**CSS**

Leereenheid 3

# CSS

INTRODUCTIE

In leereenheid 2 hebt u gelezen dat het belangrijk is de structuur en inhoud, de functionaliteit en de presentatie van elkaar te scheiden. Leereenheid 2 gaf een inleiding in het beschrijven van de structuur en inhoud van webpagina's door gebruik te maken van HTML. Deze leereenheid geeft een inleiding in het beschrijven van de presentatie. Daarbij wordt gebruikgemaakt van CSS.

Om CSS echt zo te gebruiken dat u een pagina er uit kunt laten zien zoals u wilt, is erg veel kennis en erg veel ervaring nodig. In deze leereenheid krijgt u alleen een basis aangereikt. U leert de syntaxis van CSS, u leert hoe CSS-declaraties er uitzien en hoe u HTML-elementen kunt selecteren waarop die declaraties van toepasing zijn, en u leert hoe de browser op basis van de CSS-regels bepaalt hoe elementen op de pagina getoond worden.

We laten u zien hoe u te werk kunt gaan bij het opstellen van CSS bij een HTML-bestand, en bespreken daarbij steeds de belangrijke concepten. We gebruiken de beide pagina's uit de vorige leereenheid als voorbeeld.

Daarbij volgen we een methode waarbij u eerst een ontwerp op papier maakt en daarbij (variabele) maten vastlegt, vervolgens de hoofdstructuur implementeert, en daarna de details uitwerkt. Ten slotte kunt u de code nog 'refactoren': de code herstructureren zodat de onderhoudbaarheid beter wordt.

We besteden ook aandacht aan de ontwikkelingen binnen CSS, zodat u een blik op de toekomst krijgt.

Na het bestuderen van deze leereenheid hebt u geen overzicht van alle mogelijke eigenschappen die u met CSS kunt beïnvloeden; wel hebt u een idee van de werking van CSS, en kent u de belangrijkste concepten.

LEERDOELEN

Na het bestuderen van deze leereenheid, wordt verwacht dat u:

- de basissyntaxis van een CSS-regel kunt geven
- kunt uitleggen wat de voordelen zijn om CSS-regels in een afzonderlijk bestand te plaatsen
- kunt uitleggen hoe wordt bepaald welke declaratie wordt toegepast als op een element verschillende declaraties voor dezelfde eigenschap van toepassing zijn
- kunt uitleggen waarom het de voorkeur heeft semantische namen voor id's en classes te gebruiken
- kunt uitleggen hoe het CSS box model in elkaar zit en de breedte en hoogte van een element kunt bepalen

– kunt uitleggen welke consequenties het heeft wanneer een element de eigenschap `float` krijgt
– de elementen `div` en `span` kunt gebruiken
– gebruik kunt maken van een CSS-validator en van CSSLint
– de betekenis kent van de volgende concepten: default styling, CSS-regel, CSS-declaratie, selector, CSS-eigenschap, default styling, block, inline, responsive design, pixel, em, collapsing margins, pseudo-selector, cascade, specificiteit, inheritance.

*Studeeraanwijzingen*
Het is aan te bevelen de stappen uit het werkboek zelf te implementeren (de uitgangscode is in de bouwsteen te vinden), en steeds het resultaat in de browser te bekijken.

Het is uitdrukkelijk niet de bedoeling dat u de details van CSS-eigenschappen uit uw hoofd leert. De leerdoelen beschrijven precies wat u *wel* moet weten.

Er is geen zelftoets bij deze leereenheid omdat de leereenheid al zoveel opgaven bevat; het is daarom extra aan te bevelen om alle opdrachten uit te werken.

LEERKERN

## 1 HTML zonder CSS

### 1.1 DEFAULT STYLING

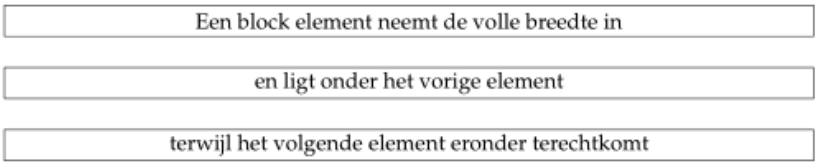
Wanneer u de pagina die in leereenheid 2 is opgesteld in de browser bekijkt (zie de pagina in de map `pagina/uitgangspunt` in de bouwsteen), valt op dat de browser er een zeer basale styling aan geeft. Zo zijn de `hx`-elementen vet gedrukt, evenals de `th`-elementen uit de tabel. De `a`-elementen zijn blauw en onderstreept. De `li`-elementen in `ul`-elementen zijn voorzien van bolletjes. De tabellen zien er herkenbaar uit als tabel. Ook is te zien dat er een marge is tussen de inhoud van de pagina en de randen van de window.

*Default styling*

Browsers hebben een *default styling* die op alle pagina's wordt toegepast. Die styling heeft de vorm van CSS-regels die de browser ook toepast wanneer de webontwikkelaar zelf geen CSS heeft toegevoegd.

### 1.2 BLOCK, INLINE, TABLE

Wat misschien minder snel opvalt, is dat de browser verschillende manieren heeft om elementen op de pagina te plaatsen. Wat volgt op een `h1`- of `h2`-element bijvoorbeeld, staat onder dat element. U ziet dat ook bij `p`-elementen. Maar wat volgt op een `a`-element staat gewoon *naast* dat element (zie bijvoorbeeld de link met de tekst 'specificatie' in het `asi de`-element).




FIGUUR 3.1 Elementen met display: block

*display:block*
*block-elementen*

De browser geeft aan sommige elementen de CSS-eigenschap *display:block* mee (we noemen die elementen *block-elementen*). Dat geldt bijvoorbeeld voor de elementen die content groeperen. Zo'n element neemt, zoals figuur 3.1 laat zien, in principe de gehele beschikbare breedte in (dat kan worden veranderd met CSS). Een element met `display:block` wordt onder het element ervoor geplaatst en elementen erna komen eronder terecht (ook wanneer het element met CSS een smallere breedte heeft gekregen).

FIGUUR 3.2 Elementen met display:inline

*display:inline*
*inline-elementen*

Andere elementen hebben de eigenschap dat de browser ze de CSS-eigenschap *display:inline* meegeeft (we noemen die elementen *inline-elementen*). Dat geldt bijvoorbeeld voor de elementen voor tekst. Deze elementen gedragen zich als tekst in een editor waar word wrap aan staat: ze worden naast het vorige element met `display:inline` geplaatst zolang er ruimte is, en schuiven anders naar onderen, zoals te zien is in figuur 3.2: de onderste blokjes passen niet meer naast de bovenste, en zijn een regel naar onderen geschoven. In de breedte nemen ze dus uitsluitend de ruimte in die ze nodig hebben, zoals woorden in een zin.

*display:table*

De browser gebruikt meer vormen, waarvan een belangrijke *display:table* is, voor de opmaak van tabellen. De browser berekent per kolom de breedte die de cel met de breedste inhoud nodig heeft.

### 1.3 RESET CSS

*Reset*

Lastig voor webontwikkelaars is dat verschillende browsers verschillende default-styling gebruiken. Het is daarom een goede gewoonte om altijd zogenaamde CSS op te nemen waarin dat soort verschillen worden opgeheven, zodat er als het ware een schone lei beschikbaar is voor CSS. Er zijn verschillende van dat soort bestanden in omloop. Bij de bouwsteen vindt u een voorbeeld. In de voorbeelden en opdrachten zullen we per geval de reset-regels meegeven die van belang zijn.

## 2 Planning op papier

De handigste aanpak om een webpagina vorm te geven is om in vier stappen te werken, uitgaande van een pagina in HTML:

– maak een planning op papier voor de structuur van de pagina
– werk die structuur uit in CSS
– vul de details in
– refactor: herstructureer de code (CSS en eventueel HTML) indien nodig.

### 2.1 RESPONSIVE DESIGN

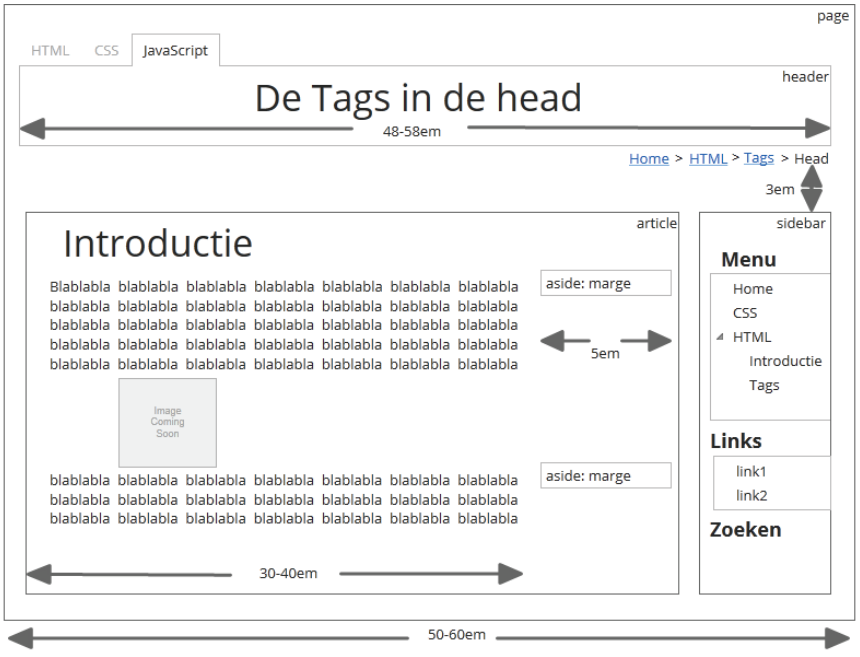
Bij de planning op papier legt u de afmetingen van de diverse onderdelen vast. Een probleem daarbij is dat een webpagina te bekijken moet zijn op schermen van heel verschillende afmetingen en resoluties: uitgaan van één vaste schermresolutie leidt tot problemen op schermen met een veel kleinere of veel grotere resolutie.

