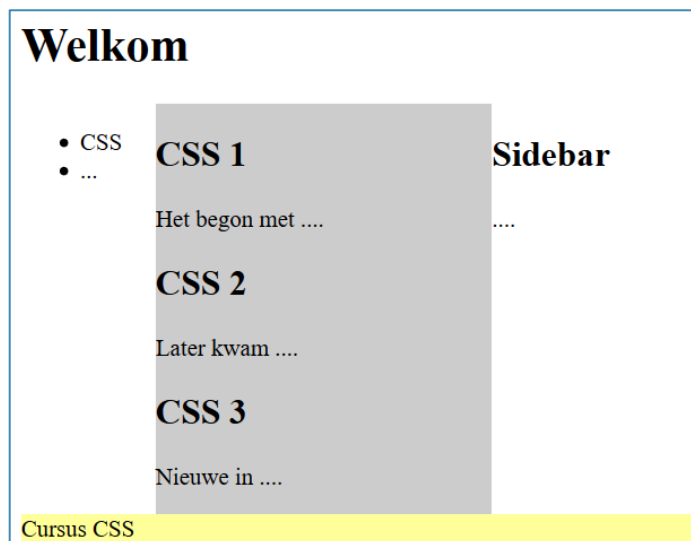
mogelijkheid om de pagina te centreren. Met de eigenschap `margin:auto`; berekent de browser hoe de vrije ruimte links en rechts moet worden verdeeld, waardoor de inhoud altijd in het midden staat.

```
div#container {
  width: 90%;
  margin:auto;
}
```

Vervolgens kiezen we `float: left` voor de volgende elementen: `<nav>`, `<div id="content">` en `<div id="sidebar">` en `clear: both` voor de 'footer'.

```
nav {
  float:left;
  width:20%;
}
div#content{
  float:left;
  width:50%;
  background-color:#CCC;
}
div#sidebar{
  float:left;
  width:30%;
}
footer{
  clear:both;
  background-color:#FF9;
}
```

Deze CSS-regels toepassen op onze basis structuur resulteert in de volgende visualisatie:



Om je te tonen waarom 'clear' zo belangrijk is, hebben we clear verwijderd. Zijn achtergrond gaat dwars door je linkerkolom en de inhoud komt onder je rechterkolom i.p.v. onder beide kolommen.

Welkom



De eigenschap clear:waarde bepaalt aan welke kant van dat element GEEN float-element mag staan.

Waarden:

- none Clear staat uitgeschakeld (standaard)
- left Het element mag NIET naast een element met float: left staan.
- right Het element mag NIET naast een element met float: right staan.
- both Het element mag NIET naast een element met float: left of float: right staan

7.7 CSS Positioning

We kunnen ook via *absoluut* en *relatief* positioneren werken. Een pagina wordt met deze manier van positioneren een statisch geheel. Niet zo'n probleem bij kleine, statische websites waarbij je op voorhand precies weet hoeveel content je hebt en waarvan je weet dat er in de toekomst niet zoveel meer aan gewijzigd zal hoeven worden. Let echter op bij grotere projecten.

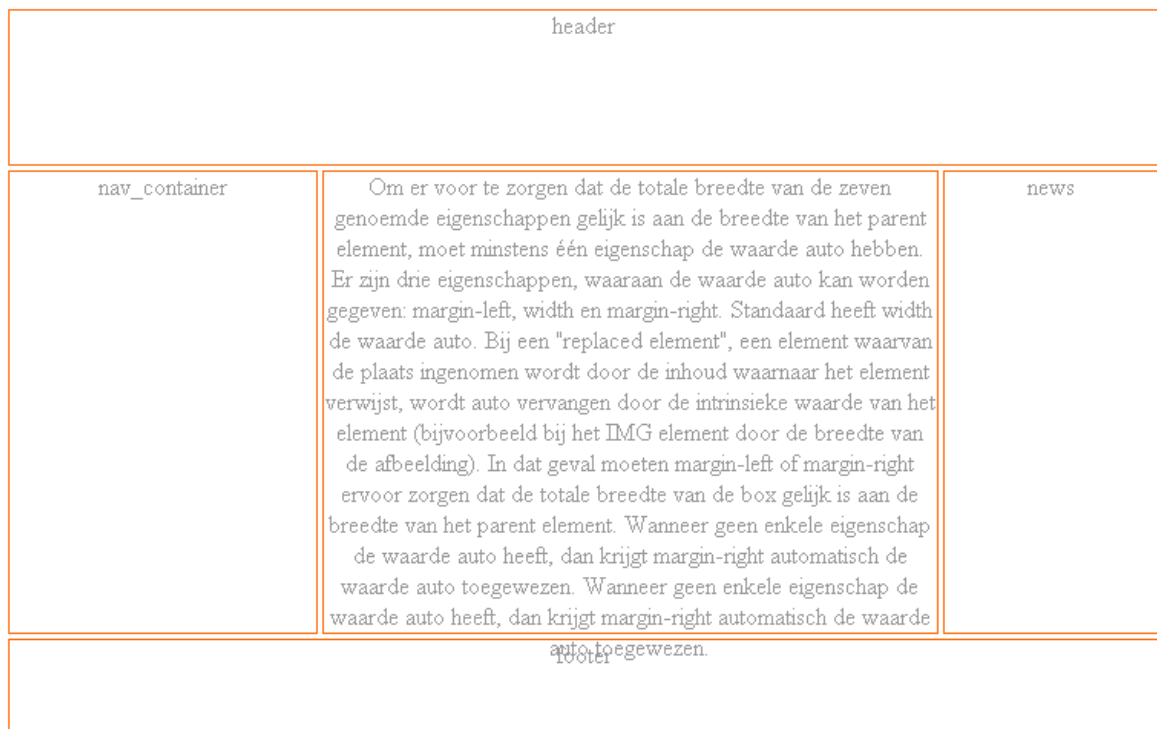
Zie hieronder een voorbeeld van een pagina opgebouwd via position: absolute.

