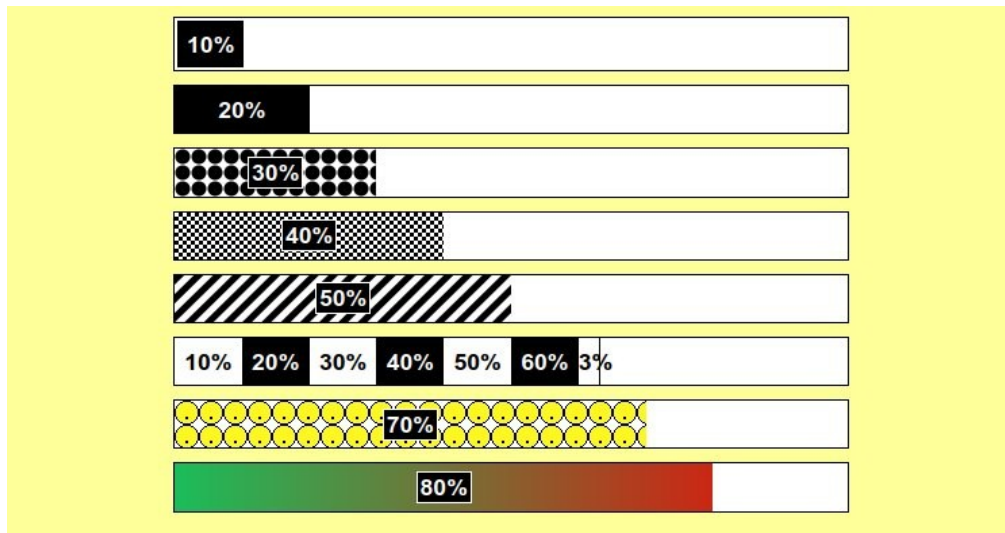


Makkelijk aan te passen horizontale staafdiagrammen



BELANGRIJKE INFORMATIE

Alles op deze site kan vrij worden gebruikt, met drie beperkingen:

* Je gebruikt het materiaal op deze site volledig op eigen risico. Het kan prima zijn dat er fouten in de hier verstrekte info zitten. Voor eventuele schade die door gebruik van materiaal van deze site ontstaat, in welke vorm dan ook, zijn www.css-voorbeelden.nl en medewerkers daarvan op geen enkele manier verantwoordelijk.

* Deze uitleg wordt min of meer regelmatig bijgewerkt. Het is daarom niet toegestaan deze uitleg op welke manier dan ook te verspreiden, zonder daarbij duidelijk te vermelden dat de uitleg afkomstig is van www.css-voorbeelden.nl en dat daar altijd de nieuwste versie is te vinden. Dit is om te voorkomen dat er verouderde versies worden verspreid.

Een link naar www.css-voorbeelden.nl wordt trouwens altijd op prijs gesteld.

* Het kan zijn dat materiaal is gebruikt dat van anderen afkomstig is. Dat materiaal kan onder een bepaalde licentie vallen, waardoor het mogelijk niet onbeperkt gebruikt mag worden. Als dat zo is, wordt dat vermeld onder [Inhoud van de download en licenties](#).

De volledige code vind je helemaal achteraan dit document. Deze code is exact hetzelfde als die van de in de download bijgesloten bestanden. Het is de bedoeling dat je die bestanden gebruikt, als je de code wilt bewerken. Kopiëren van de code achteraan dit bestand om die te bewerken – als dat al lukt – levert de wildste problemen op.

Alle code is geschreven in een afwijkende lettersoort. De code die te maken heeft met de basis van dit voorbeeld (essentiële code), is in de hele uitleg **vet blauw**. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles vanwege de leesbaarheid in een gewone letter.)

Inhoudsopgave

[Korte omschrijving](#)

[Opmerkingen](#)

[Achterliggend idee](#)

[Semantische elementen en WAI-ARIA](#)

[Semantische elementen](#)

[WAI-ARIA-codes](#)

[De code aanpassen aan je eigen ontwerp](#)

[Toegankelijkheid en zoekmachines](#)

[Getest in](#)
[Bekende problemen \(en oplossingen\)](#)
[Wijzigingen](#)
[Inhoud van de download en licenties](#)

Uitleg code:

[HTML](#) (in deze inhoudsopgave staat alleen de html, waar iets interessants over is te melden):

```
<!doctype html>
<html lang="nl">
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="060-css-dl/figuren-060-dl.css">
<h1>Acht voorbeelden van een staafdiagram.</h1>
<p>Buiten deze acht (...) achtergrond weergegeven.</p>
<div id="een" role="img" aria-label="Eerste diagram: voor 10% gevuld
    met zwarte achtergrond"><span
    aria-hidden="true">10&#8203;%</span></div>
<span aria-hidden="true">10&#8203;%</span>
```

[CSS](#) (in deze inhoudsopgave staat slechts de css die nodig is voor een werkend voorbeeld):

```
main {max-width: 90vw;}
main div {background-color: white; border: black solid 1px;}
div > span {background-color: black; color: white; display: inline-block;
    text-align: center;}
div span span {background-color: black; color: white;}
#een {padding: 2px;}
#een span {width: 10%; font-size: min(4vw, 1em);}
#binnen-twee {width: 20%;}
#binnen-drie {background-color: white; background-image:
    radial-gradient(black 5px, transparent 6px); background-size: 12px
    12px; width: 30%;}
#binnen-vier {background-color: white; background-image:
    linear-gradient(45deg, black 25%, transparent 25%, transparent 75%,
    black 75%), linear-gradient(45deg, black 25%, transparent 25%,
    transparent 75%, black 75%); background-position: 0 0, 3px 3px;
    background-size: 6px 6px; width: 40%;}
#binnen-vijf {background-image: repeating-linear-gradient(-45deg, black,
    white 1px, white 5px, black 6px, black 10px); width: 50%;}
#zes span {width: 10%; font-size: min(4vw, 1em);}
#zes span:nth-of-type(odd) {background-color: white; color: black;}
#zes span:last-of-type {width: 3%; border-right: black solid 1px;}
#binnen-zeven {background-color: white; background-image:
    url("../060-pics/smiley.gif"); width: 70%;}
#binnen-acht {background-image: linear-gradient(to right, #1bbd5a,
    #cc2714); width: 80%;}
```

Volledige code:

[HTML](#)
[CSS](#)

Korte omschrijving

Horizontale staafdiagrammen waarvan de lengte en het percentage uiterst simpel zijn aan te passen.

Opmerkingen

In een aantal diagrammen wordt voor de achtergrond een gradiënt als achtergrond-afbeelding gebruikt. Meestal wordt op deze site voor eigenschappen van de achtergrond-afbeelding de 'shorthand' `background` gebruikt. Hiermee kun je in één keer `background-attachment`, `background-clip`, `background-color`, `background-image`, `background-origin`, `background-position`, `background-repeat` en `background-size` instellen.

Hier worden steeds de afzonderlijke eigenschappen gebruikt. Deze wisselen nogal behoorlijk bij de diverse diagrammen. De shorthand `background` geeft altijd waarden voor alle acht de eigenschappen, ook als je zelf die waarden niet opgeeft. Als je bijvoorbeeld `background: url("../060-pics/smiley.gif");` gebruikt, wordt als achtergrond-afbeelding netjes 'smiley.gif' gebruikt. Maar de overige zeven eigenschappen die ook door shorthand `background` worden aangestuurd, worden nu teruggezet op hun standaardwaarden, ook al heb je daar helemaal niets voor opgegeven.

Meestal is dat geen enkel probleem, hier wel. Omdat die eigenschappen zo sterk wisselen per diagram, kun je er makkelijk eentje vergeten. En dat soort fouten is soms ook nog 'ns knap lastig te vinden.

Daarom worden hier, ook al kost dat wat meer css, steeds de afzonderlijke eigenschappen gebruikt, en niet de shorthand `background`.

Links in deze uitleg, vooral links naar andere sites, kunnen verouderd zijn. Op www.css-voorbeelden.nl/links vind je steeds de meest recente links.

Dit voorbeeld is gemaakt op een systeem met Linux ([Kubuntu](http://www.kubuntu.org)). Daarbij is vooral gebruik gemaakt van [Visual Studio Code](http://www.visualstudio.com), [GIMP](http://www.gimp.org) en [Firefox](http://www.mozilla.org) met extensies. De pdf-bestanden zijn gemaakt met [LibreOffice](http://www.libreoffice.org).

Vragen of opmerkingen? Fout gevonden? Ga naar het [forum](#).

Achterliggend idee

De buitenkant van elk staafdiagram is een gewone `<div>` met een border. Die border is er eigenlijk alleen om te laten zien, hoe breed het volledige staafdiagram, de volle honderd procent, is.

In deze `<div>` worden een of meer ``'s gezet, waarin de achtergrond van het diagram en het percentage staan. Als het diagram veertig procent moet aangeven, moet de `` veertig procent van de breedte van de `<div>` krijgen. De in de `` zittende achtergrond krijgt dan ook die breedte, waardoor de `<div>` voor veertig procent met de achtergrond wordt gevuld.

Hier doet zich echter een probleem voor: een `` is een inline-element, waardoor je er geen breedte e.d. aan kunt geven. Je zou het met `display: block;` in een blok-element kunnen veranderen, maar in het zesde diagram zitten meerdere ``'s, en die komen dan elk op een nieuwe regel te staan.

Met `display: inline-block;` worden de ``'s in een soort kruising tussen een inline- en een blok-element veranderd: ze kunnen nu wel een breedte e.d. krijgen, maar blijven toch op dezelfde regel staan.

Het eerste en tweede diagram hebben een simpele zwarte achtergrond.

Het derde diagram heeft een achtergrond die met behulp van `radial-gradient()` is gemaakt. Je kunt daarmee relatief simpel cirkels maken.

De kleine blokjes in het vierde diagram ontstaan door twee `linear-gradient()`'s elkaar te laten overlappen. Omdat de gradiënten als `background-image` worden

gebruikt, kun je gewoon meerdere gradiënten gebruiken, net zoals bij gewone `background-image`'s.

De diagonale strepen in het vijfde diagram zijn gemaakt met behulp van `repeating-linear-gradient()`. Door deze gradiënt onder een hoek te zetten, krijg je diagonale strepen.

Het zesde diagram wijkt wat af, omdat hierin niet een, maar zes ``'s zitten, afwisselend met een zwarte en witte achtergrond- en voorgrondkleur.

Het zevende diagram is het enige dat nog een afbeelding gebruikt. Een emoticon die niet beweegt is inmiddels geen enkel probleem meer, want er zijn inmiddels heel veel emoticons die net als gewone letters gebruikt kunnen worden. Het probleem zit hem in de beweging. In theorie zou het misschien wel mogelijk zijn om met behulp van iets als `animation` in combinatie met iets als SVG ook dit zonder afbeelding te maken, maar dat is meer iets voor zeer gemotiveerde puzzelaars. Hier is gewoon een simpele gif gebruikt.

Het achtste diagram heeft weer een gewone `linear-gradient()`, nu met een verlopende kleur.

De percentages staan elk in een eigen ``. Bij het zesde diagram levert dat een klein probleem op, als de lettergrootte meer dan ongeveer 140 procent wordt. Het procentteken past dan niet meer in de ``, waardoor dit (gedeeltelijk) wegvalt. Dit wordt opgelost door een spatie zonder breedte tussen het getal en het procent te zetten. Meer hierover is te vinden bij [`10​%`](#).

De `<div>`'s met het diagram worden nooit breder dan het venster van de browser. Omdat de breedte van de ``'s met de achtergronden een percentage van de `<div>`'s is, blijven de verhoudingen altijd hetzelfde, ook in smallere vensters: tachtig procent blijft tachtig procent van een bepaalde breedte.

Omdat voor de vulling van de diagrammen achtergrond-afbeeldingen zijn gebruikt, is het percentage van de diagrammen heel simpel te wijzigen. Als je het percentage wijzigt, hoef je alleen de breedte van de `` met de diagrammen en het getal bij het percentage te veranderen naar het nieuwe percentage.

In vorige versies werden geen gradiënten gebruikt, maar achtergrond-afbeeldingen. Hier wordt alleen voor het zevende diagram (met de smileys) nog een afbeelding gebruikt. De gradiënten hebben een paar voordelen boven afbeeldingen. Je hoeft ze niet te downloaden, ze kunnen zonder enig kwaliteitsverlies eindeloos worden vergroot, en op hogeresolutieschermen worden ze beter weergegeven.

Een afbeelding bestaat uit pixels, een gradiënt uit formules. In een afbeelding van een bepaalde grootte zit een bepaald aantal pixels. Bij een vergroting of op een hogeresolutiescherm levert dat vroeger of later (fors) kwaliteitsverlies op.

Een gradiënt wordt opgebouwd aan de hand van formules: maak een zwarte cirkel met een doorsnede van 1 centimeter. Als dat op een hogeresolutiescherm wordt weergegeven, of de cirkel moet 2 centimeter groot worden, worden gewoon meer pixels gebruikt.

Afgezien van de opbouw werken gradiënten verder precies hetzelfde als gewone achtergrond-afbeeldingen.

Schermlezers kunnen geen diagrammen voorlezen. Alleen de percentages worden voorgelezen, want dat is gewone tekst. Dit zou de ietwat eigenaardige zin '10% 20% 30% 40% 50% 10% 20% 30% 40% 50% 60% 3% 70% 80%' opleveren. Met behulp van WAI-ARIA-codes wordt daarom voor schermlezers een beschrijving van elk diagram gegeven. Deze is niet te zien, maar wordt wel voorgelezen. De percentages worden voor schermlezers

verborgen, want die staan al in de beschrijving van het diagram. Meer hierover is te vinden bij [WAI-ARIA-codes](#).

Semantische elementen en WAI-ARIA

Deze twee onderwerpen zijn samengevoegd, omdat ze veel met elkaar te maken hebben.

Semantische elementen

De meeste elementen die in html worden gebruikt, hebben een semantische betekenis. Dat wil zeggen dat je aan de gebruikte tag al (enigszins) kunt zien, wat voor soort inhoud er in het element staat. In een `<h1>` staat een belangrijke kop. In een `<h2>` staat een iets minder belangrijke kop. In een `<p>` staat een alinea. In een `<table>` staat een tabel (en geen lay-out, als het goed is!). Enz.

Door het op de goede manier gebruiken van semantische elementen, kunnen zoekmachines, schermlezers, enz. de structuur van een pagina begrijpen. De spider van een zoekmachine is redelijk te vergelijken met een blinde. Het is dus ook in je eigen belang om semantische elementen zo goed mogelijk te gebruiken. Een site die toegankelijk is voor mensen met een handicap, is in de regel ook goed te verwerken door een zoekmachine en maakt dus een grotere kans gevonden en bezocht te worden.

Als het goed is, wordt het uiterlijk van de pagina bepaald met behulp van css. Het uiterlijk staat hierdoor (vrijwel) los van de semantische inhoud van de pagina. Met behulp van css kun je een `<h1>` heel klein weergeven en een `<h6>` heel groot, terwijl schermlezers, zoekmachines, e.d. nog steeds weten dat de `<h1>` een belangrijke kop is.

Slechts enkele elementen, zoals `<div>` en ``, hebben geen semantische betekenis. Daardoor zijn deze elementen uitstekend geschikt om met behulp van css het uiterlijk van de pagina aan te passen: de semantische betekenis verandert niet, maar het uiterlijk wel. Voor een schermlezer of zoekmachine verandert er (vrijwel) niets, voor de gemiddelde bezoeker krijgt het door de css een heel ander uiterlijk. (De derde laag, naast html voor de inhoud en css voor het uiterlijk, is JavaScript. Die zorgt voor de interactie tussen site en bezoeker. De min of meer strikte scheiding tussen css en html aan de ene kant en JavaScript aan de andere kant is met de komst van css3 en html5 veel vager geworden. Je kunt nu bijvoorbeeld ook met css dingen langzaam verplaatsen en met html deels de invoer in formulieren controleren.)