*Responsive design*

Een ontwerp waarbij u ervoor zorgt dat de pagina zich aanpast aan het scherm en de resolutie van het apparaat waarop de pagina wordt bekeken, heet *Responsive design*. Het komt erop neer dat u waar mogelijk elementen 'elastisch' maakt, met een ondergrens en een bovengrens, en dat u iets aparts regelt voor schermen die onder de ondergrens of boven de bovengrens komen.

Weblink: Tool voor schetsen

In het plan dat wij op papier hebben gezet, in figuur 3.3, kunt u daar iets van zien. U vindt de afbeelding in het groot in de bouwsteen (we hebben hier een tool voor gebruikt; u vindt die tool bij de weblink).




FIGUUR 3.3 De layout op papier

### 2.1.1 *Eenheden*

*Percentage*
*Pixel*

Responsive design stelt eisen aan de eenheden waarmee u afmetingen vastlegt. *Percentage*s zijn erg handig: die eenheid is altijd relatief. Maar vaak is het nodig om absolute maten te gebruiken. *Pixel*s (de 'korrelgrootte' van een scherm) zijn lastig, omdat er enorme verschillen zijn in *dots per inch* (dpi), oftewel het aantal pixels per inch. Pixels zijn ook lastig omdat daarme instellingen in de browser voor bijvoorbeeld een groter font dan standaard niet worden gehonoreerd.

*em*

Het is daarom bijzonder aan te bevelen om absolute afmetingen in *em* te geven. De maat `em` staat voor de *hoogte* van de letter `m` in het *huidige font*. De breedte van een `em` is ongeveer anderhalf karakter, gemiddeld.

Voor de pagina gaan we uit van de breedte van de tekst van het artikel. Een tekst leest het plezierigst wanneer hij `30` tot `40em` breed is (45-60 karakters). De tekst komt `1em` van de rand af. Voor de tekst in de marge houden we `5em` aan; ook hier is aan weerszijden open ruimte van `1em`. Totaal hebben we dan `38-48em`. Voor de sidebar met het menu houden we in totaal `10em` aan (voor de tekst `8em`, met aan weerszijden ruimte).

Het geheel willen we omkaderd midden op het scherm hebben (als dat groter is dan de omkadering). Die omkadering is dan `50-60em` breed. In de CSS zullen we ervoor gaan zorgen dat het nog minder breed kan als dat nodig is: dat het menu naar onder het artikel kan verschuiven als er te weinig ruimte is.

Op deze manier hebben we de maten van de structuur van de pagina vastgelegd. Het is dan gemakkelijker om ze in CSS over te zetten dan wanneer u direct in CSS zou beginnen.

## 2.2 GROEPEREN ZONDER BETEKENIS

In het plan dat we op papier hebben opgesteld ziet u dat de header, de footer, het article en de sidebar omhuld worden door een element dat we 'page' hebben genoemd. In de HTML uit de vorige leereenheid hebben we zo'n element niet opgenomen. Er is geen element-met-betekenis te vinden dat zou kunnen dienen voor dat omhullende element. Hier kunnen we dus prima het `div`-element gebruiken. Dat `div`-element geven we als class `page` mee: `<div class="page">`. In de bouwsteen, onder `uitgangspunt`, vindt u het HTML-bestand met dit extra element.

Wanneer u een pagina met behulp van CSS gaat opmaken, zal het vaker voorkomen dat u extra elementen nodig hebt. Meestal gaat het dan om `div`- en `span`-elementen.

OPGAVE 3.1
In leereenheid 2 hebt u een formulier ontwikkeld voor het aanvragen van een offerte voor een autoverzekering.
In de bouwsteen (in de map `formulier`) vindt u dit formulier (`offerte.html`), waarbij het formulier robuuster is gemaakt (u leert daar meer over in leereenheid 12).
a Bekijk de HTML van deze pagina en open de pagina vervolgens in een browser. Wanneer u de pagina in een browser opent die alle typen voor het `input`-element ondersteunt, zijn de HTML5-controls zichtbaar.
b Voeg een `div`-element toe met een `class`-attribuut met waarde 'page'.
c Maak een schets van de structuur van deze pagina zoals deze uiteindelijk op het scherm moet worden getoond. Geef het formulier een breedte van `30em`.

## 3 De hoofdstructuur

De volgende stap is het uitwerken van de hoofdstructuur in CSS. Daarbij kan het voorkomen (net als bij de uitwerking van de details) dat de HTML moet worden uitgebreid met elementen of classes.

### 3.1 HET CSS-BESTAND

In de vorige leereenheid hebt u al gezien dat de CSS in een apart bestand wordt gehouden, waarnaar het HTML-bestand een link legt via een `link`-element. De reden daarvan is dat de HTML, verantwoordelijk voor structuur en inhoud, en de CSS, verantwoordelijk voor opmaak, dan fysiek gescheiden worden gehouden: het principe van modulariteit. Dat is een uitbreiding van het principe van het scheiden van verantwoordelijkheden dat u in leereenheid 1 hebt gezien.

**Software design principe: Principe van modulariteit**
Het *principe van modulariteit* houdt in dat u de oplossing van een probleem verdeelt in componenten met elk een eigen verantwoordelijkheid. Componenten met een bepaalde verantwoordelijkheid houdt u fysiek gescheiden van componenten met andere verantwoordelijkheden.

```
<link rel="stylesheet" href="stijl.css">
```

*CSS-regel*

Het CSS-bestand is in principe een lijst van *CSS-regel*s. Een voorbeeld van een CSS-regel is:

```
body {
  color:#333;
}
```

*Selector*
*CSS-declaratie*

Een CSS-regel bestaat uit een *selector* (in dit voorbeeld `body`), gevolgd door één of meer *CSS-declaratie*s die tussen accolades staan.

*CSS-eigenschap*

Zo'n declaratie bestaat uit een *CSS-eigenschap* (in dit geval is dat `color`), gevolgd door een dubbele punt, gevolgd door een *waarde*, gevolgd door een puntkomma. Na een dubbele punt of puntkomma mag een spatie staan, als dat de leesbaarheid ten goede komt.

Weblink: Kleuren

De weblink geeft informatie over mogelijke waarden voor `color`; we gaan daar hier niet verder op in.

Commentaar in een CSS-bestand is mogelijk door het commentaar te omsluiten met `/*` en `*/`. Het is bijzonder aan te bevelen om het CSS-bestand direct al te ordenen in een aantal delen, met behulp van commentaar, zoals in het volgende voorbeeld:

```
/*** Reset ***/
/*** Generiek ***/
/*** Hoofdstructuur ***/
/*** Navigatie ***/
  /*** Navigatie: Tabs ***/
  /*** Navigatie: Breadcrumbs ***/
  /*** Navigatie: Menu ***/
```

In die indeling komt steeds wat generiek is vóór wat specifiek is. Tijdens het opstellen van de CSS zullen er meer groepen bij komen.

**Programmeeraanwijzing: CSS in groepen**
Deel een CSS-bestand in groepen in, waarbij steeds het generieke deel vóór een specifiek deel komt.

OPGAVE 3.2
a Maak een leeg bestand `offerte.css` aan en link de stylesheet aan het HTML-bestand `offerte.html`.
b Link tevens de file `reset.css` aan het HTML-bestand `offerte.html`.
Ziet u verschil in de browser?

## 3.2 SELECTOREN

TABEL 3.1 Selectoren

| *patroon* | *betekenis* | *voorbeeld* |
|---|---|---|
| `t` | elementen met tag `t` | `li` |
| `#myid` | element met id `myid` | `#uniek` |
| `.myclass` | elementen met klasse `myclass` | `.tab` |
| `t.myclass` | elementen met tag `t` en klasse `myclass` | `li.active` |
| `.class1.class2` | elementen met `class1` *en* `class2` | `.tab.active` |
| `[attr1]` | elementen met `attr1` | [selected] |
| `[attr1="value1"]` | elementen met `attr1` met waarde `value1` | `[name="addr"]` |
| `t[attr1]` | elementen met tag `t` en `attr1` | `a[title]` |

*Tag-selector*

Met een selector geeft u aan op welk element of op welke elementen de declaraties die erbij horen betrekking hebben. De simpelste selector is de *tag-selector* (patroon `t` in tabel 3.1). Die bestaat uit een tag-naam, en selecteert alle elementen met die tag-naam. Het element met een bepaalde id geeft u aan met de *id-selector*, die bestaat uit het teken `#`, gevolgd door het id.