Het grote voordeel van het werken met **absoluut positioneren** is de snelheid waarmee je een opzet kunt maken voor de lay-out van je website en de eenvoudige manier waarop je 'zeggen-schap-tot-op-de-pixel' hebt over wat waar komt te staan.

Bekijk het onderstaande voorbeeld:

```
header{
  position: absolute;
  top: 5px;
```

```
    left: 5px;
    width: 750px;
    height: 100px;
    border: 1px solid #F60;
    color: #999;
    text-align: center;
}
#nav_container{
    position: absolute;
    top: 110px;
    left: 5px;
    width: 200px;
    height: 300px;
    border: 1px solid #F60;
    color: #999;
    text-align: center;
}
#content{
    position: absolute;
    top: 110px;
    left: 210px;
    width: 400px;
    height: 300px;
    border: 1px solid #F60;
    color: #999;
    text-align: center;
}
#news{
    position: absolute;
    top: 110px;
    left: 615px;
    width: 140px;
    height: 300px;
    border: 1px solid #F60;
    color: #999;
    text-align: center;
}
footer{
    position: absolute;
    top: 415px;
    left: 5px;
    width: 750px;
    height: 60px;
    border: 1px solid #F60;
    color: #999;
    text-align: center;
}
```

Je ziet dat de tekst in de box 'content' dwars door de box 'footer' heen loopt. De footer geeft geen pixel mee en is in geen enkel opzicht flexibel ten opzichte van de boxen er omheen.

Er valt daar wel een mouw aan te passen, maar we vinden het resultaat weinig professioneel.

Je kunt aan de div #content het volgende toevoegen:

```
overflow: auto;
```

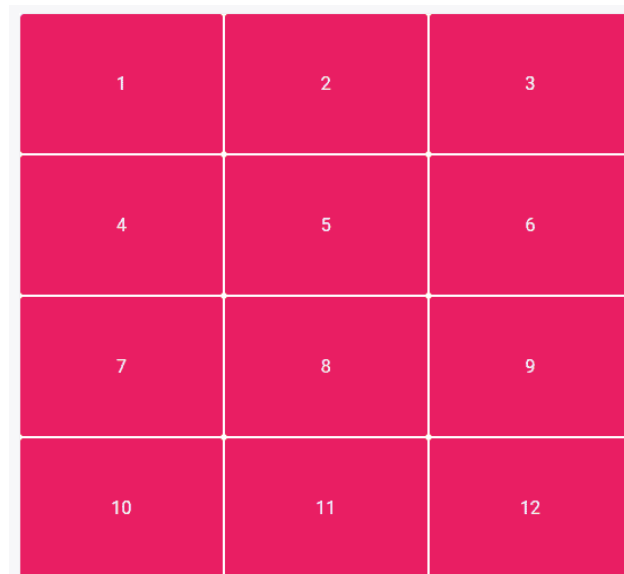
In het geval van een grotere website, waar je wat flexibiliteit in je layout in wilt bouwen, kom je niet meer weg met deze statische oplossingen. Daarom wordt deze methode zelden gekozen. De manier van lay-outen gaan we NIET gebruiken.

7.8 CSS Grid

Het is je waarschijnlijk al opgevallen dat, als je complexe lay-outs wil maken met floating elements, je soms wel extra geneste blokken nodig hebt om het design te laten werken. Wanneer een element vast zit in de structuur van de DOM, kan het enkel maar via JavaScript een nieuwe positie krijgen. Bijvoorbeeld: Het is niet eenvoudig om de footer bovenaan te plaatsen wanneer je scherm minder dan 700px breed is.