Html5 heeft een aantal nieuwe elementen, die speciaal zijn bedoeld om de opbouw van een pagina aan te geven. In dit voorbeeld wordt hiervan alleen `<main>` gebruikt. `<main>` gedraagt zich als een gewone `<div>`, maar dan een `<div>` met een semantische betekenis. Hierdoor kunnen schermlezers, zoekmachines, e.d. beter zien, hoe de pagina is samengesteld. De meeste schermlezers kunnen dit soort elementen ook gebruiken om snel door de pagina te navigeren.

`<MAIN>`

Hierbinnen staat de belangrijkste inhoud van de pagina (in dit voorbeeld is dat de hele pagina).

Met behulp van dit soort nieuwe semantische elementen kan bijvoorbeeld een schermlezer in één keer een heel menu passeren en gelijk naar de echte inhoud gaan.

WAI-ARIA-codes

WAI-ARIA wordt vaak ingekort tot ARIA. Voluit betekent het Web Accessibility Initiative – Accessible Rich Internet Applications.

Er worden in dit voorbeeld drie WAI-ARIA-codes gebruikt: `role="img"`, `aria-label` en `aria-hidden`.

ROLE="IMG"

Elk staafdiagram zit in een eigen `<div>`. Hoewel de tag `<div>` wordt gebruikt, zijn de diagrammen eigenlijk afbeeldingen. Op het scherm maakt dit niets uit, maar voor schermlezers wel, want schermlezers behandelen een afbeelding op een afwijkende manier. Met behulp van `role="img"` wordt aan schermlezers duidelijk gemaakt, dat dit eigenlijk afbeeldingen zijn:

```
<div id="een" role="img" aria-label="Eerste  
    diagram: voor 10% gevuld met zwarte  
    achtergrond">
```

Een schermlezer zal deze `<div>` nu behandelen, alsof het een afbeelding is.

ARIA-LABEL

Hier gelijk boven is met `role="img"` aan schermlezers duidelijk gemaakt, dat de `<div>` met het diagram eigenlijk een afbeelding is. Dat maakt het mogelijk om in `aria-label` een beschrijving van de `<div>` te geven. Deze beschrijving is niet te zien op het scherm, maar wordt wel voorgelezen. De beschrijvingen op deze pagina zijn een beetje mallotig, maar dit zijn natuurlijk geen echte diagrammen.

ARIA-HIDDEN

Met behulp van `aria-hidden="true"` kan een deel van de code worden verborgen voor schermlezers e.d., zodat dit niet wordt voorgelezen. Op de normale weergave op het scherm heeft dit verder geen enkele invloed. Met behulp van `aria-label` gelijk hierboven is een omschrijving van het diagram gegeven, die wordt voorgelezen. Na deze omschrijving zou dan alsnog het in het diagram staande percentage worden voorgelezen. Dat is dubbelop en bovendien verwarrend. Daarom wordt dit percentage met behulp van `aria-hidden` voor schermlezers verborgen:

```
<span aria-hidden="true">10&#8203;%</span>
```

De code aanpassen aan je eigen ontwerp

- Als je dit voorbeeld gaat aanpassen voor je eigen site, houd het dan in eerste instantie zo eenvoudig mogelijk. Ga vooral geen details invullen.
- Gebruik geen FrontPage, Publisher of Word (alle drie van Microsoft). Publisher en Word zijn niet bedoeld om websites mee te maken. FrontPage is zwaar verouderd en wordt al jaren niet meer onderhouden door Microsoft. Ook OpenOffice en LibreOffice leveren een uiterst beroerd soort html af. Tekstverwerkers met al hun toeters en bellen zijn gewoon niet geschikt om websites mee te bouwen. Je kunt beter een goed (gratis) programma gebruiken. Links naar dat soort programma's vind je op de pagina met [links](#) onder Gereedschap → wysiwyg-editor. Maar het allerbeste is om gewoon zelf html, css, enz. te leren, omdat zelfs het allerbeste programma het nog steeds zwaar verliest van 'n op de juiste manier met de hand gemaakte pagina.
- Als je in een desktopbrowser met behulp van zoomen het beeld vergroot, heeft dit hetzelfde effect, als wanneer de pagina in een kleiner browservenster wordt getoond. Je kunt hiermee dus kleinere apparaten zoals een tablet of een smartphone simuleren. Maar het blijft

natuurlijk wel een simulatie: het is nooit hetzelfde als testen op een écht apparaat. Zo kun je bijvoorbeeld aanrakingen alleen echt testen op een echt touchscreen.

Inmiddels hebben veel browsers in de ontwikkelgereedschappen mogelijkheden voor het simuleren van weergave op een kleiner scherm ingebouwd. Ook dit blijft een simulatie, maar geeft vaak wel een beter beeld dan zoomen.

- Als je 'n site maakt in Firefox, Opera, Safari, Google Chrome of Edge, is er 'n hele grote kans dat hij in alle browsers werkt. Ik geef de voorkeur aan Firefox, omdat het de enige grote browser is die niet bij een bedrijf hoort dat vooral op je centen of je data uit is. Google Chrome wordt ook door veel mensen gebruikt, maar ik heb dus wat moeite met hoe Google je hele surfgedrag, je schoenmaat en de kleur van je onderbroek vastlegt. Daarom gebruik ik Google Chrome zelf alleen om in te testen.
- Het allereerste dat je moet invoeren, is het doctype, vóór welke andere code dan ook. Een lay-out met een missend of onvolledig doctype ziet er totaal anders uit dan een lay-out met een geldig doctype. Wát er anders is, verschilt ook nog 'ns tussen de diverse browsers. Als je klaar bent en dan nog 'ns 'n doctype gaat invoeren, weet je vrijwel zeker dat je van voren af aan kunt beginnen met de lay-out.

Geldige doctypes vind je op www.w3.org/QA/2002/04/valid-dtd-list.

Gebruik het volledige doctype, inclusief de eventuele url, anders werkt het niet goed.

- Gebruik een 'strict' doctype of (beter!) het doctype voor html5. Deze zijn bedoeld voor nieuwe sites. Het transitional doctype is bedoeld voor al bestaande sites, niet voor nieuwe. Het transitional doctype staat talloze tags toe, die in html5 zijn verboden. Deze tags worden al zo'n tien jaar afgeraden. Het transitional doctype is echt alleen bedoeld om de puinhoop van vroeger, toen niet volgens standaarden werd gewerkt, enigszins te herstellen. Het strict doctype staat verouderde tags niet toe. Daardoor kan met 'n strict doctype, of het nu html of xhtml is, probleemloos worden overgestapt naar html5. Met een transitional doctype en het gebruik van afgekeurde tags kun je niet overstappen naar html5. Je moet dan eerst alle verouderde tags verwijderen, wat echt ontzettend veel werk kan zijn.

Het doctype voor html5 is uiterst simpel: `<!doctype html>`. Omdat het doctype voor html5 in alle browsers werkt, zelfs in de gelukkig vrijwel uitgestorven nachtmerrie Internet Explorer 6, is er geen enkele reden dit uiterst simpele doctype niet te gebruiken.

- De eerste regel binnen de `<head>` moet de charset zijn. Dit vertelt de browser, welke tekenset er gebruikt moet worden, zodat letters met accenten e.d. overal goed worden weergegeven. Het beste kun je utf-8 nemen. Als je later van charset verandert, loop je 'n grote kans dat je alle aparte tekens als letters met accenten weer opnieuw moet gaan invoeren. In html5 is het simpele `<meta charset="utf-8">` voldoende.
- Test vanaf het allereerste begin in zoveel mogelijk verschillende browsers in 'n aantal resoluties (schermgroottes). Onder het kopje [Getest in](#) kun je in deze uitleg vinden, waar dit voorbeeld in is getest.
- Voor alle voorbeelden geldt: breng veranderingen stapsgewijs aan. Als je bijvoorbeeld foto's wilt laten weergeven, begin dan met het alleen veranderen van de namen van de foto's, zodat je eigen foto's worden weergegeven. Maakt niet uit als de maten niet kloppen en de teksten fout zijn. Als dat werkt, ga dan bijvoorbeeld de maten aanpassen. Dan de teksten. En controleer steeds, of alles nog goed werkt.
- Als het om een lay-out of iets dergelijks gaat: zorg eerst dat header, kolommen, footer, menu, en dergelijke staan en bewegen, zoals je wilt. Ga daarna pas details binnen die blokken invullen. In eerste instantie gebruik je dus bijvoorbeeld 'n leeg blok op de plaats, waar uiteindelijk het menu komt te staan.

Als je begint met allerlei details, is er 'n heel grote kans dat die de werking van de blokken gaan verstoren. Bouw eerst het huis, en ga dan pas de kamers inrichten. Zorg eerst dat de blokken werken, zoals je wilt. Dan zul je het daarna gelijk merken, als 'n toegevoegd detail als tekst of 'n afbeelding iets gaat verstoren. Daarvoor moet je natuurlijk wel regelmatig controleren in verschillende browsers, of alles nog wel goed werkt.

Je kunt de blokken tijdens het aanpassen opvullen met bijvoorbeeld `
1
2
3` enz., tot ze de juiste hoogte hebben. Het is handig om aan het einde even iets toe te voegen als 'laatste', zodat je zeker weet dat er niet ongemerkt drie regels onderaan naar 't virtuele walhalla zijn verhuisd.

Om de breedte te vullen, kun je het best 'n kort woord als 'huis' duizend keer of zo herhalen. Ook hier is het handig om aan 't eind (en hier ook aan 't begin) 'n herkenningsteken te maken, zodat je zeker weet dat je de hele tekst ziet.

- Zolang je in grotere dingen zoals 'n lay-out aan 't wijzigen bent, kan het helpen de verschillende delen een achtergrondkleur te geven. Je ziet dan goed, waar 'n deel precies staat. Een achtergrondkleur heeft – anders dan bijvoorbeeld een border – verder geen invloed op de lay-out, dus die is hier heel geschikt voor.
- Als je eigenschappen verandert in de css, verander er dan maar één, hooguit twee tegelijk. Als je er zeventien tegelijk verandert, is de kans groot dat je niet meer weet, wat je hebt gedaan. En dat je 't dus niet meer terug kunt draaien.
- `margin`, `padding` en `border` worden bij de hoogte en breedte van het element opgeteld. Hier worden vaak fouten mee gemaakt. Als je bijvoorbeeld in een lay-out 'n border toevoegt aan een van de 'hoofdvakken' (header, footer, kolommen), dan wordt deze er bij opgeteld. Bij 'n border van 2 px rondom de linkerkolom wordt deze dus plotseling 4 px breder (2 px aan beide kanten), en 4 px hoger. Zoiets kan je hele lay-out verstoren, omdat iets net te breed of te hoog wordt. Je moet dan elders iets 4 px kleiner maken. Dat zal vaak zo zijn: als je één maat verandert, zul je vaak ook 'n andere moeten aanpassen. Css geeft de mogelijkheid om met behulp van `box-sizing` de padding en border binnen de breedte en hoogte van de inhoud te zetten, als je dat handiger vindt. Met nieuwere css-eigenschappen als `grid` en `flexbox`, die speciaal zijn gemaakt om een lay-out mee te maken, spelen dit soort problemen veel minder. In alle browsers waarop hier nog wordt getest, werken `flexbox` en `grid` prima. Maar als je oudere browsers moet ondersteunen, kan dat wel problemen opleveren en moet je ook in die oudere browsers testen.
- In plaats van een absolute eenheid als `px` kun je ook een relatieve eenheid gebruiken, met name `em` en `rem`. Voordeel van `em` en `rem` is dat een lettergrootte, regelhoogte, e.d. in `em` en `rem` in alle browsers kan worden veranderd. Nadeel is dat het de lay-out sneller kan verstoren dan bijvoorbeeld `px`. Dit moet je gewoon van geval tot geval bekijken. Voor weergave in mobiele apparaten zijn relatieve eenheden als `em` en `rem` vrijwel altijd beter dan absolute eenheden als `px`.

(De minder bekende `rem` is ongeveer hetzelfde als de `em`. Alleen is de lettergrootte bij `rem` gebaseerd op de lettergrootte van het `<html>`-element, waardoor de `rem` overal op de pagina precies even groot is. Bij de `em` kan de lettergrootte worden beïnvloed door de voorouders van het element, bij de `rem` niet.)

Zoomen kan trouwens altijd, ongeacht welke eenheid je gebruikt.

- Valideren, valideren, valideren en dan voor 't slapen gaan nog 'ns valideren. Valiwié???

Valideren is het controleren van je html en css op 'n hele serie fouten. Computers zijn daar vaak veel beter in dan mensen. Als je 300 keer `<h2>` hebt gebruikt en 299 keer `</h2>` vindt 'n computer die ene missende `</h2>` zonder enig probleem. Jij ook wel, maar daarna ben je misschien wel aan vakantie toe.

Valideren kan helpen om gekmakende fouten te vinden. Valide code garandeert ook dat de weergave in verschillende browsers (vrijwel) hetzelfde is. En valide code is over twintig jaar ook nog te bekijken.

Valideren moet trouwens ook niet worden overdreven. Het is een hulpmiddel om echte fouten te vinden, meer niet. Het gaat erom dat je site goed werkt, niet dat je het braafste kind van de klas bent. Als de code niet valideert, maar daar is een goede reden voor, is daar niets

op tegen. Zeker met nieuwere html en css wil de validator nog wel eens achterlopen, terwijl dat al prima is te gebruiken.

Op deze site is alle css en html gevalideerd. Als de code niet helemaal valide is (wat regelmatig voorkomt), staat daar onder [Bekende problemen \(en oplossingen\)](#) de reden van. Je kunt je css en html valideren als 't online staat, maar ook als het nog in je computer staat. Html kun je valideren op: validator.w3.org/nu.

Css kun je valideren op: jigsaw.w3.org/css-validator.

Toegankelijkheid en zoekmachines

De tekst in dit hoofdstukje is een algemene tekst, die voor elke pagina geldt. Eventueel specifiek voor dit voorbeeld geldende problemen en eventuele aanpassingen om die problemen te voorkomen staan bij [Bekende problemen \(en oplossingen\)](#).

Toegankelijkheid (in het Engels 'accessibility') is belangrijk voor bijvoorbeeld blinden die een schermlezer gebruiken, of voor motorisch gehandicapte mensen die moeite hebben met het bedienen van een muis. Een spider van een zoekmachine (dat is het programmaatje dat de site indexeert voor de zoekmachine) is te vergelijken met een blinde. Als je je site goed toegankelijk maakt voor gehandicapten, is dat gelijk goed voor een hogere plaats in een zoekmachine. Dus als je 't niet uit sociale motieven wilt doen, kun je 't uit egoïstische motieven doen.

(Op die plaats in de zoekmachine heb je maar beperkt invloed. De toegankelijkheid van je site is maar één van de factoren, maar zeker niet onbelangrijk.)

Als je bij het maken van je site al rekening houdt met toegankelijkheid, is dat nauwelijks extra werk. 't Is ongeveer te vergelijken met inbraakbescherming: doe dat bij 'n nieuw huis en 't is nauwelijks extra werk, doe 't bij 'n bestaand huis en 't is al snel 'n enorme klus. Enkele tips die helpen bij toegankelijkheid:

- Gebruik altijd een alt-beschrijving bij een afbeelding. De alt-tekst wordt gebruikt, als afbeeldingen niet kunnen worden getoond of gezien (dat geldt dus ook voor zoekmachines). Als je iets wilt laten zien, als je over de afbeelding hovert, gebruik daar dan het title-attribuut voor, niet de alt-beschrijving.