*Id-selector*

*Class-selector*

Elementen met een bepaalde klasse geeft u aan met de *class-selector*: een punt, gevolgd door de naam van de klasse. U kunt daarbij ook nog selecteren op de tagnaam: wanneer er bijvoorbeeld zowel `input-` als

`li`-elementen zijn met class `active`, kunt u die laatste selecteren met `li.active`. Het is mogelijk om elementen die zowel klasse `tab` als klasse `active` hebben, te selecteren met `tab.active`.

*Attribuut-selector*

Ten slotte is het ook mogelijk om elementen te selecteren waarvan een bepaald attribuut gezet is, of waarvan een attribuut een bepaalde waarde heeft, met de *attribuut-selector*, zoals in `[selected]` of `input[type="search"]`.

TABEL 3.2 Combinatoren

| *combinator* | *betekenis* |
|---|---|
| Sel1, Sel2 | declaraties gaan op voor beide selectoren |
| Sel1 Sel2 | Sel2-elementen die als (voor)ouder een Sel1-element hebben |
| Sel1 > Sel2 | Sel2-elementen die als directe ouder een Sel1-element hebben |

Selectoren kunnen gecombineerd worden, zoals tabel 3.2 laat zien. Wanneer declaraties opgaan voor meerdere selectoren, kunt u die selectoren door een komma scheiden (bijvoorbeeld als `h1, h2, h3`). Een `img`-element dat als één van de voorouders een element heeft met klasse `introduction`, geeft u aan met `introduction img`. Een `h2`-element dat als directe ouder een element heeft met klasse `introduction`, geeft u aan met `introduction > h2`. Later in deze leereenheid zult u zien dat u die twee laatste combinatoren moet vermijden, maar het is handig als u het bestaan ervan kent.

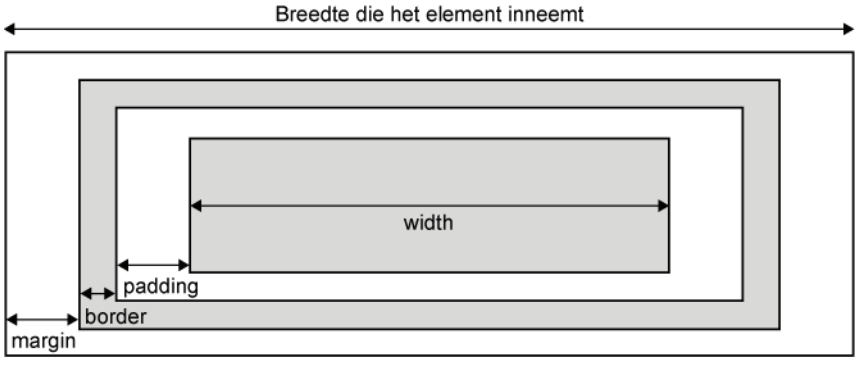
Weblink: Selectoren

Verderop in deze leereenheid komt u nog meer selectoren tegen. Een compleet overzicht vindt u via de weblink.

*Syntaxis*

Naast deze kennis over de *syntaxis* van CSS moet u weten hoe u elementen een bepaalde breedte kunt meegeven. Daartoe bespreken we eerst het box model.

## 3.3 HET BOX MODEL




FIGUUR 3.4 Het box-model

*width*

*Box-model*

Elementen met `display:block` kunnen een waarde hebben voor de CSS-eigenschappen `width`, `padding`, `border` en `margin`. De eigenschap *width* bepaalt de breedte die de *inhoud* van het element tot z'n beschikking heeft. De manier waarop die waarden samenhangen, heet het *box-model*.

Let op!

Dit is niet van toepassing op elementen met `display:inline`. We zullen in paragraaf 3.5 zien hoe we van een inline-element een block-element kunnen maken, waardoor we wel eigenschappen als `width` en `height` zinvol kunnen definiëren.

*padding*

De *padding* is de ruimte tussen de eventuele rand (border) om het element heen en de ruimte voor de inhoud, en kan verdeeld worden in `padding-top`, `padding-right`, `padding-bottom` en `padding-left`, die allemaal een verschillende waarde mogen krijgen. Wanneer ze allemaal dezelfde waarde hebben, kunt u die waarde simpelweg aan `padding` geven. Wanneer de top en de bottom dezelfde waarde hebben (bijvoorbeeld `0`) en left en right ook (bijvoorbeeld `3em`) kunt u dat opgeven met:

```
padding: 0 3em;
```

Wanneer ze allemaal een andere waarde hebben, kunt u dat ook in één keer aangeven. U begint dan bij de top, en draait met de klok mee via right naar bottom naar left:

```
padding: 0 1em 2em 3em;
```

*border*

*margin*

Datzelfde geldt ook voor de breedte van de *border*, die bepaald wordt door `border-top-width` enzovoort, en in één keer gezet kan worden met `border-width`. Ook voor de *margin* (de afstand tussen de rand en het omhullende block-element) kunnen afzonderlijke waarden worden gegeven, met `margin-top` enzovoort.

Zoals figuur 3.4 laat zien, wordt de breedte die een element inneemt berekend door de volgende formule:

```
ingenomen breedte =
  width +
  padding-left + padding-right +
  border-left + border-right +
  margin-left + margin-right
```

*auto*

De belangrijkste eenheden die u kunt gebruiken voor deze waarden zijn, zoals gezegd, `em` en percentage. Voor de `border-width` kunt u ook de woorden `thin`, `medium` en `thick` gebruiken. Voor de `margin` kunt u ook de waarde *auto* gebruiken. Wanneer `margin-left` en `margin-right` die waarde hebben, en het element heeft een bepaalde breedte, wordt het element netjes in het midden van het omhullende element geplaatst, terwijl de waarde van de margin `0` bedraagt: de waarde `auto` draagt niet bij aan de breedte die het element inneemt.

*Collapsing margins*

Margins hebben een eigenschap die misschien niet direct voor de hand ligt. Wanneer twee block-elementen boven elkaar liggen, en de bovenste heeft een `margin-bottom` van `3em`, terwijl de onderste een `margin-top` heeft van `2em`, ligt het in eerste instantie voor de hand dat de tussenruimte `5em` wordt. Dat is niet het geval: de tussenruimte wordt `3em`. De margins schuiven als het ware in elkaar. De term daarvoor is *collapsing margins*. Hetzelfde geldt voor `margin-left` en `margin-right` bij block-elementen die naast elkaar liggen.

Nu kunnen we beginnen de hoofdstructuur naar CSS te vertalen. U begint dan met de maat voor het `div`-element met class `page`, dat u in het midden van de pagina zet. De maximum-maat was `60em`; als minimum nemen we de minimum-breedte van het artikel plus de ruimte van `1em` aan weerszijden: dan hebben we een minimum-breedte waarbij de sidebar naar onderen zou kunnen verschuiven. De breedte die het element met class `page` inneemt is dus minimaal `40em` en maximaal `60em`. Dat houdt in:

```
40em tot 60em =
  width +
  padding-left + padding-right +
  border-left + border-right +
  margin-left + margin-right
```

Het element willen we in het midden plaatsen. De `margin` krijgt dus als waarde `0 auto`. In bovenstaande berekening krijgen `margin-left` en `margin-right` dus als waarde `0`. We geven het element geen `border`, dus ook die waarden zijn `0`. We geven het element wel een `padding`, zodat overal een lege ruimte van `1em` ontstaat. De formule wordt dus:

```
40em tot 60em = width + 1em + 1em
width = 38em tot 58em
```

Om het element te kunnen onderscheiden van het gedeelte van de `body` dat er buiten valt, geven we ze een verschillende achtergrond mee. We krijgen nu:

```
/*** Generiek ***/
body {
  background: linear-gradient(to bottom, Ivory, #A20013);
}
/*** Hoofdstructuur ***/
.page {
  width: 100%;
  min-width: 38em;
  max-width: 58em;
  margin: 1em auto;
  background-color: Ivory;
  padding: 0 1em;
  box-shadow: 3px 3px 5px #888;
}
```

U ziet dat er voor het `page`-element als selector `.page` is gebruikt en niet `div.page`. Dat heeft een reden. In het tweede geval zal de browser eerst een lijst opstellen van alle `div`-elementen, en vervolgens de lijst langslopen op zoek naar het element met class `page`. In het eerste geval zoekt de browser alleen naar elementen met class `page`. De eerste notatie is dus veel efficiënter en sneller.

**Programmeeraanwijzing: Korte selectoren**
Houd selectoren zo kort mogelijk. Zet in het geval van een klasse of id niet de tag ervoor, tenzij dat echt noodzakelijk is.

Weblink CSS eigenschappen

Er is een enorm aantal CSS-eigenschappen, en er komen regelmatig eigenschappen bij. We bespreken daarom lang niet alle eigenschappen. De weblink geeft een referentie van de door Firefox ondersteunde eigenschappen, en een pagina met uitleg en voorbeelden.

OPGAVE 3.3

Geef de body van de pagina voor het aanvragen van een offerte een effen achtergrondkleur. Definieer tevens de opmaak voor het `div`-element met class `page`.

