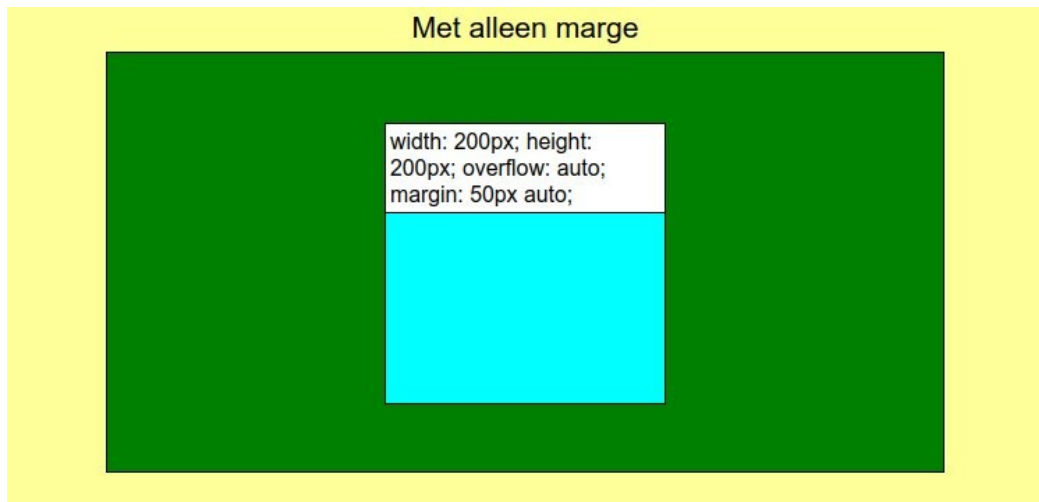


Horizontaal én verticaal centreren van blok-element met bekende hoogte en breedte



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](#)

[Toetsenbordnavigatie en tabindex](#)

[Muis, toetsenbord, touchpad en touchscreen](#)

[:hover](#)

[:focus](#)

[:active](#)

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

[Toegankelijkheid en zoekmachines](#)

[Getest in](#)

[Bekende problemen \(en oplossingen\)](#)

[Zonder JavaScript](#)

[Zonder css](#)

[Toetsenbordnavigatie](#)

[Tekstbrowsers](#)

[Schermlezers](#)

[Zoomen en andere lettergrootte](#)

[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="068-css-dl/positioneren-068-dl.css">](#)

[\(...\)](#)

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

[css voor alle vensters:](#)

[Algemene lay-out:](#)

[.buiten](#) {width: 600px; height: 300px;}

[#bekend .binnen, #combinatie .binnen](#) {width: 200px; height: 200px; overflow: auto;}

[Binnenste <div> heeft een vaste breedte en hoogte van 200 px:](#)

[#alleen-marge .binnen](#) {margin: 50px auto;}

[#marge-padding .buiten](#) {box-sizing: border-box; padding-top: 50px;}

[#marge-padding .binnen](#) {margin: 0 auto;}

[#absolute-negatieve-marge .buiten](#) {position: relative;}

[#absolute-negatieve-marge .binnen](#) {margin: -100px 0 0 -100px; position: absolute; top: 50%; left: 50%;}

[#absolute-calc .buiten](#) {position: relative;}

[#absolute-calc .binnen](#) {position: absolute; top: calc(50% - 100px); left: calc(50% - 100px);}

[#absolute-breedte-marge .buiten](#) {position: relative;}

[#absolute-breedte-marge .binnen](#) {margin: auto auto; position: absolute; top: 0; right: 0; bottom: 0; left: 0;}

[#fixed .buiten](#) {transform: translateX(0);}

[#fixed .binnen](#) {margin: auto auto; position: fixed; top: 0; right: 0; bottom: 0; left: 0;}

[#tabel-text-align .buiten](#) {display: table-cell; text-align: center; vertical-align: middle;}

[#tabel-text-align .binnen](#) {display: inline-block; text-align: left;}

[#tabel-marge .buiten](#) {display: table-cell; vertical-align: middle;}

[#tabel-marge .binnen](#) {margin: 0 auto;}

[#flexbox .buiten](#) {display: flex; justify-content: center;}

[#flexbox .binnen](#) {align-self: center;}

[#grid .buiten](#) {display: grid; justify-content: center;}

[#grid .binnen](#) {align-self: center;}

[Binnenste <div> heeft een vaste breedte van 200 px en een relatieve hoogte:](#)

[#text-align-inline-block .buiten](#) {height: 20rem; text-align: center; line-height: 20rem;}

[#text-align-inline-block .binnen](#) {display: inline-block; height: auto; max-height: 20rem; line-height: normal; text-align: left; vertical-align: middle;}

[Binnenste <div> heeft een relatieve breedte en hoogte:](#)

[#percentage .buiten](#) {box-sizing: border-box; padding: 50px 0;}

[#percentage .binnen](#) {width: 33%; height: 100%; overflow: auto; margin: 0 auto;}

[#absolute-translate .buiten](#) {position: relative;}

[#absolute-translate .binnen](#) {max-height: 30%; overflow: auto; position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%);}

[Binnenste <div> heeft een onbekende breedte en hoogte:](#)

[#onbekend-absolute .buiten](#) {position: relative;}

[#onbekend-absolute .binnen](#) {width: 0; height: 0; margin: auto auto; position: absolute; top: 0; right: 0; bottom: 0; left: 0;}

[#onbekend-absolute .binnen span](#) {max-height: 300px; overflow: auto; transform: translate(-50%, -50%);}

[#onbekend-padding .buiten](#) {box-sizing: border-box; padding: 150px min(300px, 45vw);}

[#onbekend-padding .binnen](#) {display: inline-block; max-height: 300px; overflow: auto; transform: translate(-50%, -50%);}

[css voor vensters minimaal 760 px breed](#)

Volledige code:

[HTML](#)

[CSS](#)

Korte omschrijving

Vijftien verschillende manieren om een blok-element horizontaal én verticaal te centreren.

GROENE BLOK

Buitenste element.

BLAUWE BLOK

Binnenste gecentreerde element (de laatste twee zijn wit).

TEKST IN WITTE BLOKJES

Benodigde css (alleen de voor het centreren benodigde css wordt weergegeven).

Opmerkingen

Binnen de buitenste groene en de binnenste blauwe <div> staat binnen de witte blokjes de css die voor het centreren nodig is (de essentiële css). Alleen die essentiële css staat binnen de blokjes, de css die nodig is om die blokjes weer te geven is weggelaten. (Die wordt wel hier in de uitleg besproken, maar wordt niet weergegeven in het voorbeeld zelf.)

Normaal genomen zouden veel regels van de css worden gecombineerd, omdat bij nogal wat selectors de erachter staande eigenschappen en waarden (vrijwel) hetzelfde zijn. Hier staat

het grootste deel van de regels die voor het centreren nodig zijn bij elk van de vijftien voorbeelden. Dat levert wel wat herhalingen op, maar is veel duidelijker.

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 ([KDE neon](https://kde.org/en/about/what-is-kde)). Daarbij is vooral gebruik gemaakt van [Visual Studio Code](https://code.visualstudio.com/), [GIMP](https://www.gimp.org/) en [Firefox](https://www.mozilla.org/en-US/firefox/) met extensies. De pdf-bestanden zijn gemaakt met [LibreOffice](https://www.libreoffice.org/).

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

Achterliggend idee

Eigenlijk is hier weinig idee. Gewoon vijftien verschillende manieren om een blok-element horizontaal én centraal te centreren. In dit voorbeeld gaat het om een <div> die binnen een andere <div> wordt gecentreerd, maar in principe kan elk blok-element op deze manier worden gecentreerd.

Toetsenbordnavigatie en tabindex

Op een desktopcomputer gebruiken de meeste mensen een muis om over een pagina te navigeren, links te volgen, e.d. Soms gebruiken mensen hier echter een toetsenbord voor, omdat ze geen muis kunnen of willen gebruiken. Navigeren, links volgen, e.d. gebeurt dan met behulp van de Tab-toets, Enter, Spatiebalk, pijltjestoetsen, e.d. (Overigens wordt ook bij touchscreens soms een toetsenbord en/of muis gebruikt.)

Als je alleen simpele html zou gebruiken, werkt deze toetsenbordnavigatie fantastisch. Maar soms kan een bepaalde constructie in de html of in de css de werking ervan verstoren. Bij het maken van een website moet je er vooral op letten dat alles bereikbaar is met de Tab-toets. Voor andere toetsen zijn dan meestal geen aanpassingen nodig.

Links, invoervelden in formulieren, e.d. kunnen met behulp van de Tab-toets één voor één worden bezocht, in de volgorde waarin ze in de html voorkomen. Shift+Tab-toets keert de volgorde van de Tab-toets om. Dit is een belangrijk hulpmiddel voor mensen die om een of andere reden de muis niet kunnen of willen gebruiken. (En het is vaak ook veel sneller dan de muis, vooral in formulieren.)

In sommige browsers en/of besturingssystemen is dit vreemd genoeg standaard uitgeschakeld en is een zoektocht in de instellingen nodig om dit aan te zetten. Maar gebruikers van de Tab-toets zullen dit al hebben gedaan.

Als je met behulp van de Tab-toets een element hebt bereikt, heeft dit [focus](#): als het een link is en je drukt op Enter, wordt de link gevolgd. Bij een tekstveld kun je tekst gaan invoeren. Enz.

De Tab-toets volgt normaal genomen de volgorde van de elementen in de html. Het maakt niet uit, in welke volgorde ze op het scherm staan. Als je met behulp van css de elementen van plaats verwisselt op het scherm, wordt toch gewoon de volgorde in de html gevolgd.

De volgorde van de Tab-toets kan worden veranderd met behulp van het `tabindex`-attribuut: `<div tabindex="3">`. Deze `<div>` zal nu als derde worden bezocht, ook al krijgt een simpele `<div>` normaal genomen nooit bezoek van de Tab-toets.

Normaal genomen is het gebruik van een `tabindex` niet nodig. Het is zeker niet bedoeld om de bezoeker als een kangoeroe op een hindernisbaan van onder via links over rechts naar boven te laten springen. Maar soms kan het handig zijn voor kleinere correcties, als de normale volgorde in de html niet optimaal is. Of om een element bereikbaar te maken voor de Tab-toets, zoals de hierboven genoemde `<div>`.

Schermlezers blijven altijd de volgorde van de html volgen, dus als de tabindex sterk afwijkt van de volgorde in de html, kan dat behoorlijk verwarrend zijn.
De tabindex kan drie verschillende waarden hebben: -1, 0 of een positief getal.

In principe is de volgorde bij gebruik van de Tab-toets als volgt: eerst worden alle positieve getallen in volgorde afgewerkt. Als twee tabindexen dezelfde waarde hebben, wordt de volgorde in de html aangehouden. Daarna worden elementen die automatisch al de [focus](#) kunnen krijgen (zoals een link en een knop) en elementen met een `tabindex="0"` afgewerkt in de volgorde, waarin ze in de html staan.

TABINDEX="-1"

Een negatieve waarde van -1 zorgt ervoor dat een element, dat normaal genomen door het gebruik van de Tab-toets de [focus](#) kan krijgen, volledig wordt genegeerd door de Tab-toets. Zelfs een link met een negatieve tabindex wordt volledig genegeerd.

Ook kan aan een element met een negatieve tabindex dat normaal genomen geen focus kan krijgen, toch de focus worden gegeven met behulp van JavaScript, een klik of een aanraking. Waarbij gebruikers van de Tab-toets daar geen last van hebben, omdat `tabindex="-1"` wordt genegeerd door de Tab-toets.

In dit voorbeeld wordt `tabindex="-1"` niet gebruikt.

TABINDEX="0"

Als je `tabindex="0"` gebruikt, kan een element [focus](#) krijgen met behulp van de Tab-toets. Ook als dat element normaal genomen geen focus kan krijgen.

In het voorbeeld uit de download hebben de buitenste ``'s in de binnenste `<div>` een `tabindex="0"`:

```
<span tabindex="0"><span>Gecentreerde element:
    </span>width: 200px; height: 200px; overflow:
    auto; align-self: center;</span>
```

Als de tekst sterk wordt vergroot, kan de tekst in die ``'s zo hoog worden dat de tekst (ver) onder de buitenste `<div>` komt te staan. En daardoor eventueel andere delen van de pagina kan afdekken. Om dit te voorkomen is een maximumhoogte aan de binnenste `<div>`'s gegeven. Als de tekst in de `` hoger is dan die maximumhoogte, kan de tekst worden gescrold.

Scrollen kan ook met behulp van de pijltjes op het toetsenbord. Maar daarvoor moet de `` wel de [focus](#) kunnen krijgen. Alleen Firefox geeft zelf al de focus aan een element (in dit geval de ``), als daarin gescrold kan worden.

Door het attribuut `tabindex="0"` toe te voegen aan de ``, kan deze door het indrukken van de Tab-toets ook in andere browsers de focus krijgen, waarna met de pijltjestoetsen gescrold kan worden.

(Hierboven wordt alleen een `` genoemd, maar in het laatste voorbeeld ontbreekt die ``. Daarin staat de css voor de binnenste `<div>` in die `<div>` zelf, daarom heeft daar de binnenste `<div>` `tabindex="0"` gekregen. De rest van het verhaal is hetzelfde.)

TABINDEX="..."

Op de plaats van de puntjes moet een positief getal worden ingevuld: het volgnummer. Een element met een positieve tabindex wordt altijd bezocht bij gebruik van de Tab-toets, ook als dit element normaal genomen zou worden genegeerd. Elementen met een tabindex met een positief volgnummer worden altijd

als eerste bezocht, voor elementen als een link of tekstveld zonder tabindex, en ook voor elementen met `tabindex="0"`.

In dit voorbeeld wordt een positieve tabindex niet gebruikt.

Muis, toetsenbord, touchpad en touchscreen

Vroeger, toen het leven nog mooi was en alles beter, waren er alleen monitors. Omdat kinderen daar niet af konden blijven met hun tengels, besloten fabrikanten dan maar touchscreens te gaan maken, omdat je die mag aanraken. Het bleek makkelijker te zijn om volwassenen ook te leren hoe je 'n scherm vies maakt, dan om kinderen te leren met hun vingers van de monitor af te blijven.

Zo ontstonden touchscreens en kreeg het begrip ellende een geheel nieuwe lading. In de perfecte wereld van vroeger, waarin alleen desktops bestonden, werkten dingen als hoveren, klikken en slepen gewoon. Zonder dat je eerst 'n cursus hogere magie op Zweinstein hoefde te volgen. Zelfs in JavaScript was het nog wel te behappen, ook voor mensen zoals ik die toevallig niet Einstein heten.

Op dit moment kun je computerschermen ruwweg in twee soorten indelen: schermen die worden aangeraakt, en schermen die worden bediend met hulpmiddelen als een toetsenbord, muis of touchpad. Omdat ook computerschermen zich kennelijk vermengen, bestaan er inmiddels ook schermen die zowel van aanraken als van muizen houden.

Hieronder staat een lijstje met dingen die zijn aangepast voor de verschillende soorten schermen, zodat dit voorbeeld overal werkt. Een touchpad werkt ongeveer hetzelfde als een muis. Als hieronder iets over een muis staat, geldt dat ook voor een touchpad.

:HOVER

Omdat `:hover` mogelijk niet werkt, als css uitstaat, ontbreekt of onvolledig is geïmplementeerd, moet je nooit belangrijke informatie alleen met behulp van `:hover` tonen.

Je hoovert over een element, als je met behulp van muis of touchpad de cursor boven dat element brengt. Hoveren kan over alle elementen. Het wordt veel gebruikt om iets van uiterlijk te laten veranderen, een pop-up te laten verschijnen, e.d.

Op een touchscreen wordt hoveren vaak hetzelfde afgehandeld als een aanraking. Bij gebruik van een muis is er een verschil tussen hoveren en klikken, maar op een touchscreen is dat verschil er niet: je raakt een touchscreen aan of niet. Dat levert vooral soms problemen op, als bij een element `:hover` én klikken worden gebruikt, zoals bij een link die bij hoveren erover verkleurt.

Hoveren wordt in dit voorbeeld niet gebruikt.

:FOCUS

Omdat `:focus` mogelijk niet werkt, als css uitstaat, ontbreekt of onvolledig is geïmplementeerd, moet je nooit belangrijke informatie alleen met behulp van `:focus` tonen.

De meeste mensen gaan met een muis naar een link, invoerveld, e.d. Waarna ze vervolgens klikken om de link te volgen, tekst in te voeren, of wat ze ook maar willen doen. (Dit geldt uiteraard niet voor touchscreens.)

Er is echter 'n tweede manier om naar links, invoervelden, e.d. te gaan: met behulp van de Tab-toets kun je naar links, invoervelden, e.d. 'springen'. (Over het gebruik van de Tab-toets staat meer bij [Toetsenbordnavigatie en tabindex](#).)

Waar je staat, wordt door alle browsers aangegeven met een of ander kadertje. De link e.d. met het kadertje eromheen 'heeft focus'. Dat wil zeggen dat je die link volgt, als je op de Enter-toets drukt, of dat je in dat veld tekst kunt gaan invoeren, enz.

Het kadertje dat de focus aangeeft, moet nooit zonder meer worden weggehaald. Gebruikers van de Tab-toets hebben dan geen idee meer, waar ze zijn. Normaal genomen worden een `` en een `<div>` door de Tab-toets genegeerd, maar door het toevoegen van `tabindex="0"` kunnen de ``'s met de css voor de binnenste `<div>` en de binnenste `<div>` in het laatste voorbeeld toch focus krijgen. Als de Tab-toets nogmaals wordt ingedrukt, verliezen de ``'s en de `<div>` de focus weer. Waarom dit van belang is, staat bij `tabindex="0"`.