Als een afbeelding alleen maar voor de sier wordt gebruikt, zet je daarbij `alt=""`, om aan te geven dat de afbeelding niet belangrijk is voor het begrijpen van de tekst of zo.

- Gebruik in links een tekst die duidelijk aangeeft, waar de link naartoe gaat. Een tekst als 'pagina met externe links' is waarschijnlijk duidelijk genoeg, een tekst als alleen 'links' waarschijnlijk niet. Een duidelijke zwart-witregel is niet te geven, omdat dit ook van tekst e.d. in de omgeving van de link afhangt.

Schermlezers kunnen een lijst van alle links in de pagina weergeven, en een duidelijke tekst is daarbij belangrijk. Alleen 'volgende' zegt niets, als dat in 'n lijst met alleen links staat.

- Accesskeys (sneltoetsen) kun je beter niet gebruiken, deze geven te veel problemen, omdat ze vaak dubbelop zijn met sneltoetsen voor de browser of andere al gebruikte sneltoetsen. Bovendien is voor de gebruiker meestal niet duidelijk, welke toetsen het zijn.

Op zichzelf zijn accesskeys een heel goed idee. Maar helaas zijn ze ook in html5 volstrekt onvoldoende gedefinieerd. Er is nog steeds geen standaard voor de meest gebruikelijke accesskeys, zoals Zoek of Home.

Er is nog steeds niet vastgelegd, hoe accesskeys zichtbaar gemaakt kunnen worden. Voor de makers van browsers zou dit 'n relatief kleine moeite zijn, voor de makers van 'n site is het bergen extra werk.

Hierdoor zijn accesskeys (vrijwel) niet te gebruiken. Misschien kunnen ze nog enig nut hebben op sites, die gericht zijn op 'n specifieke groep gebruikers. Maar voor algemene sites is het advies: normaal genomen niet gebruiken.

- Met behulp van de Tab-toets (of op 'n soortgelijke manier) kun je in de meeste browsers door links, invoervelden, e.d. lopen. Elke tab brengt je één link, invoerveld, e.d. verder,

Shift+Tab één plaats terug. Met behulp van het attribuut `tabindex` kun je de volgorde aangeven, waarin de Tab-toets werkt. Zonder `tabindex` wordt de volgorde van de html aangehouden bij gebruik van de Tab-toets, maar soms is een andere volgorde logischer. In principe is het beter, als `tabindex` niet nodig is, maar gewoon de volgorde van de html wordt aangehouden. Bij verkeerd gebruik kan `tabindex` heel verwarrend zijn. Het is niet bedoeld om van de pagina een hindernisbaan voor kangoeroes te maken, waarop van beneden via links over rechts naar boven wordt gesprongen.

- Als, zoals hierboven beschreven, een gebruiker van de Tab-toets bij een link, invoerveld, e.d. is aangekomen, heeft dit element 'focus'. Dit wordt aangegeven door de link, invoerveld, e.d. extra te markeren met een kadertje. Dat kadertje mag je alleen weghalen, als op een andere manier wordt duidelijk gemaakt, welk element focus heeft. Een gebruiker van de Tab-toets kan anders niet zien, waar zij of hij zit, en welk element gaat reageren op bijvoorbeeld een Enter.
- In het verleden werd vaak aangeraden de volgorde van de code aan te passen. Een menu bijvoorbeeld kon in de html onderaan worden gezet, terwijl het op het scherm met behulp van css bovenaan werd gezet. Inmiddels zijn schermlezers e.d. zo sterk verbeterd dat dit niet meer wordt aangeraden. De volgorde in de html kan tegenwoordig beter hetzelfde zijn als die op het scherm, omdat het anders juist verwarrend kan werken.

- Een zogenaamde skip-link is vaak nog wel zinvol. Dat is een link die je buiten het scherm parkeert met behulp van css, zodat hij normaal genomen niet te zien is. Zo'n link is wel gewoon zichtbaar in speciale programma's zoals tekstbrowsers en schermlezers, want die kijken gewoon naar wat er in de broncode staat.

(Alleen in de schermlezer TalkBack op oudere versies van Android werkt zo'n buiten het scherm geplaatste link niet. TalkBack leest zo'n link wel voor, maar de link kan niet worden gevolgd, als deze buiten het scherm staat. Met ingang van versie 8.1 van Android is dit eindelijk opgelost en werkt een skip-link ook fatsoenlijk in TalkBack.)

Een skip-link staat bovenaan de pagina, nog boven menu, header, e.d., en linkt naar de eigenlijke inhoud van de pagina. Hierdoor kunnen mensen met één toetsaanslag naar de eigenlijke inhoud van de pagina gaan.

Een skip-link is vooral nuttig voor gebruikers van de Tab-toets. Zodra de normaal genomen onzichtbare link door het indrukken van de Tab-toets focus krijgt, kun je hem op het scherm plaatsen, waardoor hij zichtbaar wordt. Bij een volgende tab wordt hij dan weer buiten het scherm geplaatst en is dus niet meer zichtbaar, zodat de lay-out niet wordt verstoord.

Op pagina's en in voorbeelden waar dat nuttig is, wordt op deze site een skip-link gebruikt.

(Althans: nog niet in alle voorbeelden die daarvoor in aanmerking komen, zit een skip-link. Maar geleidelijk aan worden dat er steeds meer.)

- Van oorsprong is html een taal om wetenschappelijke documenten weer te geven, pas later is hij gebruikt voor lay-out. Maar daar is hij dus eigenlijk nooit voor bedoeld geweest. Het gebruiken van html voor lay-out leidt tot enorme problemen voor gehandicapten en tot een lage plaats in zoekmachines.

De html hoort alleen inhoud te bevatten, lay-out doe je met behulp van css. Die css moet in een externe stylesheet staan of, als hij alleen voor één bepaalde pagina van toepassing is, in de `<head>` van die pagina.

- Breng een logische structuur aan in je document. Gebruik een `<h1>` voor de belangrijkste kop, een `<h2>` voor een subkop, enz. Schermlezers e.d. kunnen van kop naar kop springen. En een zoekmachine gaat ervan uit dat `<h1>` belangrijke tekst bevat.

Dit geldt voor al dit soort structuurbepalende tags.

Als een `<h1>` te grote letters geeft, maak daar dan met behulp van je css 'n kleinere letter van, maar blijf die `<h1>` gewoon gebruiken. Op dezelfde manier kun je al dit soort dingen oplossen.

- `<table>` is fantastisch, maar alleen als die wordt gebruikt om een echte tabel weer te geven, niet als hij voor opmaak wordt misbruikt. In het verleden is dat op grote schaal gebeurd bij

gebrek aan andere mogelijkheden. Een tabel is, als je niet heel erg goed oplet, volstrekt ontoegankelijk voor gehandicapten en zoekmachines. Het lezen van een tabel is ongeveer te vergelijken met het lezen van een krant van links naar rechts: niet per kolom, maar per regel. Dat gaat dus alleen maar goed bij een echte tabel, zoals een spreadsheet. In alle andere gevallen garandeert 'n tabel volstreekte ontoegankelijkheid voor schermlezers e.d. en als extra bonus vaak 'n lagere plaats in een zoekmachine.

- Frames horen bij een volstrekt verouderde techniek, die heel veel nadelen met zich meebrengt. <iframe>'s hebben voor een deel dezelfde nadelen. Eén van die nadelen is dat de verschillende frames voor zoekmachines, schermlezers, e.d. als los zand aan elkaar hangen, omdat ze los van elkaar worden weergegeven. Ze staan wel naast elkaar op het scherm, maar er zit intern geen verband tussen.

Als je 'n stuk code vaker wilt gebruiken, zoals 'n menu dat op elke pagina hetzelfde is, voeg dat dan in met PHP. Dan wordt de pagina niet pas in de browser, maar al op de server samengesteld. Hierdoor zien zoekmachines, schermlezers, e.d. één pagina, net zoals wanneer je maar één pagina met html zou hebben geschreven.

(Je kunt ook invoegen met behulp van SSI (Server Side Includes). Maar tegenwoordig kun je beter PHP dan SSI gebruiken, omdat SSI min of meer aan het uitsterven is en PHP veel meer mogelijkheden heeft. Op deze site wordt in enkele voorbeelden nog SSI gebruikt, maar zodra die worden bijgewerkt, gaat dat vervangen worden door PHP.)

- Geef de taal van het document aan, en bij woorden en dergelijke die afwijken van die taal de afwijkende taal met behulp van `lang="..."`. Op deze site gebeurt dat maar af en toe, omdat de tekst (en vooral de code) een mengsel is van Engels, Nederlands en eigengemaakte namen. Dat soort teksten is gewoon niet goed in te delen in een taal. Maar bij enigszins 'normale' teksten hoor je een taalwisseling aan te geven.

Op deze site wordt de lijst op woordenlijst.org gebruikt om te bepalen, of een woord inmiddels 'Nederlands' is. Als het woord in deze lijst voorkomt, wordt geen `lang`-attribuut gebruikt, ook niet als het woord oorspronkelijk uit een andere taal komt.

- Gebruik de tag <abbr> bij afkortingen. Doe dat de eerste keer op een pagina samen met de title-eigenschap: <abbr title="ten opzichte van">t.o.v.</abbr>. Daarna kun je op dezelfde pagina volstaan met <abbr>t.o.v.</abbr>. Doe je dit niet, dan is er 'n grote kans dat 'n schermlezer 't.o.v.' uit gaat spreken als 'tof', en 'n zoekmachine kan er ook geen chocola van maken.

- Geef een verandering niet alleen door kleur aan. Een grote groep mensen heeft moeite met het onderscheiden van kleuren en/of het herkennen van kleuren. Verander bijvoorbeeld een ronde rode knop niet in een ronde groene knop, maar in een vierkante groene knop. Door ook de vorm te veranderen, is het herkennen van de verandering niet alleen van een kleur afhankelijk.

- Zorg voor voldoende contrast tussen achtergrond- en voorgrondkleur, tussen `background-color` en `color`. Soms zie je heel lichtgrijze tekst op een donkergrijze achtergrond, en dan ook nog in een mini-formaat. Dat is dus voor heel veel mensen stomweg volledig onleesbaar. Op de pagina met [links](#) staat onder het kopje Toegankelijkheid → Contrast en kleurenblindheid een hele serie sites, waar je kunt controleren of het contrast groot genoeg is.

- De spider van 'n zoekmachine, schermlezers, en dergelijke kunnen geen plaatjes 'lezen'. Het is soms verbazingwekkend om te zien hoe veel, of eigenlijk: hoe weinig tekst er overblijft op een pagina, als de plaatjes worden weggehaald.

Op Linux kun je met Lynx kijken, hoe je pagina eruitziet zonder plaatjes e.d., als echt alleen de tekst overblijft. Een installatie-programma voor Lynx op Windows is te vinden op invisible-island.net/lynx.

Ook kun je in Windows het gratis programma WebbIE installeren. WebbIE laat de pagina zien, zoals een tekstbrowser e.d. hem ziet. WebbIE is te downloaden vanaf www.webbie.org.uk.

- Ten slotte kun je je pagina nog online op toegankelijkheid laten controleren op 'n behoorlijk aantal sites, zoals:

lowvision.support: laat zien hoe een kleurenblinde de site ziet. Engelstalig.

wave.webaim.org: deze laat grafisch zien, hoe de toegankelijkheid is. Engelstalig. Deze tester is ook als extensie in Firefox en Google Chrome te installeren.

Op de pagina met [links](#) kun je onder Toegankelijkheid links naar meer tests e.d. vinden.

Getest in

Laatst gecontroleerd op 14 februari 2023.

Onder dit kopje staat alleen maar, hoe en waarin is getest. Alle eventuele problemen, ook die met betrekking tot zoomen, lettergroottes, toegankelijkheid, uitstaan van JavaScript en/of css, enz. staan iets hieronder bij [Bekende problemen \(en oplossingen\)](#). Het is belangrijk dat deel te lezen, want uit een test kan ook prima blijken dat iets totaal niet werkt!

Dit voorbeeld is getest op de volgende systemen:

DESKTOPCOMPUTERS

Linux (Kubuntu 20.04 LTS, 'Focal Fossa') (2560 x 1080 px, resolution: 96 ppi):

Firefox, Google Chrome en Vivaldi, in grotere en kleinere browservensters.

In Vivaldi is ook ruimtelijke navigatie ('Spatial Navigation') getest.

LAPTOPS

Windows 10 (1600 x 900 px, resolution: 106 ppi):

Firefox, Google Chrome en Edge, in grotere en kleinere browservensters.

OS X 11.7.3 ('Big Sur') (1440 x 900 px, resolution: 96 ppi, device-pixel-ratio: 1):

Firefox, Safari, Google Chrome en Microsoft Edge, in grotere en kleinere browservensters.

TABLETS

iPad met iOS 12.5.7 (2048 x 1536 px, device-pixel-ratio: 2):

Safari, Chrome, Firefox, Microsoft Edge (alle portret en landschap).

iPad met iOS 13.3 (gesimuleerd in Xcode):

Safari (portret en landschap).

iPad met iPadOS 16.3 (2160 x 1620 px, 264 ppi):

Safari, Chrome, Firefox, Microsoft Edge (alle portret en landschap).

Android 6.0 ('Marshmallow') (1920 x 1200 px, resolution: 224 ppi):

Samsung Internet, Firefox en Chrome (alle portret en landschap).

Android 8.1 ('Oreo') (1920 x 1200 px, resolution: 218 ppi):

Samsung Internet, Firefox, Microsoft Edge en Chrome (alle portret en landschap).

Android 13 (2000 x 1200 px, resolution: 225 ppi):

Samsung Internet, Firefox, Microsoft Edge en Chrome (alle portret en landschap).

SMARTPHONES

iPhone 7 met iOS 12.4 (gesimuleerd in Xcode):
Safari (portret en landschap).

iPhone 8 met iOS 13.4 (gesimuleerd in Xcode):
Safari (portret en landschap).

iPhone met iOS 15.7.3 (1334 x 750, 326 ppi):
Safari, Chrome, Firefox en Microsoft Edge (alle portret en landschap).

Android 7.0 ('Nougat') (1280 x 720 px, resolution: 294 ppi):
Samsung Internet, Firefox, Microsoft Edge en Chrome (alle portret en landschap).

Android 9.0 ('Pie') (1920 x 1080 px, resolution: 424 ppi):
Samsung Internet, Firefox, Microsoft Edge en Chrome (alle portret en landschap).

Er is op de aan het begin van dit hoofdstukje genoemde controledatum getest in de meest recente versie van de browser, die op het betreffende besturingssysteem kon draaien. Het aantal geteste browsers en systemen is al tamelijk fors, en als ook nog rekening gehouden moet worden met (zwaar) verouderde browsers, is het gewoon niet meer te doen. Surfen met een verouderde browser is trouwens vragen om ellende, want updates van browsers hebben heel vaak met beveiligingsproblemen te maken.

In- en uitzoomen en – voor zover de browser dat kan – een kleinere en grotere letter zijn ook getest. Er is ingezoomd en vergroot tot zover de browser kan, maar niet verder dan 200%.

Er is getest met behulp van muis en toetsenbord, behalve op iOS, iPadOS en Android, waar een touchscreen is gebruikt. Op Windows 10 is getest met touchscreen, touchpad, toetsenbord, muis, en – waar dat zinvol was – op een combinatie daarvan. Op OS X 11.7.3 is getest met (een combinatie van) toetsenbord, touchpad en muis.

Als in een voorbeeld JavaScript is gebruikt, is ook getest of het werkt zonder JavaScript. Dat is alleen gedaan in de browsers, waarin in de instellingen JavaScript kan worden uitgeschakeld.

Ook is getest zonder css en – als afbeeldingen worden gebruikt – zonder afbeeldingen.