De `header` (met de `ul`-elementen voor de tabs en de broodkruimels) laten we nog even voor wat hij is, maar de `h1` uit de header kunnen we al wel vormgeven. De paginatitel onderscheiden we van eventuele andere `h1`-elementen door middel van een class: `pagetitle`. De eigenschappen die van toepassing zijn op alle `h1` (en `h2` en `h3`-elementen) zetten we daarbij onder de groep Generiek, terwijl we alleen de eigenschappen die specifiek zijn voor deze ene `h1` onder Hoofdstructuur zetten.

```
/*** Generiek ***/
h1, h2, h3 {
  font-variant: small-caps;
  color: #E2001B;
  margin: 0.5em 0;
  box-shadow: 3px 3px 5px #888;
}
/*** Hoofdstructuur ***/
.pagetitle {
  padding: 0.25em;
  text-align: center;
  font-size: 4em;
  border: thin solid #EF001D;
  border-radius: 0.25em;
}
```

In het algemeen geldt: probeer eigenschappen zo generiek mogelijk te houden.

**Programmeeraanwijzing: Meest generieke selector**
Breng declaraties onder bij zo een zo generiek mogelijke selector.

OPGAVE 3.4

Geef het element `h1` in `offerte.html` een attribuut class met waarde `pagetitle`.
Definieer vervolgens in de CSS-file de opmaak voor dit element.

### 3.4 DE CASCADE, SPECIFICITEIT EN INHERITANCE

Op het `h1`-element zijn nu verschillende CSS-regels van toepassing: de regels met `h1, h2, h3` als selector, en de regel met `.pagetitle` als selector. Er zijn in dit geval geen conflicten, maar het is handig om te weten wat er gebeurt als dat wel het geval zou zijn.

*Cascade*

Het woord *cascade* (CSS staat voor Cascading StyleSheet) heeft te maken met de verschillende bronnen waarin CSS-regels kunnen voorkomen, en de manier waarop die worden toegepast. Wanneer meerdere bronnen een declaratie voor een bepaald element hebben, is de volgorde van belangrijkheid (de belangrijkste staat bovenaan):

1 de stijl van de gebruiker met het kenmerk `!important` (`!important` behandelen we niet)
2 de stijl van de auteur met het kenmerk `!important`
3 de stijl van de auteur
4 de stijl van de gebruiker
5 de stijl van de browser zelf.

De stijl van de gebruiker is de stijl die kan worden ingesteld in de browser. De meeste gebruikers maken geen gebruik van die mogelijkheid, maar slechtzienden kunnen hiermee bijvoorbeeld de standaard font-grootte beïnvloeden.

Hier is te zien dat een reset-stijl werkt doordat 3 belangrijker is dan 5.

*Specificiteit*

Wanneer er in de stijl van de auteur meerdere declaraties staan met regels voor hetzelfde element die conflicteren, wordt er volgens een algoritme bepaald welke declaratie gebruikt zal worden: *specificiteit*.

De specificiteit is een getal dat als volgt wordt berekend:

– het aantal *tagnamen* in de selector maal `1`, plus:
– het aantal *attributen* en *klassen* in de selector maal `10`, plus:
– het aantal *id's* in de selector maal `100`.

Hoe groter het getal, hoe specifieker de selector, en hoe meer prioriteit hij heeft. Dit betekent dus bijvoorbeeld dat `.tab.active` specifieker (en dus belangrijker) is dan `.active`.

Weblink: Cascade

Als er dan nog steeds declaraties zijn die even belangrijk zijn, telt de volgorde: de laatste overschrijft dan de voorgaande declaratie(s). De weblink geeft een uitgebreide specificatie van specificiteit en de cascade.

Wanneer er voor een element geen declaraties zijn voor een bepaalde eigenschap, hangt het van twee zaken af of de browser toch een waarde rekent voor die eigenschap: inheritance en een eventuele default-waarde.

*inherited*

Sommige eigenschappen (`color` en `font-size` zijn twee voorbeelden) hebben een speciale eigenschap: ze zijn *inherited*. Dat houdt in dat wanneer die eigenschap een bepaalde waarde heeft gekregen in een element, die waarde ook wordt toegepast op alle elementen die (klein)kinderen zijn van dat element. Wanneer de tekstkleur in de `body` dus een bepaalde waarde krijgt, krijgt de tekstkleur in alle elementen van de pagina die waarde:

```
/*** Generiek ***/
body {
  color: #333;
}
```

*Default-waarde*

Ten slotte hebben veel elementen een *default-waarde* die van toepassing is als er voor die eigenschap geen regels zijn voor een element. De default-waarde voor `background-color` bijvoorbeeld is `transparent`.

Weblink: CSS-specificatie

Of eigenschappen `inherited` zijn, en of ze een bepaalde default-waarde hebben, kunt u vinden in de officiële specificatie. De weblink geeft links naar de verschillende specificaties van CSS (zowel de huidige standaard als nieuwe ontwikkelingen).

U kunt met behulp van Firebug bekijken welke declaraties er op een bepaald element van toepassing zijn, en wat de volgorde is waarin ze worden toegepast.

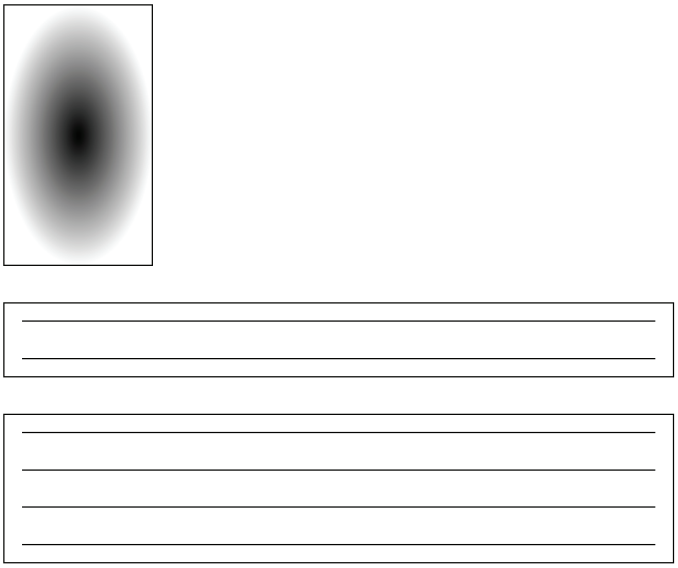
OPGAVE 3.5

Installeer Firebug (zie de handleiding op Studienet), en laadt de HTML-pagina uit de bouwsteen in de map `pagina/hoofdstructuur`. Start Firebug met F12, en zorg dat de tab HTML is geselecteerd. Klik nu op het pijltje om een element te selecteren (zie handleiding op studienet) en klik op de titel van de pagina. U ziet in de HTML dat het `h1`-element is geselecteerd. Rechts ziet u de declaraties die van toepassing zijn op dit element. Wat valt u op aan de volgorde waarin die declaraties worden weergegeven?

## 3.5 FLOATS EN POSITIONING

Van de hoofdstructuur moeten nu nog het `article`-element, de sidebar en de footer op hun plek worden gezet (de tabs en de breadcrumbs vallen onder de details). Het `article`-element en het `aside`-element dat als sidebar fungeert, moeten naast elkaar komen. Dat kan door middel van *floats*.

Stel dat een block-element dat we een bepaalde breedte hebben gegeven (bijvoorbeeld een `figure`-element) boven twee block-elementen staat zonder opgegeven breedte, die daardoor de gehele beschikbare breedte innemen (bijvoorbeeld twee `p`-elementen), zoals weergegeven in figuur 3.5.



FIGUUR 3.5 Drie block-elementen

*float*

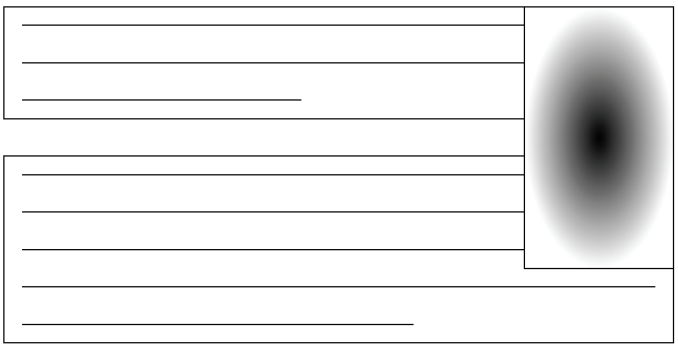
In zo'n situatie is het mogelijk om het bovenste element een waarde te geven voor de eigenschap `float`. De default waarde van die eigenschap is `none`. De mogelijke andere waarden zijn `left` en `right`. In dit geval geven we de waarde `right`.

Het effect, zoals getoond in figuur 3.6, is:

– Het floatende element schuift naar rechts op, tot de grens van het omhullende element (bij een waarde `left` blijft het floatende element uiteraard links staan).
– Voor de andere elementen op de pagina is het alsof het floatende element geen ruimte inneemt. De `p`-elementen komen er daardoor onder te liggen.
– De *content* van de `p`-elementen schuift netjes op, zodat die om het floatende element heen komt te liggen.

*Float layout*

*Float layout* is een vierde variant naast block layout, inline layout en table layout.



