

## Animatie verandert vooruit sneller dan achteruit



### 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](http://www.css-voorbeelden.nl) en medewerkers daarvan op geen enkele manier verantwoordelijk.

- \* Deze uitleg wordt 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](http://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.

- \* 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](#).

Een link naar [www.css-voorbeelden.nl](http://www.css-voorbeelden.nl) wordt trouwens altijd op prijs gesteld.

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 rood gekleurd. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles in een gewone letter vanwege de leesbaarheid.)

### Inhoudsopgave

[Korte omschrijving](#)

[Opmerkingen](#)

[Achterliggend idee](#)

[De voorvoegsels -moz-, -ms- en -webkit-](#)

[Semantische elementen en WAI-ARIA](#)

[Semantische elementen](#)

[WAI-ARIA-codes](#)

[Tabindex](#)

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

[:hover](#)

[:focus](#)

[:active](#)

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

[Toegankelijkheid en zoekmachines](#)

[Specifiek voor dit voorbeeld](#)

[Getest in](#)

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

[Wijzigingen](#)

[Inhoud van de download en licenties](#)

Uitleg code:

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

[<!DOCTYPE html>](#)

[<html lang="nl">](#)

[<meta charset="utf-8">](#)

[<meta name="viewport" content="width=device-width, initial-scale=1">](#)

[<link rel="stylesheet" href="118-css-dl/animatie-118-dl.css">](#)

[<div id="vooruit" tabindex="0" aria-label="Animatie speelt vooruit als div wordt aangeraakt of -geklikt, focus heeft of wordt gehoverd">Vooruit</div>](#)

[<div id="terug" tabindex="0" aria-label="Animatie speelt achteruit als div wordt aangeraakt of -geklikt">Terug</div>](#)

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

[main {display: block; width: 300px;}](#)

[#vooruit {background-color: white; color: blue; height: 280px; transform: rotate\(0deg\); transition: 5s;}](#)

[#vooruit:focus, #vooruit:hover {background-color: black; color: white; transform: rotate\(360deg\); transition: .5s;}](#)

[#terug {line-height: 3em;}](#)

[JavaScript:](#)

[document.getElementById\("vooruit"\).addEventListener\("touchstart", function \(\) {return null; }\);](#)

[document.getElementById\(terug\).addEventListener\("touchstart", function \(\) {return null; }\);](#)

Volledige code:

[HTML](#)

[CSS](#)

[JavaScript](#)

## Korte omschrijving

Met behulp van `transition` een animatie met verschillende snelheden vooruit en achteruit afspelen.

## Opmerkingen

Soms kan deze techniek nuttig zijn voor gebruikers van een muis. Als bijvoorbeeld een submenu wordt geopend door hoveren over een knop, kan het sluiten van het submenu iets worden vertraagd. Als de bezoeker dan per ongeluk met de muis buiten het submenu komt, kan worden teruggegaan naar het submenu. 'n Halve seconde vertraging is al voldoende om 'n eraf floepende muis te kunnen corrigeren.

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

Dit voorbeeld is gemaakt op een systeem met Linux ([Kubuntu](http://kubuntu.org)). Daarbij is vooral gebruik gemaakt van [Komodo Edit](http://kumodoedit.com), [GIMP](http://gimp.org) en [Firefox](http://firefox.com) met extensies. De pdf-bestanden zijn gemaakt met [LibreOffice](http://libreoffice.org).

Vragen of opmerkingen? Fout gevonden? Ga naar het [forum](http://forum).

Iets gevonden waar je wat aan hebt? Mooi. Als je je waardering wilt uiten, maak dan een donatie over aan War Child Nederland, een organisatie die kinderen uit oorlogsgebieden helpt hun trauma's te verwerken. Of - nog beter - wordt donateur:



## Achterliggend idee

Een groot aantal css-eigenschappen kan bij hoveren, aanraken, klikken, e.d. worden veranderd. Normaal genomen gebeurt deze verandering bliksemsnel, maar met behulp van `transition` kan de verandering over een in te stellen tijdsverloop geleidelijk aan plaatsvinden. Een achtergrondkleur bijvoorbeeld kan over een tijdsverloop van bijvoorbeeld tien seconden geleidelijk aan veranderen van zwart naar wit. Eventueel met een vertraging aan het begin, snel beginnend en aan het einde traag, en meer van dat soort dingen.