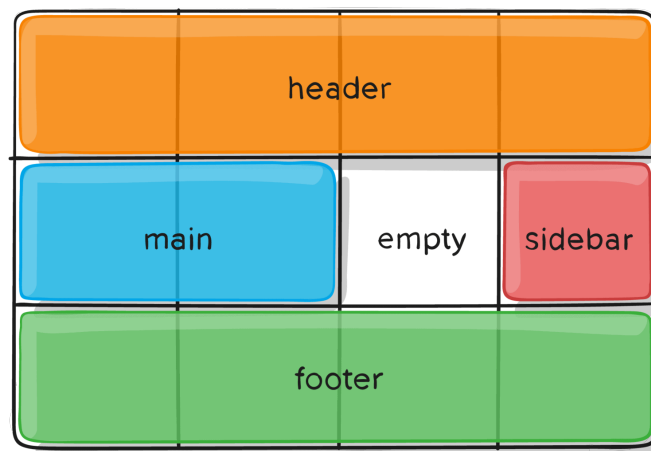
Door middel van CSS Grid (raaster) kunnen we veel vrijer omspringen met de positie van elementen. Bij Grid definiëren we eerst de grid structuur van onze lay-out, het aantal rijen en het aantal kolommen. Daarna kunnen we aangeven welk blok er op welke positie moet komen in de Grid en hoeveel grid blokjes er ingenomen moeten worden.

CSS Grid is een krachtig hulpmiddel voor het maken van tweedimensionale lay-outs



Met het gridsysteem kun je alle denkbare combinaties maken. De enige voorwaarde is dat ze rechthoekig moeten zijn. L-vormen kun je bijvoorbeeld niet construeren.

Ook een andere bijzonderheid duikt op: niet alle vakjes hoeven gevuld te zijn met inhoud. Je kunt ook lege inhoud toewijzen in de CSS Grid:



In de marge: CSS Grid is niet hetzelfde als Flexbox

Hoe ziet CSS Grid er in de praktijk uit?

Eerst bepalen we een container waaraan we de gridfuncties toewijzen.

```
.container {
  display: grid;}
```

Er zijn vele andere waarden die je kunt toekennen om de grid naar je wens op te bouwen:

grid-template-columns, **grid-template-rows** en **grid-template-areas** (gezamenlijk kunnen deze ook allemaal worden ingesteld met `grid-template`). Deze eigenschappen bevatten informatie over het

aantal en de grootte van de rijen en kolommen. Je kunt ook de afstanden tussen de rijen en kolommen definiëren met de eigenschappen `column-gap` en `row-gap` (of kortweg: `gap`).