FIGUUR 3.6 Een floatend element

*clear*

Bij elk van de `p`-elementen kan de eigenschap *clear* worden gezet (default-waarde: `none`). De regel `clear: left` zorgt ervoor dat voorgaande links-floatende elementen weer zichtbaar worden voor het element; `clear: right` doet hetzelfde voor rechts-floatende elementen, en `clear: both` doet het voor alle floatende elementen. Figuur 3.7 laat zien wat er gebeurt als het tweede `p`-element de waarde `left` voor de eigenschap `clear` meekrijgt.

FIGUUR 3.7 De onderste p heeft clear: left

Wanneer een aantal block-elementen met ingestelde `width` achter elkaar de waarde `left` voor `float` hebben, gedragen ze zich ten opzichte van elkaar als inline-elementen, en worden ze op de pagina gezet als in figuur 3.2. Wanneer de waarde `right` is, wordt het eerste element rechts geplaatst, het volgende element daarnaast, enzovoort.

*Floaten van inline-elementen*

Door een inline-element te floaten, krijgt het de eigenschappen die horen bij een block-element. Hierdoor kunnen we bijvoorbeeld een label een breedte geven.

Er is een aantal ontwikkelingen binnen CSS waarmee het gemakkelijker wordt om de layout van elementen vast te leggen. De ondersteuning daarvoor is nog vrij instabiel, en daarom behandelen we ze hier niet. Achteraan in deze leereenheid gaan we er kort op in.

*position*

De positie van een element op een pagina kan ook worden bepaald door de eigenschap *position* te gebruiken. Mogelijke waarden zijn: `static` (de default-waarde), `relative`, `absolute` en `fixed`. Door bovengenoemnde ontwikkelingen zullen deze positioneringstechnieken minder belangrijk worden, maar vandaag de dag worden ze nog wel veel gebruikt. We gaan hier alleen kort in op absolute en relatieve positionering.

*Absolute positionering*

*Absolute positioning* wordt bereikt door de waarde `absolute` aan de eigenschap `position` van een element te geven. Dat heeft een aantal effecten:

– De positie van het gepositioneerde element is standaard de plek waar het normaal ook zou komen te staan.
– Voor de andere elementen op de pagina is het alsof het gepositioneerde element niet bestaat. Ze schuiven op en nemen een nieuwe positie, volgens de normale regels, in. Het absoluut gepositioneerde element wordt daarbij boven de andere elementen geplaatst.
– De positie kan veranderd worden door een waarde te geven aan `top` en/of `right` en/of `bottom` en/of `left`. Het element wordt daarmee geplaatst *ten opzichte van het eerste omhullende element dat een andere waarde voor* `position` *heeft dan* `static`. Bij `top: 0; left: 0;` komt de linkerbovenhoek van het element dus te liggen op de linkerbovenhoek van het omhullende element met een andere waarde voor `position` dan `static` (of van de body).

*z-index*

Om het element op het scherm achter het volgende element in de HTML-boom te plaatsen, kan een negatieve waarde worden gegeven aan de eigenschap *z-index*. Wanneer twee elementen met absolute positionering elkaar overlappen, kan met de `z-index` worden bepaald welk element boven ligt (dat is het element met de hoogste `z-index`).

*relative*
*Relative positioning*

Een andere mogelijke waarde voor de eigenschap `position` is *relative*, voor *Relative positioning*. Hierbij houden de andere elementen op de pagina rekening met de oorspronkelijke plaats van het element, en wordt dat element zelf met waarden voor `top` en/of `right` en/of `bottom` en/of `left` verplaatst ten opzichte van die oorspronkelijke plaats.

OPGAVE 3.6

In de bouwstenen vindt u de bestanden `positionering.html` en `positionering.css`. Open het HTML-bestand in een browser.
De HTML is:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Positionering</title>
    <link rel="stylesheet" href="positionering.css">
  </head>
  <body>
    <div>
     <span id="id1">xx</span>
     <span id="id2">yy</span>
     <span id="id3">zz</span>
    </div>
  </body>
</html>
```

De CSS ziet er als volgt uit:

```
div {
 width: 10em;
 height: 4em;
 padding: 4em;
 border:  thin solid #EF001D;
 margin: auto;
}
```

```
span {
  padding: 1em;
  border: thin solid #EF001D;
}

#id1 {
 background-color: yellow;
}

#id2 {
 background-color: blue;
}

#id3 {
 background-color: red;
}
```

U ziet dat de drie `span`-elementen naast elkaar in een `div`-element zijn gepositioneerd.
Het `div`-element is in het midden van de pagina geplaatst. De `span`-elementen zijn zichtbaar als gekleurde vierkantjes.
a Geef de positioneringseigenschap van het `span`-element met id `id2` de waarde `absolute`. Geef tevens de eigenschappen `top` en `left` de waarde `1em`. Wat ziet u in de browser? Kunt u dit verklaren?
b Hoe kunt u ervoor zorgen dat het blauwe vierkantje ten opzichte van het `div`-element wordt verplaatst?
N.B. Experimenteer ook met de eigenschap `relative`.

### 3.5.1 *Float voor de hoofdstructuur*

Met de eigenschap `float` is het nu mogelijk om het `article`-element links naast het `aside`-element voor de sidebar te plaatsen, en zelfs op zo'n manier dat wanneer er niet voldoende plaats is, het `aside`-element naar onderen zal verschuiven.

We geven het `article`-element een `padding` van `1em` rondom. De breedte die het element inneemt, mocht liggen tussen `38em` en `48em`. Dat betekent dus dat de `width` `36em` tot `46em` wordt. We geven het element ook een rand om het te kunnen zien liggen. Ten slotte geven we het element aan de bovenkant ruimte:

```
/*** Generiek ***/
article {
  float: left;
  min-width: 36em;
  max-width: 46em;
  padding: 1em;
  margin-top: 3em;
  border: thin solid #EF001D;
  border-radius: 1em;
  box-shadow: 3px 3px 5px #888;
}
```

Deze regels gelden voor elk `article`-element op de pagina. Als de pagina meerdere `article`-elementen zou bevatten, zouden deze declaraties voor alle `article`-elementen geschikt zijn. Er is dus geen id of klasse nodig in dit geval.

Wanneer u de pagina bekijkt (zie de map `floatprobleem` in de bouwsteen), ziet u dat het element met class `page` nu het floatende `article`-element niet meer omhult. Dat klopt, want de overige elementen worden geplaatst alsof `article` niet bestaat; alleen de content van het `aside`-element wordt er netjes naast geplaatst. Een truc om ervoor te zorgen dat het `.page`-element weer netjes doorloopt, is om dat element `overflow: auto` mee te geven.

Ten slotte kan het `aside`-element met class `sidebar` nog netjes worden geplaatst. Het krijgt ook de eigenschap `float: left` mee, en een breedte. Er was `10em` beschikbaar, maar omdat het `article`-element een `border` heeft die weliswaar smal is maar wel ruimte inneemt, houden we `9em` aan. Als we dan een `padding-left` van `1em` meegeven, wordt de `width` `8em`:

```
/*** Hoofdstructuur ***/
.sidebar {
  float: left;
  width: 8em;
  padding: 1em 0 1em 1em;
  margin-top: 3em;
}
```

Om de `footer` weer onderaan te krijgen, geven we die een `clear:left` mee, en we lijnen de tekst rechts uit:

```
footer {
  clear: left;
  padding: 1em 0;
  font-size: small;
  text-align: right;
}
```

OPGAVE 3.7

In deze opdracht vragen we u de hoofdstructuur van het formulier voor het aanvragen van een offerte in CSS te definiëren.

a In tegenstelling tot `pagina.html` bestaat `offerte.html` niet uit een linker- en een rechterkolom. In plaats daarvan hebben we hier een soort van rijen, de fieldsets, met daarin steeds een label (links) en een input control (rechts). Geef aan hoe de hoofdstructuur met behulp van float-positionering kan worden gespecificeerd.
Bedenk daarbij dat zowel labels als input-elementen inline-elementen zijn.

b Implementeer vervolgens de benodigde CSS code.

## 4 De lay-out in detail

De hoofdstructuur is nu vastgelegd. Bij het invullen van de details beginnen we met de indeling van het `article`-element. De indeling daarvan lijkt erg op het gedeelte van de hoofdstructuur waarin het `aside`-element naast het `article`-element ligt. Het verschil is dat er hier een aantal `aside`-elementen is, en dat die op specifieke plekken naast een element binnen het `article`-element staan.

Voor de aside-elementen hebben we een vaste breedte van 5em ontworpen, met rechts een ruimte van 1em die al geïmplementeerd is door de padding van het article-element, en met links ook een ruimte van 1em. De eerste stap is dan ook om de elementen links van de aside-elementen te instrueren om die ruimte open te laten. Dit lijkt niet generiek voor p- en andere elementen. Het ligt daarom in eerste instantie voor de hand om het volgende in de CSS op te nemen:

```
/*** specifiek voor article ***/
article p, article figure, article table, article h2,
article h3 {
  margin-right: 7em;
}
```

Het is echter een goede gewoonte om elementen zo te stylen dat een element zonder een klasse of id overal op de pagina op dezelfde manier wordt getoond. Daarom declareren we de eigenschappen die we van die elementen binnen het article-element nodig hebben als generiek, en voegen we een klasse toe aan de h2-elementen in de sidebar, die een margin-right van 0 moeten hebben:

```
/*** Generiek ***/
h2, h3 {
  margin: 0.5em 7em 0.5em 0;
}
p {
  margin-right: 7em;
}
figure {
  margin-right: 7em;
}
table {
  margin-right: 7em;
}
.sidebarh2 {
  margin-right: 0;
}
```

Er zijn uitzonderingen op deze regel. Wanneer u een ul-element heeft met een bepaalde klasse (bijvoorbeeld tasklist) met erg veel li-elementen als (klein)kinderen, wordt de HTML wel erg 'vol' wanneer al die li-elementen van een klasse voorzien moeten worden om ze een andere styling te geven dan de styling voor li-elementen zonder die klasse. In zo'n geval kunt u overwegen om ze te stylen met de selector .tasklist li. Maar in het algemeen probeert u dat te vermijden:

**Programmeeraanwijzing: Geen kind-selectoren**
Zorg ervoor dat een element zonder class of id overal op de pagina hetzelfde wordt getoond. Vermijd dus de selectoren sel1 sel2 of sel1 > sel2.

Nu is het eenvoudig om de aside-elementen naar rechts op te schuiven. Om duidelijker te maken dat het hier om zaken buiten de eigenlijke tekst om gaat, maken we de font-grootte klein. Die kleine font-grootte lijkt ook een goed idee voor het aside-element met het menu, en doordat in de declaraties voor .sidebar niets is opgenomen over font-size, geldt die declaratie ook voor de sidebar.

```
/*** Generiek ***/
aside {
  font-size: small;
  float: right;
  width: 6em;
}
```

OPGAVE 3.8
Hoe komt het dat de sidebar links blijft floaten, en een `width` van `8em` houdt in plaats van de eigenschappen voor `aside` over te nemen? U kunt dat controleren met Firebug, met het bestand in de map `article` in de bouwsteen.

We geven de `p`-elementen wat ruimte aan de onder- en bovenkant:

```
/*** Generiek ***/
p {
  margin: 0.6em 7em 0.6em 0;
}
```

Hier is duidelijk waarom het handig is dat CSS *collapsing margins* kent: nu blijft de ruimte onder en boven een `p`-element altijd `0.6em`, onafhankelijk van de vraag of erboven of eronder een ander `p`-element ligt of een element zonder eigen `margin`.

Ten slotte brengen we nog wat styling aan bij de `figure`-elementen. We gebruiken daarbij een truc waarmee afbeeldingen en filmpjes meeschalen met de beschikbare ruimte: de `figure`-elementen krijgen een vaste breedte van `30em` (zodat de breedte in pixels afhankelijk is van de instellingen van het apparaat en/of de gebruiker), terwijl de `img`- en `iframe`-elementen daarbinnen een breedte krijgen van `100%`. Daardoor schalen ze in de breedte mee, en de browser berekent zelf de bijbehorende schalingsfactor in de hoogte.

```
/*** Generiek ***/
figure {
  border: thin dotted #EF001D;
  width: 30em;
  padding: 1em;
  margin: 0 7em 1em 0;
  text-align: center;
}
figcaption {
  font-size: small;
  font-weight: bold;
  font-variant: small-caps;
}
figure img, figure iframe  {
  width: 100%;
}
```

OPGAVE 3.9
Specificeer naar eigen inzicht de details van de CSS voor het formulier om een offerte aan te vragen.

4.1 LIJSTEN

De tabs, de broodkruimels en het menu zien er nog erg onappetijtelijk uit. Bij de details hoort beslist ook dat die beter worden vormgegeven. Het is duidelijk dat de browser bolletjes invoegt voor de `li`-items, en in het menu is te zien dat de bolletjes open worden bij het tweede niveau, en vierkantjes worden bij het derde niveau. Voor mooie tabs en een mooi menu moeten we uiteraard van die bolletjes af. Dat doen we niet op generiek niveau, omdat het in de tekst van het `article`-element erg handig is. We voegen dus een CSS-klasse toe in de HTML, `navul`, aan alle `ul`-elementen die we zonder bolletjes willen zien.

```
/*** Navigatie ***/
.navul {
  list-style-type: none;
}
```

De tabs geven we op de volgende manier vorm:

```
/*** Navigatie: tabs ***/
.tabs {
      margin-left: 2em;
}
.tab {
  float: left;
  margin: 0 0.2em;
  line-height: 2em;
  width: 6em;
  text-align: center;
  font-variant: small-caps;
  color: Ivory;
  background: linear-gradient(to bottom, Gray, Black);
  border-radius: 0.5em;
}
.tab.active {
  background: linear-gradient(to bottom, #EF001D, #62000C);
}
```

Het werkt als volgt. `li`-elementen zijn block-elementen: daarom verschijnen de `li`-elementen onder elkaar. We laten ze naast elkaar verschijnen door ze een `float: left` mee te geven. Ze krijgen bovendien een kleine `margin` mee rechts en links. Door de `line-height` van `2em` nemen ze in de hoogte wat meer ruimte in dan normaal. Ze krijgen een vaste breedte. Voor de achtergrond is een gradient gebruikt, en ze krijgen ronde hoeken.

Doordat de `li`-elementen nu floatend zijn geworden, lijkt het voor de rest van de pagina alsof ze niet bestaan. Het `h1`-element schuift dan ook naar boven. Om de tabs toch netjes bovenop de rand van het `h1`-element te krijgen, krijgt `.pagetitle` een `clear:left`.

De actieve tab krijgt uiteraard een andere achtergrond. Het geheel schuiven we iets naar rechts op door middel van een `margin-left` op het `ul`-element met class `tabs`.

OPGAVE 3.10
Voorzie de submit-button van een gradient en ronde hoeken.

Op soortgelijke manier kunnen we de broodkruimels vormgeven. Daarbij vallen de regels voor klasse `active` onder generiek: die klasse komt op meerdere plekken voor en kan daar op dezelfde manier worden gebruikt (in tegenstelling tot de regels voor de actieve tab). Het `ul`-element in de broodkruimels krijgt een extra klasse omdat die rechtsfloatend gemaakt moet worden.

```
/*** Generiek ***/
.active {
  background-color: #E2001B;
  color: Ivory;
  padding: 0 1em;
}
/*** Navigatie: breadcrumbs ***/
.breadcrumbsul {
  float: right;
  list-style-image: url(../arrow.png);
}
.crumb {
  float: left;
  margin: 0 1em;
  font-size: small;
}
```

De broodkruimels krijgen nu een eigen gekozen image als `list-type-image`.

Omdat de broodkruimels wel erg ver onder het `h1`-element komen te staan, geven we `.pagetitle` een `margin-bottom` van `0.2em`. En voor alle zekerheid krijgt `article` nu een `clear:right` mee, zodat het nooit naast de broodkruimels kan schuiven.

*Semantische namen*

De `li`-items met class `active` hebben nu een rode achtergrond gekregen. De verleiding is misschien groot om daarom als naam voor de klasse niet `active` te kiezen, maar `red`. Zo'n keuze zou u in de problemen brengen wanneer u de stijl van de pagina zou willen veranderen (bijvoorbeeld met blauw als overwegende kleur): u krijgt dan erg vreemde klassennamen, of u moet de klassennamen zowel in de HTML als in de CSS veranderen. Daarom is het van belang om zoveel mogelijk *semantische namen* voor de klassen te gebruiken, die slaan op de betekenis, en niet op het uiterlijk.

De lijsten in de sidebar kunnen we op soortgelijke manier vormgeven. Ook daarbij zorgen we ervoor dat het door middel van een extra klasse duidelijk wordt dat `section` en `ul` zich anders gedragen dan elders.