:ACTIVE

Omdat `:active` mogelijk niet werkt, als css uitstaat, ontbreekt of onvolledig is geïmplementeerd, moet je nooit belangrijke informatie alleen met behulp van `:active` tonen.

Een element is actief, als de muis wordt ingedrukt boven dat element. Op sommige touchscreens is het element actief, als het wordt aangeraakt. `:active` wordt niet gebruikt in dit voorbeeld.

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, Vivaldi of Edge, is er 'n hele grote kans dat die in alle browsers werkt. Ik geef de voorkeur aan Firefox, omdat het een van de weinige browsers 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 het doctype voor html5. Dat is bedoeld voor nieuwe sites.
Het oudere transitional doctype staat talloze tags toe, die in html5 zijn verboden. Deze tags worden al zo'n tien jaar afgeraden. Het transitional doctype is alleen bedoeld om de puinhoop van vroeger, toen niet volgens standaarden werd gewerkt, enigszins te herstellen. Het (ook al oudere) 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 of een outline te geven. Je ziet dan goed, waar 'n deel precies staat. Een achtergrondkleur en een outline hebben – anders dan bijvoorbeeld een border – verder geen invloed op de lay-out, dus die zijn 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 layout 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 layout 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. Valiwie???

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 door schermlezers en 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 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. (Meer over de Tab-toets is te vinden [Toetsenbordnavigatie en tabindex](#).)

- 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 die zit, en welk element gaat reageren op bijvoorbeeld een Enter. (Meer over focus is te vinden bij [focus](#).)

- 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. Schermlezers houden namelijk altijd de volgorde van de html aan en niet een eventueel afwijkende volgorde op het scherm.
- Een zogenaamde skip-link is vaak nog wel zinvol. Dat is een link die je buiten het scherm parkeert met behulp van css, zodat die 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 de link op het scherm plaatsen, waardoor deze zichtbaar wordt. Bij een volgende tab wordt de link 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.
- Van oorsprong is html een taal om wetenschappelijke documenten weer te geven, pas later is deze gebruikt voor lay-out. Maar daar is de taal 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 deze 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 <table> 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 papieren 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 volstrekte 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.)

- 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. deze 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 5 september 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 (KDE Neon 5.27) (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 11.7.9 ('Big Sur') (1440 x 900 px, resolution: 96 ppi):
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 iPadOS 16.6 (2160 x 1620 px, resolution: 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 met iOS 15.7.8 (1334 x 750, resolution: 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).

Android 13 ('Tiramisu') (2408 x 1080 px, resolution: 401 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 11.7.9 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, Orca 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 15.7.8, iPadOS 16.6 en OS X 11.7.9.

Orca is een schermlezer voor Linux. Er is getest in combinatie met Firefox op KDE Neon 5.27.

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 \(en oplossingen\)](#).

Waar dat zinvol was, is ook nog getest op combinaties als een grote letter in een schermlezer met toetsenbordbediening.

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.

JavaScript wordt niet gebruikt, dus het aan- of uitstaan van JavaScript heeft geen invloed.

ZONDER CSS

Geen problemen.

Zonder css is de lay-out uiteraard verdwenen, maar alle tekst is gewoon nog aanwezig.

TOETSENBORDNAVIGATIE

Geen problemen.

Als tekst sterk wordt vergroot, kan de tekst in de binnenste <div> worden gescrold. Door het aanbrengen van `tabindex="0"` bij de 's (en één <div>) met tekst, kan de tekst ook met de pijltjestoetsen worden gescrold.

TEKSTBROWSERS

Geen problemen.

Tekstbrowsers geven netjes alle tekst weer.

SCHERMLEZERS

Geen problemen.

Omdat met behulp van css alleen het uiterlijk wordt veranderd, is er voor schermlezers geen enkel probleem. Aan de tekst zelf wordt niets veranderd, deze wordt gewoon voorgelezen.

Zonder lay-out is niet duidelijk, waarvoor de css in de witte blokjes dient. Daarom is, waar nodig, voor elke buitenste <div> 'Container: ' en voor elke binnenste <div> 'Gecentreerde element: ' toegevoegd. Deze tekstjes zijn bij `span span, #onbekend-padding .binnen span` buiten het scherm geplaatst. Ze zijn nu niet zichtbaar, maar worden wel voorgelezen.

Achter het tekstje staat een spatie om het tekstje te scheiden van het woord erna.

ZOOMEN EN ANDERE LETTERGROOTTE

Geen problemen.

Als bij een grotere letter de tekst te hoog wordt, kan deze worden gescrold.

Wijzigingen

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

Nieuw opgenomen.

24 februari 2011:

Op 'n aantal plaatsen `color: black;` toegevoegd vanwege de toegankelijkheid.
De reden staat in de uitleg bij de betreffende code.

5 september 2023:

- Volledig herschreven. Hier staan alleen de belangrijkste wijzigingen.
- xhtml omgezet naar html.
- Alles voor Internet Explorer verwijderd.
- Aantal voorbeelden van één naar vijftien uitgebreid, waaronder voorbeelden met flexbox en grid.
- Enkele nieuw hoofdstukken over toegankelijkheid e.d. toegevoegd.
- Kleuren van de <div>'s veranderd, omdat de nu gebruikte kleuren beter te onderscheiden zijn bij kleurenblindheid.
- Tig kleine veranderingen

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.

positioneren-068positioneren-068-dl.html: de pagina met het voorbeeld.

positioneren-068.pdf: deze uitleg (aangepast aan de inhoud van de download).

positioneren-068-inhoud-download-en-licenties.txt: een kopie van de tekst onder dit kopje (Inhoud van de download en licenties).

068-css-dl:

positioneren-068-dl.css: stylesheet voor positioneren-068-dl.html.

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 de regel 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 deze 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 de breedte van de <div>'s 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 gediensig 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="068-css-dl/positioneren-068-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 schelen. 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.)

```
<span tabindex="0"><span>Gecentreerde element: </span>width:
200px; height: 200px; overflow: auto; position: absolute; top:
50%; left: 50%; margin: -100px 0 0 -100px;</span>
```

Binnen de binnenste `<div>` staat de css die bij die `<div>` wordt gebruikt voor het centreren. Deze css staat binnen een ``. Als de tekst wordt vergroot en daardoor te hoog wordt, kan de tekst met de css worden gescrold.

Mensen die navigeren met behulp van het [toetsenbord](#) kunnen scrollen met behulp van de pijltjestoetsen. Maar dan moet de `` wel de [focus](#) kunnen krijgen. Door het toevoegen van [tabindex="0"](#) aan de `` wordt dat mogelijk.

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 alle 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, krijg je gegarandeerd 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, CLS, 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.)

css voor alle vensters

Algemene lay-out

`/* positioneren-068-dl.css */`

Om vergissingen te voorkomen is het een goede gewoonte bovenaan het script 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: #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 laan 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.

`margin: 0;`

Slim om te doen vanwege verschillen tussen browsers.

`main`

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

`padding: 5px;`

Kleine afstand tussen de buitenkant van en de inhoud in `<main>`

`abbr[title]`

`abbr`: alle `<abbr>`'s.

`[title]`: maar alleen als ze een bepaald attribuut hebben. Dat attribuut staat tussen twee teksthaken. In dit geval moet de `<abbr>` het `title`-attribuut hebben.

De hele selector in gewone taal: alle `<abbr>`'s, maar alleen als ze een `title`-attribuut hebben.

```
text-decoration: none; border-bottom: none;
```

Met een onderstreping wordt aangegeven dat het hier om een afkorting gaat.

Verskillende browsers doen dat op een verschillende manier. Hier worden alle onderstrepingen verwijderd. Maar alleen als de browser een `title`-attribuut heeft.

De reden daarvan staat gelijk hieronder.

```
abbr[title]::after
```

`abbr`: alle `<abbr>`'s.

`[title]`: maar alleen als ze een bepaald attribuut hebben. Dat attribuut staat tussen twee teksthaken. In dit geval moet de `<abbr>` het `title`-attribuut hebben.

`::after`: met behulp van `::after` wordt bij het de `<abbr>`'s met een `title`-attribuut een pseudo-element gemaakt.

De hele selector in gewone taal: zet achter de `<abbr>`'s die een `title`-attribuut hebben met behulp van `::after` een pseudo-element. Dit pseudo-element wordt gebruikt om de tekst in het `title`-attribuut weer te geven.

```
content: " (" attr(title) ")";
```

Met behulp van `content` kan iets worden weergegeven in het met behulp van `::after` gemaakte pseudo-element. Vaak is dat gewone tekst, maar hier is het de inhoud van het `title`-attribuut plus twee haakjes.

`" ("`: als eerste wordt een haakje, voorafgegaan door een spatie, weergegeven.

Omdat spatie en haakje letterlijk moeten worden weergegeven, staan ze tussen aanhalingstekens.

De spatie is nodig omdat het met `::after` gemaakte pseudo-element anders direct tegen de afkorting in de `<abbr>` komt te staan.

`attr()`: dit geeft aan dat de inhoud van een attribuut moet worden weergegeven.

`attr(title)`: tussen de haakjes staat de naam van het attribuut dat moet worden weergegeven. Omdat de selector `abbr[title]` alleen van toepassing is op `<abbr>`'s met een `title`-attribuut, is dit attribuut altijd aanwezig. De inhoud van het `title`-attribuut wordt getoond.

`") "`: het afsluitende haakje. Dit moet ook weer letterlijk worden weergegeven, dus staat het tussen aanhalingstekens.

Eigenschap en waarde zorgen ervoor dat de in de `<abbr>` zittende `title` tussen haakjes achter de afkorting wordt geplaatst.

```
<abbr lang="en" title="cascading style  
sheets">css</abbr>-tabel
```

verschijnt op het scherm als

css (cascading style sheets)-tabel

(Het attribuut `lang="en"` geeft aan de inhoud van `<abbr>` Engels is. Dit is van belang voor schermlezers, die anders mogelijk iets als 'kaskaading steile sjeets' uitspreken, wat mogelijk niet geheel en al direct duidelijk is.)

Afkorting is uitermate lastige dingen. Sommige mensen kennen ze niet en met een beetje pech herkent een schermlezer een afkorting niet, maar probeert de afkorting als een woord uit te spreken. Wat in het geval van 'css' waarschijnlijk een Chinese nies of zoiets oplevert.

Daarom worden op deze site afkortingen binnen een <abbr> gezet. De eerste keer dat een afkorting voorkomt, wordt daarbij de afkorting voluit in het `title`-attribuut gezet. Door dat attribuut vervolgens ook nog achter de afkorting te zetten, is dit voor iedereen zichtbaar. En schermlezers lezen het netjes voor.

Bij een herhaling van de afkorting kan iemand dan, als de afkorting onbekend is, zoeken naar de eerste keer dat de afkorting voorkomt.

```
font-size: 0.9em;
```

Iets kleinere letter.

Als eenheid wordt de relatieve eenheid `em` 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.

```
font-style: italic;
```

Cursief.

Als de gebruikte lettersoort cursief als stijl heeft, wordt die gebruikt. Heel veel lettersoorten hebben die stijl. Een cursieve stijl wordt speciaal ontworpen, het is iets anders dan 'alle letters gewoon 'n beetje scheef neerzetten'. Sommige letters kunnen er zelfs heel anders uitzien.

Als de lettersoort geen cursief als stijl heeft, maakt de browser die. Dat wil zeggen dat de browser gewoon alle letters een soort schop geeft, waardoor ze allemaal 'n beetje omvallen. Dat kan mooi zijn, maar vaak is het foeilelijk. Het is heel iets anders dan een speciaal ontworpen font. Elke rechtgeaarde typograaf gruwet hiervan.

h1

Alle <h1>'s. Dat is er maar één: de belangrijkste kop van de pagina.

```
font-size: 1em;
```

Een <h1> heeft van zichzelf een wel heel erg grote letter. Hier wordt dat teruggebracht naar de standaardgrootte.

Als eenheid wordt de relatieve eenheid `em` 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.

```
text-align: center;
```

Tekst horizontaal centreren.

```
margin-bottom: 0;
```

Standaard heeft een <h1> een marge aan boven- en onderkant. Hier wordt de marge aan de onderkant weggehaald.

p

Alle <p>'s.

```
text-align: center;
```

Tekst horizontaal centreren.

```
font-size: 0.9em;
```

Iets kleinere letter.

Als eenheid wordt de relatieve eenheid `em` 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;
```

Standaard heeft een <p> een marge aan boven- en onderkant. Die wordt hier weggehaald.

h2 + p

Voor deze elementen 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.

`p {text-align: center; font-size: 0.9em; margin: 0;}`

h2: alle <h2>'s.

+: het element achter de + moet in de html direct volgen op het element voor de +. In dit geval gaat het om een <p> die gelijk op een <h2> volgt. Beide elementen moeten ook nog dezelfde ouder hebben.

p: alle <p>'s.

Er zijn twee <p>'s die direct op een <h2> volgen én die dezelfde ouder als die <h2> hebben, een <section>. Hierin staan ondertitels bij de tweede en derde vette kop.

`margin: -10px 0 15px;`

Omdat voor links geen waarde is opgegeven, krijgt links automatisch dezelfde waarde als rechts. Hier staat dus eigenlijk `-10px 0 15px 0` in de volgorde boven – rechts – onder – links.

Boven en onder geen marge.

Boven een negatieve marge. Hierdoor komt de <p> met de ondertitel dicht bij de <h2> erboven te staan.

Onder een marge van 15 px voor wat afstand tussen de ondertitel en het eronder staande kopje.

section

Alle <section>'s. De voorbeelden zijn opgedeeld in vier groepen, die elk in een <section> staan. Binnen die <section> staat elk voorbeeld ook weer in een eigen <section>.

`margin-bottom: 15px;`

Wat afstand tussen elke <section> en wat eronder staat.

h2, h3

Alle <h2>'s en <h3>'s. Binnen de <h2>'s staan de vier vette titels boven elke groep, binnen de <h3>'s staan de kopjes boven elk voorbeeld.

`font-size: 0.9em;`

De standaardmaat van de <h2>'s en <h3>'s iets verkleinen.

Als eenheid wordt de relatieve eenheid `em` 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.

`text-align: center;`

Tekst horizontaal centreren.

`margin-bottom: 15px;`

Marge aan de onderkant.

h3

Voor deze elementen 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.

```
h2, h3 {font-size: 0.9em; text-align: center; margin-bottom: 15px;}
```

Alle <h3>'s. De kopjes boven elk voorbeeld.

```
font-weight: normal;
```

Standaard is een <h3> vet. Hier wordt dat veranderd in normaal.

```
margin: 0;
```

De marge die gelijk hierboven aan de <h2>'s en <h3>'s is gegeven bij de <h3>'s weer weghalen.

.buiten

Alle elementen met class="buiten". Dit zijn de buitenste <div>'s met de groene achtergrond. Een deel van de css is voor al deze <div>'s hetzelfde en kan hier voor allemaal worden opgegeven.

De css die nodig is voor het centreren staat steeds bij de voorbeelden zelf. Behalve de breedte en hoogte van de buitenste <div>: die worden hier ook voor alle buitenste <div>'s opgegeven.

(Behalve bij het laatste voorbeeld zijn de breedte en hoogte van de buitenste <div> eigenlijk ook niet nodig voor het centreren. Alleen wordt het een volslagen losgeslagen zootje omdat breedte en hoogte van de buitenste <div>, afhankelijk van de precieze code voor het voorbeeld, enorm gaan verschillen. 'n Wilde tuin is mooi, wilde <div>'s niet.)

```
background: green;
```

Groene achtergrond.

```
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.

Dit is ook al bij <body> opgegeven, maar sommige mensen hebben bij alle elementen de kleuren veranderd. Het heeft immers weinig zin, als ze dat alleen bij de body doen, terwijl de sitebouwer de kleuren ook bij bijvoorbeeld de paragrafen heeft aangepast.

```
width: 600px;
```

Breedte.

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

Hier gelijk boven is een breedte van 600 px opgegeven. In smallere browservensters zou daardoor horizontaal gescrold moeten worden om alles 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. De buitenste <div> wordt hierdoor nooit breder dan 90% van de breedte van het venster, ongeacht de breedte van het venster.

height: 300px;

Hoogte.

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 de <div> altijd horizontaal gecentreerd binnen ouder <section> te staan. <section> is een blok-element en wordt daardoor normaal genomen automatisch even breed als de ouder ervan: ook weer een <section>. Die dus normaal genomen ook weer even breed wordt als ouder <main>. Ook <main> is een blok-element en wordt daardoor normaal genomen even breed als de ouder ervan: <body>. Het wordt eentonig: ook weer een blok-element dat dus normaal genomen even breed wordt als ouder <html>. Omdat <html> het buitenste element is, wordt dit normaal genomen even breed als het venster van de browser.

Hierdoor staat uiteindelijk div.buiten altijd horizontaal gecentreerd binnen het venster van de browser, ongeacht de breedte van het venster. En daarmee ook de in de <div> zittende tekst en binnenste <div>.

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 hierboven heeft de <div> een breedte van 600 px en een maximumbreedte van 90 vw gekregen.

border: black solid 1px;

Randje.

.binnen

Alle elementen met class="binnen". Dit zijn de binnenste gecentreerde <div>'s met de blauwe achtergrond. Een deel van de css is voor al deze <div>'s hetzelfde en kan hier voor allemaal worden opgegeven.

De css die nodig is voor het centreren staat steeds bij de voorbeelden zelf.

background: cyan;

Blauwe achtergrondkleur.

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.

Dit is ook al bij <body> opgegeven, maar sommige mensen hebben bij alle elementen de kleuren veranderd. Het heeft immers weinig zin, als ze dat alleen bij de body doen, terwijl de sitebouwer de kleuren ook bij bijvoorbeeld de paragrafen heeft aangepast.

border: black solid 1px;

Zwart randje.

#bekend .binnen, #combinatie .binnen

Voor deze elementen 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.