Een afgewerkt Grid zou er dan zo uit kunnen zien aan de CSS kant:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  gap: .5rem .8rem;
  justify-items: stretch;
}
```

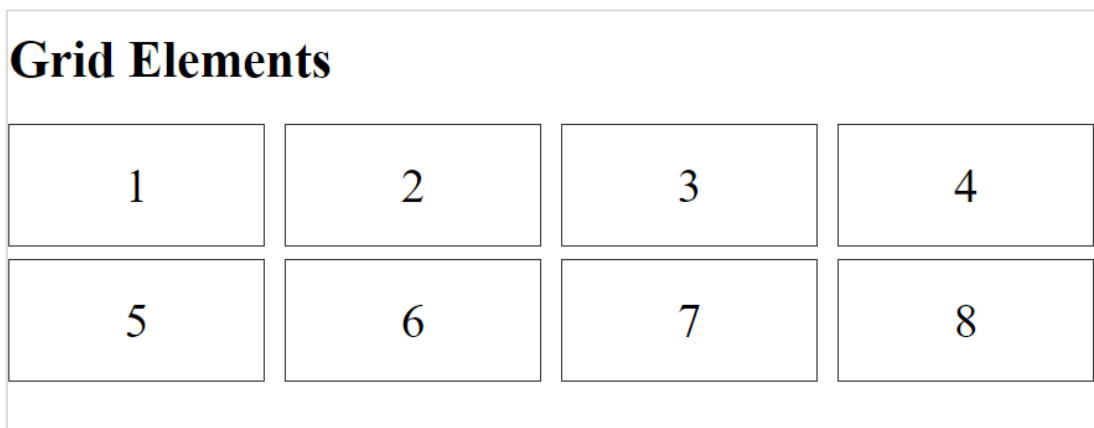
Een fraction (1fr) is een deel van een hele rij/kolom. Er worden hier vier kolommen en 3 rijen gedefinieerd. Alle kolommen/rijen zijn even groot.

Nu je je grid container hebt opgebouwd, is het tijd om deze te vullen met de grid items.

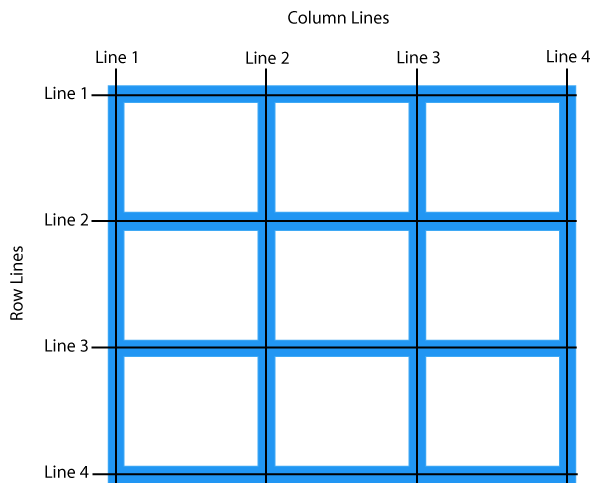
Voorbeeld:

<pre><code><h1>Grid Elements</h1> <div class="grid-container"> <div class="grid-item">1</div> <div class="grid-item">2</div> <div class="grid-item">3</div> <div class="grid-item">4</div> <div class="grid-item">5</div> <div class="grid-item">6</div> <div class="grid-item">7</div> <div class="grid-item">8</div> </div></code></pre>	<pre><code>.grid-container { display: grid; grid-template-columns: 1fr 1fr 1fr 1fr; grid-template-rows: 1fr 1fr ; gap: .5rem .8rem; justify-items: stretch; } .grid-item { background-color: rgba(255, 255, 255, 0.8); border: 1px solid rgba(0, 0, 0, 0.8); padding: 20px; font-size: 30px; text-align: center; }</code></pre>
---	--

Resultaat:



Het bijzondere van het CSS gridsysteem is dat de indeling in de HTML secundair kan zijn, omdat je elk item ook een begin- en eindcoördinaat kunt geven. Dit wordt gedaan met `grid-column-start`, `grid-column-end`, `grid-row-start`, `grid-row-end`.



De coördinaten wijzen naar de gridlijnen en tellen van links naar rechts, of van boven naar beneden. Deze kunnen ook worden afgekort als `grid-column` en `grid-row`.

Voorbeeld: `<h1>Grid Lines</h1>`

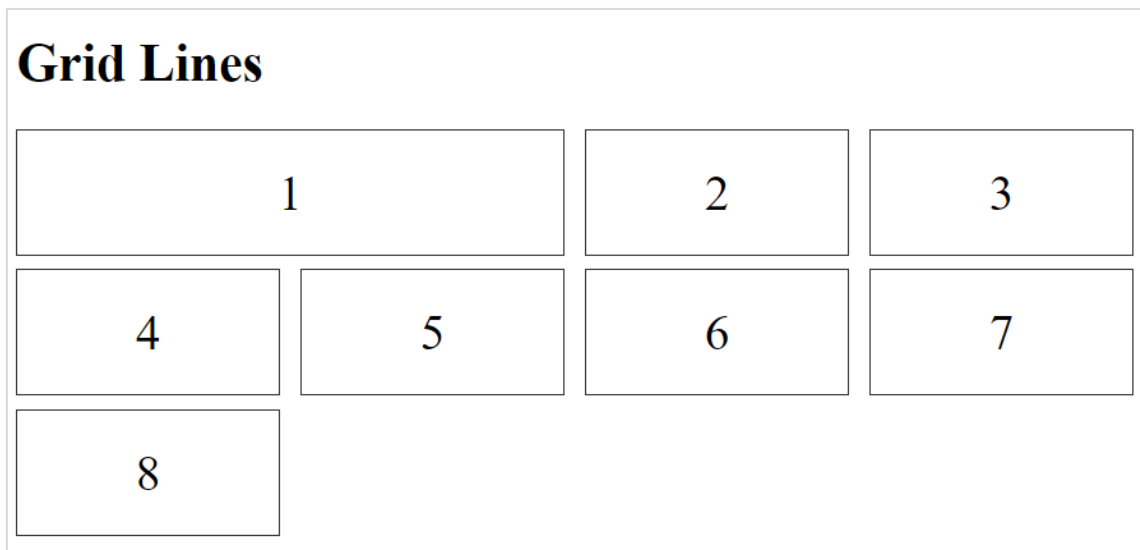
```
<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
</div>
```

```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr ;
  gap: .5rem .8rem;
  justify-items: stretch;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.8);
  padding: 20px;
  font-size: 30px;
  text-align: center;
}

.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

Resultaat:



Via de Developer Console van Chrome kan je eenvoudig de structuur van een Grid debuggen.