SCHERMLEZERS E.D.

Naast deze 'gewone' browsers is ook getest in Lynx, WebbIE, NVDA, TalkBack, VoiceOver en Verteller.

[Lynx](#) is een browser die alleen tekst laat zien en geen css gebruikt. Er is getest op Linux.

[WebbIE](#) is een browser die gericht is op mensen met een handicap. Er is getest op Windows 10.

[NVDA](#) is een schermlezer, zoals die door blinden wordt gebruikt. Er is getest op Windows 10 in combinatie met Firefox.

TalkBack is een in Android ingebouwde schermlezer. Er is getest in combinatie met Chrome op Android 6.0, 7.0, 8.1, 9 en 13.

VoiceOver is een in iOS en OS X ingebouwde schermlezer. Er is getest in combinatie met Safari op iOS 12.5.7 en 15.7.3, iPadOS 16.3 en OS X 11.7.3.

Verteller (Narrator) is een in Windows 10 ingebouwde schermlezer. Er is getest in combinatie met Edge.

(Voor de bovenstaande programma's zijn links naar sites met uitleg e.d. te vinden op de pagina met [links](#) onder Toegankelijkheid → Schermlezers, tekstbrowsers, en dergelijke.)

Als het voorbeeld in deze programma's toegankelijk is, zou het in principe toegankelijk moeten zijn in alle aangepaste browsers en dergelijke. En dus ook voor zoekmachines, want een zoekmachine is redelijk vergelijkbaar met een blinde. Eventuele problemen in schermlezers (en eventuele aanpassingen om die te voorkomen) staan iets hieronder bij [Bekende problemen \(en oplossingen\)](#).

Alleen op de hierboven genoemde systemen en browsers is getest. Er is dus niet getest op bijvoorbeeld 'n Blackberry. Er is een kans dat dit voorbeeld niet (volledig) werkt op niet-geteste systemen en apparaten. Om het wel (volledig) werkend te krijgen, zul je soms (kleine) wijzigingen en/of (kleine) aanvullingen moeten aanbrengen, bijvoorbeeld met JavaScript.

Er is ook geen enkele garantie dat iets werkt in een andere tablet of smartphone dan hierboven genoemd, omdat fabrikanten in principe de software kunnen veranderen. Dit is anders dan op de desktop, waar browsers altijd (vrijwel) hetzelfde werken, zelfs op verschillende besturingssystemen. Iets wat in Samsung Internet op Android werkt, zal in de regel overal werken in die browser, maar een garantie is er niet. De enige garantie is het daadwerkelijk testen op een fysiek apparaat. En aangezien er duizenden mobiele apparaten zijn, is daar geen beginnen aan.

De html is gevalideerd met de [html-validator](#), de css met de [css-validator](#) van w3c. Als om een of andere reden niet volledig gevalideerd kon worden, wordt dat bij [Bekende problemen \(en oplossingen\)](#) vermeld.

Nieuwe browsers worden pas getest, als ze uit het bèta-stadium zijn. Anders is er 'n redelijke kans dat je tegen 'n bug zit te vechten, die voor de uiteindelijke versie nog gerepareerd wordt.

Dit voorbeeld is alleen getest in de hierboven met name genoemde browsers. Vragen over niet-geteste browsers kunnen niet worden beantwoord, en het melden van fouten in niet-geteste browsers heeft ook geen enkel nut. (Melden van fouten, problemen, enz. in wel geteste browsers: graag! Dat kan op het [forum](#).)

Bekende problemen (en oplossingen)

Waarop en hoe is getest, kun je gelijk hierboven vinden bij [Getest in](#).

Als je hieronder geen oplossing vindt voor een probleem dat met dit voorbeeld te maken heeft, kun je op het [forum](#) proberen een oplossing te vinden voor je probleem. Om forumspam te voorkomen, moet je je helaas wel registreren, voordat je op het forum een probleem kunt aankaarten.

Bij toegankelijkheid is er vaak geen goed onderscheid te maken tussen oplossing en probleem. Zonder (heel simpele) aanpassingen heb je vaak 'n probleem, en omgekeerd. Daarom staan wat betreft toegankelijkheid aanpassingen en problemen hier bij elkaar in dit hoofdstukje.

Voor zover van toepassing wordt eerst het ontbreken van JavaScript, css en/of afbeeldingen besproken. Vervolgens problemen en aanpassingen met betrekking tot toegankelijkheid voor specifieke groepen bezoekers, zoals zoomen en andere lettergrootte, Tab-toets, tekstbrowsers en schermlezers. Als laatste volgen de overige problemen in één of meer specifieke browsers.

Als in een onderdeel geen problemen aanwezig zijn, staat in een smal groen kadertje 'Geen problemen'. Bij een onderwerp over toegankelijkheid zijn er soms geen problemen, maar alleen aanpassingen. Ook in dat geval staat bovenaan in een smal groen kadertje 'Geen problemen'. Daaronder staan dan de aanpassingen.

Als in een onderdeel één of meer problemen worden besproken, staat van elk probleem in een breed rood kadertje een korte samenvatting daarvan.

Als bij het probleem een oplossing wordt gegeven, staat de samenvatting in een rode stippellijn. Bij een onderwerp over toegankelijkheid zijn er soms, naast de opgeloste problemen, ook aanpassingen. In dat geval staan die aanpassingen boven de kadertjes met opgeloste problemen.

Als bij het probleem geen oplossing is gevonden, staat de samenvatting in een rode ononderbroken lijn. Bij een onderwerp over toegankelijkheid zijn er soms, naast de problemen, ook aanpassingen. In dat geval staan die aanpassingen boven de kadertjes met problemen.

ZONDER JAVASCRIPT

Geen problemen.

Omdat JavaScript helemaal niet wordt gebruikt, heeft het aan- of uitstaan van JavaScript geen enkele invloed.

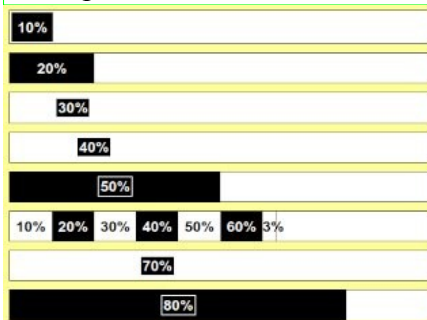
ZONDER CSS

Probleem: alleen de omschrijving binnen de `<h1>` en de `<p>` en de percentages zijn zichtbaar.

Omdat omtrek en achtergrond van de diagrammen volledig met behulp van CSS worden gemaakt, vallen die weg als CSS uitstaat. Dit is niet goed op te lossen. Bij een 'echt' diagram zal er tekst bij het diagram staan. Mogelijk geeft dat een goede beschrijving van het diagram, maar ook dan blijven de percentages zichtbaar, wat vooral bij het zesde diagram verwarrend kan zijn: '10%20%30%40%50%60%3%'. Nou zal dit niet vaak meer voorkomen en mensen die CSS uit hebben staan, zullen dit soort dingen gewend zijn. Hoe dan ook: niets aan te doen.

ZONDER AFBEELDINGEN

Geen problemen.



In veel browsers kunnen afbeeldingen worden geblokkeerd. Maar voor de diagrammen worden geen gewone afbeeldingen, maar achtergrond-afbeeldingen gebruikt. (Ook al zijn het vaak gradiënten, ze worden behandeld als een `background-image`.) In de meeste browsers worden achtergrond-afbeeldingen niet geblokkeerd, als gewone afbeeldingen wel worden geblokkeerd. Een overzicht van wie wat

blokkeert heeft geen nut, want mensen kunnen ook nog extensies hebben geïnstalleerd, die het gedrag van de browser veranderen.

Als een browser alle afbeeldingen blokkeert, zoals Firefox in sommige uitvoeringen, dan is het resultaat zoals op de afbeelding. Bij alle diagrammen is het percentage nog duidelijk zichtbaar, en vaak ook een zwarte achtergrond die het percentage aangeeft. Dan is er ook nog 'n enkele vreemde snuiter die alleen de smileys blokkeert, als afbeeldingen worden geblokkeerd. Die excentriekeling toont alle diagrammen zoals bedoeld, behalve de zevende, want die gebruikt voor de achtergrond smileys. In alle gevallen is echter duidelijk te zien, wat het percentage is.

ZONDER CSS ÉN ZONDER AFBEELDINGEN

Probleem: alleen de omschrijving binnen de <h1> en de <p> en de percentages zijn zichtbaar.

Dit is hetzelfde als bij Zonder css iets hierboven.

TEKSTBROWSERS

Probleem: alleen de omschrijving binnen de <h1> en de <p> en in Lynx ook de percentages zijn zichtbaar.

In tekstbrowsers zijn deze diagrammen niet goed te begrijpen. Je ziet alleen de korte beschrijving van de pagina boven de diagrammen (die eigenlijk voor schermlezers is bedoeld), en in Lynx zie je dan ook nog een onbegrijpelijke serie percentages. Alleen door in de omringende tekst de diagrammen te beschrijven, is dit (enigszins) op te vangen.

SCHERMLEZERS

Geen problemen.

Zonder aanpassingen voor schermlezers zouden deze alleen de percentages voorlezen. Daarom is voor schermlezers een korte omschrijving voor de hele pagina en een korte omschrijving van elk diagram aangebracht.

De korte omschrijving voor de hele pagina staat buiten het scherm en is daardoor onzichtbaar, maar deze wordt wel gewoon voorgelezen. Meer hierover is te vinden bij [<h1>...](#) en [<p>...](#).

Ook de omschrijving bij elk afzonderlijk diagram is onzichtbaar, maar wordt wel voorgelezen door schermlezers. Meer hierover is te vinden bij [WAI-ARIA-codes](#).

ZOOMEN EN ANDERE LETTERGROOTTE

Geen problemen.

Bij inzoomen (vergroten) is er geen enkel probleem.

Bij een lettergrootte van meer dan ongeveer 140 procent verdween in een vorige versie het procentteken in het zesde diagram gedeeltelijk of volledig. Door het toevoegen van een spatie zonder breedte tussen getal en procent, komt het procentteken nu op een nieuwe regel te staan. Meer hierover is te vinden bij [10​%](#).

Wijzigingen

Alleen grotere wijzigingen worden hier vermeld, geen dingen als een link die is geüpdatet.
12 mei 2009:

Nieuw opgenomen.

27 januari 2011:

Vanwege toegankelijkheid `color: black;` toegevoegd aan css voor <body>.

2 januari 2012:

Bij Bekende problemen (en oplossingen) stukje over Firefox en Safari toegevoegd.

14 februari 2023:

Volledig herschreven. Belangrijkste wijzigingen:

- Alles voor Internet Explorer verwijderd.
- xhtml veranderd naar html.
- Is nu ook goed te bekijken op kleine schermen.
- Aantal nieuwe hoofdstukken.
- Voor schermlezers beschrijving van elk diagram toegevoegd.
- Selectors zo kort mogelijk gemaakt.

- Nieuwere selectors gebruikt zoals `:nth-of-type`, waardoor de meeste classes en id's konden vervallen.
- Vrijwel alle afbeeldingen vervangen door gradiënten.
- Bij een grotere letter valt het percentage-teken in het zesde diagram niet meer weg.
- Tig kleine wijzigingen in de tekst.

Inhoud van de download en licenties

De inhoud van deze download kan vrij worden gebruikt, met drie beperkingen:

- * Sommige onderdelen die van 'n andere site of zo afkomstig zijn, vallen mogelijk onder een of andere licentie. Dat is hieronder bij het betreffende onderdeel te vinden.
- * Je gebruikt het materiaal uit deze download volledig op eigen risico. Het kan prima zijn dat er fouten in de hier verstrekte code e.d. zitten. Voor eventuele schade die door gebruik van materiaal uit deze download ontstaat, in welke vorm dan ook, zijn www.css-voorbeelden.nl en medewerkers daarvan op geen enkele manier verantwoordelijk.
- * Dit voorbeeld (en de bijbehorende uitleg e.d.) wordt min of meer regelmatig bijgewerkt. Het is daarom niet toegestaan dit voorbeeld (en de bijbehorende uitleg e.d.) op welke manier dan ook te verspreiden, zonder daarbij duidelijk te vermelden dat voorbeeld, uitleg, e.d. afkomstig zijn van www.css-voorbeelden.nl en dat daar altijd de nieuwste versie is te vinden. Dit is om te voorkomen dat er verouderde versies worden verspreid. Een link naar www.css-voorbeelden.nl wordt trouwens altijd op prijs gesteld.

figuren-060-dl.html: de pagina met het voorbeeld.

figuren-060.pdf: deze uitleg (aangepast aan de inhoud van de download).

figuren-060-inhoud-download-en-licenties.txt: een kopie van de tekst onder dit kopje (Inhoud van de download en licenties).

060-css-dl:

figuren-060-dl.css: stylesheet voor figuren-060-dl.html.

060-pics:

De smiley die wordt gebruikt in het zevende diagram.

HTML

De code is geschreven in een afwijkende lettersoort. De code die te maken heeft met de basis van dit voorbeeld (essentiële code), is in de hele uitleg **vet blauw**. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles in een gewone letter vanwege de leesbaarheid.)

In de html hieronder wordt alleen de html besproken, waarover iets meer is te vertellen. Een `<h1>` bijvoorbeeld wordt in de regel niet genoemd, omdat daarover weinig interessants valt te melden. (Als bijvoorbeeld het uiterlijk van de `<h1>` wordt aangepast met behulp van css, staat dat verderop bij de bespreking van de [css](http://www.css-voorbeelden.nl).)

Zaken als een `doctype` en `charset` hebben soms wat voor veel mensen onbekende effecten, dus daarover wordt hieronder wel een en ander geschreven.

`<!doctype html>`

Een document moet met een `doctype` beginnen om weergaveverschillen tussen browsers te voorkomen. Zonder `doctype` is de kans op verschillende (en soms volkomen verkeerde) weergave tussen verschillende browsers heel erg groot.

Geldige doctypes vind je op www.w3.org/QA/2002-04/valid-dtd-list.

Gebruik het volledige `doctype`, inclusief de eventuele url, anders werkt het niet goed.

Het hier gebruikte `doctype` is dat van html5. Dit kan zonder enig probleem worden gebruikt: het werkt zelfs in Internet Explorer 6.

<html lang="nl">

Het attribuut `lang="nl"` bij `<html>` geeft aan dat de pagina in het Nederlands is. De taal is van belang voor schermlezers, automatisch afbreken, automatisch genereren van aanhalingstekens, juist gebruik van decimale punt of komma, e.d.

<meta charset="utf-8">

Zorgt dat de browser letters met accenten e.d. goed kan weergeven.

utf-8 is de beste charset (tekenset), omdat deze alle talen van de wereld (en nog heel veel andere extra tekens) bestrijkt, maar toch niet meer ruimte inneemt voor de code, dan nodig is. Als je utf-8 gebruikt, hoef je veel minder entiteiten (`ä`; e.d.) te gebruiken, maar kun je bijvoorbeeld gewoon `ä` gebruiken.

Deze regel moet zo hoog mogelijk komen te staan, als eerste regel binnen de `<head>`, omdat hij anders door sommige browsers niet wordt gelezen.

In html5 hoeft deze regel niet langer te zijn, dan wat hier staat.

<meta name="viewport" content="width=device-width, initial-scale=1">

Mobiele apparaten variëren enorm in grootte. En dat is een probleem. Sites waren, in ieder geval tot enkele jaren geleden, gemaakt voor desktopbrowsers. En die hebben, in vergelijking met bijvoorbeeld een smartphone, heel brede browservensters. Hoe moet je op 'n smartphone een pagina weergeven, die is gemaakt voor de breedte van een desktop? Je kunt natuurlijk wachten tot alle sites zijn omgebouwd voor smartphones, tablets, enz., maar dan moet je waarschijnlijk heel erg lang wachten.