Als in dit voorbeeld de bovenste `<div>` (`div#vooruit`, de `<div>` met 'Vooruit' erin) wordt geactiveerd door aanraken of -klikken, door erover te hoveren, of door de `<div>` [focus](#) te geven, draait de `<div>` in 'n halve seconde volledig rond. Tevens verandert in die tijd de achtergrondkleur van wit naar zwart, en de tekstkleur van blauw naar wit.

Zodra de onderste `<div>` (`div#terug`, de `<div>` met 'Terug' erin) wordt geactiveerd door aanraken of -klikken, door erover te hoveren, of door de `<div>` focus te geven, wordt weer teruggedaan naar de beginstand: volledig achteruit terugdraaien, witte achtergrond en blauwe tekst. (Afhankelijk van browser en/of besturingssysteem kan dit ook gebeuren, als het scherm ergens buiten het bovenste blok wordt aangeraakt of -geklikt.)

Tot zover is er niets bijzonders aan deze animatie: gewoon vooruit en achteruit afspelen. Tijdsduur, vertraging en verloop van de animatie zijn in beide richtingen precies hetzelfde. Wat echter weinig bekend is: het is heel simpel om beide richtingen een andere tijdsduur, vertraging en/of verloop te geven.

Meestal wordt in de css alleen bij de selector met `:hover`, `:focus`, e.d. `transition` gebruikt. De daar opgegeven waarden bij `transition` worden gebruikt voor het voorwaarts afspelen, en ook voor het achterwaarts terugspelen naar de beginstand.

Als je echter ook bij de selector zonder `:hover` e.d. `transition` gebruikt, worden de daar opgegeven waarden gebruikt bij het terugspelen. Je kunt op deze manier bij voorwaarts afspelen een andere tijdsduur, vertraging en/of verloop opgeven dan bij terugspelen.

In het voorbeeld gaat het om deze twee regels:

```
#vooruit {transition: 5s;}  
#vooruit:focus, #vooruit:hover {transition: .5s;}
```

Het vooruit afspelen duurt slechts 'n halve seconde, het terugspelen duurt vijf seconden. Andere waarden dan de tijdsduur, zoals een vertraging aan het begin, zijn hier niet gebruikt, maar dan kan dus wel.

## De voorvoegsels -moz-, -ms- en -webkit-

Voordat een nieuwe css-eigenschap wordt ingevoerd, is er in de regel een experimentele fase. Browsers passen het dan al toe, maar met een aangepaste naam. Tijdens deze fase

kunnen problemen worden opgelost en worden veldslagen uitgevochten, over hoe de standaard precies moet worden toegepast.

Als iedereen het overal over eens is en alle problemen zijn opgelost, wordt de officiële naam uit de standaard gebruikt.

De belangrijkste browsers hebben elk een eigen voorvoegsel:

Firefox: `-moz-`, naar de maker: Mozilla.

Op webkit gebaseerde browsers, zoals Google Chrome, Opera, Safari en Android browser: `-webkit-`.

(Google Chrome is van webkit overgestapt op een eigen weergave-machine: blink. Blink gaat geen voorvoegsels gebruiken. Het is echter een aftakking van webkit, dus het zal nog wel even duren voor `-webkit-` hier helemaal uit is verdwenen. Ook Opera gebruikt blink.)

Internet Explorer: `-ms-`, naar de maker: Microsoft. (Edge gebruikt geen voorvoegsels, maar vanwege compatibiliteit met oudere sites kunnen er nog wat aanwezig zijn.)

Zodra de experimentele fase voorbij is, wordt het voorvoegsel weggelaten. Omdat dat moment niet bij alle browsers hetzelfde is, zet je nu ook al de officiële naam erbij. Deze wordt als laatste opgegeven. Bijvoorbeeld Android browser herkent `-webkit-linear-gradient`. Zodra Android browser `linear-gradient` gaat herkennen, zal dit `-webkit-linear-gradient` overrulen, omdat het er later in staat. Dat ze er beide in staan, is dus geen enkel probleem.

In dit voorbeeld worden `transform` en `transition` gebruikt.