Alle mogelijkheden bespreken in deze cursus is onmogelijk. Via de volgende site: <https://css-tricks.com/snippets/css/complete-guide-grid/#top-of-site> kan je je verdiepen in deze techniek.

Andere bronnen:



- https://www.w3schools.com/css/css_grid.asp
- <https://css-tricks.com/snippets/css/complete-guide-grid/>
- <https://www.youtube.com/watch?v=AqwPrR7hkIE>
- <https://www.youtube.com/watch?v=7kVeCgQCxIk>

7.9 FlexBox

De afgelopen jaren is de CSS-lay-out volwassen geworden, met speciale tools voor complexe lay-outs die de verschillende oplossingen van het gebruik zwevende, absolute positionering enzovoort vervangen. Flexbox (of de Flexibele Box Layout Module) was de eerste van deze speciale lay-outmodules, gevolgd door CSS Grid Layout.

Flexbox is een eendimensionaal lay-out systeem dat dus altijd maar in een richting tegelijk werkt. Dat maakt het een stuk eenvoudiger dan CSS Grid, maar krachtig genoeg om eenvoudige lay-outs snel te realiseren. (Je kunt Flexbox ook nesten, waardoor je toch twee richtingen op kunt)

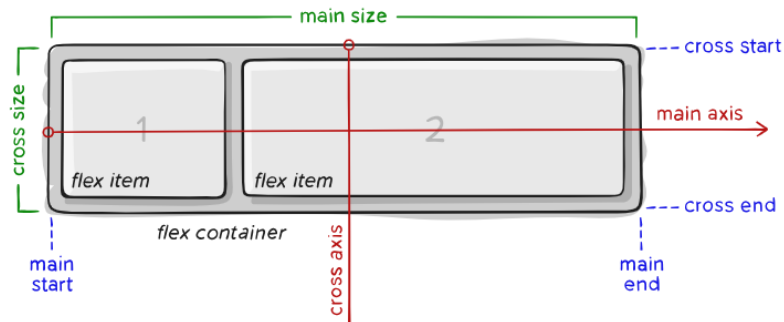
Daarnaast is Flexbox ook bedoeld om een aantal lay-out problemen op te lossen waarmee webbouwers al jaren tobben: verticale centrering, gelijkmatige verdeling van elementen binnen de ruimte, kolommen met gelijke hoogte ongeacht de inhoud, het is allemaal mogelijk.

Enkele voordelen van flexbox zijn:

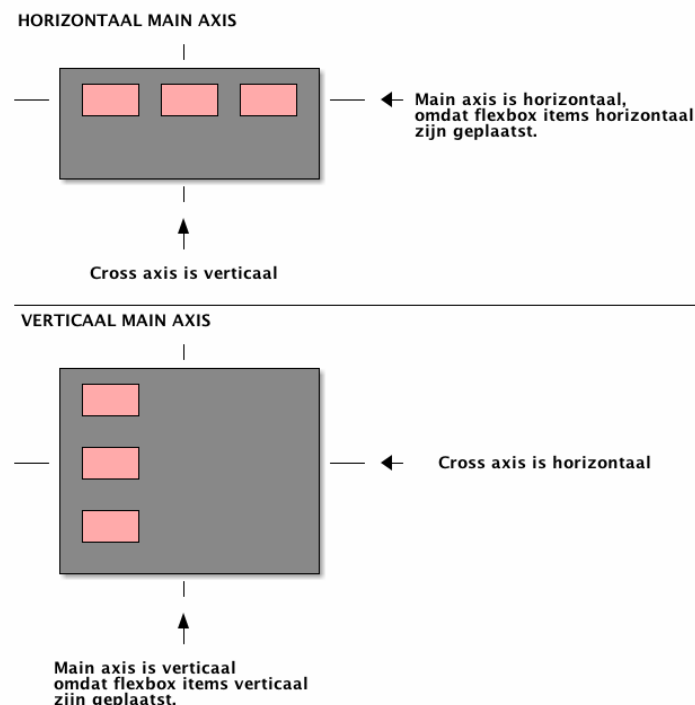
- pagina-inhoud kan in elke richting worden opgemaakt (naar links, naar rechts, naar beneden of zelfs naar boven)

- De visuele volgorde van stukjes inhoud kan worden omgekeerd of opnieuw worden gerangschikt
- het bereiken van lay-outs met gelijke kolommen (ongeacht de hoeveelheid inhoud in elke kolom) zonder hacks.

Flexbox is een lay-out systeem waarbij flex-elementen (flex-items) flexibel in een container worden verdeeld. De verdeling op de pagina is afhankelijk van de beschikbare ruimte in combinatie met zowel de eigenschappen van de flex-container als de eigenschappen van de flex-items.