Mobiele browsers gokken erop dat een pagina een bepaalde breedte heeft. Safari voor mobiel bijvoorbeeld gaat ervan uit dat een pagina 980 px breed is. De pagina wordt vervolgens zoveel versmald dat hij binnen het venster van het apparaat past. Op een iPhone wordt de pagina dus veel smaller dan op een iPad. Vervolgens kan de gebruiker inzoomen op het deel van de pagina dat deze wil zien.

Dit betekent ook dat bij het openen van de pagina de tekst meestal heel erg klein wordt weergegeven. (Meestal, want niet alle browsers en apparaten doen het op dezelfde manier.) Niet erg fraai, maar bedenk maar 'ns 'n betere oplossing voor bestaande sites.

Nieuwe sites of pagina's kunnen echter wel rekening houden met de veel kleinere vensters van mobiele apparaten. In dit voorbeeld bijvoorbeeld wordt breedte van de diagrammen aangepast aan de breedte van het venster.

Maar die stomme mobiele browser weet dat niet, dus die gaat ervan uit dat ook deze pagina 980 px breed is, en verkleint die dan. Dat is ongeveer even behulpzaam als de gediensige kelner die behulpzaam de stoel naar achteren trekt, net als jij wilt gaan zitten.

Om de door de browser aangeboden hulp vriendelijk maar beslist te weigeren, wordt deze tag gebruikt. Hiermee geef je aan dat de pagina is geoptimaliseerd voor mobiele apparaten. De kreet `width=device-width` zegt tegen de mobiele browser dat de breedte van de weer te geven pagina gelijk is aan de breedte van het apparaat. Als een iPad in portretstand bijvoorbeeld 768 px breed is, wordt de pagina ook 768 px breed.

Er staat nog een tweede deel in de tag: `initial-scale=1`. Sommige mobiele apparaten zoomen een pagina gelijk in of uit. Ook weer in een poging behulpzaam te zijn. Ook dat is hier niet nodig. Er is ook een instructie om zoomen helemaal onmogelijk te maken, maar die wordt niet gebruikt. De bezoeker kan zelf nog gewoon zoomen, wat belangrijk is voor mensen die wat slechter zien.

`<link rel="stylesheet" href="060-css-dl/figuren-060-dl.css">`

Dit is een koppeling naar een externe stylesheet (stijlbestand), waarin de css staat. In html5 is de toevoeging `type="text/css"` niet meer nodig, omdat dit standaard al zo staat ingesteld. Je moet uiteraard de naam van en het pad naar de stylesheet aanpassen aan de naam en plaats, waar je eigen stylesheet staat.

Voordeel van een externe stylesheet is o.a. dat deze geldig is voor alle pagina's, waaraan deze is gelinkt. 'n Verandering in de lay-out hoeft je dan maar in één enkele stylesheet aan te brengen, in plaats van in elke pagina apart. Op een grotere site kan dit ontzettend veel werk schele n. Bovendien hoeft de browser zo'n externe stylesheet maar één keer te downloaden, ongeacht hoeveel pagina's er gebruik van maken. Zou je de css in elke pagina opnieuw aanbrengen, dan worden de te downloaden bestanden veel groter.

In dit voorbeeld heeft een extern stylesheet eigenlijk geen nut, omdat er maar één pagina is die dit stylesheet gebruikt. In dit geval kun je de css beter in de `<head>` van de html-pagina zelf zetten. Voor de omvang maakt het hier niets uit, want de css wordt hoe dan ook altijd precies één keer gedownload, en nooit vaker. Voor het onderhoud maakt het ook geen verschil, want ook hier hoeft je de css maar op één plaats te wijzigen. Maar het scheelt wel een extra aanroep naar de server, omdat geen apart stylesheet hoeft te worden gedownload. Dat opnemen in de `<head>` gaat heel simpel: je kopieert gewoon het hele stylesheet en zet die bovenin de `<head>`, tussen `<style>` en `</style>`:

```
<style>
    body {color: black;}
        (...) rest van de css (...)
    div {color: red;}
</style>
```

Maar zodra een stylesheet op meerdere pagina's wordt gebruikt, wat meestal het geval zal zijn, is een extern stylesheet beter.

(De reden dat er toch externe stylesheets zijn, terwijl hierboven omstandig wordt beweerd dat dat in dit voorbeeld eigenlijk geen nut heeft: overzichtelijkheid. Nu kun je html en css los van elkaar bekijken.)

`<h1>Acht voorbeelden van een staafdiagram.</h1>`

`<p>Buiten deze acht staafdiagrammen is deze pagina leeg. Het percentage dat is gevuld, wordt bij elk diagram met een andere achtergrond weergegeven.</p>`

Binnen de `<h1>` staat de belangrijkste kopregel van de pagina. Binnen de `<p>` staat een korte omschrijving van de pagina. Beide zijn bedoeld voor schermlezers, want voor een schermlezer bestaan de diagrammen bij gebrek aan tekst niet. Alleen de percentages binnen de diagrammen worden voorgelezen, wat de ietwat eigenaardige zin '10% 20% 30% 40% 50% 10% 20% 30% 40% 50% 60% 3% 70% 80%' oplevert.

Daarom staat voor schermlezers bovenaan een korte beschrijving van de pagina. Deze wordt met behulp van css buiten het scherm neergezet, zodat de lay-out niet wordt verstoord.

Schermlezers lezen deze tekst gewoon voor, ook al zie je die niet.

De eerste zin is in een `<h1>` gezet, omdat veel gebruikers van schermlezers niet gewoon met lezen beginnen, maar van `<h>` naar `<h>` 'springen' om een snel overzicht van de pagina te krijgen. Ook in dat geval wordt de korte beschrijving gevonden.

(Overigens kunnen ook spiders van zoekmachines diagrammen niet lezen, dus ook daarvoor is dit van belang.)

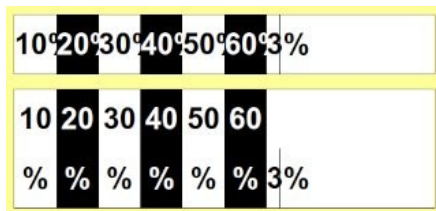
```
<div id="een" role="img" aria-label="Eerste diagram: voor 10%
gevuld met zwarte achtergrond"><span aria-
hidden="true">10&#8203;%</span></div>
```

Elk diagram staat in een eigen <div>. Elke <div> heeft het attribuut `role="img"` en een `aria-label` met een korte omschrijving van het diagram gekregen. Dit zorgt ervoor dat de tekst binnen `aria-label` wordt voorgelezen door schermlezers. Op het scherm is hiervan niets te zien.

Omdat voorlezen van het percentage dubbelop zou zijn, wordt het percentage met behulp van `aria-hidden="true"` voor schermlezers verborgen. Op het scherm wordt het percentage gewoon weergegeven.

Meer hierover is te vinden bij [WAI-ARIA-codes](#).

```
<span aria-hidden="true">10&#8203;%</span>
```



Als de tekst wordt vergroot tot meer dan ongeveer 140%, wordt de tekst te breed voor het eerste en het zesde diagram, want het diagram wordt niet breder. Hierdoor verdwijnt het procentteken grotendeels. Dit is op de bovenste afbeelding te zien. (In de andere diagrammen gebeurt dit niet, want daar is genoeg ruimte.)

Je zou het diagram ook breder kunnen maken bij een grotere lettergrootte, maar in smallere browservensters zou je dan horizontaal moeten scrollen om het hele diagram te zien.

Tussen het getal en het procentteken is de utf-8-code `​` neergezet: een spatie zonder breedte (de officiële naam is 'zero width space'). Je ziet niets van deze spatie, omdat die geen breedte heeft. Maar omdat het een spatie is, zijn het getal en het procentteken nu twee aparte woorden, net als twee gewone losstaande woorden. Als ze niet meer op één regel passen, komen ze op twee regels te staan. Dat is wat op de onderste afbeelding gebeurt. Misschien niet heel erg mooi, maar er verdwijnt in ieder geval niets.

De spatie staat tussen `&` en `;`. Daarmee geef je in html het begin en eind van een 'entiteit', een apart teken aan, zoals `ä` voor een ä. Lang niet voor alle mogelijke tekens bestaat zo'n makkelijk te onthouden 'afkorting' als 'auml', daarvoor zijn het er gewoon te veel. Daarom wordt voor de spatie zonder breedte een utf-8-code gebruikt. utf-8 is een standaard, waarin alle tekens van alle talen ter wereld, symbolen, dominostenen, smileys, noem maar op een eigen code hebben gekregen. Ook de spatie zonder breedte heeft een eigen code. `&` en `;` geven weer begin en einde van de entiteit aan. De `#` geeft aan, dat hier een gewoon decimaal getal op volgt (je kunt utf-8-codes ook met hexadecimale, zestientallige, getallen definiëren). `8203` is de decimale code voor een spatie zonder breedte.

`10​` is dus niets anders dan gewoon '10%', met een onzichtbare spatie ertussen.

(Als je niet `8203`, maar `0032` (of `32`) zou gebruiken, zou je 'n gewone spatie zien, want `32` is de code voor een gewone spatie.)

CSS

De code is geschreven in een afwijkende lettersoort. De code die te maken heeft met de basis van dit voorbeeld (essentiële code) is in de hele uitleg **vet blauw**. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles in een gewone letter vanwege de leesbaarheid.) Omdat deze site nou eenmaal (voornamelijk) op css is gericht, wordt hieronder álle css besproken.

Technisch gezien is er geen enkel bezwaar om de css in de stylesheet allemaal achter elkaar op één regel te zetten:

```
div#header-buiten {position: absolute; right: 16px;
width: 100%; height: 120px; background: yellow;} div p
{margin-left 16px; height: 120px; text-align: center;}
```

Maar als je dat doet, garandeer ik je hele grote problemen, omdat het volstrekt onoverzichtelijk is. Beter is het om de css netjes in te laten springen:

```
div#header-buiten {
    position: absolute;
    right: 16px;
    width: 100%;
    height: 120px;
    background: yellow;
}

div p {
    margin-left: 16px;
    height: 120px;
    text-align: center;
}
```

Hiernaast is het heel belangrijk voldoende commentaar (uitleg) in de stylesheet te schrijven. Op dit moment weet je waarschijnlijk (hopelijk...), waarom je iets doet. Maar over vijf jaar kan dat volstrekt onduidelijk zijn. Op deze site vind je nauwelijks commentaar in de stylesheets, maar dat heeft een simpele reden: deze uitleg is in feite één groot commentaar. Op internet zelf is het goed, als de stylesheet juist zo klein mogelijk is. Dus voor het uploaden kun je normaal genomen het beste het commentaar weer verwijderen. Veel mensen halen zelfs alles wat overbodig is weg, voordat ze de stylesheet uploaden. Inspringingen bijvoorbeeld zijn voor mensen handig, een computer heeft ze niet nodig. Je hebt dan eigenlijk twee stylesheets. De uitgebreide versie waarin je dingen uitprobeert, verandert, enz., met commentaar, inspringingen, e.d. Dat is de mensvriendelijke versie. Daarnaast is er dan een stylesheet die je op de echte site gebruikt: een gecomprimeerde versie.

Dat comprimeren kun je met de hand doen, maar er bestaan ook hulpmiddelen voor. Op de pagina met [links](#) kun je onder het kopje Gereedschap → Snelheid, testen, gzip, comprimeren (inclusief theorie) links naar sites vinden, waar je bestanden kunt comprimeren. (Stylesheets op deze site zijn niet gecomprimeerd. Omdat het vaak juist om de css gaat, kunnen mensen dan zonder al te veel moeite de css bekijken.)

```
/* figuren-060-d1.css */
```

Om vergissingen te voorkomen is het een goede gewoonte bovenaan het stijlbestand even de naam neer te zetten. Voor je het weet, zit je anders in het verkeerde bestand te werken.

body

Het element waarbinnen de hele pagina staat. Veel instellingen die hier worden opgegeven, worden geërfd door de nakomelingen van `<body>`. Ze gelden voor de hele pagina, tenzij ze later worden gewijzigd. Dit geldt bijvoorbeeld voor de lettersoort, de lettergrootte en de voorgrondkleur.

```
background-color: #ff9;
```

Achtergrondkleurtje.

```
color: black;
```

Voorgrondkleur zwart. Dit is o.a. de kleur van de tekst.

Hoewel dit de standaardkleur is, wordt deze toch specifiek opgegeven. Hierboven is een achtergrondkleur opgegeven. Sommige mensen hebben zelf de voorgrond- en/of achtergrondkleur veranderd, bijvoorbeeld omdat ze slecht kleuren kunnen onderscheiden. Als nu de achtergrondkleur wordt veranderd, maar niet de voorgrondkleur, loop je het risico dat tekstkleur en achtergrondkleur te veel op elkaar gaan lijken.

Door beide op te geven, is redelijk zeker dat achtergrond- en tekstkleur genoeg van elkaar blijven verschillen. Als de gebruiker `!important` heeft gebruikt in een eigen stylesheet, is er nog niets aan de hand, want dan veranderen achtergrond- en voorgrondkleur geen van beide.

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

Als Arial is geïnstalleerd op de machine van de bezoeker, wordt deze gebruikt, anders Helvetica. Als die ook niet wordt gevonden, wordt in ieder geval een schreefloze letter (zonder dwarsstreepjes) gebruikt.

```
font-size: 110%;
```

Iets groter dan standaard. 't Zal de leeftijd zijn, maar ik vind de standaardgrootte wat te klein.

Als eenheid wordt de relatieve eenheid `%` gebruikt, omdat bij gebruik van een absolute eenheid zoals `px` niet alle browsers de lettergrootte kunnen veranderen.

Zoomen kan wel altijd, ongeacht welke eenheid voor de lettergrootte wordt gebruikt.

```
margin: 0;
```

Slim om te doen vanwege verschillen tussen browsers.

main

Alle `<main>`'s. Dat is er maar een. Hierbinnen staat de belangrijkste inhoud van de pagina. In dit voorbeeld is dat de hele pagina.

```
width: 500px;
```

Breedte.

Als eenheid wordt de absolute eenheid `px` gebruikt. Dat wil zeggen dat de breedte altijd hetzelfde blijft en niet mee verandert met een andere lettergrootte.

```
max-width: 90vw;
```

Hier gelijk boven is een breedte van 500 px opgegeven. In browservensters die smaller zijn dan 500 px, zou je daardoor horizontaal moeten scrollen om het hele diagram te zien. Daarom wordt hier een maximumbreedte opgegeven.

De eenheid vw is gebaseerd op de breedte van het venster van de browser. 1 vw is 1% van de breedte van het venster, en 90 vw is 90% van de breedte. <main> wordt hierdoor nooit breder dan 90% van de breedte van het venster, ongeacht de breedte van het venster. Hierdoor wordt ook alles wat binnen <main> zit niet breder dan 90% van de breedte van het venster.

```
margin: 0 auto;
```

Omdat voor onder en links geen waarden zijn opgegeven, krijgen die automatisch dezelfde waarde als boven en rechts. Hier staat dus eigenlijk 0 auto 0 auto in de volgorde boven – rechts – onder – links.

Boven en onder geen marge.

Links en rechts auto, wat hier hetzelfde betekent als evenveel. Hierdoor komt <main> altijd horizontaal gecentreerd binnen z'n ouder <body>. <body> is een blok-element en wordt daardoor normaal genomen automatisch even breed als z'n ouder <html>. Omdat <html> het buitenste element is, wordt dit normaal genomen even breed als het venster van de browser.

Hierdoor staat uiteindelijk <main> altijd horizontaal gecentreerd binnen het venster van de browser, ongeacht de breedte van het venster. En daarmee ook de in <main> zittende diagrammen.