[.binnen](#) {background: cyan; color: black; border: black solid 1px;}

Twee selectors, gescheiden door een komma.

#bekend. binnen, de eerste selector:

#bekend: het element met id="bekend". Binnen section#bekend staan de

gecentreerde <div>'s met een vaste breedte en hoogte.

.binnen: de elementen met class="bekend". De binnenste gecentreerde <div>'s met blauwe achtergrond.

#combinatie .binnen, de tweede selector:

#combinatie: het element met id="combinatie". Binnen section#combinatie staan de <div>'s met een absolute breedte en een relatieve hoogte.

.binnen: de elementen met class="bekend". De binnenste gecentreerde <div>'s met blauwe achtergrond.

De twee selectors bij elkaar: de <div>'s met class="binnen" binnen het element met id="bekend" en binnen het element met id="combinatie". Een deel van de css voor de gecentreerde <div>'s met een vaste breedte en/of hoogte is hetzelfde. Die kan hier in een keer worden opgegeven. De css die echt nodig is voor het eigenlijke centreren staat bij de voorbeelden zelf.

width: 200px;

Breedte.

height: 200px;

Hoogte.

overflow: auto;

Als de tekst wordt vergroot, kan deze hoger worden dan de hierboven opgegeven hoogte van 200 px. Standaard wordt die tekst dan toch weergegeven. Mogelijk verstoort het de lay-out, maar er verdwijnt in ieder geval geen tekst.

Maar daardoor zou deze tekst beneden de <div> komen te staan en mogelijk zelfs buiten de buitenste <div>. Daardoor zouden andere delen van de pagina afgedekt kunnen worden. Door deze waarde kan tekst worden gescrolld, als deze hoger wordt dan 200 px. Afhankelijk van browser en besturingssysteem kan hierbij een verticale scrollbar verschijnen.

Dit is ook de reden dat dit tot de essentiële css wordt gerekend: het is niet de bedoeling dat bij een grotere letter mogelijk andere delen van de pagina verdwijnen achter de <div>.

span

Alle 's. Dit zijn de witte blokjes, waarin de benodigde css voor het centreren staat. Op het scherm is duidelijk zichtbaar of de css voor de buitenste of binnenste <div> is. Als een [schermlezer](#) het voorleest, is dat niet duidelijk. Daarom zijn voor schermlezers extra 's aangebracht met 'Container: ' en 'Gecentreerde element: '.

background: white;

Witte achtergrond.

color: black;

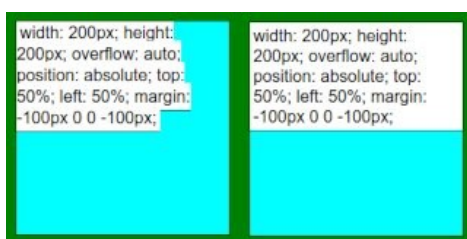
Voorgrondkleur zwart. Dit is o.a. de kleur van de tekst.

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.

Dit is ook al bij `<body>` opgegeven, maar sommige mensen hebben bij álle elementen de kleuren veranderd. Het heeft immers weinig zin, als ze dat alleen bij de body doen, terwijl de sitebouwer de kleuren ook bij bijvoorbeeld de paragrafen heeft aangepast.

```
display: inline-block;
```



Een `` is van zichzelf een inline-element. Hierdoor zijn eigenschappen als padding niet goed te gebruiken. Bij een inline-element komt de padding alleen aan de buitenkant van de tekst te staan: aan het begin en eind van de tekst, aan de bovenkant van de eerste regel en aan de onderkant van de onderste regel. Dit is

op de linkerafbeelding te zien.

Een inline-block is een soort kruising tussen een inline- en een blok-element. Het komt niet op een nieuwe regel te staan, maar de padding komt nu, net als bij een blok-element, rondom de hele tekst te staan, zoals op de afbeelding rechts is te zien. Bovendien vult de `` nu, net als een blok-element, de volle breedte van ouder `<div>`. (Dit geldt niet voor de ``'s binnen de buitenste `div.buiten`, want die worden later absoluut gepositioneerd en daardoor niet breder dan nodig is om de tekst erin weer te geven).

```
font-size: 0.9em;
```

Iets kleinere letter.

Als eenheid wordt de relatieve eenheid `em` 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.

```
border-bottom: black solid 1px;
```

Zwart randje aan de onderkant.

```
padding: 3px;
```

Wat afstand tussen tekst in en buitenkant van de ``.

.buiten > span

Voor deze elementen 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.
[span](#) {background: white; color: black; display: inline-block; font-size: 0.9em; border-bottom: black solid 1px; padding: 3px;}

Een deel van de css is voor al deze 's hetzelfde, die kan hier in één keer worden opgegeven. Niet alle css hier is voor elk van die 's nodig, maar dat maakt in dit geval niets uit.

.buiten: de elementen met class="buiten". De groene buitenste <div>'s.

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

Onderstaande span#kind is een direct kind van div.buiten:

```
<div class="buiten">
  <span id="kind"></span>
</div>
```

De binnenste span#binnenste hieronder is geen direct kind van div.buiten, omdat er een div.binnen tussen div.buiten en span#binnenste zit:

```
<div class="buiten">
  <span id="kind"></span>
  <div class="binnen">
    <span id="binnenste"></span>
  </div>
</div>
```

span: de 's die een direct kind van div.buiten zijn.

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

Dit zijn de 's die in de meeste voorbeelden linksboven in de buitenste groene <div> staat. Hierin staat de css voor de buitenste <div>, voor zover die nodig is voor het centreren.

opacity: 0.8;

Bij een (veel) grotere letter komt de met de css voor de binnenste <div> soms onder de met de css voor de buitenste <div> te staan. Door deze voor de buitenste <div> wat doorzichtig te maken, is in ieder geval nog te zien dat de binnenste <div> ook met een grotere letter nog steeds horizontaal en verticaal gecentreerd is.

(Dit is trouwens wel iets om rekening mee te houden. Als ook in de ouder van het te centreren element tekst of zo staat, moet op een of andere manier worden voorkomen dat die tekst verdwijnt onder het te centreren element.)

border-right: black solid 1px;

Zwart randje rechts. Omdat deze 's niet de volle breedte vullen, krijgen ze rechts een zwart randje.

line-height: normal;

Bij sommige voorbeelden wordt voor het centreren o.a. een grote regelhoogte voor de buitenste <div> gebruikt. Een regelhoogte wordt geërfd door de nakomelingen van de <div>, waaronder de met de css voor de buitenste <div>. Hier wordt de regelhoogte voor die naar een standaardregelhoogte teruggezet.

`text-align: left;`

Tekst links uitlijnen. Bij sommige voorbeelden wordt de buitenste `<div>` met `text-align: center;` gecentreerd. `text-align` wordt geërfd door de nakomelingen van de `<div>`, waaronder de `` met de css voor de buitenste `<div>`. Hier wordt de `` gewoon weer links uitgelijnd.

`position: absolute;`

Om de `` 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. Dat is hier de buitenste `<div>` `div.buiten`, die – waar nodig – een relatieve positie krijgt.

Door de ``'s absoluut te positioneren verstoren ze het centreren van de binnenste `<div>` niet, omdat een absoluut gepositioneerd element volledig wordt genegeerd door de andere elementen.

Bij twee voorbeelden wordt [display: flex](#); en [display: grid](#); gebruikt bij de buitenste `<div>`. Daardoor worden de directe kinderen van die `<div>`'s, waaronder deze ``, op een aparte manier weergegeven. Dat levert hier problemen op. Door de ``'s absoluut te positioneren, wordt dit voorkomen. `display: flex`; en `display: grid`; hebben dan geen invloed meer op de weergave.

`top: 0; left: 0;`

In de linkerbovenhoek neerzetten.

`z-index: 10;`

Normaal genomen worden elementen weergegeven in de volgorde, waarin ze in de html staan. Elementen die later in de html staan, worden boven eerdere elementen weergegeven.

Daardoor zou deze `` bij een grotere lettergrootte in sommige voorbeelden (afhankelijk van hoe het voorbeeld werkt) kunnen verdwijnen onder `div.binnen`.

Door de `` een hogere z-index te geven, blijft deze altijd bovenaan staan.

Een z-index werkt alleen in bepaalde omstandigheden. Eén van die omstandigheden is een absolute positie. Die is iets hierboven aan de `` gegeven, dus dat is geregeld.

`span span, #onbekend-padding .binnen span`

Voor deze elementen 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.

[span](#) {background: white; color: black; display: inline-block; font-size: 0.9em; border-bottom: black solid 1px; padding: 3px;}

Twee selectors, gescheiden door een komma.

`span span`: de eerste selector. Alle ``'s die binnen een andere `` staan. Dit zijn de ``'s waarin een tekstje voor [schermlezers](#) staat, omdat bij voorlezen van de css niet gelijk duidelijk is, voor welke `<div>` de css is bedoeld.

De tweede selector `#onbekend-padding .binnen span`:

`#onbekend-padding`: het element met `id="onbekend-padding"`.

`.binnen`: de elementen met `class="binnen"` binnen `#onbekend-padding`. Dat is er maar één: de binnenste `<div>` `div.binnen`.

`span`: alle ``'s binnen `div.binnen`. Dat is er ook maar één.

De binnenste `<div>` bij [#onbekend-padding .binnen](#) is een wat apart geval, omdat de css voor die `<div>` niet binnen een `` staat, maar gewoon rechtstreeks in de `<div>`. Binnen die `<div>` staat het tekstje 'Gecentreerd element: ' voor schermlezers wel weer in een ``. Maar dat is geen geneste ``, vandaar dat daarvoor een aparte selector nodig is.

Alles bij elkaar bestrijken deze twee selectors alle ``'s met tekstjes voor een schermlezer. Deze tekstjes worden links buiten het scherm geplaatst, zodat ze onzichtbaar zijn, maar ze worden gewoon voorgelezen.

De selector `span span` is een wat riskante selector. In de html hieronder gaat het goed:

```
<span><span>Gecentreerde element: </span>width: 200px;
height: 200px; overflow: auto; margin: 0
auto;</span>
```

Maar in de html hieronder niet:

```
<span>Mijn naam is Repelsteeltje (<span
lang="de">Rumpelstilzchen</span> in het
Duits)</span>
```

Hier valt de naam van de Duitse tweelingzus van Repelsteeltje ook onder deze selector, omdat ze binnen een geneste `` staat. Die geneste `` geeft alleen maar de taal aan, maar daardoor wordt de arme tweelingzus van Repelsteeltje toch onbedoeld onzichtbaar links buiten het scherm komt te staan. Een geneste `` is een geneste ``, waar je die ook voor gebruikt. Kortom: een wat breed werkende selector, waar je wat mee op moet passen.

```
position: absolute;
```

Om de ``'s 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. Hier is dat `div.buiten`, waarbinnen deze ``'s staan, want elke `div.buiten` met zo'n `` wordt later relatief gepositioneerd.

(Dat de `` waarvoor de tweede selector is bedoeld binnen `div.binnen` staat, maakt niet uit. `div.binnen` is niet gepositioneerd, dus ook daar is de eerste voorouder die is gepositioneerd `div.buiten`.)

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

Ver links buiten het scherm neerzetten. Hoewel de tekst in de ``'s nu niet is te zien, lezen [schermlezers](#) deze gewoon voor.

Binnenste `<div>` heeft een vaste breedte en hoogte van 200 px

De binnenste `<div>`'s binnen deze groep hebben allemaal een vaste breedte en hoogte van 200 px. Zonder deze vaste breedte en hoogte werken de voorbeelden binnen deze groep niet.

#alleen-marge .binnen

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.

```
.binnen {background: cyan; color: black; border: black solid 1px;}
```

```
#bekend .binnen, #combinatie .binnen {width: 200px; height: 200px; overflow: auto;}
```

De elementen met `class="binnen"` binnen het element met `id="alleen-marge"`. De binnenste `<div>` binnen `section#alleen-marge` (het voorbeeld onder het kopje 'Met alleen marge').

```
margin: 50px auto;
```

Omdat voor onder en links geen waarden zijn opgegeven, krijgen die automatisch dezelfde waarde als boven en onder. Hier staat dus eigenlijk `50px auto 50px auto` in de volgorde boven – rechts – onder – links.

Boven en onder een marge van 50 px. De buitenste <div> heeft bij [.buiten](#) een hoogte van 300 px gekregen. De binnenste <div> heeft bij [#bekend .binnen](#), [#combinatie .binnen](#) een hoogte van 200 px gekregen. Als aan de boven- en onderkant van de binnenste <div> een marge van 50 px wordt gegeven, komt de binnenste <div> dus verticaal precies in het midden van de buitenste <div>. Links en rechts auto, wat hier hetzelfde betekent als evenveel. Hierdoor komt de binnenste <div> altijd horizontaal gecentreerd binnen ouder div.buiten te staan, ongeacht de breedte van div.buiten.

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 [#bekend .binnen](#), [#combinatie .binnen](#) heeft div.binnen een breedte 200 px gekregen.

#marge-padding .buiten

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.

```
.buiten {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}
```

De elementen met class="buiten" binnen het element met id="marge-padding". De buitenste <div> binnen section#marge-padding (het voorbeeld onder het kopje 'Met marge en padding').

box-sizing: border-box; padding-top: 50px;



Aan de bovenkant wordt aan div.buiten een padding van 50 px gegeven.

div.binnen komt tegen deze padding aan te staan en komt daardoor op 50 px van de bovenkant van div.buiten te staan.

De buitenste <div> heeft bij [.buiten](#) een hoogte van 300 px gekregen, de binnenste <div> heeft bij [#bekend .binnen](#), [#combinatie .binnen](#) een

hoogte van 200 px gekregen. Als de binnenste <div> 50 px omlaag wordt verplaatst, komt deze dus verticaal precies in het midden van de buitenste <div> te staan.

Normaal genomen wordt een padding bij de hoogte opgeteld. Dat is wat op de afbeelding gebeurt: de binnenste <div> staat door de padding aan de bovenkant van div.buiten wel 50 px omlaag, maar die 50 px padding komt er aan de onderkant van de buitenste <div> weer bij. Dat schiet dus niet op.

Door box-sizing: border-box; wordt de padding niet bij de hoogte van div.buiten opgeteld, maar komt deze binnen de hoogte te staan. De buitenste <div> wordt nu niet hoger en de binnenste <div> staat netjes verticaal gecentreerd binnen de buitenste <div>.

```
position: relative;
```

Om nakomelingen van div.buiten te kunnen positioneren ten opzichte van div.buiten, moet de <div> zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op div.buiten zelf, omdat niets voor top e.d. wordt opgegeven.

#marge-padding .binnen

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.

[.binnen](#) {background: cyan; color: black; border: black solid 1px;}

[#bekend .binnen](#), [#combinatie .binnen](#) {width: 200px; height: 200px; overflow: auto;}

De elementen met class="binnen" binnen het element met id="marge-padding". De binnenste <div> binnen section#marge-padding (het voorbeeld onder het kopje 'Met marge en padding').

margin: 0 auto;

Omdat voor onder en links geen waarde is opgegeven, krijgen deze 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 de binnenste <div> altijd horizontaal gecentreerd binnen ouder div.buiten te staan, ongeacht de breedte van div.buiten.

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 [#bekend .binnen](#), [#combinatie .binnen](#) heeft div.binnen een breedte 200 px gekregen.

#absolute-negatieve-marge .buiten

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.

[.buiten](#) {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}

De elementen met class="buiten" binnen het element met id="absolute-negatieve-marge". De buitenste <div> binnen section#absolute-negatieve-marge (het voorbeeld onder het kopje 'position: absolute met negatieve marge').

position: relative;

Om nakomelingen van div.buiten te kunnen positioneren ten opzichte van div.buiten, moet de <div> zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op div.buiten zelf, omdat niets voor top e.d. wordt opgegeven.

#absolute-negatieve-marge .binnen

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.

`.binnen {background: cyan; color: black; border: black solid 1px;}`

`#bekend .binnen, #combinatie .binnen {width: 200px; height: 200px; overflow: auto;}`

De elementen met `class="binnen"` binnen het element met `id="absolute-negatieve-marge"`. De binnenste `<div>` binnen `section#absolute-negatieve-marge` (het voorbeeld onder het kopje 'position: absolute met negatieve marge').

margin: -100px 0 0 -100px;



Als dat niet wordt aangepast, zou de binnenste `<div>` in de linkerbovenhoek van de buitenste `<div>` komen te staan.

Op de afbeelding gebeurt dat ook, maar vervolgens is de binnenste `<div>` met een negatieve marge aan boven- en linkerkant 100 px naar boven en naar links verschoven. Hierdoor komt de binnenste `<div>` veel

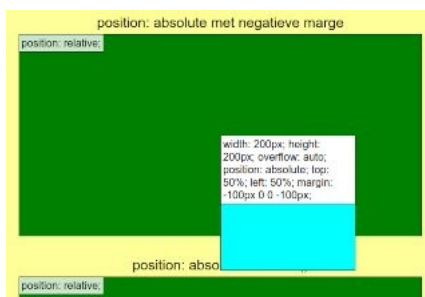
te ver naar links en naar boven te staan.

position: absolute;

Om `div.binnen` 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. Dat is hier `div.buiten`, die bij `#absolute-negatieve-marge .buiten` relatief is gepositioneerd.

top: 50%; left: 50%;



De negatieve marge van iets hierboven is op de afbeelding even weggelaten. De linkerbovenhoek van de binnenste `<div>` wordt horizontaal en verticaal precies in het midden van de buitenste `<div>` gezet. Nu staat de binnenste `<div>` weer veel te veel naar rechts en naar beneden.