Om te begrijpen welk effect een aantal Flexbox properties hebben op flex-items is het belangrijk om te begrijpen wat de **main axis** en **cross axis** zijn. De **main axis** is de as waarop de flexbox items worden geplaatst. Met andere woorden, als de flexbox items horizontaal (row, dus) worden geplaatst, dan is de horizontale as de **main axis**. Verticaal is dan de **cross axis**. Bij verticaal geplaatste flexbox items is dat precies andersom: de **main axis** is verticaal en de **cross axis** is horizontaal.

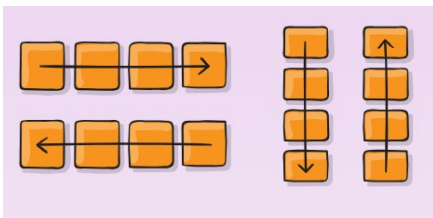


7.9.1 Flex container: display

Dit geeft aan dat het element een flex container is. Het zorgt ervoor dat alle directe kind-elementen flex-items zijn.

```
.container {
  display: flex;
}
```

7.9.2 Flex container : flex-direction



De richting waarin flex elementen worden geplaatst, wordt bepaald door flex-direction. Flexbox elementen worden geplaatst in horizontale rijen of verticale kolommen. Eventueel is ook omloop (wrapping) mogelijk, dit moet dan apart worden opgegeven.

```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
}
```

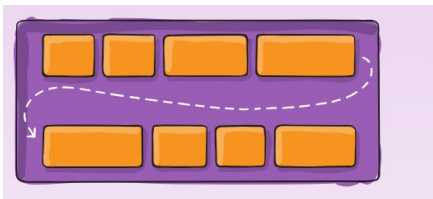
row (standaard): van links naar rechts

row-reverse: van rechts naar links

column: van boven naar beneden

column-reverse: van beneden naar boven

7.9.3 Flex container : flex-wrap



Via flex-wrap kan de omloop van de elementen in een flex container worden aangegeven. Standaard is er geen omloop.

```
.container{
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

nowrap (standaard) : alle flex elementen in dezelfde rij

wrap: flex elementen lopen door over meerdere rijen, van boven naar beneden

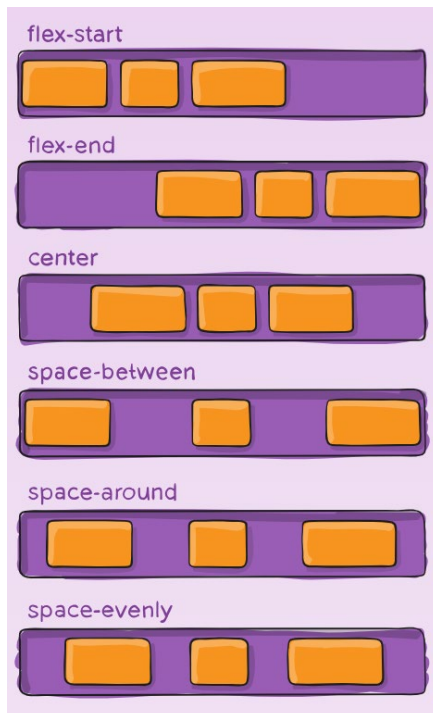
wrap-reverse: flex elementen lopen door over meerdere rijen, van beneden naar boven

7.9.4 Flex container : flex-flow

Voor de flex container kunnen de eigenschappen flex-direction en flex-wrap samengevoegd worden in flex-flow. Standaard is row nowrap.

```
flex-flow: 'flex-direction' || 'flex-wrap'
```

7.9.5 Flex container : justify-content



met gelijke ruimte tussen

space-around: elementen worden gelijk verdeeld over de rij met gelijke ruimte om hen heen

De uitlijning over de hoofd-as (main-axis) wordt aangegeven via justify-content. Het bepaalt de verdeling van de vrije ruimte die overblijft als alle elementen een vaste grootte hebben, of als deze hun maximumgrootte hebben bereikt. Het geeft ook enige controle over de uitlijning als elementen overlopen naar de volgende rij.