Deze manier van horizontaal centreren van een blok-element werkt alleen, als het blok-element een breedte heeft, want anders zou het normaal genomen even breed worden als de ouder ervan. Dat is geregeld, want bij hierboven heeft <main> een breedte van 500 px en een maximumbreedte van 90 vw gekregen.

```
h1, h1 + p
```

Twee selectors, gescheiden door een komma. De eerste selector h1:

h1: alle <h1>'s. Dat is er maar een.

De tweede selector h1 + p:

h1: alle <h1>'s. Dat is er maar een.

+: het element achter de + moet in de html direct volgen op het element voor de +. In dit geval gaat het om <p>'s gelijk op een <h1> volgen. Beide elementen moeten ook nog dezelfde ouder hebben. Dat is hier het geval, ze hebben beide als ouder <main>.

p: alle <p>'s die gelijk op een <h1> volgen.

De hele tweede selector in gewone taal: de <p> die in de html gelijk op de <h1> volgt. 'De' <p>, want er is maar één <h1>, dus er kan ook maar één <p> gelijk daarop volgen.

In deze <h1> en <p> zit een korte beschrijving van de pagina voor [schermlezers e.d.](#) Meer hierover is te vinden bij [<h1>...](#) en [<p>...](#).

```
position: absolute;
```

Om de elementen op een bepaalde plaats neer te kunnen zetten.

Er wordt gepositioneerd ten opzichte van het 'containing block'. Dat is de eerste voorouder die zelf een bepaalde eigenschap heeft, zoals een andere positie dan statisch. Als zo'n voorouder er niet is, zoals hier het geval is, wordt gepositioneerd ten opzichte van het venster van de browser.

```
left: -20000px;
```

Ver links buiten het scherm neerzetten. De tekst in de <h1> en de <p> wordt gewoon voorgelezen door schermlezers, maar is niet te zien op het scherm.

main div

Alle <div>'s binnen <main>. Acht <div>'s met in elk een diagram. Vrijwel alle eigenschappen voor deze <div>'s zijn hetzelfde. Ze kunnen daardoor hier in een keer voor alle <div>'s worden opgegeven.

background-color: white;

Witte achtergrondkleur.

De achtergronden staan binnen 's. Als dat nodig is, wordt de achtergrondkleur van een later aangepast, waardoor het wit van de <div> niet meer te zien is.

margin-top: 10px;

Kleine marge aan de bovenkant voor wat afstand tussen de <div>'s.

border: black solid 1px;

Zwart randje. Dit is eigenlijk alleen om de breedte van de <div> goed te laten zien.

Die breedte staat voor 100%, de achtergrond van de diagrammen vult een wisselend percentage daarvan.

div > span

div: alle <div>'s. In dit voorbeeld zijn alleen <div>'s aanwezig, waarbinnen een diagram zit.

>: de elementen achter dit teken moeten een direct kind van het element voor dit teken zijn. De hierachter staande moet een direct kind van een <div> zijn.

Onderstaande is een direct kind van de <div>:

```
<div>
    <span></span>
</div>
```

De hieronder is geen direct kind van de <div>, omdat er een tussen de <div> en de zit:

```
<div>
    <em>
        <span></span>
    </em>
</div>
```

span: alle 's binnen een <div> die een direct kind van die <div> zijn.

De hele selector in gewone taal: doe iets met de 's die een direct kind van een <div> zijn.

In een aantal diagrammen staat het percentage in een aparte , die genest is binnen een andere :

```
<div>
    <span>
        <span>50%</span>
    </span>
</div>
```

De 's met het percentage vallen niet onder deze selector, omdat ze geen direct kind van de <div> zijn.

De ``'s die wel een direct kind van de `<div>` zijn, hebben voor een deel dezelfde css. Die kan hier in één keer voor allemaal worden opgegeven. Als het nodig is, wordt dat later weer aangepast voor individuele ``'s.

background-color: black;

Achtergrondkleur zwart.

color: white;

Voorggrondkleur wit. Dit is o.a. de kleur van de tekst. Hier gelijk boven is de achtergrondkleur zwart gemaakt, en een zwarte letter op een zwarte achtergrond leest wat lastig. Vandaar.

display: inline-block;

Een `` is van zichzelf een inline-element. Hierdoor zijn eigenschappen als breedte en hoogte niet te gebruiken. Een inline-block is een soort kruising tussen een inline- en een blok-element. Het komt niet op een nieuwe regel te staan, maar eigenschappen als hoogte en breedte zijn wel te gebruiken.

font-weight: bold;

Vette letter.

Als de gebruikte lettersoort een vette variant heeft, wordt die gebruikt. De meeste lettersoorten hebben wel een vette variant, soms zelfs meerdere. Een vette variant wordt speciaal ontworpen, het is iets anders dan 'alle lijntjes wat dikker maken'. Sommige letters kunnen er zelfs heel anders uitzien.

Als de lettersoort geen vette variant heeft, maakt de browser de letter vet. Dat wil zeggen dat de browser gewoon alle lijntjes wat dikker maakt. Dat kan mooi zijn, maar vaak is het foeilelijk. Het is heel iets anders dan een speciaal ontworpen font. Elke rechtgeaarde typograaf gruwet hiervan.

line-height: 2rem;

Regelhoogte.

Omdat geen gewone hoogte voor de ``'s is opgegeven, is dit ook gelijk de hoogte van de ``'s. Tekst wordt automatisch halverwege de regelhoogte neergezet, dus de tekst staat gelijk netjes verticaal gecentreerd binnen de ``'s, en daarmee binnen het diagram.

Als eenheid wordt de relatieve eenheid `rem` gebruikt, omdat bij gebruik van een absolute eenheid zoals `px` de hoogte van de ``'s niet mee verandert met de lettergrootte. Zoomen kan wel altijd, ongeacht welke eenheid wordt gebruikt. De minder bekende `rem` is ongeveer hetzelfde als de `em`. Alleen is de lettergrootte bij `rem` gebaseerd op de lettergrootte van het `<html>`-element, waardoor de `rem` overal op de pagina precies even groot is, ook als de bezoeker de lettergrootte heeft veranderd. Bij de `em` kan de lettergrootte worden beïnvloed door de voorouders van het element, bij de `rem` niet.

Dat is hier van belang, omdat later bij het eerste en zesde diagram in heel smalle browservensters de lettergrootte wordt verkleind. De `em` is gebaseerd op de lettergrootte van het element zelf. Als de hoogte van de `` in `em` zou worden uitgedrukt, wordt de hoogte van de `` bij een kleinere letter lager. Nu heeft de lettergrootte van de `` geen invloed op de hoogte van de ``.

text-align: center;

Tekst horizontaal centreren. De ``'s variëren in breedte, maar de tekst wordt nu altijd horizontaal gecentreerd binnen de ``, ongeacht de breedte van de ``.

div span span

Alle ``'s die binnen een andere `` zitten, die weer binnen een `<div>` zit.

In een aantal diagrammen staat het percentage in een `` die binnen een andere `` zit. De css hier is voor die ``'s bedoeld.

background-color: black;

Zwarte achtergrond.

color: white;

Voorgrondkleur. Dit is o.a. de kleur van de tekst.

Deze ``'s zitten allemaal genest binnen een andere ``. Die andere `` heeft bij [div > span](#) ook al een witte voorgrondkleur gekregen, en een voorgrondkleur wordt – anders dan een `background-color` – geërfd door de nakomelingen. Dus eigenlijk is dit hier niet nodig.

Sommige mensen hebben zelf de voorgrond- en/of achtergrondkleur veranderd, bijvoorbeeld omdat ze slecht kleuren kunnen onderscheiden. Als nu de achtergrondkleur wordt veranderd, maar niet de voorgrondkleur, loop je het risico dat tekstkleur en achtergrondkleur te veel op elkaar gaan lijken. Daarom is het een goede gewoonte altijd beide kleuren op te geven, ook als dat niet strikt noodzakelijk is. Dan loop je bij een latere wijziging van de css of de html niet het risico, dat de voorgrondkleur niet meer genoeg contrast met de achtergrondkleur heeft.

`border: white solid 1px;`

Wit randje.

`padding: 1px 2px;`

Omdat voor onder en links geen waarden zijn opgegeven, krijgen die automatisch dezelfde waarde als boven en rechts. Hier staat dus eigenlijk `1px 2px 1px 2px` in de volgorde boven – rechts – onder – links. Kleine ruimte aan alle kanten tussen de witte rand om en het percentage in de ``'s.

#een

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

`main div {background-color: white; margin-top: 10px; border: black solid 1px;}`



Het element met id="een". De `<div>` met het eerste diagram.

padding: 2px;

Kleine afstand tussen de `` binnen de `<div>` en de buitenkant van de `<div>`.

#een span

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

`div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}`

De ``'s binnen het element met id="een". Dat is er maar één: de `` die bij het eerste diagram hoort.

width: 10%;

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier `div#een`. Een `<div>` is een blok-element en wordt daarom normaal genomen even breed als z'n ouder `<main>`, die bij [main](#) 500 px breed is gemaakt met een maximumbreedte van 90 vw, 90% van het venster van de browser.

`div#een` wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het venster. De 10% breedte van de `` geldt ten opzichte van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is). Omdat % een relatieve eenheid is, zal de `` altijd in verhouding dezelfde breedte ten opzichte van de `<div>` hebben: 10%. Ongeacht de breedte van de `<div>` blijven de verhoudingen altijd hetzelfde.

font-size: min(4vw, 1em);

In heel smalle browservensters, zoals op heel kleine smartphones, is in sommige browsers de lettergrootte net iets te groot. Niet in alle browsers, er zitten heel kleine variaties in de lettergrootte. Hierdoor komt het procentteken op de volgende regel te staan, omdat de gelijk hierboven aan de `` gegeven breedte van 10% net te smal is om '10%' op één regel te zetten.

Met de `min()`-functie kun je twee maten opgeven, waarvan de browser dan de kleinste maat gebruikt. De twee maten worden door een komma van elkaar gescheiden.

Voor de komma staat `4vw`. Dit is de maat die is bedoeld voor heel smalle browservensters. De eenheid `vw` is gebaseerd op de breedte van het venster. 1 `vw` is 1% van de breedte van het venster, en 4 `vw` is 4% van de breedte. In heel smalle vensters wordt de letter hierdoor iets verkleind, waardoor '4%' net op één regel past. Zodra het browservenster (veel) breder wordt, levert een lettergrootte van 4 `vw` een (veel) te grote letter op. Als het venster bijvoorbeeld 1000 px breed is, wordt de lettergrootte 4% van 1000 = 40 px. In nog bredere vensters zou de letter nog (veel) groter worden. Het probleem is dan opgelost in heel smalle vensters, maar nu past '10%' in bredere vensters niet meer in de ``. Het beleid van Rutte zeggend: we lossen het probleem niet op, maar schuiven het vriendelijk glimlachend door naar de volgende generatie.

Als een browservenster ongeveer 400 px breed is, is de lettergrootte met 4 `vw` ongeveer 16 px. Dat is de standaardlettergrootte. Groter mag de lettergrootte in dit voorbeeld niet worden. 1 `rem` is de standaard lettergrootte: 16 px. Zodra het venster breder wordt dan ongeveer 400 px, zal 1 `rem` minder zijn dan 4 `vw`. In dat geval is 1 `rem` de kleinste maat en wordt die gebruikt.

Als eenheid wordt de relatieve eenheid `em` gebruikt, omdat bij gebruik van een absolute eenheid zoals `px` niet alle browsers de lettergrootte kunnen veranderen. Nu kan de bezoeker nog steeds zelf de lettergrootte aanpassen. Zoomen kan wel altijd, ongeacht welke eenheid voor de lettergrootte wordt gebruikt.

#binnen-twee

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

```
div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}
```



Het element met `id="binnen-twee"`. De `` die bij het tweede diagram hoort.

width: 20%;

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier de `div` met het tweede diagram. Een `<div>` is een blok-element en wordt daarom normaal genomen even breed als z'n ouder `<main>`, die bij [main](#) 500 px breed is gemaakt met een maximumbreedte van 90 `vw`, 90% van het venster van de browser.

De <div> met het tweede diagram wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het browservenster. De 20% breedte van de geldt ten opzicht van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is).

Omdat % een relatieve eenheid is, zal de altijd in verhouding dezelfde breedte ten opzichte van de <div> hebben: 20%. Ongeacht de breedte van de <div> blijven de verhoudingen altijd hetzelfde.

#binnen-drie

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

```
div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}
```



Het element met id="binnen-drie". De die bij het derde diagram hoort.

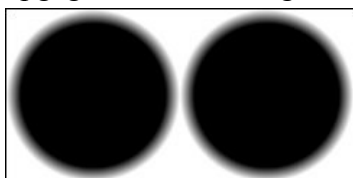
background-color: white;

Witte achtergrondkleur.

background-image: radial-gradient(black 5px, transparent 6px);

Achtergrond-afbeelding.

Vaak zul je een gewone afbeelding als achtergrond nemen, maar een gradiënt kan ook als achtergrond-afbeelding worden gebruikt. De browser maakt de gradiënt zelf, dus een url met een adres is niet nodig. De gradiënt gedraagt zich net zo als een gewone achtergrond-afbeelding. Omdat voor background-repeat niets wordt opgegeven, wordt de gradiënt herhaald tot de hele is gevuld.



Op de afbeelding zijn twee gradiënts te zien, tien keer vergroot. (Omdat het wit wat moeilijk is te zien, is op de afbeelding om de gradiënts een zwart randje gezet.)

Hier gelijk onder wordt de gradiënt met background-size 12 px breed en hoog gemaakt. De afbeelding

hiernaast bestaat dus eigenlijk uit twee vierkanten van elk 12 x 12 px.

Bij een gradiënt veranderen de kleuren. Die verandering kan scherp begrensd zijn of geleidelijk aan verlopen. Bij een radiale gradiënt zijn de grenzen tussen de kleuren min of meer rond. Je kunt er bijvoorbeeld een ellips mee maken, waarbij de kleuren van binnen naar buiten veranderen. Hier wordt een cirkel gemaakt.

Je kunt met behulp van het sleutelwoord circle opgeven dat de gradiënt een cirkel moet worden, en ook waar het middelpunt van die cirkel komt te liggen. Hier is dat niet nodig. Als je niets opgeeft, begint een gradiënt in het midden en groeit van daaruit naar buiten. De gradiënt wordt gelijk hieronder 12 px breed en hoog gemaakt, dus het midden ligt op 6 px vanaf alle kanten. Omdat de gradiënt vanuit dat midden aan alle kanten evenveel naar buiten groeit, krijg je automatisch een cirkel (Zou je hieronder niet 12 x 12 px hebben opgegeven, maar bijvoorbeeld 20 x 12 px, dan zou je een ellips krijgen. Om dan toch een cirkel te krijgen, zou je wel het sleutelwoord circle moeten gebruiken.)

Deze gradiënt is heel simpel: er worden maar twee kleuren gebruikt, gescheiden door een komma.

De eerste kleur black 5px begint in het middelpunt en wordt dan 5 px naar alle kanten uitgebreid. Dit levert de zwarte cirkel op, die een diameter van 10 px krijgt (vanaf het middelpunt 5 px naar alle kanten). Omdat dit de eerste kleur is, hoeft je geen beginpunt op te geven: het beginpunt is automatisch het middelpunt.

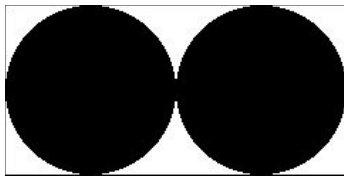


De tweede kleur is transparent 6px. Dit is dus eigenlijk geen kleur, het is doorzichtig. Het wit dat je ziet is de iets hierboven opgegeven witte achtergrondkleur van de zelf. Op de afbeelding heeft de een rode achtergrondkleur, en nu zie je door het transparent de rode achtergrond van de .