```
/*** Navigatie: menu ***/
.sidebarsection {
  margin: 0 0 2em 00;
  padding-left: 1em;
}
.sidebarul {
  list-style-image: url(../arrow.png);
  margin-left: 1em;
}
```

De secties krijgen wat ruimte onderling. De lijsten in de sidebar krijgen in plaats van bolletjes een image, en doordat we ze een `margin-left` meegeven, springt elke sublijst een stukje in.

Het zoekveld ten slotte is lelijk breed: daar geven we een vaste breedte aan. Bovendien stylen we de knop voor het zoeken op dezelfde manier als de actieve tab:

```
/*** Generiek ***/
input[type="submit"] {
  background: linear-gradient(to bottom, #EF001D, #62000C);
  color: Ivory;
  width: 9em;
}
```

## 4.2 LINKS EN PSEUDO-CLASSES

Links zijn standaard blauw en onderstreept. Wanneer ze bezocht zijn, verandert de kleur in paars, en terwijl een pagina van zo'n link aan het laden is in een nieuwe tab, verschijnt er een randje om de actieve link. Ook die standaard-instellingen van de browser zijn te veranderen met behulp van CSS.

*Pseudo-selector*

Om te kunnen aangeven dat een `a`-element in een bepaalde toestand naar bepaalde stijlregels moet luisteren, kent CSS *pseudo-selector*en. Met `a:hover` selecteert u bijvoorbeeld `a`-elementen waar de gebruiker met de muis overheen beweegt:

```
/*** Links ***/
a:hover {
  background-color: blue;
  color: white;
}
```

De kleur blauw is plezierig voor links, omdat mensen gewend zijn dat die kleur op een link duidt. Voor de broodkruimels stellen we wel aangepaste kleuren in:

```
/*** Links ***/
.crumba {
  color: #E2001B;
  text-decoration: none;
}
.crumba:hover, .crumba:active {
  outline: thin, dotted, #E2001B;
  color: Ivory;
}
.crumba:visited {
  color: #62000C;
}
```

OPGAVE 3.11

Voeg de `hover`-eigenschap toe aan de submit-button.
Zorg er bijvoorbeeld voor dat de achtergrondkleur wordt veranderd.
Zoek eventueel in de specificaties hoe dit moet.

Het `table`-element en de elementen daarbinnen maken we even breed als de `figure`-elementen, die `30em` plus `2` maal `1em` breed waren. Verder kunnen we die elementen op de volgende manier stylen:

```
/*** Generiek ***/
table {
  width: 32em;
  border: thin dotted #EF001D;
  margin: 0 7em 1em 0;
}
td, th {
  border: thin dotted #EF001D;
  padding: 0.2em;
  font-size: small;
}
th {
  color: #E2001B;
  font-variant: small-caps;
}
```

Nu is te zien dat de cellen in de tabel dubbele randen krijgen: elke cel krijgt een eigen rand, die op een afstandje ligt van de naburige cellen. Dat is te voorkomen door aan het `table`-element mee te geven: `border-collapse: collapse;`.

Verder kan `caption` dezelfde eigenschappen meekrijgen die `figcaption` al had:

```
/*** Generiek ***/
figcaption, caption {
  font-size: small;
  font-weight: bold;
  font-variant: small-caps;
}
```

## 5 Refactoring

De laatste taak is om de code nauwkeurig door te lopen. Een eerste stap daarbij is om te controleren of er geen fouten in de code zitten. Zoals er een HTML-validator is, is er ook een CSS-validator. De weblink brengt u naar die validator.

Weblink: CSS validator

OPGAVE 3.12

Valideer een van de CSS-bestanden uit de bouwsteen met behulp van de validator (door directe invoer of via file upload). Doe dat nogmaals nadat u er een fout in hebt aangebracht (bijvoorbeeld door een dubbele punt weg te halen).

*Refactoring*

*Refactoring* houdt in: de code herstructureren zodat de functionaliteit gelijk blijft, terwijl de code beter onderhoudbaar wordt.

Weblink: CSSLint

Een hulpmiddel daarbij is CSSLint (zie weblink). CSSLint bekijkt CSS-code met een aantal vuistregels die gedocumenteerd zijn (zie weblink). Van een aantal waarschuwingen hoeft u zich niets aan te trekken: er wordt

Weblink: documentatie CSSLint

bijvoorbeeld gewaarschuwd wanneer u `margin`, `border` of `padding` gebruikt, omdat dat gemakkelijk fouten oplevert in de breedte die een element inneemt, vanwege het box model. Wanneer u een ontwerp op papier hebt gemaakt en alles netjes hebt uitgerekend, zullen die eigenschappen geen fouten opleveren.

Maar er zijn ook vuistregels die ervoor zorgen dat de pagina (of de site) beter onderhoudbaar wordt. Zo is er bijvoorbeeld een vuistregel om headings slechts één keer te definiëren. Wij hebben dat als algemeen principe geformuleerd: dat elementen zonder class of id zich overal in de pagina op dezelfde manier behoren te gedragen, wat inhoudt dat de selectoren `sel1 sel2` en `sel1 > sel2` uit den boze zijn.

OPGAVE 3.13
Voer het CSS-bestand uit de map `pagina/foutincsslint` in de bouwsteen in, in CSSLint. Klik eerst op het pijltje naast 'Lint', en haal het vinkje bij 'Beware of broken box size' weg: dat is op ons niet van toepassing.

In de meldingen is te zien dat CSSLint niet 'slim' genoeg is om een aantal situaties van elkaar te onderscheiden. Een regel in het Reset-gedeelte voor elementen, die verderop overschreven wordt, is geen fout, en geen ongewenste situatie. Een element dat als (klein)kind van een ander element andere declaraties krijgt dan buiten dat element (zoals `h2` versus `article h2`) is *wel* een ongewenste situatie, tenzij het om uitzonderingen gaat zoals `li`-elementen. Bekijk de meldingen van CSSLint dus altijd kritisch, en denk niet dat ze allemaal moeten verdwijnen om een onderhoudbaar CSS-bestand te hebben.

## 6 Ontwikkelingen

CSS is sterk in ontwikkeling. Er zijn belangrijke aanvullingen onderweg waarmee het veel gemakkelijker wordt om de hoofdstructuur te implementeren, of om elementen binnen bijvoorbeeld een formulier, een sidebar of een artikel netjes op een bepaalde plek te krijgen.

De belangrijkste nieuwe ontwikkelingen voor dat soort layout-kwesties zijn:

– Grid template layout, of kortweg template layout. Daarmee is het mogelijk om binnen een element (zoals bijvoorbeeld `.page`) een aantal 'slots' te definiëren, inclusief hun verhoudingen en onderlinge positie. Vervolgens kunt u zo'n slot aan een element geven.
– Grid layout, waarmee elementen in een tabel-achtige structuur kunnen worden gezet (zonder de betekenis van een tabel): ideaal voor formulieren.
– Flexbox, waarmee het gemakkelijker wordt om indelingen te creëren die adaptief reageren op de afmetingen van een scherm.

Weblink: Layout ontwikkelingen

De weblink geeft de (voorlopige) specificaties van die aanvullingen.

Een andere ontwikkeling is dat het mogelijk zal worden om variabelen te definiëren en als waarde te gaan gebruiken. Dat is ideaal voor kleuren en maten (bijvoorbeeld om brede en smalle marges consistent te houden).

Het zal op de volgende manier werken:

```
:root {
  main-color: #06c;
  accent-color: #006;
}
/* The rest of the CSS file */
.pagetitle {
  color: var(main-color);
}
h2, h3 {
  color: var(accent-color);
}
```

Weblink: SASS

Variabelen worden al langere tijd gemist in CSS. Er is een aantal tools ontwikkeld waarmee het mogelijk is de stijl te schrijven in een taal *met* variabelen. Die code wordt dan vertaald naar 'gewone' CSS. De bekendste van dat soort tools is SASS. Deze biedt naast het gebruik van variabelen meer mogelijkheden. Omdat de ontwikkelingen van CSS dat soort tools in de toekomst misschien overbodig maken, hebben we SASS niet in de cursus opgenomen. Tot die tijd is het erg aantrekkelijk om ermee te werken wanneer u veel CSS gaat schrijven.

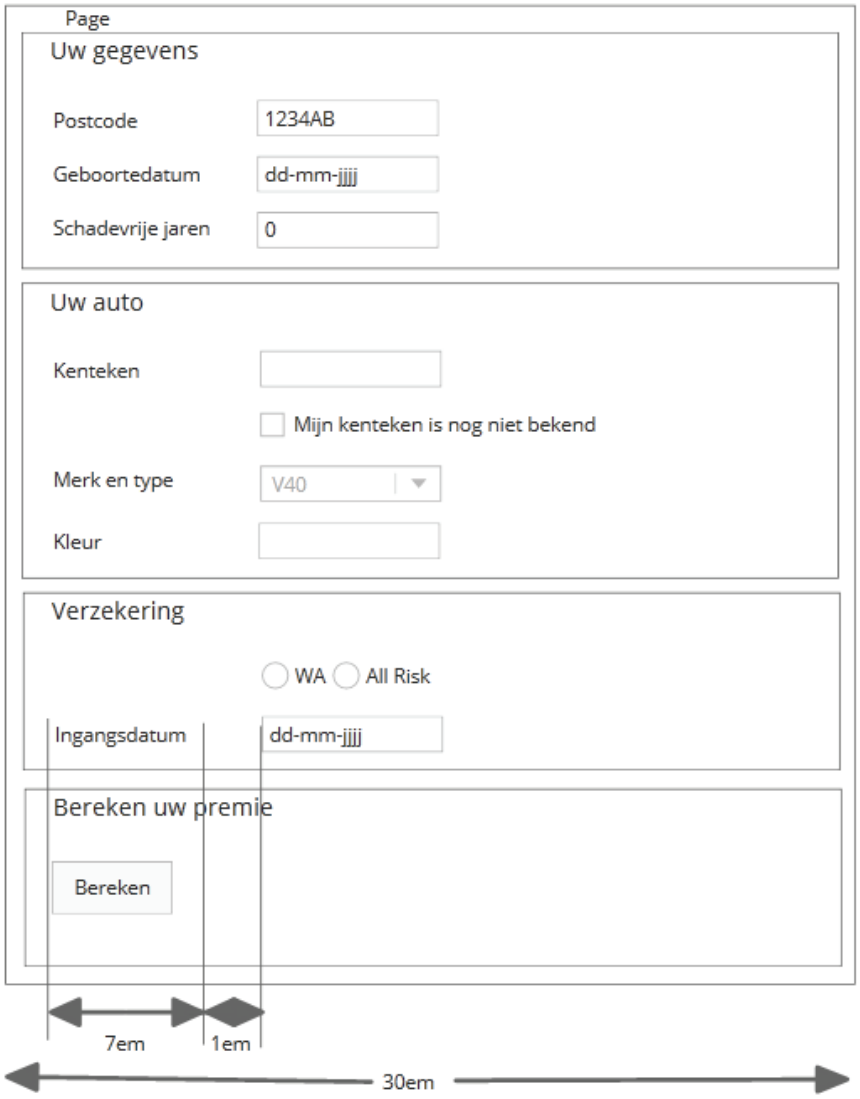
TERUGKOPPELING

**Uitwerking van de opgaven**

3.1 a Bij diverse onderdelen van het formulier zijn HTML5-controls toegepast. Als eerder aangegeven zijn deze niet in elke browser zichtbaar, maar in bijvoorbeeld Opera wel. In andere browsers worden de meeste van deze controls als tekstvelden getoond.
Het attribuut `required` zorgt ervoor dat een veld ingevuld moet zijn voordat het formulier kan worden verstuurd.
Bij een veld dat niet is ingevuld, komt dan een mededeling te staan.
Het `placeholder`-attribuut zorgt ervoor dat een gebruiker enig houvast heeft bij het invullen van een veld.
b De code is te vinden in het bestand `offerte1.html` in de bouwstenen.