```
.container {
  justify-content: flex-start | flex-end | center
  | space-between | space-around;
}
```

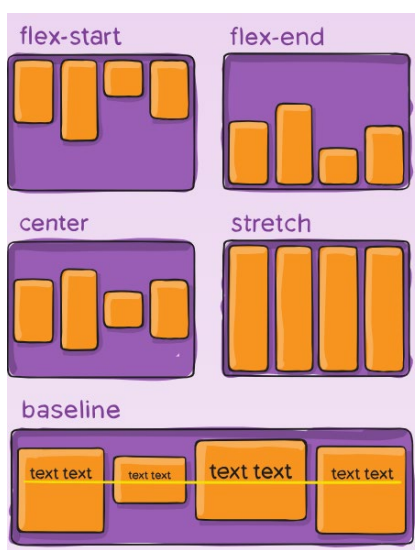
flex-start (standaard): elementen worden uitgelijnd richting de start

flex-end: elementen worden uitgelijnd richting het einde

center: elementen worden gecentreerd

space-between: elementen worden gelijk verdeeld over de rij

7.9.6 Flex container : align-items



De uitlijning in de richting die haaks staat op de rij, wordt aangegeven via align-items.

```
.container {
  align-items: flex-start | flex-end | center |
  baseline | stretch;
}
```

flex-start: plakken aan de haakse beginlijn

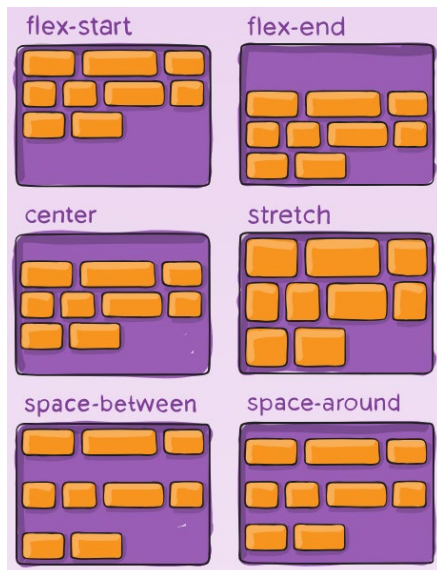
flex-end: plakken aan de haakse eindlijn

center: gecentreerd in de haakse richting

baseline: elementen worden uitgelijnd zodanig dat hun basislijn gelijk is

stretch (standaard): elementen uitrekken om de container te vullen in de haakse richting (minimums en maximums worden wel aangehouden)

7.9.7 Flex container : align-content



Als de inhoud van een container uit meerdere rijen bestaat, dan kan met align-content worden aangegeven hoe deze moeten worden uitgelijnd.

```
.container {
  align-content: flex-start | flex-end | center |
  space-between | space-around | stretch;
}
```

flex-start: rijen uitgelijnd richting start van de container

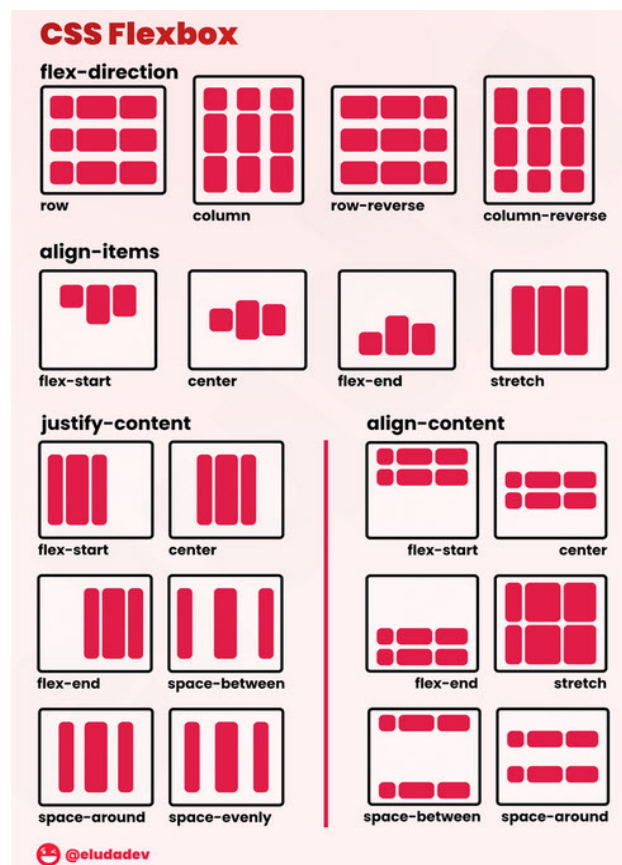
flex-end: rijen uitgelijnd richting einde van de container

center: rijen gecentreerd in de container

space-between: rijen gelijk verdeeld over de hele container

space-around: rijen gelijk verdeeld met evenveel ruimte om hen heen

stretch (standaard): rijen uitgerekt om de ruimte te vullen



7.9.8 Flex element: order

Met order (aangegeven met een geheel getal) kan een element een andere plaats in de container krijgen dan deze zou krijgen op basis van de volgorde in de broncode.