De binnenste `<div>` heeft bij `#bekend .binnen, #combinatie .binnen` een breedte en hoogte van 200 px gekregen. Als je de `<div>` vanuit dit middelpunt

de helft van die breedte en hoogte naar boven en naar links zet, komt de binnenste `<div>` uiteindelijk horizontaal en verticaal gecentreerd in de buitenste `<div>` te staan. Dat is precies wat de negatieve marge van iets hierboven aan boven- en linkerkant doet: negatieve marge en absolute positie halverwege heffen elkaar als het ware op, waardoor `div.binnen` gecentreerd staat.

Ook dit werkt weer alleen bij een bekende breedte en hoogte van de binnenste `<div>`, want je moet weten hoeveel px de negatieve marges moeten zijn. Een marge in procenten werkt hier niet, omdat een marge in procenten ten opzichte van de breedte van de ouder wordt genomen. En `div.binnen` moet de helft van z'n eigen breedte en hoogte worden verplaatst, niet ten opzichte van de breedte van ouder `div.buiten`.

#absolute-calc .buiten

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.

[.buiten](#) {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}

De elementen met class="buiten" binnen het element met id="absolute-calc". De buitenste <div> binnen section#absolute-calc (het voorbeeld onder het kopje 'position: absolute met calc()').

position: relative;

Om nakomelingen van div.buiten te kunnen positioneren ten opzichte van div.buiten, moet de <div> zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op div.buiten zelf, omdat niets voor top e.d. wordt opgegeven.

#absolute-calc .binnen

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.

[.binnen](#) {background: cyan; color: black; border: black solid 1px;}

[#bekend .binnen](#), [#combinatie .binnen](#) {width: 200px; height: 200px; overflow: auto;}

De elementen met class="binnen" binnen het element met id="absolute-calc". De binnenste <div> binnen section#absolute-calc (het voorbeeld onder het kopje 'position: absolute met calc()').

position: absolute;

Om div.binnen 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. Dat is hier div.buiten, die bij [#absolute-calc .buiten](#) relatief is gepositioneerd.

top: calc(50% - 100px);

Met behulp van calc() kunnen berekeningen worden gemaakt. In dit geval wordt het verticale midden van de buitenste div.buiten berekend. Daar wordt dan de bovenkant van div.binnen neergezet, zodat die verticaal gecentreerd binnen div.buiten staat.

50%: bij top is een eenheid in procenten ten opzichte van de ouder van het element.

De ouder van div.binnen is div.buiten, dus dit is precies halverwege de hoogte van de buitenste <div>.

100px: bij [#bekend .binnen](#), [#combinatie .binnen](#) heeft div.binnen een hoogte van 200 px gekregen. De helft daarvan is 100 px.

De berekening wordt hier gemaakt met twee verschillende eenheden: % en px. Dat kan niet, eerst moeten allen eenheden worden omgerekend naar dezelfde eenheid. Daarom rekent de browser de eenheden om naar de eenheid px.

Bij het schrijven van de code kan dat omrekenen vaak niet, omdat je niet weet hoe hoog de ouder van het element is. Hier zou dat wel kunnen, omdat de hoogte van div.buiten bekend is: 300 px. Maar deze methode werkt altijd, zelfs als de hoogte van de ouder van het element niet bekend is.

De berekening wordt dan 50% van de hoogte van de ouder min 100 px. Die hoogte van de ouder is hier 300 px, de helft daarvan is 150 px. Min 100 px is 50 px. div.binnen komt dus op 50 px van de bovenkant te staan. div.buiten is 300

px hoog, `div.binnen` is 200 px hoog, dus aan boven- en onderkant van `div.binnen` is de afstand tot `div.buiten` 50 px: verticaal gecentreerd.

left: calc(50% - 100px);

Om `div.binnen` horizontaal te centreren binnen `div.buiten`, is het verhaal precies hetzelfde als gelijk hierboven bij `top: calc(50% - 100px)`.

Hierboven is de hoogte van `div.buiten` gebruikt: 300 px. Hier wordt de breedte van `div.buiten` gebruikt: 600 px. Maar dat maakt niets uit, omdat met procenten wordt gerekend. 50% is altijd halverwege, hoe groot de breedte of hoogte ook is.

#absolute-breedte-marge .buiten

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.

`.buiten` {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}

De elementen met `class="buiten"` binnen het element met `id="absolute-breedte-marge"`. De buitenste `<div>` binnen `section#absolute-breedte-marge` (het voorbeeld onder het kopje 'position: absolute met margin: auto').

position: relative;

Om nakomelingen van `div.buiten` te kunnen positioneren ten opzichte van `div.buiten`, moet de `<div>` zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op `div.buiten` zelf, omdat niets voor `top` e.d. wordt opgegeven.

#absolute-breedte-marge .binnen

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.

`.binnen` {background: cyan; color: black; border: black solid 1px;}

`#bekend.binnen, #combinatie.binnen` {width: 200px; height: 200px; overflow: auto;}

De elementen met `class="binnen"` binnen het element met `id="absolute-breedte-marge"`. De binnenste `<div>` binnen `section#absolute-breedte-marge` (het voorbeeld onder het kopje 'position: absolute met margin: auto').

margin: auto auto;

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

Eerst de marges links en rechts: `auto`, wat hier hetzelfde betekent als evenveel.

Oftewel: `div.binnen` staat altijd horizontaal gecentreerd ten opzichte van z'n ouder `div.buiten`, ongeacht de breedte van `div.buiten`.

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.

Maar hier doet zich een eigenaardig verschijnsel voor. `div.binnen` wordt iets hieronder absoluut gepositioneerd, en bij een absoluut gepositioneerd element werkt de truc om met `auto` te centreren normaal genomen niet. Hier werkt die echter toch. Iets hieronder wordt met `right: 0;` en `left: 0;` opgegeven dat `div.binnen` van de linker- tot de rechterkant van `div.buiten` moet lopen. Bij `#bekend.binnen, #combinatie.binnen` is echter een breedte van 200 px opgegeven.

Dit is een onmogelijke opdracht. `div.buiten` heeft bij [.buiten](#) een breedte tot 600 px gekregen (afhankelijk van de breedte van het browservenster). `div.binnen` moet van links tot rechts lopen, dus ook tot 600 px breed zijn, én heeft een breedte van 200 px. Dat is hooguit mogelijk als het browservenster toevallig één bepaalde breedte van iets van 240 px of zo zou hebben.

De browser raakt hier dermate van in de war, dat `right: 0;` en `left: 0;` gewoon worden genegeerd en `margin: 0 auto;` toch werkt. Oftewel: bij een absolute positie werkt `margin: 0 auto;` niet, tenzij je ook `left: 0;` en `right: 0;` opgeeft.

(Deze weinig bekende truc werkt trouwens ook bij `position: fixed;`. Je kunt de horizontale plaatsing nog beïnvloeden door bij `right` of `left` een andere waarde dan 0 in te vullen. En het is ook niet zo dat de browser in de war raakt. Dit staat gewoon in de [specificatie](#). Alleen is het zo weerzinwekkend technisch opgeschreven dat naar verluidt zelfs Einstein het pas na 38 keer lezen begreep. Inmiddels is er een nieuwe ontwerp-specificatie met een andere benadering, maar het resultaat is hetzelfde.)

Dan zijn er nog de marges aan boven- en onderkant: ook `auto`. Hiervoor geldt precies hetzelfde verhaal. `div.binnen` wordt iets hieronder met `top: 0;` en `bottom: 0;` even hoog gemaakt als `div.buiten`, die bij [.buiten](#) een hoogte van 300 px heeft gekregen. `div.binnen` heeft bij [#bekend.binnen](#), [#combinatie.binnen](#) een hoogte van 200 px gekregen. Weer een onmogelijke opdracht dus. Mogelijk zou uitrekken op een pijnbank werken, maar dan krijg je Amnesty International op je dak, dus dat kan ook niet.

De browser lost dit op precies dezelfde manier op als bij het centreren in horizontale richting: `top` en `bottom` worden genegeerd en `auto` aan boven- en onderkant werkt: `div.binnen` staat ook verticaal gecentreerd binnen `div.buiten`.

position: absolute;

Om `div.binnen` 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. Dat is hier `div.buiten`, die bij [#absolute-breedte-marge.buiten](#) relatief is gepositioneerd.

top: 0; right: 0; bottom: 0; left: 0;

`div.binnen` aan alle kanten tegen ouder `div.buiten` zetten. Waarom dit toch niet gebeurt, staat iets hierboven bij [margin: auto auto;](#).

#fixed .buiten

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.

```
.buiten {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}
```

De elementen met `class="buiten"` binnen het element met `id="fixed"`. De buitenste `<div>` binnen `section#fixed` (het voorbeeld onder het kopje 'position: fixed met margin: auto').

transform: translateX(0);

Met de bij `transform` horende functie `translateX()` kun je 'n element horizontaal ten opzichte van zichzelf verplaatsen. Tussen de haakjes komt de afstand van die verplaatsing te staan. Hier is dat de indrukwekkende afstand van 0 px, wat aan de visie van Rutte doet denken. (Waar de man godbetert nog trots op is ook.)

Een verplaatsing van 0 px lijkt wat nutteloos. Voor die verplaatsing is dat ook zo. Maar dit vormt wel een nieuw 'containing block'. Hier gelijk onder wordt `div.binnen` fixed gepositioneerd. Normaal genomen wordt een fixed gepositioneerd element gepositioneerd ten opzichte van het browservenster en scrolt het niet mee met de pagina: het venster vormt het containing block.

Maar als een voorouder van een fixed gepositioneerd element bepaalde eigenschappen heeft, vormt die voorouder het containing block. Het fixed element wordt dan gepositioneerd ten opzichte van die voorouder, en het scrolt mee met die voorouder.

Een van de eigenschappen die een nieuw containing blok vormen is `transform` met een andere waarde dan `none`. Daarom wordt `transform` hier gebruikt met een wat rare waarde: 0 px horizontaal verplaatsen. Het gaat niet om die verplaatsing, het gaat om een nieuw containing block. En daarvoor is een waarde nodig die anders is dan `none`. En omdat een browser niet zo snugger is, denkt die dat een verplaatsing van 0 px iets anders is dan `none`. Net zoals Rutte denkt dat het niet hebben van een visie iets anders is dan het zijn van een leeghoofd. Dus maakt de browser braaf een nieuw containing block.

#fixed .binnen

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.

`.binnen` {background: cyan; color: black; border: black solid 1px;}

`#bekend .binnen, #combinatie .binnen` {width: 200px; height: 200px; overflow: auto;}

De elementen met `class="binnen"` binnen het element met `id="fixed"`. De binnenste `<div>` binnen `section#fixed` (het voorbeeld onder het kopje 'position: fixed met margin: auto').

`margin: auto auto; position: fixed; top: 0; right: 0; bottom: 0; left: 0;`

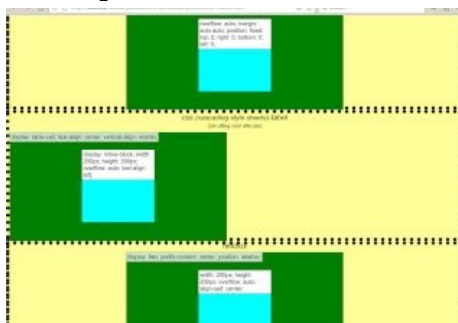
Deze css is vrijwel hetzelfde als die bij [#absolute-negatieve-marge .binnen](#) iets hierboven. Er is maar één verschil: de `<div>` hierboven is absoluut gepositioneerd, de `<div>` hier is fixed gepositioneerd.

Normaal genomen scrolt een fixed gepositioneerd element mee met de pagina. Hier is dat niet het geval, omdat ouder `div.buiten` het zogenaamde 'containing block' vormt voor deze `div.binnen`. Hierdoor scrolt de `<div>` gewoon mee met de pagina. Meer over het containing block is hierboven te vinden bij [#fixed .buiten](#).

`#tabel-text-align, #tabel-marge`

De elementen met `id="tabel-text-align"` en `id="tabel-marge"`. Dit zijn de twee `<section>`'s met een css-tabel.

`width: 600px;`



Een `<section>` is een blok-element en wordt daardoor normaal genomen automatisch even breed als de ouder ervan. Die ouder is hier `<main>`, ook een blok-element. `<main>` wordt daardoor normaal genomen ook weer even breed als ouder `<body>`. Ook weer een blok-element, dat daardoor normaal genomen even breed wordt als ouder `<html>`. Omdat `<html>` het buitenste element is, wordt dit normaal

genomen even breed als het venster van de browser. Uiteindelijk wordt de `<section>` daardoor even breed als het venster, ongeacht de breedte van het venster.

Hier zie je niets van, omdat de buitenste `<div>` bij [.buiten](#) horizontaal is gecentreerd. Maar ook al zie je die `<section>`'s niet, ze zijn er wel. Op de afbeelding zijn de `<section>`'s zichtbaar gemaakt met een gestippelde outline.

Voor het centreren worden de twee buitenste `<div>`'s in deze `<section>`'s iets hieronder met `display: table-cell;` weergegeven als de cel van een tabel.

Dit gebeurt met css, in de html is niets van `<table>`, `<td>`, `<tr>`, of andere dingen die bij `<table>` horen te vinden. Alleen de weergave van de pagina wordt veranderd, niet de structuur van de pagina.

Als in de html elementen ontbreken bij een tabel, worden die door de browser ingevoegd. Dat is bij een css-tabel net zo. Hier is alleen een cel aanwezig. De browser voegt de ontbrekende elementen toe. Ook dat zijn geen 'echte' `<table>`-elementen, maar css-tabel-elementen. Daarom leveren deze geen enkel probleem op wat betreft toegankelijkheid en zoekmachines: alleen de weergave wordt veranderd, niet de structuur van de inhoud.

Als je een html-`<table>` gebruikt om het uiterlijk te veranderen, kan dat grote problemen opleveren voor toegankelijkheid en zoekmachines, omdat ook de structuur van de inhoud wordt veranderd. Bij een css-tabel speelt dat niet. Een `<table>` in de html hoort alleen voor tabulaire gegevens gebruikt te worden.

Hier speelt echter wel een ander probleem. Die door de browser ingevoegde elementen zie je niet, zelfs niet als je de code in het ontwikkelgereedschap van de browser bekijkt. Hierdoor zijn ze niet met iets als `margin: 0 auto;` te centreren, want je kunt de ingevoegde elementen niet met een selector bereiken. Daardoor wordt de ingevoegde css-tabel gewoon standaard helemaal links in de ouder ervan neergezet. En dat is hier de `<section>`, die dus even breed is als het venster van de browser. Daardoor komt deze tabel helemaal links te staan. En de inhoud ervan ook. Hierdoor staan deze twee voorbeelden als enige helemaal links. Dat is wat er op de afbeelding gebeurt.

Door de `<section>` met de twee voorbeelden met een css-tabel 600 px breed te maken en horizontaal te centreren, wordt dit voorkomen.

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

Hier gelijk boven is een breedte van 600 px opgegeven. In smallere browservensters zou daardoor horizontaal gescrold moeten worden om alles 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. De `<section>` wordt hierdoor nooit breder dan 90% van de breedte van het venster, ongeacht de breedte van het venster.

```
margin: 0 auto 30px;
```

Omdat voor links geen waarde is opgegeven, krijgt links automatisch dezelfde waarde als rechts. Hier staat dus eigenlijk `0 auto 30px auto` in de volgorde boven – rechts – onder – links.

Boven geen marge. Onder een marge voor wat afstand tot het volgende voorbeeld. Links en rechts `auto`, wat hier hetzelfde betekent als evenveel. Hierdoor staat de `<section>` altijd horizontaal gecentreerd binnen ouder `<main>`. `<main>` is een blok-element en wordt daardoor normaal genomen even breed als de ouder ervan: `<body>`.

Ook `<body>` is weer een blok-element dat dus normaal genomen even breed wordt als ouder `<html>`. Omdat `<html>` het buitenste element is, wordt dit normaal genomen even breed als het venster van de browser.

Hierdoor staan de twee `<section>`'s uiteindelijk altijd horizontaal gecentreerd binnen het venster van de browser, ongeacht de breedte van het venster. En daarmee ook alles wat in de `<section>` zit.

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 hierboven heeft de `<section>` een breedte van 600 px en een maximumbreedte van 90 vw gekregen.

Nu staan de twee voorbeelden met een css-tabel horizontaal net zo gecentreerd als de andere voorbeelden. Wat wel zo netjes is.

`#tabel-text-align h3 span, #tabel-marge h3 span`

Voor deze elementen 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.
`span {background: white; color: black; display: inline-block; font-size: 0.9em; border-bottom: black solid 1px; padding: 3px;}`

Alle ``'s binnen de `<h3>`'s binnen de elementen met `id="tabel-text-align"` en `id="tabel-marge"`.

Binnen de `<h3>`'s staan de kopjes van deze twee voorbeelden. Onder die kopjes staat de toevoeging '(zie uitleg voor alle css)'. Die toevoeging staat in deze ``'s.

Eigenlijk zou de css voor `section#tabel-text-align` en `section#tabel-marge` hier ook in moeten staan, maar dat paste niet meer. (Die css is niet nodig om de binnenste `<div>` ten opzichte van de buitenste `<div>` te centreren, maar wel om de buitenste `<div>` horizontaal te centreren ten opzichte van het venster van de browser. Dus het voelt een beetje als valsspelen, als je die css weglaat.)

`background: none;`

Alle ``'s hebben eerder een witte achtergrond gekregen. Deze ``'s krijgen geen achtergrondkleur.

`border: none;`

Alle ``'s hebben eerder een zwarte border aan de onderkant gekregen. Die wordt hier weggehaald.

`transform: translateY(-5px);`

Met de bij `transform` horende functie `translateY()` kun je 'n element ten opzichte van zichzelf in verticale richting verplaatsen. Omdat de waarde negatief is, wordt naar boven verplaatst. Hierdoor komt de toevoeging iets dichterbij het kopje te staan, wat iets netter is.