```
<body>
  <div class="page">
    <h1>Bereken uw premie</h1>
    ...
  </div>
</body>
</html>
```

c Figuur 3.8 geeft een mogelijke schets.




FIGUUR 3.8 De layout op papier

Het formulier krijgt een breedte van `30em`. We zorgen er straks voor dat het formulier gecentreerd in het midden van het scherm zichtbaar is. Vooral bij meer brede schermen leest dit prettig.
We onderscheiden voor elke input een linker- en een rechterkant: links het label en rechts het veld dat de gebruiker moet invullen.
De labels geven we een breedte van `7em`.
Tussen de labels en de inputvelden voegen we `1em` ruimte toe.
Na elk inputveld hebben we voldoende ruimte over om later een feedbackveld toe te voegen, waarin we bij foutieve invoer een foutboodschap kunnen afbeelden.

NB. Later kunnen we, als blijkt dat dit nodig is, de breedte van bijvoorbeeld de labels aanpassen.

3.2 a De volgende regel voegt u toe aan het title-element:

```
<link rel="stylesheet" href="offerte.css" />
```

b Na het linken van het bestand `reset.css` is allerlei default-opmaak verdwenen. N.B. De code van beide opdrachten is te vinden in het bestand `offerte2.html`.

3.3 De opmaak van beide elementen kan als volgt zijn gedefinieerd:

```
/*** Generiek ***/
body {
  background-color: black;
}
/*** Hoofdstructuur ***/
.page {
  width: 30em;
  margin: 2em auto;
  background-color: Ivory;
  padding: 1em;
  box-shadow: 3px 3px 5px #888;
}
```

Belangrijk is om de breedte van de 'page' op `30em` te zetten, overeenkomstig het ontwerp.

3.4 Het `h1`-element wordt als volgt:

```
<h1 class="pagetitle">Bereken uw premie</h1>
```

Een mogelijke CSS-definitie is als volgt:

```
.pagetitle {
  padding: 0.25em;
  text-align: center;
  font-size: 2em;
  border: thin solid #EF001D;
  border-radius: 0.25em;
}
```

3.5 Als u naar beneden scrollt in het venster met de regels die van toepassing zijn, komt u declaraties tegen die zijn doorgestreept, zoals `border:0 none;`. De selector die bij die declaratie hoort is minder specifiek dan de selector met een declaratie voor een andere waarde voor `border`, namelijk de declaratie in de regel met `.pagetitle` als selector. U kunt in Firebug dus precies volgen welke declaraties er van toepassing zijn, en welke worden overschreven door andere.

De regels staan op volgorde van specificiteit: de meest specifieke staat bovenaan.

3.6 a We vullen de CSS-regel voor `#id2` als volgt aan:

```
position: absolute;
top: 1em;
right: 1em;
```

Het resultaat is dat het blauwe vierkantje naar de rechter-bovenhoek wordt veplaatst.

De reden hiervan is dat er geen voorouder is gepositioneerd en dus wordt het element ten opzichte van het `body`-element gepositioneerd. Verder zien we dat het rode vierkantje is opgeschoven en nu de plaats inneemt van het blauwe vierkantje.
Wanneer u de eigenschappen `top` en `right` verwijdert, ziet u dat het blauwe vierkantje boven het rode vierkantje wordt afgebeeld.

b Dit kan door het `div`-element, als eerste voorouder, met waarde `relative` te positioneren. Hiermee behoudt het `div`-element zijn plaats.

3.7 a We kijken hiervoor naar een HTML-fragment uit de eerste `fieldset` 'Uw gegevens':

```
<label>Postcode</label>
<input type="text" name="postcode" value=""
       placeholder="1234AB"  required />
```

Het `label`- en het `input`-element zijn beide inline-elementen en worden standaard naast elkaar geplaatst. Door het label naar links te floaten, verkrijgt het de eigenschappen van een block-element. Daarmee kunnen we het label een breedte (zeg `7em`) plus een rechter marge (zeg `1em`) geven. Het `input`-element schuift automatisch tegen het label aan. Daarmee zijn bijna alle `input`-elementen automatisch uitgelijnd.

Alleen de `checkbox` in het onderdeel 'Uw auto' en de `radiobuttons` in 'Verzekering' moeten nog naar rechts worden opgeschoven. Uitgaande van de breedte en rechtermarge van de labels geven we de `checkbox` en de `radiobuttons` een linkermarge van `8em`.

b Alle labels, behalve bij de checkbox en de radio buttons, krijgen een `class`-attribuut met waarde `heading`. De bijbehorende CSS voor deze klasse is:

```
.heading {
  float: left;
  width: 7em;
  margin-right: 1em;
}
```

De `div`-elementen die de `checkbox` en beide `radiobuttons` omsluiten, inclusief de bijbehorende labels, krijgen een `class`-attribuut met waarde `choice`.
De bijbehorende CSS voor deze klasse is:

```
.choice {
  margin-left: 8em;
}
```

Alle `input`-elementen geven we nog een linkermarge met waarde nul mee, omdat in de file `reset.css` de `margin` van de `checkboxes` en `radiobuttons` niet op nul staat. Hierdoor zijn de `checkbox` en de `radioboxes` niet netjes uitgelijnd. We zouden dit in `reset.css` kunnen aanpassen, maar hebben ervoor gekozen om dit hier nu te doen.

Alle code is te vinden in de bouwstenen in de bestanden `offerte2.html` en `offerte.css`.

3.8 De selector `.sidebar` is specifieker dan `aside`. De declaraties bij `.sidebar` overschrijven dus die bij `aside`.

3.9 Hieronder ziet u de gehele CSS-code tot nu toe. De belangrijke toevoegingen zijn vetgedrukt weergegeven. De totale uitwerking is te vinden in `offerte2.css` in de bouwstenen.

```
/*** Generiek ***/
body {
  background-color: black;
}
#pagetitle {
 padding: 0.25em;
 text-align: center;
 font-size: 2em;
 border: thin solid #888;
 border-radius: 0.25em;
}
/*** Hoofdstructuur ***/
.page {
  width: 30em;
  margin: 2em auto;
  background-color: Ivory;
  padding: 1em;
  box-shadow: 3px 3px 5px #888;
  border-radius: 0.25em;
}
.heading {
  float: left;
  width: 7em;
  margin-right: 1em;
  text-align: right;
}
/*** Formulier ***/
.choice {
  margin-left: 8em;
}
input {
   margin-left: 0em;
}
fieldset {
  margin-top: 1em;
  padding: 1em;
  box-shadow: 3px 3px 5px #888;
  border-radius: 0.25em;
  background: #F5F5DC;
}
legend {
  font-weight: bold;
  background-color: Ivory;
  padding: 0.2em;
  box-shadow: 3px 3px 5px #888;
  border-radius: 0.25em;
}
```

3.10 Een voorbeeld is:

```
input[type="submit"] {
  background: linear-gradient(to bottom, #B0B0B0, #F0F0F0);
  border-radius: 0.75em;
}
```

Deze specificatie is in de file `offerte.css` opgenomen.

3.11 De `hover`-eigenschap kan als volgt worden toegevoegd.

```
input:hover[type="submit"] {
  background: #DEB887;
}
```

3.12 Wanneer u een dubbele punt weghaalt, krijgt u een foutmelding te zien, met de aanduiding van het regelnummer.

3.13 De fouten hebben vooral te maken met de headings. CSSLint waarschuwt ook tegen het gebruik van id's. CSSLint is gericht op herbruikbare elementen, en een element dat met een id wordt geselecteerd is per definitie niet herbruikbaar. We zouden dat kunnen oplossen door beide id's te veranderen in klassen, maar in dit geval spreken we CSSLint tegen: met een id geven we aan dat we van deze elementen echt slechts één element per pagina willen zien.