```
.item {
  order: integer;
}
```

De eerste stap is het plaatsen van de elementen naast elkaar. Zonder Flexbox zouden we waarschijnlijk alle drie de elementen laten zweven (Float), maar het zou niet zo eenvoudig zijn om het naar wens te laten werken. Bovendien zou de traditionele manier van werken een bekend probleem met zich meebrengen: elke column is net zo hoog als de inhoud ervan. Als gevolg hiervan zou je een gelijke hoogte moeten instellen voor alle drie de kolommen om dezelfde lengte te hebben, of een soort hack moeten gebruiken.

Alle mogelijkheden bespreken in deze cursus is onmogelijk. Via de volgende site: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> kan je je verdiepen in deze techniek.

Andere bronnen:

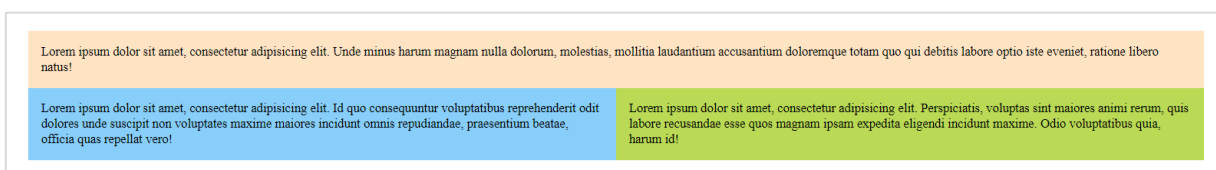


- https://www.w3schools.com/css/css3_flexbox.asp
- <https://flexboxfroggy.com/#nl> (leuk spel)

7.10 Samenvatting

Laten we even drie technieken toepassen op dezelfde html-code.

Volgende lay-out wensen we te bekommen:



Dit is onze HTML-pagina

```
<body>
<div class="container">
  <main>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde minus harum magnam nulla dolorum, molestias, mollitia laudantium accusantium doloremque totam quo qui debitis labore optio iste eveniet, ratione libero natus!
</main>
  <section>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Perspiciatis, voluptas sint maiores animi rerum, quis labore recusandae esse quos magnam ipsam expedita eligendi incidunt maxime. Odio voluptatibus quia, harum id!
</section>
</div>
</body>
```

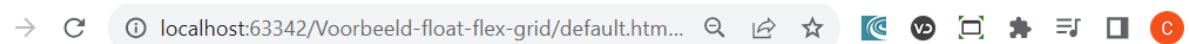
```

    </section>
    <section>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
    Perspiciatis, voluptas sint maiores animi rerum, quis labore recusandae esse quos
    magnam ipsam expedita eligendi incidunt maxime. Odio voluptatibus quia, harum id!
    </section>

</div>

```

Dit is de natuurlijke flow van de pagina (de twee section-elementen krijgen een achtergrondkleur om alles duidelijk te maken).



Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde minus harum magnam nulla dolorum, molestias, mollitia laudantium accusantium doloremque totam quo qui debitis labore optio iste eveniet, ratione libero natus!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Id quo consequuntur voluptatibus reprehenderit odit dolores unde suscipit non voluptates maxime maiores incidunt omnis repudiandae, praesentium beatae, officia quas repellat vero!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Perspiciatis, voluptas sint maiores animi rerum, quis labore recusandae esse quos magnam ipsam expedita eligendi incidunt maxime. Odio voluptatibus quia, harum id!

7.10.1 Eerst benadering via Float

```

*{
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
.container{
  max-width: 90%;
  margin: 1rem auto;
}
main{
  background-color: bisque;
  padding: 5px;
}
section{
  width: 50%;
  padding: 5px;
}
section:nth-child(2){
  float: left;
  background-color: lightskyblue;
}
section:last-of-type{
  float: right;
  background-color: #bada55;
}

```

7.10.2 Tweede benadering: CSS-Grid

```

*{
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
.container{
  max-width: 90%;
  margin: 1rem auto;
}
.gridy {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-areas:
    "a a"
    "b c";
}

main{
  grid-area: a;
  background-color: bisque;
  padding: 1rem;
}
section{
  padding: 5px;
}
section:nth-child(2){
  grid-area: b;
  background-color: lightskyblue;
}
section:last-of-type{
  grid-area: c;
  background-color: #bada55;
}

```

7.10.3 Derde benadering: FLEX

```

*{
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
.container{
  max-width: 90%;
  margin: 1rem auto;
}

```

```
}  
.flexy{  
  display: flex;  
  flex-wrap: wrap; /* two rows ok now */  
}  
main{  
  width: 100%;  
  background-color: bisque;  
  padding: 1rem;  
}  
section{  
  padding: 5px;  
  width: 50%;  
}  
section:nth-child(2){  
  background-color: lightskyblue;  
}  
section:last-child{  
  background-color: #bada55;  
}
```