De 6px geeft aan dat deze 'kleur' op 6 px vanaf het middelpunt moet beginnen.

Omdat dit de laatste kleur is, hoef je geen eindpunt op te geven: de kleur loopt door tot de buitenkant van de gradiënt. (Dat het begin- en eindpunt van de doorzichtige kleur hier hetzelfde zijn, maakt niets uit. Een computer denkt zeer rechtlijnig...)

De zwarte kleur loopt door tot 5 px vanaf het middelpunt, en het doorzichtige deel begint pas op 6 px. Er zit dus een 'kier' van 1 px tussen beide kleuren. Omdat het om een gradiënt gaat, verandert het zwart over die ene pixel geleidelijk aan in doorzichtig.



Op de afbeelding hiernaast is niet transparent 6px maar transparent 5px gebruikt: het doorzichtige deel begint gelijk waar het zwarte deel eindigt, een harde grens. Op de tien keer vergrote afbeelding is al te zien dat dit niet echt heel mooi is, zo'n

harde overgang. Op de werkelijke grootte is dit nog veel lelijker. Die kleine overgang van 1 px maakt de cirkels veel minder blokkerig en laat ze – in ieder geval voor het oog – beter op elkaar aansluiten.

background-size: 12px 12px;

De gradiënt 12px breed en hoog maken.

span#binnen-drie heeft eerder een hoogte van 2 em gekregen en krijgt gelijk hieronder een breedte van 30% van de <div>, waar de in zit. Hier krijgt de gradiënt een breedte en hoogte van 12 px. Net als een gewone achtergrond-afbeelding wordt de gradiënt herhaald, tot de hele is gevuld.

width: 30%;

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier de <div> met het derde diagram. Een <div> is een blok-element en wordt daarom normaal genomen even breed als z'n ouder <main>, die bij [main](#) 500 px breed is gemaakt met een maximumbreedte van 90 vw, 90% van het venster van de browser.

De <div> met het derde diagram wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het browservenster. De 30% breedte van de geldt ten opzicht van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is).

Omdat % een relatieve eenheid is, zal de altijd in verhouding dezelfde breedte ten opzichte van de <div> hebben: 30%. Ongeacht de breedte van de <div> blijven de verhoudingen altijd hetzelfde.

#binnen-vier

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

```
div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}
```



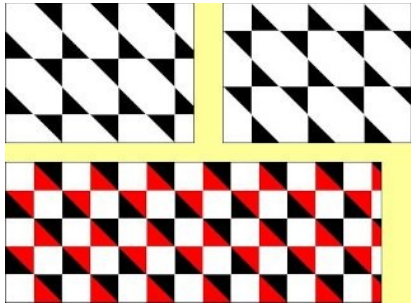
Het element met id="binnen-vier". De die bij het vierde diagram hoort.

background-color: white;

Witte achtergrond.

```
background-image: linear-gradient(45deg, black 25%,
transparent 25%, transparent 75%, black 75%), linear-
gradient(45deg, black 25%, transparent 25%, transparent
75%, black 75%);
```

Hier worden twee gradiënts gebruikt als achtergrond-afbeelding. Ook dit werkt precies hetzelfde als bij gewone achtergrond-afbeeldingen: de gradiënts worden over elkaar heen gezet.



Op de bovenste twee afbeeldingen zijn de twee gradiënts apart weergegeven, tien keer vergroot. Linksboven staat alleen de eerste gradiënt een aantal keren, rechtsboven alleen de tweede gradiënt. Elk vierkant blokje is een gradiënt. Net als een gewone achtergrond-afbeelding worden deze weer herhaald, tot de hele achtergrond van de `` is gevuld.

Beide gradiënts zijn eigenlijk precies hetzelfde. Ze zijn opgebouwd uit zwart afgewisseld met doorzichtig. Dit levert horizontale zwarte balken op, afgewisseld met doorzichtige horizontale balken.

Door de doorzichtige kleur zie je de witte achtergrondkleur van de ``. Wat op de afbeeldingen wit is, is dus eigenlijk doorzichtig.

De richting van de gradiënts wordt vervolgens 45 graden gedraaid, waardoor de horizontale balken veranderen in diagonale balken. Iets hieronder worden de gradiënts 6 x 6 px groot gemaakt, waardoor het grootste deel van de zwarte balken wegvalt, omdat dat er niet meer in past.

Gelijk hieronder wordt de tweede gradiënt 3 px naar onder en naar rechts verplaatst. Omdat beide gradiënts 6 px breed en 6 px hoog zijn, komt het zwart van de tweede gradiënt nu precies tegen het zwart van de eerste gradiënt aan te staan. En omdat beide naar een diagonale richting zijn gedraaid, levert dit vierkante zwarte blokjes op.

Op de onderste afbeelding is het resultaat te zien. Het zwarte driehoekje hoort bij de eerste gradiënt, het rode driehoekje bij de tweede. Waar beide gradiënts geen zwart hebben, zijn ze doorzichtig en zie je de witte achtergrondkleur van de ``.

De gradiënts zijn precies hetzelfde, alleen de positie van de tweede gradiënt wordt hier gelijk onder aangepast. Als we de draaiing even weglaten, is de gradiënt:

```
linear-gradient(black 25%, transparent 25%,
transparent 75%, black 75%);
```



Dit levert de gradiënt op de afbeelding op. (Omdat de gradiënts heel klein zijn, staan op de afbeelding meerdere gradiënts naast elkaar.)

Omdat geen richting is opgegeven, loopt de gradiënt in de standaardrichting van boven naar beneden.

De eerst opgegeven kleur is `black 25%`. Omdat geen andere beginkleur is opgegeven, begint zwart bij 0%: helemaal bovenaan. En loopt door tot 25%.

De tweede kleur is `transparent 25%`. Deze begint bij 25%, precies waar zwart eindigt. Hierdoor krijg je een scherp begrensde grens tussen zwart en doorzichtig.

De derde kleur is `transparent 75%`. Het doorzichtige deel loopt van 25% tot 75%. (Omdat het doorzichtig is, zie je weer de witte achtergrondkleur van de `` erdoorheen.)

De laatste kleur is black 75%. Zwart begint precies waar doorzichtig is geëindigd: weer een scherpe overgang. Omdat verder niets is opgegeven, loop zwart door tot het eind: 100%.

Het eindresultaat: een horizontale zwarte balk over 25% van de hoogte, een doorzichtig deel over de volgende 50%, en onderaan weer een zwarte balk over 25% van de hoogte.



Het enige dat nu nog hoeft te gebeuren is het toevoegen van een draaiing. Op de afbeelding staat nog steeds vijf keer dezelfde gradiënt, maar nu met een toenemende draaiing. De eerste afbeelding is 9 graden gedraaid, de tweede 18 graden, de derde 27 graden, de vierde 36 graden, en de vijfde uiteindelijk zoals die is gebruikt: met 45 graden:

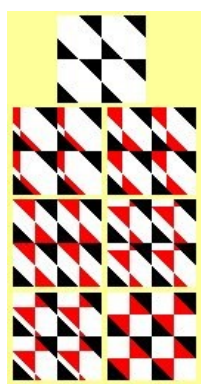
```
linear-gradient(45deg, black 25%, transparent 25%,  
transparent 75%, black 75%);
```

(Kleine vertekeningen op de afbeelding zijn ontstaan door de extreme vergroting.)

background-position: 0 0, 3px 3px;

Beide gradiënten zijn precies hetzelfde. Gelijk hieronder krijgen ze ook nog dezelfde maat. Alleen de positie van beide verschilt.

Als er meerdere achtergrond-afbeeldingen zijn, kun je daarvoor een verschillende positie opgeven, gescheiden door een komma. De maten voor de komma gelden dan voor de eerste, die achter de komma voor de tweede (en als er nog meer zouden zijn, volgen nog meer komma's en maten).



Op de zeven afbeeldingen hiernaast zijn op elke afbeelding acht gradiënten weergegeven: vier keer de eerste, en vier keer de tweede. De tweede gradiënt heeft een rode kleur gekregen, de eerste is zwart gebleven. Hierdoor kun je het verschil tussen de eerste en de tweede gradiënt zien.

Als er meerdere achtergrond-afbeeldingen zijn, komt de eerste afbeelding uit de html bovenop te staan. Het wit op de afbeelding is weer de witte achtergrond van de , die door het doorzichtige deel van de gradiënten zichtbaar is.

Op de afbeelding midden boven staan beide gradiënten op dezelfde positie. Daardoor is alleen de eerste gradiënt te zien, de zwarte, want die staat als eerste in de html. En komt dus over de tweede rode gradiënt heen te staan.

Die eerste zwarte gradiënt blijft steeds op dezelfde positie staan: 0 0. 0 px vanaf de bovenkant, 0 px vanaf de linkerkant.

Op de afbeelding linksboven is de tweede rode gradiënt met 1px 0 één pixel naar rechts verplaatst, waardoor deze gedeeltelijk zichtbaar wordt.

Op de afbeelding rechtsboven is de tweede rode gradiënt met 2px 0 twee pixels naar rechts verplaatst.

Op de afbeelding midden links is de tweede rode gradiënt met 3px 0 drie pixel naar rechts verplaatst. Dat is precies de helft van de breedte van de gradiënt. Nu staat het rode deel van de tweede gradiënt precies boven het zwarte deel van de eerste gradiënt.

Op de afbeelding midden rechts is de tweede rode gradiënt met 3px 1px drie pixel naar rechts en één pixel naar beneden verplaatst.

Op de afbeelding linksonder is de tweede rode gradiënt met 3px 2px drie pixel naar rechts en twee pixels naar beneden verplaatst. Dat begint er al op te lijken.

Op de afbeelding rechtsonder is de uiteindelijke positie bereikt: 3px 3px. De tweede rode gradiënt 3 px naar rechts en 3 px naar beneden verplaatst, de helft van

de breedte en hoogte van de gradiënt. Het rood van de tweede gradiënt sluit nu aan op het zwart van de eerste gradiënt: daar is het vierkantje.

background-size: 6px 6px;

Grootte van de achtergrond-afbeeldingen. De eerste maat is voor de breedte, de tweede voor de hoogte. Omdat er maar één breedte en hoogte is opgegeven, gelden deze voor beide gradiënten.

width: 40%;

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier de <div> met het vierde diagram. Een <div> is een blok-element en wordt daarom normaal genomen even breed als z'n ouder <main>, die bij [main](#) 500 px breed is gemaakt met een maximumbreedte van 90 vw, 90% van het venster van de browser.

De <div> met het vierde diagram wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het browservenster. De 40% breedte van de geldt ten opzicht van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is).

Omdat % een relatieve eenheid is, zal de altijd in verhouding dezelfde breedte ten opzichte van de <div> hebben: 40%. Ongeacht de breedte van de <div> blijven de verhoudingen altijd hetzelfde.

#binnen-vijf

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

```
div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}
```



Het element met id="binnen-vijf". De die bij het vijfde diagram hoort.

background-image: repeating-linear-gradient(-45deg, black, white 1px, white 5px, black 6px, black 10px);

Hoe je deze schuine strepen krijgt, is het best te zien als je de gradiënt stap voor stap maakt. Eerst een gewone lineaire gradiënt zonder draaiing:

```
linear-gradient(black, white 1px, white 5px, black 6px, black 10px);
```



Dit levert de hiernaast staande gradiënt op.

De zwarte rechthoek linksonder en de daarboven zittende witte rechthoek horen bij de gradiënt. De witte rechthoek en de zwarte rand aan boven-,

onder- en linkerkant zijn van de <div>, waar de met de gradiënt in zit. Die <div> is maar gedeeltelijk afgebeeld.

De heeft 50% van de breedte van de <div>. De gradiënt heeft dezelfde breedte als span#binnen-vijf: 50% van de breedte van de <div>. De verticale grens tussen de zwarte en witte rechthoek ligt precies op die 50%.

Boven de zwarte rechthoek is de rand aan de bovenkant iets dikker dan boven de witte rechthoek. Die verdikking hoort bij de gradiënt, de rand zelf hoort bij de <div>. Omdat geen richting of draaiing is opgegeven, loopt de gradiënt in de standaardrichting: van boven naar beneden.

linear-gradient bevat drie kleuren met vijf posities, gescheiden door een komma:

`black`: de eerste kleur is `black`. Omdat dit de eerste kleur is, begint deze gelijk aan het begin van de gradiënt, aan de bovenkant. Je zou ook `black 0` kunnen schrijven, waarbij de `0` voor `0px` staat, maar dit mag je weglaten bij de kleur die aan het begin staat.

Dit zwart vormt de kleine verdikking van de zwarte rand aan de bovenkant.

Dit dunne zwarte randje heeft hier geen enkel nut, maar het helpt voorkomen dat de uiteindelijke diagonale strepen blokkerig worden.

`white 1px, white 5px`: de tweede kleur. Wit begint op `1 px` vanaf de bovenkant en eindigt op `5 px` vanaf de bovenkant: van `1px` vanaf de bovenkant tot `5 px` vanaf de bovenkant is de kleur effen wit.

Omdat het een gradiënt is, gaat de gelijk hierboven opgegeven zwarte kleur helemaal bovenaan (op `0 px`) geleidelijk aan over in de witte kleur, die op `1 px` vanaf de bovenkant begint. Dat zie je hier nauwelijks, maar die kleine overgang van `1 px` zorgt ervoor dat het bij de uiteindelijke diagonale strepen minder snel blokkerig wordt.

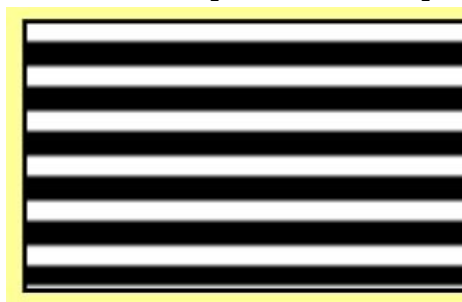
`black 6px, black 10px`: de derde kleur. Zwart begint op `6 px` vanaf de bovenkant en eindigt op `10 px` vanaf de bovenkant: van `6 px` vanaf de bovenkant tot `10 px` vanaf de bovenkant is de kleur effen zwart.

Zwart begint op `6 px` vanaf de bovenkant. Dat is `1 px` lager dan waar het gelijk hierboven opgegeven wit eindigt. Omdat het een gradiënt is, verloopt de overgang van wit naar zwart weer geleidelijk over die ene pixel. Ook dat zie je hier nauwelijks, maar het helpt weer bij het voorkomen van blokkerige diagonale strepen.

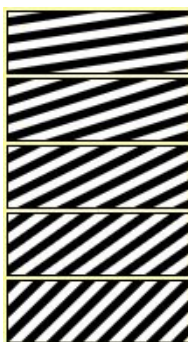
Omdat zwart de laatste kleur is, loopt die helemaal door tot aan de onderkant van de ``. Die laatste `black 10px` zou je daarom hier weg kunnen laten, maar die zijn straks wel nodig voor het uiteindelijke resultaat: de diagonale strepen.

Als `repeating` wordt toegevoegd, gaat het er heel anders uitzien:

```
repeating-linear-gradient(black, white 1px, white 5px, black 6px, black 10px);
```



Door `linear-gradient` te veranderen in `repeating-linear-gradient` wordt de gradiënt nu steeds herhaald, tot de volledige hoogte van de `` is opgevuld.