`#tabel-text-align .buiten`

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.
`.buiten {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}`

De elementen met `class="buiten"` binnen het element met `id="tabel-text-align"`. De buitenste `<div>` binnen `section#tabel-text-align` (het voorbeeld onder het kopje 'css-tabel met text-align: center').

`display: table-cell;`

`div.buiten` weergeven alsof het een cel in een tabel is. In html zou je hier `<td>` voor gebruiken.

Een tabel gebruiken voor lay-out doeleinden? In 2023? Eh, ja.

Vroeger kon je eigenlijk alleen een tabel gebruik om te lay-outen. Dat leverde gigantische problemen op voor toegankelijkheid, zoekmachines, onderhoud, noem

maar op. Tegenwoordig heb je veel betere en meer mogelijkheden met css dan html ooit heeft geboden. Daarom wordt al jaren geen `<table>` meer gebruikt voor lay-out doeleinden.

Dit is echter een css-tabel, of eigenlijk: een css-cel. Bij een html-tabel wordt de structuur echt aangepast, bij een css-tabel niet. Daarom levert een css-tabel, en daarmee ook dingen als een css-tabelcel, geen enkel probleem op wat betreft toegankelijkheid, zoekmachines of onderhoud.

Je hebt dus niet de nadelen van een `<td>`, maar je hebt wel de voordelen: in een `<td>` kun je de eigenschap `vertical-align` gebruiken om de inhoud ervan verticaal te centreren. Ook een blok-element – zoals `div.binnen` – kan hiermee simpel verticaal worden gecentreerd. En omdat een css-tabelcel zich net zo gedraagt als een `<td>`, kan dat hier ook.

`text-align: center;`

Tekst horizontaal centreren.

De binnen deze `div.buiten` zittende `div.binnen` wordt iets hieronder met `display: inline-block;` veranderd in een soort kruising tussen een inline- en een blok-element. Hierdoor kan de `div.binnen` met behulp van `text-align` horizontaal worden gecentreerd, wat bij een gewoon blok-element niet zou werken.

`vertical-align: middle;`

Omdat `div.buiten` hierboven met `display: table-cell;` is veranderd in de cel van een tabel, kan deze eigenschap worden gebruikt om `div.binnen` verticaal te centreren binnen `div.buiten`. Normaal genomen werkt dit niet voor een blok-element, maar binnen een tabelcel werkt het wel.

`position: relative;`

Om nakomelingen van `div.buiten` te kunnen positioneren ten opzichte van `div.buiten`, moet de `<div>` zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op `div.buiten` zelf, omdat niets voor top e.d. wordt opgegeven.

`#tabel-text-align .binnen`

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.

`.binnen {background: cyan; color: black; border: black solid 1px;}`
`#bekend .binnen, #combinatie .binnen {width: 200px; height: 200px; overflow: auto;}`

De elementen met `class="binnen"` binnen het element met `id="tabel-text-align"`. De binnenste `<div>` binnen `section#tabel-text-align` (het voorbeeld onder het kopje 'css-tabel met `text-align: center`').

`display: inline-block;`

Een `<div>` is van zichzelf een blok-element. Een inline-block is een soort kruising tussen een inline- en een blok-element. Het komt niet op een nieuwe regel te staan, maar het kan nu wel worden gecentreerd met behulp van `text-align: center;`, wat iets hierboven aan ouder `div.buiten` is gegeven.

`text-align: left;`

Ouder `div.buiten` heeft iets hierboven `text-align: center;` gekregen om `div.binnen` te kunnen centreren. `text-align` is erfelijk, waardoor ook de tekst binnen `div.binnen` wordt gecentreerd. Hier wordt die tekst weer links uitgelijnd.

#tabel-marge .buiten

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.

```
.buiten {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}
```

De elementen met class="buiten" binnen het element met id="tabel-marge". De buitenste <div> binnen section#tabel-marge (het voorbeeld onder het kopje 'css-tabel met marge').

display: table-cell; vertical-align: middle; position: relative;

Deze css is hetzelfde als die bij [#tabel-text-align .buiten](#). De beschrijving is daar te vinden. Daar wordt ook nog text-align: center; gebruikt om

div.binnen horizontaal te centreren, maar dat gebeurt hier op een andere manier, dus dat ontbreekt hier.

#tabel-marge .binnen

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.

```
.binnen {background: cyan; color: black; border: black solid 1px;}  
#bekend .binnen, #combinatie .binnen {width: 200px; height: 200px; overflow: auto;}
```

De elementen met class="binnen" binnen het element met id="tabel-marge". De binnenste <div> binnen section#tabel-marge (het voorbeeld onder het kopje 'css-tabel met marge').

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 div.binnen altijd horizontaal gecentreerd binnen ouder div.buiten te staan, ongeacht de breedte van div.buiten.

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 [#bekend .binnen, #combinatie .binnen](#) heeft div.binnen een breedte van 200 px gekregen.

#flexbox .buiten

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.

```
.buiten {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}
```

De elementen met class="buiten" binnen het element met id="flexbox". De buitenste <div> binnen section#flexbox (het voorbeeld onder het kopje 'flexbox').

display: flex;

De waarde flex is onderdeel van een hele reeks eigenschappen en waarden, die bij elkaar 'flexbox' worden genoemd. Met display: flex; wordt div.buiten in een zogenaamde 'flex container' veranderd. Dit maakt het veel makkelijker om de directe kinderen van dit element, de 'flex items', op een bepaalde plaats neer te zetten. De directe kinderen van div.buiten zijn hier de met de css voor

`div.buiten` en de binnenste `<div> div.binnen`. Omdat de `` bij [.buiten](#) `span` absoluut is gepositioneerd, is deze echter geen flex item. `display: flex;` heeft dus alleen invloed op `div.binnen`, precies de bedoeling.

`justify-content: center;`

De flex items horizontaal centreren. Hier is dat de binnenste `<div> div.binnen`. Voor deze eigenschap is ook de eigenschap `flex-direction` van belang. Deze geeft de richting aan, waarin flex items worden geplaatst: horizontaal of verticaal, en van de eerste naar de laatste, of in omgekeerde volgorde. Standaard heeft `flex-direction` de waarde `row`: horizontaal. Daardoor worden de flex items – hier is dat `div.binnen` – horizontaal gecentreerd.

Zou je hier `flex-direction: column;` gebruiken, dan zouden de flex items verticaal worden gecentreerd. De waarde van `flex-direction` is ook van belang voor andere eigenschappen van flexbox, zoals het hieronder bij `div.binnen` gebruikte `align-self`.

`position: relative;`

Om nakomelingen van `div.buiten` te kunnen positioneren ten opzichte van `div.buiten`, moet de `<div>` zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op `div.buiten` zelf, omdat niets voor top e.d. wordt opgegeven.

`#flexbox .binnen`

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.

[.binnen](#) {background: cyan; color: black; border: black solid 1px;}
[#bekend .binnen](#), [#combinatie .binnen](#) {width: 200px; height: 200px; overflow: auto;}

De elementen met `class="binnen"` binnen het element met `id="flexbox"`. De binnenste `<div>` binnen `section#flexbox` (het voorbeeld onder het kopje 'flexbox').

`align-self: center;`

Bij [#flexbox .buiten](#) is ouder `div.buiten` veranderd in een 'flex container'.

Hierdoor zijn de directe kinderen veranderd in een 'flex item'. `div.binnen` is zo'n flex item. Met behulp van de bij flexbox horende eigenschap `align-self` kan een flex item horizontaal worden gecentreerd binnen de flex container. In dit geval wordt `div.binnen` horizontaal gecentreerd binnen `div.buiten`.

`#grid .buiten`

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.

[.buiten](#) {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}

De elementen met `class="buiten"` binnen het element met `id="grid"`. De buitenste `<div>` binnen `section#grid` (het voorbeeld onder het kopje 'grid').

`display: grid; justify-content: center;` `position: relative;`

Deze css is vrijwel hetzelfde als die bij [#flexbox .buiten](#). De uitleg is daar te vinden. Het enige verschil: hier wordt `display: grid;` gebruikt in plaats van `display: flex;`.

`grid` is meer bedoeld voor lay-outs in twee richtingen, `flex` meer voor een horizontale óf verticale lay-out.

#grid .binnen

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.

```
.binnen {background: cyan; color: black; border: black solid 1px;}  
#bekend .binnen, #combinatie .binnen {width: 200px; height: 200px; overflow: auto;}
```

De elementen met class="binnen" binnen het element met id="grid". De binnenste <div> binnen section#grid (het voorbeeld onder het kopje 'grid').

align-self: center;

Dit is dezelfde css als bij #flexbox .binnen, alleen gaat het hier niet om een 'flex item', maar om een 'grid item'.

Binnenste <div> heeft een vaste breedte van 200 px en een relatieve hoogte

De binnenste <div> binnen deze groep heeft een vaste breedte van 200 px. De hoogte is afhankelijk van de hoogte van ouder div.buiten.

#text-align-inline-block .buiten

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.

```
.buiten {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto;  
border: black solid 1px;}
```

De elementen met class="buiten" binnen het element met id="text-align-inline-block". De buitenste <div> binnen section#text-align-inline-block (het voorbeeld onder het kopje 'text-align met display: inline-block').

height: auto;

Eerder heeft div.buiten bij .buiten een vaste hoogte van 300 px gekregen. Hier wordt die weggehaald. Daardoor gedraagt deze div.buiten zich weer op de standaardmanier van een blok-element: de hoogte wordt bepaald door de inhoud ervan, of door een opgegeven hoogte. Hier is het de iets hieronder opgegeven regelhoogte van 20 rem die de hoogte van div.buiten bepaalt.

text-align: center;

Tekst horizontaal centreren.

De binnen deze div.buiten zittende div.binnen wordt iets hieronder met display: inline-block; veranderd in een soort kruising tussen een inline- en een blok-element. Hierdoor kan de <div> met behulp van text-align horizontaal worden gecentreerd, wat bij een gewoon blok-element niet zou werken.

line-height: 20rem;

Regelhoogte.

Deze regelhoogte is enorm groot, een standaardregelhoogte is ongeveer 1,2 rem. Een regelhoogte van 20 rem is ongeveer 320 px. Door de regelhoogte zo hoog te maken, ontstaat de mogelijkheid div.binnen halverwege deze regelhoogte neer te zetten: verticaal gecentreerd.

Als eenheid wordt de relatieve eenheid rem gebruikt, omdat bij gebruik van een absolute eenheid zoals px de regelhoogte niet mee verandert met de lettergrootte. Hierdoor zou bij een (heel) grote letter de tekst in div.binnen onder div.buiten komen te staan en mogelijk andere delen van de pagina afdekken. Zoomen kan wel altijd, ongeacht welke eenheid wordt gebruikt.

Omdat geen gewone hoogte is opgegeven, is dit tevens de hoogte van `div.buiten`. Deze `div.buiten` is daardoor de enige buitenste `<div>` van de vijftien voorbeelden, waarvan de hoogte mee verandert met de lettergrootte. 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.

`position: relative;`

Om nakomelingen van `div.buiten` te kunnen positioneren ten opzichte van `div.buiten`, moet de `<div>` zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op `div.buiten` zelf, omdat niets voor top e.d. wordt opgegeven.

#text-align-inline-block .binnen

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.

`.binnen` {background: cyan; color: black; border: black solid 1px;}

`#bekend .binnen, #combinatie .binnen` {width: 200px; height: 200px; overflow: auto;}

De elementen met `class="binnen"` binnen het element met `id="text-align-inline-block"`. De binnenste `<div>` binnen `section#text-align-inline-block` (het voorbeeld onder het kopje 'text-align met display: inline-block').

display: inline-block;

Een `<div>` is van zichzelf een blok-element. Een inline-block is een soort kruising tussen een inline- en een blok-element. Het komt niet op een nieuwe regel te staan, maar het kan nu wel worden gecentreerd met behulp van `text-align`:

`center;`, wat iets hierboven aan ouder `div.buiten` is gegeven.

height: auto;

Eerder heeft `div.binnen` bij `#bekend .binnen, #combinatie .binnen` een vaste hoogte van 200 px gekregen. Hier wordt die weggehaald. Daardoor gedraagt deze `div.binnen` zich weer op de standaardmanier van een blok-element: de hoogte wordt bepaald door de inhoud ervan. Hier is dat de tekst in de `<div>` en de iets hieronder opgegeven `padding-bottom`.

max-height: 20rem;

Ouder `div.buiten` heeft bij `#text-align-inline-block .buiten` een regelhoogte van 20 rem gekregen. Omdat verder geen hoogte is opgegeven bij `div.buiten`, is dat ook gelijk de hoogte van `div.buiten`.

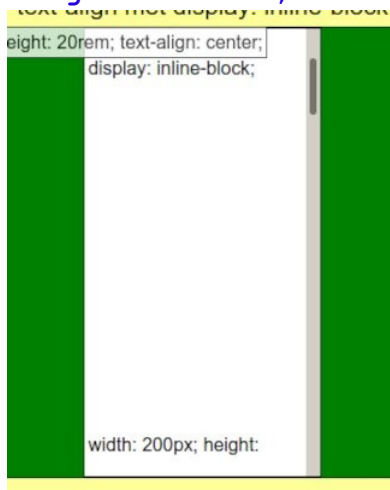
`div.binnen` wordt nu ook nooit hoger dan deze 20 rem en komt dus nooit buiten `div.buiten` te staan, waardoor mogelijk andere delen van de pagina zouden worden afgedekt.

Omdat eerder bij `#bekend .binnen, #combinatie .binnen` `overflow: auto;` is opgegeven, kan eventueel worden gescrold als de inhoud van `div.binnen` te hoog is. Op sommige systemen verschijnt in sommige browsers een verticale scrollbar, als gescrold kan worden.

Als eenheid wordt de relatieve eenheid `rem` gebruikt, omdat bij gebruik van een absolute eenheid zoals `px` de hoogte niet mee verandert met de lettergrootte. Zoomen kan wel altijd, ongeacht welke eenheid voor de lettergrootte 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.

`line-height: normal;`



Ouder `div.buiten` heeft een regelhoogte van 20 `rem` gekregen. Een regelhoogte is erfelijk, dus ook de tekst in `div.binnen` zou deze gigantische regelhoogte krijgen.

Op de afbeelding is de regelhoogte van 20 `rem` niet weggehaald bij `div.binnen`. Dat ziet er weliswaar heel vredig en meditatief uit, maar het leest toch wat rottig, en dat verpest de vredige meditatieve stemming dan toch weer.

Daarom wordt hier de regelhoogte naar de standaardregelhoogte teruggezet.

`text-align: left;`

Bij ouder `div.buiten` is `text-align: center;` gebruikt om `div.binnen` horizontaal te centreren. Die eigenschap is erfelijk, waardoor de tekst in `div.binnen` ook wordt gecentreerd. Hier wordt de tekst weer links uitgelijnd.

`vertical-align: middle;`

Omdat `div.binnen` iets hierboven in een inline-block is veranderd, kan de `<div>` halverwege de regelhoogte worden neergezet. En omdat die regelhoogte ook de hoogte van `div.buiten` is, staat `div.binnen` daarmee verticaal gecentreerd binnen `div.buiten`.

`padding-bottom: 60px;`

Aan de onderkant een padding om te laten zien dat `div.binnen` er echt nog is, want de padding heeft de bij `.binnen` aan `div.binnen` opgegeven achtergrondkleur.

Binnenste `<div>` heeft een relatieve breedte en hoogte

Bij de binnenste `<div>`'s binnen deze groep zijn breedte en hoogte afhankelijk van de hoogte en breedte van ouder `div.buiten`.

`#percentage .buiten`

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.

`.buiten` {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}

De elementen met `class="buiten"` binnen het element met `id="percentage"`. De buitenste `<div>` binnen `section#percentage` (het voorbeeld onder het kopje 'Alleen percentages').

`box-sizing: border-box;`

Bij `.buiten` heeft `div.buiten` een hoogte van 300 px gekregen. Hier gelijk onder wordt aan `div.buiten` aan boven- en onderkant een padding van 50 px gegeven

voor wat afstand tussen de buitenste en de binnenste <div>. Standaard worden een border en een padding bij de breedte en de hoogte opgeteld. `div.buiten` zou hierdoor geen 300 px, maar 400 px hoog worden.

Dat is niet de bedoeling: alle buitenste <div>'s zijn even hoog. Met de hier opgegeven waarde bij `box-sizing` komen border en padding binnen de opgegeven breedte en hoogte te staan, waardoor `div.buiten` toch niet hoger wordt dan 300 px.

padding: 50px 0;

Omdat voor onder en links geen waarden zijn opgegeven, krijgen die automatisch dezelfde waarde als boven en rechts. Hier staat dus eigenlijk `50px 0 50px 0` in de volgorde boven – rechts – onder – links.

Rechts en links geen padding. Boven en onder een padding van 50 px. Hier iets onder wordt `div.binnen` 100% hoog gemaakt, waardoor `div.binnen` de overblijvende ruimte tussen de padding aan boven- en onderkant vult. En dus 50 px van de boven- en onderkant van `div.buiten` staat: verticaal gecentreerd.

`position: relative;`

Om nakomelingen van `div.buiten` te kunnen positioneren ten opzichte van `div.buiten`, moet de <div> zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op `div.buiten` zelf, omdat niets voor top e.d. wordt opgegeven.

#percentage .binnen

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.

`.binnen {background: cyan; color: black; border: black solid 1px;}`

De elementen met `class="binnen"` binnen het element met `id="percentage"`. De binnenste <div> binnen `section#percentage` (het voorbeeld onder het kopje 'Alleen percentages').

width: 33%;

Een breedte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier `div.buiten`, die bij `.buiten` een breedte van 600 px heeft gekregen en een maximumbreedte van 90 vw. `div.binnen` is altijd 33% van de breedte van `div.buiten`, ook als `div.buiten` in smallere browservensters smaller dan 600 px is.