`transform`:

Op dit moment moet je nog het volgende schrijven:

```
{-webkit-transform: ...; transform: ...;}
```

In de toekomst kun je volstaan met:

```
{transform: ...;}
```

`transition`:

Op dit moment moet je nog het volgende schrijven:

```
{-webkit-transition: ...; transition: ...;}
```

In de toekomst kun je volstaan met:

```
{transition: ...;}
```

Inmiddels is de algemene mening dat 'vendor prefixes', zoals deze voorvoegsels in het Engels heten, geen groot succes zijn. Eén van de grootste problemen: veel sitemakers gebruiken alleen de `-webkit-` variant. Daar kwamen ze in het verleden nog mee weg, omdat Apple op mobiel zo'n beetje 'n monopolie had. Inmiddels is dat niet meer zo, maar deze gewoonte bestaat nog steeds. Waardoor 'n site alleen in op webkit georiënteerde browsers goed is te bekijken.

Dit is zo'n groot probleem dat andere browsers soms de variant met `-webkit-` ook maar zijn gaan implementeren, naast de standaard. Want als 'n site het niet goed doet in 'n bepaalde browser, krijgt in de regel niet de site maar de browser de schuld.

Vanwege alle problemen met 'vendor prefixes' worden deze door steeds meer browsers niet meer gebruikt. Nieuwe, experimentele css-eigenschappen zitten inmiddels in bijvoorbeeld Firefox, Google Chrome en Safari achter een zogenaamde vlag: de gebruiker moet iets veranderen in de instellingen, waarna de eigenschap gebruikt (en getest) kan worden. Als alles werkt, zoals het hoort te werken, schakelt de browsermaker de vlag standaard in.

Voorlopig zijn we echter nog niet van deze voorvoegsels af. Als je ze gebruikt, gebruik dan alle varianten, en eindig met de variant zonder voorvoegsel, zoals die uiteindelijk ooit gebruikt gaat worden. Als je alleen de `-webkit-` variant gebruikt, ben je in feite 'n onbetaalde reclamemaker voor Apple.

## Semantische elementen en WAI-ARIA

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

### Semantische elementen

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

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

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

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

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

`<MAIN>`

Hierbinnen staat de belangrijkste inhoud van de pagina (in dit voorbeeld zijn dit alleen de twee witte `<div>`'s).

Met behulp van dit soort nieuwe semantische elementen kan bijvoorbeeld een schermlezer in één keer een heel menu passeren en gelijk naar de echte inhoud gaan. Alleen hadden deze nieuwe elementen tot voor kort één probleem: ze hadden in de praktijk nog weinig nut, omdat schermlezers e.d. ze nog niet herkenden. Daarom werd een zogenaamde WAI-ARIA-code toegevoegd aan deze elementen. Dat is een

al veel langer bestaande code, die schermlezers e.d. wel herkennen. Voor <main> ziet dat er zo uit:

```
<main role="main">
```

Inmiddels is dit behoorlijk veranderd. Het advies is nu om deze speciale toevoeging niet meer te gebruiken, omdat de meeste schermlezers e.d. dit soort nieuwe elementen inmiddels herkennen.

## WAI-ARIA-codes

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

Er wordt in dit voorbeeld één WAI-ARIA-code gebruikt: `aria-label`.

### ARIA-LABEL

Als een schermlezer deze pagina voorleest, wordt de indrukwekkende tekst 'Vooruit Terug' uitgesproken. Nu zijn sommige moderne dichters wereldberoemd geworden met dit soort teksten, maar schrijvers dezes is helaas geen begenadigd dichter. Te vrezen valt daarom dat deze tekst mogelijk ietwat raadselachtig is.

Als je kunt zien, wat er gebeurt, is de bedoeling van deze twee woorden gelijk duidelijk. Maar als je niet kunt zien, zou je makkelijk kunnen denken dat dit het dagboek van een veerbaas of een opstel van een schommelend kind is. Of het medisch eindoordeel van de ruggengraat van een PvdA-prominent bij een debat over het Kinderpardon.

Met behulp van het attribuut `aria-label` kan een omschrijving worden gegeven, die wordt voorgelezen als de schermlezer de <div> bereikt. Deze omschrijving is niet te zien, dus de lay-out wordt niet verstoord:

```
<div id="vooruit" tabindex="0" aria-label="
  Animatie speelt vooruit als div wordt
  aangeraakt of -geklikt, focus heeft of
  wordt gehoverd">Vooruit</div>
<div id="terug" tabindex="0" aria-label="
  Animatie speelt achteruit als div wordt
  aangeraakt of -geklikt">Terug</div>
```

Uiteraard is dit nog steeds niet ideaal, als je niet (goed) kunt zien, wat er gebeurt. Maar in ieder geval heeft de bezoeker nu enigszins een idee, wat de duistere kreet 'Vooruit Terug' betekent.