Ten slotte wordt de draaiing toegevoegd. De uiteindelijke draaiing is `-45deg`:

```
repeating-linear-gradient(-45deg, black, white 4px, white 5px, black 6px, black 10px);
```

Omdat voor `deg` een negatief getal staat, is de draaiing tegen de klok in. Op de afbeeldingen is de draaiing in verschillende stappen aangebracht, zodat is te zien, wat er gebeurt. Op de bovenste afbeelding is de draaiing `-9deg`, op de tweede `-18deg`, op de derde `-27deg`, op de vierde `-36deg` en op de laatste uiteindelijk `-45deg`.

width: 50%;

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier de `<div>` met het vijfde diagram. Een `<div>` is een blok-element en wordt daarom normaal genomen even breed als z'n ouder `<main>`, die bij [main](#) 500 px breed is gemaakt met een maximumbreedte van 90 vw, 90% van het venster van de browser.

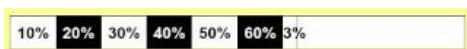
De `<div>` met het vijfde diagram wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het browservenster. De 50% breedte van de `` geldt ten opzicht van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is).

Omdat % een relatieve eenheid is, zal de `` altijd in verhouding dezelfde breedte ten opzichte van de `<div>` hebben: 50%. Ongeacht de breedte van de `<div>` blijven de verhoudingen altijd hetzelfde.

#zes span

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

```
div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}
```



De ``'s binnen het element met `id="zes"`. De ``'s die bij het zesde diagram horen.

width: 10%;

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier `div#zes`. Een `<div>` is een blok-element en wordt daarom normaal genomen even breed als z'n ouder `<main>`, die bij [main](#) 500 px breed is gemaakt met een maximumbreedte van 90 vw, 90% van het venster van de browser. `div#zes` wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het browservenster. De 10% breedte van de ``'s geldt ten opzicht van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is).

Omdat % een relatieve eenheid is, zullen de ``'s altijd in verhouding dezelfde breedte ten opzichte van de `<div>` hebben: 10%. Ongeacht de breedte van de `<div>` blijven de verhoudingen altijd hetzelfde.

font-size: min(4vw, 1em);

In heel smalle browservensters zijn getal plus procentteken in sommige browsers net te breed om op één regel te passen in deze smalle ``'s. Daarom wordt hier de lettergrootte beperkt tot 4 vw (vier procent van de browser van het venster) of 1 em (de standaardlettergrootte), afhankelijk van welke van de twee het kleinst is. Een uitgebreidere uitleg over de `min()`-functie staat bij [font-size: min\(4vw, 1em\);](#).

#zes span:nth-of-type(odd)

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

```
div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}
```

```
#zes span {width: 10%; font-size: min(4vw, 1em);}
```

Kinderen van dezelfde ouder kunnen worden geteld, net zoals dat bij kinderen uit een gezin het geval is: eerste kind, tweede kind, derde kind, enz. Je kunt ook alleen 'n bepaald soort element tellen, net zoals je alleen kinderen van jonger of ouder dan zes jaar kunt tellen. Met

de pseudo-class `:nth-of-type()`, onderdeel van deze selector, worden alleen elementen van een bepaald soort geteld.

`#zes`: het element met `id="zes"`. De `<div>` met het zesde diagram.

`span`: alle ``'s binnen `#zes`.

`:nth-of-type(odd)`: het element met een bepaald volgnummer. Het volgnummer staat tussen de haakjes. In dit geval is het 'volgnummer' geen getal, maar een sleutelwoord: `odd`. Alle oneven ``'s. Omdat vaak alleen de oneven (of alleen de even) elementen moeten worden geselecteerd, zijn er sleutelwoorden voor oneven en even.

Omdat voor `:nth-of-type(odd)` een `span` staat, worden alleen ``'s geteld. Als binnen `#zes 327 <p>`'s zitten, tellen die niet mee. Hadden ze maar 'n `` moeten zijn.

In deze ``'s zitten de oneven tientallen met percentages.

De hele selector in gewone taal: de oneven ``'s binnen het element met `id="zes"`.

background-color: white;

Eerder hebben bij [div > span](#) alle ``'s die een direct kind van een `<div>` zijn een zwarte achtergrond gekregen. Hier krijgen de oneven ``'s binnen `div#zes` een zwarte achtergrond. De even ``'s houden gewoon hun zwarte achtergrond. Hierdoor krijg je afwisselende zwarte en witte blokken met elk een breedte van tien procent van `div#zes`.

color: black;

Hier gelijk boven is de voorgrondkleur, waaronder de kleur van de tekst, wit gemaakt. Op een witte achtergrond leest dat wat lastig, daarom wordt de voorgrondkleur hier zwart gemaakt.

#zes span:last-of-type

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

[div > span](#) {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}

[#zes span](#) {width: 10%; font-size: min(4vw, 1em);}

[#zes span:nth-of-type\(odd\)](#) {background-color: white; color: black;}

Kinderen van dezelfde ouder kunnen worden geteld, net zoals dat bij kinderen uit een gezin het geval is: eerste kind, tweede kind, derde kind, enz. Je kunt ook alleen 'n bepaald soort element tellen, net zoals je alleen kinderen van jonger of ouder dan zes jaar kunt tellen. Met de pseudo-class `:nth-of-type()` worden alleen elementen van een bepaald soort geteld.

De pseudo-class `:nth-last-of-type()`, onderdeel van deze selector, werkt hetzelfde, maar telt van achter naar voren. Omdat het vaak voorkomt dat je het laatste element van 'n bepaald type wilt hebben, heeft dat een aparte pseudo-class, het in deze selector gebruikte `:last-of-type`.

`#zes`: het element met `id="zes"`.

`span`: alle ``'s binnen `#zes`.

`:last-of-type`: het element met een bepaald volgnummer, geteld van achter naar voren.

In dit geval wordt geen volgnummer gebruikt, maar een speciaal voor het laatste element bedoelde pseudo-class: `:last-of-type`. Voor alle eerdere elementen gebruik je 'n soortgelijke pseudo-class, maar dan met een volgnummer: `:nth-`

`last-of-type()`. Tussen de haakjes komt het volgnummer. `:nth-last-of-type(1)` is precies hetzelfde als `:last-of-type`, maar de laatste is wat mensvriendelijker.

Omdat voor `:last-of-type` (of voor eerdere elementen `:nth-last-of-type()`) een `span` staat, worden alleen ``'s geteld. Als binnen `div#zes` 327 `<p>`'s zitten, tellen die niet mee. Hadden ze maar 'n `` moeten zijn.

De hele selector in gewone taal: elke laatste `` binnen `div#zes`.

`width: 3%;`

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier `div#zes`. Een `<div>` is een blok-element en wordt daarom normaal genomen even breed als z'n ouder `<main>`, die bij [main](#) 500 px breed is gemaakt met een maximumbreedte van 90 vw, 90% van het venster van de browser. `div#zes` wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het venster. De 3% breedte van de `` geldt ten opzicht van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is). Omdat % een relatieve eenheid is, zal de `` altijd in verhouding dezelfde breedte ten opzichte van de `<div>` hebben: 3%. Ongeacht de breedte van de `<div>` blijven de verhoudingen altijd hetzelfde.

De zes eerdere ``'s in `div#zes` hebben elk een breedte van 10%. Dit diagram moet voor 63% gevuld zijn, daarom krijgt deze zevende en laatste `` een breedte van slechts 3%.

`border-right: black solid 1px;`

Zwarte border rechts.

Deze border geeft de grootte van drie procent aan. De tekst '3%' is hier wat te breed voor, waardoor deze ook iets aan de 'verkeerde' kant van de border komt te staan.

Je zou '3%' wel binnen de `` kunnen krijgen, maar dat vereist nogal wat extra css. Bovendien moet je de tekst dan ook fors verkleinen. Het is zo ook prima leesbaar, dus dit wordt gewoon zo gelaten.

`#binnen-zeven`

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

```
div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}
```



Het element met id="binnen-zeven". De `` die bij het zevende diagram hoort.

`background-color: white;`

Witte achtergrondkleur.

`background-image: url("../060-pics/smiley.gif");`

Hier is weinig bijzonders aan: gewoon een smiley als achtergrond-afbeelding. Dit is in dit voorbeeld de enige echte afbeelding die nog wordt gebruikt. Je zou zo'n smiley desnoods nog zelf kunnen maken met SVG of zelfs met css, maar dat wordt tamelijk ingewikkeld. Zeker als je de smiley ook wilt laten bewegen, zoals hier het geval is.

`width: 70%;`

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier de `<div>` met het zevende diagram. Een `<div>` is een blok-element en wordt daarom normaal genomen even breed als z'n ouder `<main>`, die bij

[main](#) 500 px breed is gemaakt met een maximumbreedte van 90 vw, 90% van het venster van de browser.

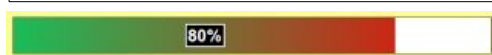
De <div> met het zevende diagram wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het browservenster. De 70% breedte van de geldt ten opzicht van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is).

Omdat % een relatieve eenheid is, zal de altijd in verhouding dezelfde breedte ten opzichte van de <div> hebben: 70%. Ongeacht de breedte van de <div> blijven de verhoudingen altijd hetzelfde.

#binnen-acht

Voor dit element is eerder css opgegeven. Deze wordt binnen dit blokje herhaald in de volgorde, waarin deze in de stylesheet staat, zodat alles hier overzichtelijk bij elkaar staat.

```
div > span {background-color: black; color: white; display: inline-block; font-weight: bold; line-height: 2em; text-align: center;}
```



Het element met id="binnen-acht". De die bij het achtste diagram hoort.

background-image: linear-gradient(to right, #1bbd5a, #cc2714);

Als achtergrond wordt weer een gradiënt gebruikt: een verlopende kleur.

linear-gradient valt in een aantal delen uiteen:

to right: de kant waar de gradiënt naartoe gaat. Omdat alleen rechts is opgegeven, loopt de gradiënt horizontaal van links naar rechts.

#1bbd5a: de eerste kleur is #1bbd5a (groenachtig). Omdat geen beginpunt is opgegeven, begint de gradiënt helemaal links met deze kleur. (Je kunt ook #1bbdf5a 0 opgeven, maar dat is precies hetzelfde.)

#cc2714: de tweede (en laatste) kleur is #cc2714 (roodachtig). Omdat geen eindpunt is opgegeven, eindigt de gradiënt helemaal rechts met deze kleur.)

De geleidelijke overgang van de kleur #1bbd5a helemaal links naar hier #cc2714 rechts wordt door de browser geregeld.

width: 80%;

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier de <div> met het achtste diagram. Een <div> is een blok-element en wordt daarom normaal genomen even breed als z'n ouder <main>, die bij [main](#) 500 px breed is gemaakt met een maximumbreedte van 90 vw, 90% van het venster van de browser.

De <div> met het achtste diagram wordt dus ook 500 px breed, met ook een maximumbreedte van 90% van het browservenster. De 80% breedte van de geldt ten opzicht van die 500 px (of 90% van de breedte van het venster, als het venster minder dan ongeveer 550 px breed is).

Omdat % een relatieve eenheid is, zal de altijd in verhouding dezelfde breedte ten opzichte van de <div> hebben: 80%. Ongeacht de breedte van de <div> blijven de verhoudingen altijd hetzelfde.

Volledige code

HTML

De code is geschreven in een afwijkende lettersoort. De code die te maken heeft met de basis van dit voorbeeld (essentiële code) is in de hele uitleg **vet blauw** gekleurd. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles in een gewone letter vanwege de leesbaarheid.)

```
<!doctype html>
<html lang="nl">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
    initial-scale=1">
  <meta name="description" content="Makkelijk aan te passen
    horizontale staafdiagrammen - voorbeeld. Gebruikt alleen
    html en css.">
  <title>Makkelijk aan te passen horizontale staafdiagrammen -
    voorbeeld</title>
  <link rel="stylesheet" href="060-css-dl/figuren-060-dl.css">
</head>
<body>
<main>
  <h1>Acht voorbeelden van een staafdiagram.</h1>
  <p>Buiten deze acht staafdiagrammen is deze pagina leeg. Het
    percentage dat is gevuld, wordt bij elk diagram met een
    andere achtergrond weergegeven.</p>
  <div id="een" role="img" aria-label="Eerste diagram: voor 10%
    gevuld met zwarte achtergrond"><span aria-
      hidden="true">10&#8203;%</span></div>
  <div role="img" aria-label="Tweede diagram: voor 20% gevuld
    met zwarte achtergrond"><span id="binnen-twee" aria-
      hidden="true">20%</span></div>
  <div role="img" aria-label="Derde diagram: voor 30% gevuld met
    zwarte stippen"><span id="binnen-drie"><span aria-
      hidden="true">30%</span></span></div>
  <div role="img" aria-label="Vierde diagram: voor 40% gevuld
    met kleine blokjes"><span id="binnen-vier"><span aria-
      hidden="true">40%</span></span></div>
  <div role="img" aria-label="Vijfde diagram: voor 50% gevuld
    met diagonale strepen"><span id="binnen-vijf"><span aria-
      hidden="true">50%</span></span></div>
```

```

<div id="zes" role="img" aria-label="Zesde diagram: voor 63%
    gevuld met afwisselende zwarte en witte blokken"><span
    aria-hidden="true">10&#8203;%</span><span aria-
    hidden="true">20&#8203;%</span><span aria-
    hidden="true">30&#8203;%</span><span aria-
    hidden="true">40&#8203;%</span><span aria-
    hidden="true">50&#8203;%</span><span aria-
    hidden="true">60&#8203;%</span><span aria-
    hidden="true">3%</span></div>
<div role="img" aria-label="Zevende diagram: voor 70% gevuld
    met smileys"><span id="binnen-zeven"><span aria-
    hidden="true">70%</span></span></div>
<div role="img" aria-label="Achtste diagram: voor 80% gevuld
    met een gradiënt"><span id="binnen-acht"><span aria-
    hidden="true">80%</span></span></div>
</main>
</body>
</html>

```

CSS

De code is geschreven in een afwijkende lettersoort. De code die te maken heeft met de basis van dit voorbeeld (essentiële code) is in de hele uitleg **vet blauw** gekleurd. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles in een gewone letter vanwege de leesbaarheid.)

```

/* figuren-060-dl.css */

body {
    background-color: #ff9;
    color: black;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 110%; margin: 0;
}

main {
    width: 500px;
    max-width: 90vw;
    margin: 0 auto;
}

h1, h1 + p {
    position: absolute;
    left: -20000px;
}

```

```
main div {
    background-color: white;
    margin-top: 10px;
    border: black solid 1px;
}

div > span {
    background-color: black;
    color: white;
    display: inline-block;
    font-weight: bold;
    line-height: 2rem;
    text-align: center;
}

div span span {
    background-color: black;
    color: white;
    border: white solid 1px;
    padding: 1px 2px;
}

#een {padding: 2px;}

#een span {
    width: 10%;
    font-size: min(4vw, 1em);
}

#binnen-twee {width: 20%;}

#binnen-drie {
    background-color: white;
    background-image: radial-gradient(black 5px, transparent 6px);
    background-size: 12px 12px;
    width: 30%;
}

#binnen-vier {
    background-color: white;
    background-image: linear-gradient(45deg, black 25%,
        transparent 25%, transparent 75%, black 75%),
```

```

        linear-gradient(45deg, black 25%, transparent 25%,
        transparent 75%, black 75%);
background-position: 0 0, 3px 3px;
background-size: 6px 6px;
width: 40%;
}

#binnen-vijf {
    background-image: repeating-linear-gradient(-45deg, black,
        white 1px, white 5px, black 6px, black 10px);
    width: 50%;
}

#zes span {
    width: 10%;
    font-size: min(4vw, 1em);
}

#zes span:nth-of-type(odd) {
    background-color: white;
    color: black;
}

#zes span:last-of-type {
    width: 3%;
    border-right: black solid 1px;
}

#binnen-zeven {
    background-color: white;
    background-image: url("../060-pics/smiley.gif");
    width: 70%;
}

#binnen-acht {
    background-image: linear-gradient(to right, #1bbd5a, #cc2714);
    width: 80%;
}

```