`min-width: 200px;`

Hier gelijk boven heeft `div.binnen` 33% van de breedte van `div.buiten` gekregen. `div.buiten` heeft bij `.buiten` een maximumbreedte van 90 vw gekregen, 90% van de breedte van het browservenster. In smalle vensters zou 33% daarvan wel heel erg smal worden. Daarom wordt hier een minimumbreedte van 200 px opgegeven.

height: 100%;

Een hoogte in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier `div.buiten`, die iets hierboven aan boven- en onderkant een padding van 50 px heeft gekregen. De ruimte tussen die padding aan boven- en onderkant wordt opgevuld door `div.binnen`. Waardoor `div.binnen` 50 px van boven- en onderkant van `div.buiten` staat: verticaal gecentreerd.

overflow: auto;

Als de tekst wordt vergroot, kan deze hoger worden dan `div.binnen`. Standaard wordt die tekst dan toch weergegeven. Mogelijk verstoort het de lay-out, maar er verdwijnt in ieder geval geen tekst.

Maar daardoor zou deze tekst mogelijk ook beneden `div.buiten` komen te staan en mogelijk andere delen van de pagina afdekken. Door deze waarde kan tekst worden gescrold, als deze hoger wordt dan `div.binnen`. Afhankelijk van browser en besturingssysteem kan hierbij een verticale scrollbar verschijnen.

Dit is ook de reden dat dit tot de essentiële css wordt gerekend: het is niet de bedoeling dat bij een grotere letter mogelijk andere delen van de pagina verdwijnen achter de `<div>`.

margin: 0 auto;

Omdat voor onder en links geen waarde is opgegeven, krijgen deze 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 de binnenste `<div>` altijd horizontaal gecentreerd binnen ouder `div.buiten` te staan, ongeacht de breedte van `div.buiten`.

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 iets hierboven is een breedte van 33% en een minimumbreedte van 200 px gegeven.

Bij het maken van de pagina is onbekend, hoeveel 33% is. Maar bij het weergeven weet de browser dat wel, en kan dus ook worden berekend, hoe groot de marges moeten worden.

#absolute-translate .buiten

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.

`.buiten` {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}

De elementen met `class="buiten"` binnen het element met `id="absolute-translate"`. De buitenste `<div>` binnen `section#absolute-translate` (het voorbeeld onder het kopje 'position: absolute met translate()').

position: relative;

Om nakomelingen van `div.buiten` te kunnen positioneren ten opzichte van `div.buiten`, moet de `<div>` zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op `div.buiten` zelf, omdat niets voor top e.d. wordt opgegeven.

#absolute-translate .binnen

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.

`.binnen {background: cyan; color: black; border: black solid 1px;}`

De elementen met `class="binnen"` binnen het element met `id="absolute-translate"`. De binnenste `<div>` binnen `section#absolute-translate` (het voorbeeld onder het kopje 'position: absolute met translate()').

max-height: 30%;

Standaard wordt een blok-element niet hoger dan nodig is om de inhoud ervan weer te geven. Bij een grotere letter zou deze `div.binnen` – door de manier waarop deze verticaal wordt gecentreerd – aan boven- en onderkant boven `div.buiten` uit komen te steken en mogelijk delen van de rest van de pagina afdekken. Daarom wordt hier een maximumhoogte opgegeven.

Een hoogte, en ook een maximumhoogte, in procenten is normaal genomen ten opzichte van de ouder van het element. Dat is hier `div.buiten`. Nu wordt `div.binnen` nooit hoger dan 30% van de hoogte van `div.buiten`, en komt daardoor nooit buiten `div.buiten` te staan. (Daarvoor is ook nog `overflow: auto;` gelijk hieronder nodig.)

`div.binnen` is echter duidelijk veel hoger dan 30% van `div.buiten`. Dat komt door de padding van 80 px aan de onderkant die iets hieronder aan `div.binnen` wordt gegeven: deze wordt bij de hoogte opgeteld. De uiteindelijke maximumhoogte is daardoor 30% van de hoogte van ouder `div.buiten` plus 80 px.

overflow: auto;

Als de tekst wordt vergroot, kan deze hoger worden dan `div.binnen`. Standaard wordt die tekst dan toch weergegeven. Mogelijk verstoort het de lay-out, maar er verdwijnt in ieder geval geen tekst.

Maar daardoor zou deze tekst mogelijk boven en onder `div.buiten` komen te staan en mogelijk andere delen van de pagina afdekken. Door deze waarde kan tekst worden gescrold, als deze hoger wordt dan de hoogte van `div.binnen`.

Afhankelijk van browser en besturingssysteem kan hierbij een verticale scrollbar verschijnen.

Dit is ook de reden dat dit tot de essentiële css wordt gerekend: het is niet de bedoeling dat bij een grotere letter mogelijk andere delen van de pagina verdwijnen achter de `<div>`.

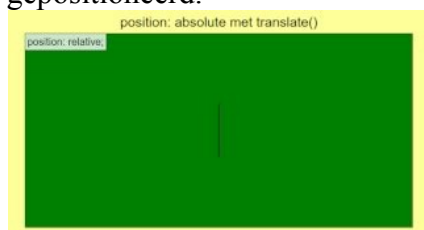
`padding-bottom: 80px;`

Wat padding aan de onderkant om te laten zien dat `div.binnen` nog steeds aanwezig is.

position: absolute;

Om `div.binnen` 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. Dat is hier `div.buiten`, die bij `#absolute-translate .buiten` relatief is gepositioneerd.

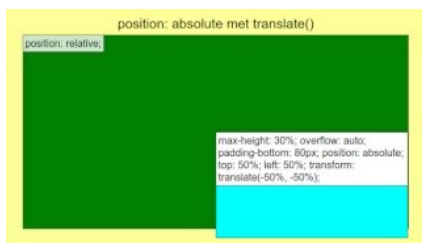


Een absoluut gepositioneerd element wordt niet breder dan nodig is om de inhoud ervan weer te geven. Die inhoud is hier de `` met de voor `div.binnen` benodigde css. Als die `` ontbreekt of leeg is, is er geen inhoud om weer te geven. Daardoor zou `div.binnen` niet breder

worden dan 0 px: onzichtbaar. Alleen de border zou je nog zien: een verticale zwarte lijn in het midden. Dat is wat er op de afbeelding gebeurt: slechts een zielig zwart lijntje blijft over, waarvan alleen liefhebbers van minimalistische moderne kunst onder de indruk schijnen te raken.

Die lijn is 80 px hoog, de hoogte van de gelijk hierboven opgegeven padding. Die padding is er nog wel, maar ook die is slechts 0 px breed. Dus uiteindelijk resteren alleen nog de linker- en rechterborder, twee verticale lijntjes die tegen elkaar aan komen te staan en samen 2 px breed zijn.

top: 50%; left: 50%;



De linkerbovenhoek van `div.binnen` horizontaal en verticaal in het midden van `div.buiten` zetten. Omdat een eenheid in procenten is opgegeven, maken breedte en hoogte van `div.buiten` niet uit: 50% is altijd halverwege.

Op de afbeelding staat de linkerbovenhoek van `div.binnen` horizontaal en verticaal precies in het midden van `div.buiten`.

Daarbij vallen twee dingen op.

Aan de onderkant komt `div.binnen` iets onder `div.buiten` te staan. Als maximumhoogte van `div.binnen` is iets hierboven 30% van de hoogte van `div.buiten` gegeven. Zolang `div.binnen` niet hoger wordt dan die 30%, wordt die weergegeven, al zou je `div.binnen` 1000 px naar beneden verplaatsen. Als je de letters zou gaan vergroten, zou `div.binnen` wel hoger worden dan die 30%. In dat geval gaat de opgegeven maximumhoogte werken en kan worden gescrollt. Een blok-element zoals een `<div>` wordt normaal genomen even breed als de ouder ervan. Maar als, zoals hier, de `<div>` absoluut is gepositioneerd, wordt deze niet breder dan nodig is om de inhoud ervan weer te geven. Hier is dat genoeg tekst om de volle breedte van ouder `div.buiten` te vullen. Maar omdat `div.binnen` 50% naar rechts is verplaatst, wordt alleen de resterende ruimte van ouder `div.buiten` gevuld. Waardoor `div.binnen` uiteindelijk niet breder wordt dan 50% van de breedte van `div.buiten`.

transform: translate(-50%, -50%);

Hier gelijk boven is de linkerbovenhoek van `div.binnen` precies in het midden van het `div.buiten` gezet. Als je nou `div.binnen` de helft van z'n eigen breedte en hoogte terug naar links en naar boven zet, staat `div.binnen` horizontaal en verticaal precies gecentreerd binnen `div.buiten`.

Met de bij `transform` horende functie `translate()` kun je 'n element ten opzichte van zichzelf verplaatsen. En als je die verplaatsing in procenten opgeeft, gelden die procenten ten opzichte van het element zelf. Omdat de browser bij het maken van de pagina alle maten weet, kan de browser altijd uitrekenen, hoeveel 50% is.

Het getal voor de komma verplaatst in horizontale richting, het getal na de komma in verticale richting. Omdat de waarde negatief is, wordt naar boven en naar links verplaatst. Met 50%, precies de helft van de breedte en de hoogte van `div.binnen`. Oftewel: de helft van `div.binnen` staat nu links van het midden van `div.buiten`, en de andere helft staat rechts daarvan: horizontaal gecentreerd. In verticale richting geldt hetzelfde: de helft van `div.binnen` staat boven het midden van `div.buiten`, de andere helft staat eronder.

Hierdoor staat `div.binnen` horizontaal en verticaal gecentreerd binnen `div.buiten`. En omdat de grootte van de verplaatsing in procenten is opgegeven, maakt de grootte van `div.binnen` niet uit: 50% is altijd de helft.

Binnenste `<div>` heeft een onbekende breedte en hoogte

Bij de binnenste `<div>`'s binnen deze groep zijn breedte en hoogte onbekend. Daardoor is er geen achtergrondkleur te zien.

#onbekend-absolute .buiten

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.

```
.buiten {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}
```

De elementen met `class="buiten"` binnen het element met `id="onbekend-absolute"`. De buitenste `<div>` binnen `section#onbekend-absolute` (het voorbeeld onder het kopje 'position: absolute met marge en translate(')).

position: relative;

Om nakomelingen van `div.buiten` te kunnen positioneren ten opzichte van `div.buiten`, moet de `<div>` zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op `div.buiten` zelf, omdat niets voor top e.d. wordt opgegeven.

#onbekend-absolute .binnen

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.

```
.binnen {background: cyan; color: black; border: black solid 1px;}
```

De elementen met `class="binnen"` binnen het element met `id="onbekend-absolute"`. De binnenste `<div>` binnen `section#onbekend-absolute` (het voorbeeld onder het kopje 'position: absolute met marge en translate(')).

width: 0; height: 0;

Breedte en hoogte 0 px. Dit verklaart, waarom hier geen achtergrondkleur van `div.binnen` is te zien: de `<div>` heeft geen achtergrond om te kleuren.

margin: auto auto; position: absolute; top: 0; right: 0; bottom: 0; left: 0;

Samen met de gelijk hierboven opgegeven breedte en hoogte zorgt deze combinatie van eigenschappen ervoor dat `div.binnen` horizontaal en verticaal gecentreerd wordt binnen `div.buiten`. Dat de `<div>` hierboven een breedte en hoogte van 0 px heeft gekregen, maakt niets uit. Een uitgebreide beschrijving hoe dit werkt, is te vinden bij [margin: auto auto](#). De `div.binnen` daar heeft een breedte en hoogte van 200 px, maar dat maakt voor de werking niets uit. Als er maar een breedte en hoogte zijn, of die nou 200 px of 0 px is.

Omdat deze `<div>` geen breedte en hoogte heeft en ook niets van een padding of zo, zou deze `<div>` volledig onzichtbaar, als de `<div>` geen inhoud heeft. Daarom wordt de `` met de css voor de binnenste `<div>` hier ook tot de essentiële code gerekend: zonder deze `` is alleen `div.buiten` zichtbaar. En dan is weliswaar `div.binnen` perfect gecentreerd, maar als die `<div>` onzichtbaar is, heb je daar niet zoveel aan.

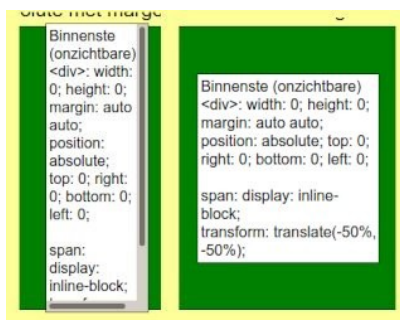
Dat de `` met tekst toch wordt weergegeven, terwijl die `` in een `<div>` staat met 0 px breedte en hoogte, komt doordat `overflow` standaard de waarde `visible` heeft: ook als de inhoud te groot is voor het element, wordt het toch weergegeven. Mogelijk wordt de lay-out dan verstoord, maar er verdwijnt in ieder geval niets.

#onbekend-absolute .binnen 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.

```
span {background: white; color: black; display: inline-block; font-size: 0.9em; border-bottom: black solid 1px; padding: 3px;}
```

De ``'s binnen de elementen met `class="binnen"`, die weer binnen een element met `id="onbekend-absolute"` zitten. Er zit maar één `div.binnen` binnen `section#onbekend-absolute`, en daarbinnen zit maar één ``. In die `` staat de css voor `div.binnen`.



Een `` is van zichzelf een inline-element, maar de `` eerder bij [span](#) veranderd in een inline-block-element. Daardoor wordt de `` even breed als ouder `div.buiten`. Dat schiet echter niet op, want `div.buiten` heeft hier iets boven een breedte van 0 px gekregen. Eigenlijk zou de `` daardoor ook maar 0 px breed worden.

Maar omdat er tekst in staat, wordt de `` toch breder. Want standaard worden tekst e.d. toch weergegeven, ook als

ze niet in het element passen. Maar de `` wordt niet breder dan nodig is. Dat is op de linkerafbeelding te zien: de `` is niet breder dan nodig is om het breedste woord weer te geven. Hierdoor wordt de `` echter zo hoog, dat deze ook zonder de tekst te vergroten al te hoog is om zonder scrollen weer te geven. 't Zal ook 'ns gewoon zonder problemen gaan.

Je kunt dit op verschillende manieren oplossen. Bijvoorbeeld door de `` een minimumbreedte te geven. Hier is het opgelost door een van de woorden uit de `` langer te maken:

```
span: display: inline-block;
      transform: &nbsp;translate(-50%, -50%);
```

Door een 'harde spatie' ` ` ('non-breaking space') tussen 'transform:' en 'translate(-50%)' te zetten, worden deze als één woord beschouwd. Nu is niet meer '(onzichtbare)' het langste woord, maar 'transform translate(-50%,)''. Het effect daarvan is op de rechterafbeelding te zien: de `` is nu veel breder en daardoor minder hoog, zodat bij een normale lettergrootte niet gescrold hoeft te worden om alles te zien.

max-height: 300px;

Maximumhoogte. Als de lettergrootte wordt verhoogd, zou de tekst buiten `div.buiten` kunnen komen te staan en daardoor mogelijk andere delen van de pagina afdekken. Daarom wordt hier een maximumhoogte opgegeven. (Om dit te voorkomen is ook nog `overflow: auto`; gelijk hieronder nodig.)

overflow: auto;

Als de tekst wordt vergroot, kan deze hoger worden dan `div.buiten`, die bij [buiten](#) een hoogte van 300 px heeft gekregen. Standaard wordt die tekst dan toch weergegeven. Mogelijk verstoort het de lay-out, maar er verdwijnt in ieder geval geen tekst.

Maar daardoor zou deze tekst mogelijk beneden en boven `div.buiten` komen te staan en mogelijk andere delen van de pagina afdekken. Door deze waarde kan tekst worden gescrold, als deze hoger wordt dan de gelijk hierboven opgegeven maximumhoogte van 300 px. Afhankelijk van browser en besturingssysteem kan hierbij een verticale scrollbar verschijnen.

Dit is ook de reden dat dit tot de essentiële css wordt gerekend: het is niet de bedoeling dat bij een grotere letter mogelijk andere delen van de pagina verdwijnen.

```
border: black solid 1px;
```

Zwart randje.

```
transform: translate(-50%, -50%);
```

De linkerbovenhoek van `div.binnen`, de ouder van deze ``, is hier iets boven horizontaal en verticaal gecentreerd binnen `div.buiten`. Daardoor wordt ook deze `` daar neergezet. Als je nou de `` de helft van z'n eigen breedte en hoogte terug naar links en naar boven zet, staat de `` horizontaal en verticaal precies gecentreerd binnen `div.buiten`.

Met de bij `transform` horende functie `translate()` kun je 'n element ten opzichte van zichzelf verplaatsen. En als je die verplaatsing in procenten opgeeft, gelden die procenten ten opzichte van het element zelf. Omdat de browser bij het maken van de pagina alle maten weet, kan de browser altijd uitrekenen, hoeveel 50% is.

Het getal voor de komma verplaatst in horizontale richting, het getal na de komma in verticale richting. Omdat de waarde negatief is, wordt naar boven en naar links verplaatst. Met 50%, precies de helft van de breedte en de hoogte van de ``. Oftewel: de helft van de `` staat nu links van het midden van `div.buiten`, en de andere helft staat rechts daarvan: horizontaal gecentreerd. In verticale richting geldt hetzelfde: de helft van de `` staat boven het midden van `div.buiten`, de andere helft staat eronder.

Hierdoor staat de ``, en daarmee ook de erin zittende tekst, horizontaal en verticaal gecentreerd binnen `div.buiten`. En omdat de grootte van de verplaatsing in procenten is opgegeven, maakt de grootte van de `` niet uit: 50% is altijd de helft.