## Tabindex

Links, invoervelden in formulieren, e.d. kunnen met behulp van de Tab-toets (of een soortgelijke 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 `tabindex`en dezelfde waarde hebben, wordt de volgorde in de html aangehouden. Een waarde van '0' wordt, afhankelijk van browser en besturingssysteem, verschillend behandeld, waarover iets hieronder bij `Tabindex="0"` meer.

#### **TABINDEX="-1"**

Een negatieve waarde van -1 zorgt ervoor dat het element volledig wordt genegeerd door de Tab-toets. Zelfs een link met een negatieve `tabindex` wordt volledig genegeerd. Normaal genomen heeft een `tabindex="-1"` maar één nut: je kunt dan met behulp van JavaScript toch [focus](#) aan het element geven, zonder dat gebruikers van de Tab-toets erdoor worden gehinderd. In dit voorbeeld wordt `tabindex="-1"` niet gebruikt.

#### **TABINDEX="0"**

Volgens de specificatie van html 4.01 moest een `tabindex="0"` pas worden bezocht, nadat `tabindex`en met een positief nummer waren bezocht. Sommige browsers hielden zich hier echter niet aan: een `tabindex="0"` werd gewoon tussen de positieve `tabindex`en gezet.

In html5 is de situatie nog fijner. Nu staat er alleen dat, wat betreft de volgorde, de gewoonte van het platform wordt gevolgd. Oftewel: doe maar raak. Maar hoe dan ook: 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.

Deze waarde wordt in dit voorbeeld gebruikt bij de twee `<div>`'s:

```
<div id="vooruit" tabindex="0" aria-label="Animatie
    speelt vooruit als div wordt aangeraakt of
    -geklikt, focus heeft of wordt
    gehoverd">Vooruit</div>
<div id="terug" tabindex="0" aria-label="Animatie
    speelt achteruit als div wordt aangeraakt of
    -geklikt">Terug</div>
```

Door de toevoeging `tabindex="0"` kunnen de `<div>`'s focus krijgen bij gebruik van de Tab-toets, terwijl een `<div>` dat normaal genomen niet krijgt. Door als selector `#vooruit:focus` te gebruiken kan de animatie nu afspelen, als de `<div>` focus heeft. Waarmee de animatie ook is af te spelen door gebruikers van de Tab-toets.

`div#terug` krijgt ook een `tabindex`. Bij de eerste keer indrukken van de Tab-toets krijgt `div#vooruit` focus en speelt de animatie af, bij nogmaals indrukken krijgt `div#terug` focus. Daarmee verliest `div#vooruit` de focus en speelt de animatie weer achterwaarts terug.

In dit voorbeeld is geen enkele ander element dat focus kan krijgen op de pagina aanwezig. Daardoor zou `div#vooruit` steeds de focus houden en de animatie niet meer terug kunnen spelen. Om dat te voorkomen is `div#terug` aangebracht. (Bij het indrukken van de Tab-toets wordt ook naar het menu van de browser e.d. gegaan. In sommige browsers verliest `div#vooruit` daarbij de focus en is `div#terug` eigenlijk overbodig. Maar in andere browsers houdt `div#vooruit` focus, ook als de Tab-toets de menubalk e.d. heeft bereikt. In die browsers is `div#terug` onmisbaar.)

Als elders op de pagina nog een `tabindex` wordt gebruikt, kan een `tabindex="0"` problemen geven met de volgorde, zoals iets hierboven beschreven. Op de site zelf, waar ook knoppen voor navigatie op de site aanwezig zijn, gaat het goed. Geen enkele knop heeft daar een `tabindex`, waardoor in alle browsers eerst de navigatieknoppen en dan pas de `<span>`'s met de `tabindex` worden bezocht. Precies zoals de bedoeling is. Maar als je wel een `tabindex` elders op de pagina gebruikt, moet je in zoveel mogelijk browsers op zoveel mogelijk systemen de volgorde bij gebruik van de Tab-toets controleren.

**TABINDEX="..."**

Op de plaats van de puntjes moet een positief getal worden ingevuld: het volgnummer. Positieve `tabindex`en worden in dit voorbeeld 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.

### **: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 hooft 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.

Bij gebruik van een muis zit er een verschil tussen hovern en klikken, maar op een touchscreen is dat verschil er niet: je raakt een touchscreen aan of niet. Dit levert problemen op in iOS. Als de bovenste <div> wordt aangeraakt, speelt de animatie niet af in Safari, Chrome for iOS en UC browser. (In Firefox en Opera Mini speelt de animatie wel volledig af, vooruit en achteruit.)