#onbekend-padding .buiten

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.

```
.buiten {background: green; color: black; width: 600px; max-width: 90vw; height: 300px; margin: 0 auto; border: black solid 1px;}
```

De elementen met `class="buiten"` binnen het element met `id="onbekend-padding"`. De buitenste `<div>` binnen `section#onbekend-padding` (het voorbeeld onder het kopje 'padding met translate()').

```
box-sizing: border-box;
```

Bij `.buiten` heeft `div.buiten` een breedte van 600 px en een hoogte van 300 px gekregen. Hieronder wordt aan de `<div>` aan alle kanten een padding gegeven. Standaard worden een border en een padding bij de breedte en hoogte opgeteld. De `<div>` zou daardoor veel breder en hoger worden dan de opgegeven breedte en hoogte.

Met de hier opgegeven waarde bij `box-sizing` komen border en padding binnen de opgegeven breedte en hoogte te staan, waardoor het element niet breder en hoger wordt dan de opgegeven breedte en hoogte.

padding: 150px min(300px, 45vw);

Omdat voor links en onder geen waarden zijn opgegeven, krijgen die dezelfde waarde als boven en rechts. Hier staat dus eigenlijk `150px min(300px, 45vw)` `150px min(300px, 45vw)` in de volgorde boven – rechts – onder – links.

Bij [buiten](#) heeft `div.buiten` een hoogte van 300 px gekregen. Boven en onder wordt hier een padding van 150 px gegeven. Die worden vanwege de `box-sizing: border-box`; hier gelijk boven van de hoogte afgetrokken. Daardoor houd je een hoogte van precies 0 px over, precies tussen de padding van 150 px boven en onder in. Dit punt ligt dus verticaal gecentreerd in `div.buiten`.

Links en rechts een padding van `min(300px, 45vw)`. Die is iets ingewikkelder, omdat niet een simpele waarde als 300 px wordt opgegeven. Dat zou prima kunnen in bredere browservensters, want daarin passen twee paddings van 300 px naast elkaar. Maar in smallere vensters ontstaan er dan problemen.

Bij [buiten](#) heeft `div.buiten` een breedte van 600 px en een maximumbreedte van 90 vw (90% van de breedte van het browservenster) gekregen. Als het venster minimaal ongeveer 667 px breed is, kan `div.buiten` een breedte van 600 px krijgen. Zodra het venster smaller is dan ongeveer 667 px wordt de maximumbreedte van 90 vw minder dan 600 px en wordt de breedte van `div.buiten` beperkt tot 90% van de breedte van het venster.

Als `div.buiten` smaller wordt dan 600 px, passen er geen twee paddings van elk 300 px breed naast elkaar meer in. In dat geval botst de breedte (minder dan 600 px) met de twee paddings (samen 600 px breed). Als de padding breder is dan de maximumbreedte van het element, wordt de maximumbreedte (en ook een gewone breedte) genegeerd: `div.buiten` blijft dan gewoon 600 px breed, waardoor je horizontaal zou moeten scrollen om de hele `<div>` te zien.

Daarom moet in smallere browservensters de padding links en rechts worden versmald. En dat is precies, waar de `min()`-functie voor is.

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 300px. Dit is de maat die is bedoeld voor browservensters, waarin `div.buiten` 600 px breed kan zijn. Daarin is ruimte voor 2 paddings van elk 300 px breed naast elkaar.

Achter de komma staat 45 vw. 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 45 vw is 45% van de breedte. Zodra het venster smaller is dan ongeveer 667 px, is 45% daarvan minder dan de 300 px voor de komma.

In browservensters smaller dan ongeveer 667 px is de breedte van 45 vw achter de komma kleiner dan de 300 px voor de komma. In die vensters wordt de breedte van de padding beperkt tot 45% van de breedte van het venster. Omdat de breedte van `div.buiten` in die vensters wordt beperkt tot 90% van de breedte van het venster, zijn de verhoudingen nog steeds hetzelfde: de paddings grenzen nog steeds precies in het midden van `div.buiten` aan elkaar.

In alle gevallen zijn de twee paddings even breed, ongeacht de breedte van `div.buiten`. En in alle gevallen vullen de paddings de hele breedte van `div.buiten`. De grens tussen de twee paddings is dus precies het horizontale midden van `div.buiten`.

De padding aan de bovenkant komt precies tot het verticale midden van `div.buiten`. Die padding is altijd 150 px hoog. De in `div.buiten` zittende `div.binnen` wordt tegen die padding aangezet: verticaal gecentreerd.

De padding links komt precies tot het horizontale midden van `div.buiten`, ongeacht de breedte van `div.buiten`. De breedte van die padding hangt af van de breedte van `div.buiten`, maar is altijd precies de helft van de breedte van `div.buiten`. De in `div.buiten` zittende `div.binnen` wordt tegen die padding aangezet: horizontaal gecentreerd.

Hierdoor komt de linkerbovenhoek van `div.binnen` precies in het midden van `div.buiten` te staan: horizontaal en verticaal gecentreerd.

```
position: relative;
```

Om nakomelingen van `div.buiten` te kunnen positioneren ten opzichte van `div.buiten`, moet de `<div>` zelf een andere positie dan statisch hebben. Een relatieve positie heeft verder geen invloed op `div.buiten` zelf, omdat niets voor top e.d. wordt opgegeven.

#onbekend-padding .binnen

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.

```
.binnen {background: cyan; color: black; border: black solid 1px;}
```

De elementen met `class="binnen"` binnen het element met `id="onbekend-padding"`. De binnenste `<div>` binnen `section#onbekend-padding` (het voorbeeld onder het kopje 'padding met translate()').

```
background: white;
```

Witte achtergrond.

```
color: black;
```

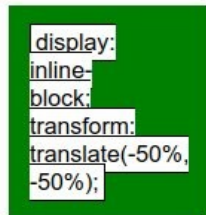
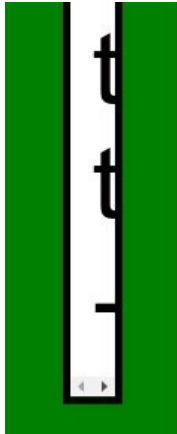
Voorgrondkleur zwart. Dit is o.a. de kleur van de tekst.

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.

Dit is ook al bij `<body>` opgegeven, maar sommige mensen hebben bij alle elementen de kleuren veranderd. Het heeft immers weinig zin, als ze dat alleen bij de body doen, terwijl de sitebouwer de kleuren ook bij bijvoorbeeld de paragrafen heeft aangepast.

display: inline-block;



Een `<div>` is van zichzelf een blok-element. Een blok-element wordt normaal genomen even breed als z'n ouder. Die ouder is hier `div.buiten`.

Zoals iets hierboven bij [padding: 150px min\(300px, 45vw\)](#); beschreven is `div.buiten` volledig opgevuld met paddings, waardoor er geen enkele vrije ruimte meer over is voor de erin zittende `div.binnen`.

Als `div.buiten` als een blok-element wordt weergegeven, wordt de breedte van `div.buiten` daardoor 0 px: meer ruimte is er niet. Dat je toch iets ziet, komt door de padding van 3 px

die iets hieronder wordt opgegeven. Als je meer wilt zien, moet je scrollen. Dit ziet eruit zoals op de afbeelding links. De pagina is vijf keer vergroot, en onderaan staat inderdaad een minuscule horizontale scrollbar.

Als je van `div.binnen` een inline-element maakt, ziet dat eruit zoals op de rechterafbeelding: de tekst wordt weergegeven, maar de achtergrond blijft beperkt tot de tekst. (De lijntjes aan de onder- en bovenkant en aan het begin en eind zijn de bij [.binnen](#) opgegeven border. Die ziet er bij een inline-element heel anders uit dan bij een blok-element.)

Door van de `<div>` een inline-block te maken, wordt de tekst weergegeven én wordt de achtergrond over de volle breedte van de tekst weergegeven.

max-height: 300px;

Maximumhoogte. Als de lettergrootte wordt verhoogd, zou de tekst buiten `div.buiten` kunnen komen te staan en daardoor andere delen van de pagina afdekken. Daarom wordt hier een maximumhoogte opgegeven. (Om dit te voorkomen is ook nog `overflow: auto`; gelijk hieronder nodig.)

overflow: auto;

Als de tekst wordt vergroot, kan deze hoger worden dan `div.buiten`, die bij [.buiten](#) een hoogte van 300 px heeft gekregen. Standaard wordt die tekst dan toch weergegeven. Mogelijk verstoort het de lay-out, maar er verdwijnt in ieder geval geen tekst.

Maar daardoor zou deze tekst beneden en boven `div.buiten` komen te staan en mogelijk andere delen van de pagina afdekken. Door deze waarde kan tekst worden gescrold, als deze hoger wordt dan de gelijk hierboven opgegeven maximumhoogte van 300 px. Afhankelijk van browser en besturingssysteem kan hierbij een verticale scrollbar verschijnen.

Dit is ook de reden dat dit tot de essentiële css wordt gerekend: het is niet de bedoeling dat bij een grotere letter mogelijk andere delen van de pagina verdwijnen achter de `<div>`.

font-size: 0.9em;

Iets kleinere letter. Als eenheid wordt de relatieve eenheid `em` 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.

padding: 3px;

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

transform: translate(-50%, -50%);

Zoals iets hierboven bij padding: 150px min(300px, 45vw); beschreven, is de linkerbovenhoek van `div.binnen` met behulp van de padding in `div.buiten` horizontaal en verticaal in het midden van `div.buiten` gezet.

Als je nou `div.binnen` de helft van z'n eigen breedte en hoogte terug naar links en naar boven zet, staat `div.binnen` horizontaal en verticaal precies gecentreerd binnen `div.buiten`.

Met de bij transform horende functie `translate()` kun je 'n element ten opzichte van zichzelf verplaatsen. En als je die verplaatsing in procenten opgeeft, gelden die procenten ten opzichte van het element zelf. Omdat de browser bij het maken van de pagina alle maten weet, kan de browser altijd uitrekenen, hoeveel 50% is.

Het getal voor de komma verplaatst in horizontale richting, het getal na de komma in verticale richting. Omdat de waarde negatief is, wordt naar boven en naar links verplaatst. Met 50%, precies de helft van de breedte en de hoogte van `div.binnen`. Oftewel: de helft van `div.binnen` staat nu links van het midden van `div.buiten`, en de andere helft staat rechts daarvan: horizontaal gecentreerd. In verticale richting geldt hetzelfde: de helft van `div.binnen` staat boven het midden van `div.buiten`, de andere helft staat eronder.

Hierdoor staat `div.binnen` horizontaal en verticaal gecentreerd binnen `div.buiten`. En omdat de grootte van de verplaatsing in procenten is opgegeven, maakt de grootte van `div.binnen` niet uit: 50% is altijd de helft.

css voor vensters minimaal 760 px breed

`@media screen and (min-width: 760px)`

De css die hier tot nu toe staat, geldt voor alle browservensters. De css die binnen deze 'media query' staat, geldt alleen voor vensters die minimaal 760 px breed zijn. In deze bredere vensters worden wat maten aangepast, omdat hier meer ruimte is.

`@media`: geeft aan dat het om css gaat die alleen van toepassing is, als aan bepaalde voorwaarden wordt voldaan. Al langer bestond de mogelijkheid om met behulp van zo'n `@media`-regel css voor bijvoorbeeld printers op te geven. css3 heeft dat uitgebreid tot veel meer eigenschappen, zoals de breedte en hoogte van het venster van de browser.

`screen`: deze regel geldt alleen voor schermweergave.

`and`: er komt nog een voorwaarde, waaraan moet worden voldaan.

`(min-width: 760px)`: het browservenster moet minimaal 760 px breed zijn. Is het venster smaller, dan wordt de css die binnen deze media query staat genegeerd. Gelijk na deze regel komt een `{` te staan, en aan het einde van de css die binnen deze query valt een bijbehorende afsluitende `}`. Die zijn in de regel hierboven weggefallen, maar het geheel ziet er zo uit:

```
@media screen and (min-width: 760px) {  
    body {color: silver;}  
    (...) rest van de css voor deze media query (...)  
    footer {color: gold;}  
}
```

Voor de eerste css binnen deze media query staat dus een extra `{`, en aan het eind staat een extra `}`.

Als je nou 'n mobieltje hebt met een resolutie van – om maar wat te noemen – 1024 x 768 px, dan geldt deze media query toch niet voor dat mobieltje. Terwijl dat toch echt meer dan 760 px breed is. Een vuig complot van gewetenloze multinationals? Voordat je je gaat beklagen bij Radar, zou ik eerst even verder lezen.

Steeds meer mobiele apparaten, maar ook steeds meer gewone beeldschermen, hebben een hogere resolutiedichtheid. Dat wil zeggen dat ze kleinere pixels hebben, die dichter bij elkaar staan. Daardoor zijn foto's, tekst, e.d. veel scherper weer te geven. Hoe kleiner de puntjes (de pixels) zijn, waaruit een afbeelding is opgebouwd, hoe duidelijker het wordt. Er ontstaat alleen één probleem: als je de pixels twee keer zo klein maakt, wordt ook wat je ziet twee keer zo klein. En inmiddels zijn er al apparaten met pixels die meer dan vier keer zo klein zijn. Een lijntje van 1 px breed zou op die apparaten minder dan 'n kwart van de oorspronkelijke breedte krijgen en vrijwel onzichtbaar zijn. Een normale foto zou in een thumbnail veranderen. Kolommen zouden heel smal worden. Tekst zou onleesbaar klein worden. Allemaal fantastisch scherp, maar je hebt 'n vergrootglas nodig om 't te kunnen zien.

Om dit te voorkomen wordt een verschil gemaakt tussen css-pixels en schermpixels (in het Engels 'device-pixels'). De css-pixels zijn gebaseerd op de – tot voor kort – normale beeldschermen van de desktop. 1 css-pixel is op zo'n beeldscherm 1 pixel. Het aantal schermpixels is het werkelijk op het apparaat aanwezige aantal pixels (dat is het aantal pixels, waarvoor je hebt betaald).

Dat eerder genoemde mobieltje van 1024 x 768 px heeft wel degelijk het aantal pixels, waarvoor je hebt betaald. Maar die zitten dichter bij elkaar. Op een gewoon beeldscherm zitten 96 pixels per inch, wat wordt uitgedrukt met de eenheid ppi ('pixels per inch'). (Vaak wordt foutief de eenheid dpi ('dots per inch') gebruikt. Die eenheid is voor printers.) Als dat mobieltje een resolutie van 192 ppi heeft, 192 pixels per inch, zijn de pixels ervan twee keer zo klein als op een origineel beeldscherm. Er zijn per inch twee keer zoveel schermpixels aanwezig.

Om nu te voorkomen dat alles op dat mobieltje twee keer zo klein wordt, geeft het mobieltje niet het echte aantal schermpixels (1024 x 768), maar een lager aantal css-pixels door bij een media query. De 192 ppi van het mobieltje is twee keer zo veel als de 96 ppi van een normaal beeldscherm. Het aantal css-pixels is dan het aantal schermpixels gedeeld door 2. 1024 x 768 gedeeld door 2 is 512 x 384 px. Het aantal css-pixels is 512 x 384 px en zit daarmee dus ruim onder de 760 px van deze media query. Je bent dus niet opgelicht, of in ieder geval niet wat betreft het aantal pixel.

Door deze truc is een lijn van 1 px breed op een normaal beeldscherm ook op het mobieltje nog steeds 1 px breed, alleen wordt die ene (css-)pixel opgebouwd uit twee schermpixels (feitelijk vier, want het verhaal geldt voor breedte én hoogte). De dikte van het lijntje is hetzelfde, maar het is veel fijner opgebouwd. Bij lijntjes is dat verschil bijvoorbeeld in bochten goed te zien.

Hetzelfde verhaal geldt voor hogere resoluties. Een tablet met een breedte van 4096 schermpixels en een ppi van 384 (vier keer de originele dichtheid) geeft 4096 gedeeld door 4 = 1024 css-pixel door. Het lijntje van 1 px breedte op de originele monitor is nog steeds 1 css-pixel breed op de tablet, maar die ene css-pixel is nu opgebouwd uit zestien schermpixel. (Overigens kun je met behulp van media query's ook testen op de resolutie met gebruik van het sleutelwoord 'resolution'. Dit kan bijvoorbeeld handig zijn om te bepalen, hoe groot een foto moet zijn.)

Kort samengevat: omdat niet het aantal schermpixels (waarvoor je hebt betaald), maar het aantal css-pixels (de door de ontwerper bedoelde afmeting) wordt doorgegeven, wordt voorkomen dat een hogeresolutiescherm onleesbaar klein wordt.

body

Deze selector werkt alleen in browservensters minimaal 700 px breed. Voor andere vensters is de uitleg hieronder niet van belang.

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. (Alleen wat binnen deze media query geldig is, wordt binnen dit blokje herhaald.)

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

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.

`font-size: 110%;`

Iets groter dan standaard in deze grotere browservensters. '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.

section

Deze selector werkt alleen in browservensters minimaal 700 px breed. Voor andere vensters is de uitleg hieronder niet van belang.

Voor deze elementen 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. (Alleen wat binnen deze media query geldig is, wordt binnen dit blokje herhaald.)

`section` {margin-bottom: 15px;}

Alle `<section>`'s. De voorbeelden zijn opgedeeld in vier groepen, die elk in een `<section>` staan. Binnen die `<section>` staat elk voorbeeld ook weer in een eigen `<section>`.

`margin-bottom: 30px;`

Wat afstand tot de volgende `<section>`

#fixed, #tabel-text-align

Deze selector werkt alleen in browservensters minimaal 700 px breed. Voor andere vensters is de uitleg hieronder niet van belang.

Voor deze elementen 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. (Alleen wat binnen deze media query geldig is, wordt binnen dit blokje herhaald.)

`section` {margin-bottom: 15px;}

`section` {margin-bottom: 30px;}

De elementen met `id="fixed"` en `id="tabel-text-align"`.

Dit zijn de twee `<section>`'s met de kopjes 'position: fixed met margin: auto' en 'css-tabel met text-align: center'.

`margin-bottom: 14px;`

Bij de twee voorbeelden die gelijk onder deze twee `<section>`'s staan, staat onder het kopje nog een onderkopje. Het verkleinen van de afstand tot die twee voorbeelden helpt om de verdeling van de voorbeelden gelijk te houden: ze staan allemaal op dezelfde afstand. Zonder deze correctie zouden de twee voorbeelden met de subkopjes op grotere afstand van de `<section>` erboven staan, wat er minder netjes uitziet.

h1

Deze selector werkt alleen in browservensters minimaal 700 px breed. Voor andere vensters is de uitleg hieronder niet van belang.
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. (Alleen wat binnen deze media query geldig is, wordt binnen dit blokje herhaald.)

```
h1 {font-size: 1em; text-align: center; margin-bottom: 0;}
```

Alle `<h1>`'s. Dat is er maar één: de belangrijkste kop van de pagina.

```
font-size: 1.3em;
```

In deze grotere browservensters mag de `<h1>` een grotere letter krijgen. Als eenheid wordt de relatieve eenheid `em` 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.

h2

Deze selector werkt alleen in browservensters minimaal 700 px breed. Voor andere vensters is de uitleg hieronder niet van belang.
Voor deze elementen 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. (Alleen wat binnen deze media query geldig is, wordt binnen dit blokje herhaald.)

```
h2, h3 {font-size: 0.9em; text-align: center; margin-bottom: 15px;}
```

Alle `<h2>`'s.

```
font-size: 1.2em;
```

In deze grotere browservensters mogen de `<h2>`'s een grotere letter krijgen. Als eenheid wordt de relatieve eenheid `em` 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.

h3

Deze selector werkt alleen in browservensters minimaal 700 px breed. Voor andere vensters is de uitleg hieronder niet van belang.
Voor deze elementen 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. (Alleen wat binnen deze media query geldig is, wordt binnen dit blokje herhaald.)

```
h2, h3 {font-size: 0.9em; text-align: center; margin-bottom: 15px;}
```

```
h3 {font-weight: normal; margin: 0;}
```

Alle `<h3>`'s.

```
font-size: 1.2em;
```

In deze grotere browservensters mogen de `<h3>`'s een grotere letter krijgen. Als eenheid wordt de relatieve eenheid `em` 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-bottom: 5px;
```

Kleine afstand tussen kopje en het eronder staande voorbeeld.

#tabel-text-align h3, #tabel-marge h3

Deze selector werkt alleen in browservensters minimaal 700 px breed. Voor andere vensters is de uitleg hieronder niet van belang.

Voor deze elementen 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. (Alleen wat binnen deze media query geldig is, wordt binnen dit blokje herhaald.)

[h2, h3](#) {font-size: 0.9em; text-align: center; margin-bottom: 15px;}

[h3](#) {font-weight: normal; margin: 0;}

[h3](#) {font-size: 1.2em; margin-bottom: 5px;}

De <h3>'s in de elementen met id="tabel-text-align" en id="tabel-marge". Dit zijn de kopjes 'css-tabel met text-align: center' en 'css-tabel met marge'.

margin-bottom: 0;

Deze twee kopjes hebben een onderkopje. Het weghalen van de marge aan de onderkant helpt om de verdeling van de voorbeelden gelijk te houden: ze staan allemaal op dezelfde afstand. Zonder deze correctie zouden de twee voorbeelden met de subkopjes op grotere afstand van de <section> erboven staan, wat er minder netjes uitziet.

#tabel-text-align h3 span, #tabel-marge h3 span

Deze selector werkt alleen in browservensters minimaal 700 px breed. Voor andere vensters is de uitleg hieronder niet van belang.

Voor deze elementen 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. (Alleen wat binnen deze media query geldig is, wordt binnen dit blokje herhaald.)

[span](#) {background: white; color: black; display: inline-block; font-size: 0.9em; border-bottom: black solid 1px; padding: 3px;}

[#tabel-text-align h3 span, #tabel-marge h3 span](#) {background: none; border: none; transform: translateY(-5px);}

De 's binnen de <h3>'s binnen het element met id="tabel-text-align" en de 's binnen de <h3>'s binnen het element met id="tabel-marge".

Dit zijn de twee onderkopjes met 'zie uitleg voor alle css'.

font-size: 0.8em;

Tekst iets kleiner weergeven. Zonder deze verkleining zouden de onderkopjes even groot zijn als de rest van de kop. Als eenheid wordt de relatieve eenheid em 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.

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="Vijftien manieren om een blok-
    element horizontaal én verticaal te centreren - voorbeeld.
    Gebruikt alleen html en css.">
  <title>Vijftien manieren om een blok-element horizontaal én
    verticaal te centreren - voorbeeld</title>
  <link rel="stylesheet"
    href="068-css-dl/positioneren-068-dl.css">
</head>
<body>
<main>
  <h1>Vijftien manieren om een blok-element horizontaal én
    verticaal te centreren</h1>
  <p>(De buitenste &lt;div> heeft overal de afmetingen width:
    600px; max-width: 90vw; height: 300px;)</p>
  <section id="bekend">
    <h2>Binnenste &lt;div> heeft een vaste breedte en hoogte
      van 200 px</h2>
    <section id="alleen-marge">
      <h3>Met alleen marge</h3>
      <div class="buiten">
        <div class="binnen">
          <span tabindex="0"><span>Gecentreerde element:
            </span>width: 200px; height: 200px; overflow: auto;
            margin: 50px auto;</span>
          </div>
        </div>
      </section>
    <section id="marge-padding">
      <h3>Met marge en padding</h3>
      <div class="buiten">
        <span><span>Container: </span>box-sizing: border-box;
          padding-top: 50px;</span>
        <div class="binnen">
```

```

        <span tabindex="0"><span>Gecentreerde element:
        </span>width: 200px; height: 200px; overflow: auto;
        margin: 0 auto;</span>
    </div>
</div>
</section>
<section id="absolute-negatieve-marge">
    <h3>position: absolute met negatieve marge</h3>
    <div class="buiten">
        <span><span>Container: </span>position: relative;</span>
        <div class="binnen">
            <span tabindex="0"><span>Gecentreerde element:
            </span>width: 200px; height: 200px; overflow: auto;
            position: absolute; top: 50%; left: 50%; margin:
            -100px 0 0 -100px;</span>
        </div>
    </div>
</section>
<section id="absolute-calc">
    <h3>position: absolute met calc()</h3>
    <div class="buiten">
        <span><span>Container: </span>position: relative;</span>
        <div class="binnen">
            <span tabindex="0"><span>Gecentreerde element:
            </span>width: 200px; height: 200px; overflow: auto;
            position: absolute; top: calc(50% - 100px); left:
            calc(50% - 100px);</span>
        </div>
    </div>
</section>
<section id="absolute-breedte-marge">
    <h3>position: absolute met margin: auto</h3>
    <div class="buiten">
        <span><span>Container: </span>position: relative;</span>
        <div class="binnen">
            <span tabindex="0"><span>Gecentreerde element:
            </span>width: 200px; height: 200px; overflow: auto;
            margin: auto auto; position: absolute; top: 0;
            right: 0; bottom: 0; left: 0;</span>
        </div>
    </div>
</section>
<section id="fixed">
    <h3>position: fixed met margin: auto</h3>
    <div class="buiten">
        <span><span>Container: </span>transform:
        translateX(0);</span>
        <div class="binnen">

```

```

        <span tabindex="0"><span>Gecentreerde element:
        </span>overflow: auto; margin: auto auto; position:
        fixed; top: 0; right: 0; bottom: 0; left: 0;</span>
    </div>
</div>
</section>
<section id="tabel-text-align">
    <h3><abbr lang="en" title="cascading style
    sheets">css</abbr>-tabel met text-align:
    center<br><span>(zie uitleg voor alle
    <abbr>css</abbr>)</span></h3>
    <div class="buiten">
        <span><span>Container: </span>display: table-cell;
        text-align: center; vertical-align: middle;</span>
        <div class="binnen">
            <span tabindex="0"><span>Gecentreerde element:
            </span>display: inline-block; width: 200px; height:
            200px; overflow: auto; text-align: left;</span>
        </div>
    </div>
</section>
<section id="tabel-marge">
    <h3><abbr>css</abbr>-tabel met marge<br><span>(zie uitleg
    voor alle <abbr>css</abbr>)</span></h3>
    <div class="buiten">
        <span><span>Container: </span>display: table-cell;
        vertical-align: middle;</span>
        <div class="binnen">
            <span><span>Gecentreerde element: </span>width: 200px;
            height: 200px; overflow: auto; margin: 0
            auto;</span>
        </div>
    </div>
</section>
<section id="flexbox">
    <h3>flexbox</h3>
    <div class="buiten">
        <span><span>Container: </span>display: flex;
        justify-content: center; position: relative;</span>
        <div class="binnen">
            <span tabindex="0"><span>Gecentreerde element:
            </span>width: 200px; height: 200px; overflow: auto;
            align-self: center;</span>
        </div>
    </div>
</section>
<section id="grid">
    <h3>grid</h3>

```

```

<div class="buiten">
  <span><span>Container: </span>display: flex;
    justify-content: center; position: relative;</span>
  <div class="binnen">
    <span tabindex="0"><span>Gecentreerde element:
      </span>width: 200px; height: 200px; overflow: auto;
      align-self: center;</span>
  </div>
</div>
</section>
</section>
<section id="combinatie">
  <h2>Binnenste <div> heeft een vaste breedte van 200 px
    en een relatieve hoogte</h2>
  <p>(De relatieve hoogte is ten opzichte van de buitenste
    <div>)</p>
  <section id="text-align-inline-block">
    <h3>text-align met display: inline-block</h3>
    <div class="buiten">
      <span><span>Container: </span>height: auto; line-height:
        20rem; text-align: center;</span>
      <div class="binnen">
        <span tabindex="0"><span>Gecentreerde element:
          </span>width: 200px; height: auto; display:
            inline-block; line-height: normal; text-align: left;
            vertical-align: middle; padding-bottom:
            100px;</span>
      </div>
    </div>
  </section>
</section>
<section>
  <h2>Binnenste <div> heeft een relatieve breedte en
    hoogte</h2>
  <p>(De relatieve breedte en hoogte is ten opzichte van de
    buitenste <div>)</p>
  <section id="percentage">
    <h3>Alleen percentages</h3>
    <div class="buiten">
      <span><span>Container: </span>box-sizing: border-box;
        padding: 50px 0;</span>
      <div class="binnen">
        <span tabindex="0"><span>Gecentreerde element:
          </span>width: 50%; height: 50%; max-height: 50%;
          overflow: auto; margin: 0 auto; transform:
            translateY(50%);</span>
      </div>
    </div>
  </section>
</div>

```

```

</section>
<section id="absolute-translate">
  <h3>position: absolute met translate()</h3>
  <div class="buiten">
    <span><span>Container: </span>position: relative;</span>
    <div class="binnen">
      <span tabindex="0"><span>Gecentreerde element:
        </span>max-height: 30%; overflow: auto;
        padding-bottom: 80px; position: absolute; top: 50%;
        left: 50%; transform: translate(-50%, -50%);</span>
      </div>
    </div>
  </section>
</section>
<section>
  <h2>Binnenste &lt;div&gt; heeft een onbekende breedte en
    hoogte</h2>
  <section id="onbekend-absolute">
    <h3>position: absolute met marge en translate()</h3>
    <div class="buiten">
      <span><span>Container: </span>position: relative;</span>
      <div class="binnen">
        <span tabindex="0">Binnenste (onzichtbare)&lt;div&gt;;
          width: 0; height: 0; margin: auto auto; position:
          absolute; top: 0; right: 0; bottom: 0; left:
          0;<br><br>span: display: inline-block;
          transform:&nbsp; translate(-50% -50%);</span>
        </div>
      </div>
    </section>
  <section id="onbekend-padding">
    <h3>padding met translate()</h3>
    <div class="buiten">
      <span><span>Container: </span>border-box; padding: 150px
        min(300px, 45vw);</span>
      <div class="binnen" tabindex="0"><span>Gecentreerde
        element: </span>display: inline-block; transform:
        translate(-50%, -50%);</div>
      </div>
    </div>
  </section>
</section>
</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.)

```
/* positioneren-068-dl.css */

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

main {padding: 5px;}

abbr[title] {
    text-decoration: none;
    border-bottom: none;
}

abbr[title]::after {
    content: " (" attr(title) ")";
    font-size: 0.9em;
    font-style: italic;
}

h1 {
    font-size: 1em;
    text-align: center;
    margin-bottom: 0;
}

p {
    text-align: center;
    font-size: 0.9em;
    margin: 0;
}

h2 + p {margin: -10px 0 15px;}

section {margin-bottom: 15px;}

h2, h3 {
    font-size: 0.9em;
    text-align: center;
    margin-bottom: 15px;
}
```

```
h3 {
    font-weight: normal;
    margin: 0;
}

.buiten {
    background: green;
    color: black;
    width: 600px;
    max-width: 90vw;
    height: 300px;
    margin: 0 auto;
    border: black solid 1px;
}

.binnen {
    background: cyan;
    color: black;
    border: black solid 1px;
}

#bekend .binnen, #combinatie .binnen {
    width: 200px;
    height: 200px;
    overflow: auto;
}

span {
    background: white;
    color: black;
    display: inline-block;
    font-size: 0.9em;
    border-bottom: black solid 1px;
    padding: 3px;
}

.buiten > span {
    opacity: 0.8;
    line-height: normal;
    text-align: left;
    border-right: black solid 1px;
    position: absolute;
    top: 0;
    left: 0;
    z-index: 10;
    border-right: black solid 1px;
}
```

```
span span, #onbekend-padding .binnen span {  
    position: absolute;  
    left: -20000px;  
}
```

```
#alleen-marge .binnen {margin: 50px auto;}
```

```
#marge-padding .buiten {  
    box-sizing: border-box;  
    padding-top: 50px;  
    position: relative;  
}
```

```
#marge-padding .binnen {margin: 0 auto;}
```

```
#absolute-negatieve-marge .buiten {position: relative;}
```

```
#absolute-negatieve-marge .binnen {  
    margin: -100px 0 0 -100px;  
    position: absolute;  
    top: 50%;  
    left: 50%;  
}
```

```
#absolute-calc .buiten {position: relative;}
```

```
#absolute-calc .binnen {  
    position: absolute;  
    top: calc(50% - 100px);  
    left: calc(50% - 100px);  
}
```

```
#absolute-breedte-marge .buiten {position: relative;}
```

```
#absolute-breedte-marge .binnen {  
    margin: auto auto;  
    position: absolute;  
    top: 0;  
    right: 0;  
    bottom: 0;  
    left: 0;  
}
```

```
#fixed .buiten {transform: translateX(0);}
```

```
#fixed .binnen {  
    margin: auto auto;  
    position: fixed;  
    top: 0;
```

```
        right: 0;
        bottom: 0;
        left: 0;
    }

#tabel-text-align, #tabel-marge {
    width: 600px;
    max-width: 90vw;
    margin: 0 auto 30px;
}

#tabel-text-align h3 span, #tabel-marge h3 span {
    background: none;
    border: none;
    transform: translateY(-5px);
}

#tabel-text-align .buiten {
    display: table-cell;
    text-align: center;
    vertical-align: middle;
    position: relative;
}

#tabel-text-align .binnen {
    display: inline-block;
    text-align: left;
}

#tabel-marge .buiten {
    display: table-cell;
    vertical-align: middle;
    position: relative;
}

#tabel-marge .binnen {margin: 0 auto;}

#flexbox .buiten {
    display: flex;
    justify-content: center;
    position: relative;
}

#flexbox .binnen {align-self: center;}

#grid .buiten {
    display: grid;
    justify-content: center;
    position: relative;
}
```

```
}

#grid .binnen {align-self: center;}

#text-align-inline-block .buiten {
    height: 20rem;
    text-align: center;
    line-height: 20rem;
    position: relative;
}

#text-align-inline-block .binnen {
    display: inline-block;
    height: auto;
    max-height: 20rem;
    line-height: normal;
    text-align: left;
    vertical-align: middle;
    padding-bottom: 60px;
}

#percentage .buiten {
    box-sizing: border-box;
    padding: 50px 0;
    position: relative;
}

#percentage .binnen {
    width: 33%;
    min-width: 200px;
    height: 100%;
    overflow: auto;
    margin: 0 auto;
}

#absolute-translate .buiten {position: relative;}

#absolute-translate .binnen {
    max-height: 30%;
    overflow: auto;
    padding-bottom: 80px;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}

#onbekend-absolute .buiten {position: relative;}
```

```

#onbekend-absolute .binnen {
    width: 0;
    height: 0;
    margin: auto auto;
    position: absolute;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
}

#onbekend-absolute .binnen span {
    max-height: 300px;
    overflow: auto;
    border: black solid 1px;
    transform: translate(-50%, -50%);
}

#onbekend-padding .buiten {
    box-sizing: border-box;
    padding: 150px min(300px, 45vw);
    position: relative;
}

#onbekend-padding .binnen {
    background: white;
    color: black;
    display: inline-block;
    max-height: 300px;
    overflow: auto;
    font-size: 0.9em;
    padding: 3px;
    transform: translate(-50%, -50%);
}

@media screen and (min-width: 760px) {
    body {font-size: 110%;}
    section {margin-bottom: 30px;}
    #fixed, #tabel-text-align {margin-bottom: 14px;}
    h1 {font-size: 1.3em;}
    h2 {font-size: 1.2em;}
    h3 {
        font-size: 1.2em;
        margin-bottom: 5px;
    }
    #tabel-text-align h3, #tabel-marge h3 {margin-bottom: 0;}
    #tabel-text-align h3 span, #tabel-marge h3 span {font-size:
        0.8em;}
}

```