Dit is vermoedelijk een van de extra's waar Applefans zoveel extra voor betalen. (Ja, sorry, maar ik word 'n beetje gallisch van dat constante geëikel van Apple, in combinatie met het ongelooflijke gebrek aan documentatie, wat bergen extra werk met zich meebrengt.) Een klein stukje [JavaScript](#) zorgt ervoor dat de animatie ook in Safari, Chrome for iOS en UC browser afspelt.

#### :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.

Er is echter 'n tweede manier om naar links, invoervelden, e.d. te gaan: met behulp van de Tab-toets (sommige browsers gebruiken andere toetsen, maar het principe is hetzelfde). Met behulp van de Tab-toets kun je naar links, invoervelden, e.d. 'springen'.

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.

Een <div> wordt normaal genomen overgeslagen door de Tab-toets, maar in dit voorbeeld hebben `div#vooruit` en `div#terug` het attribuut `tabindex="0"` gekregen, waardoor ze wel focus kunnen krijgen. Meer hierover is te vinden bij [Tabindex](#).

#### :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. In dit voorbeeld wordt `:active` niet gebruikt.

### 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 vooral geen FrontPage, Publisher of Word (alle drie van Microsoft). Deze programma's maken niet-standaard code die alleen goed te bekijken is in Internet Explorer. In alle andere browsers zie je grotendeels bagger, als je al iets ziet. Publisher en Word zijn niet bedoeld om websites mee te maken. FrontPage is zwaar verouderd en wordt 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 natuurlijk ook 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 mogelijkheden voor het simuleren van weergave op een kleiner scherm in de ontwikkelgereedschappen ingebouwd. Ook dit blijft een simulatie, maar geeft vaak wel een beter beeld dan zoomen.

- Ik maak zelf het liefst een site in Firefox. Als je 'n site maakt in Firefox, Opera, Safari, Google Chrome of Edge, is er 'n hele grote kans dat hij in alle browsers werkt. Ik geef de voorkeur aan Firefox, omdat er zoveel extensies (uitbreidingen) voor bestaan. Deze zijn te vinden op de site van Mozilla onder [Webontwikkeling](#). Verder is Firefox de enige grote browser die niet bij een bedrijf hoort dat vooral op je centen of 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](http://www.w3.org/QA/2002/04/valid-dtd-list).

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

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

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

- Als tweede voer je de charset in. 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. Ook van Internet Explorer kun je meerdere versies naast elkaar draaien. Op de pagina met [links](#) staan onder de kopjes Gereedschap → Meerdere versies van Internet Explorer draaien en Gereedschap → Weergave testen 'n aantal links die daarbij kunnen helpen. De compatibiliteitsweergave in Internet Explorer is niet geschikt om in te testen, omdat deze enigszins verschilt van de weergave in échte browsers.

- 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 `<br>1<br>2<br>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, zou ik je aanraden de verschillende delen een achtergrondkleur te geven. Je ziet dan goed, waar 'n deel precies staat. Een achtergrondkleur heeft – anders dan bijvoorbeeld een border – verder geen invloed op de lay-out, dus die is hier heel geschikt voor.
  - Als je instellingen verandert in de style, 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 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` padding en border binnen de breedte en hoogte van de inhoud te zetten, als je dat handiger vindt.
  - In plaats van een absolute eenheid als px kun je ook een relatieve eenheid gebruiken, met name em. Voordeel van em is dat een lettergrootte, regelhoogte, e.d. in em in alle browsers kan worden veranderd. Nadeel is dat het de lay-out sneller kan verstoren dan bijvoorbeeld px. Dit moet je gewoon van geval tot geval bekijken. Voor weergave in mobiele apparaten zijn relatieve eenheden als em vrijwel altijd beter dan vaste eenheden als px.  
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.

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](http://validator.w3.org/nu).  
css kun je valideren op: [jigsaw.w3.org/css-validator](http://jigsaw.w3.org/css-validator).

## **Toegankelijkheid en zoekmachines**

Het eerste deel van deze tekst is voor alle voorbeelden hetzelfde. Eventueel specifiek voor dit voorbeeld geldende dingen staan verderop onder het kopje [Specifiek voor dit voorbeeld](#).

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

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

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

Enkele tips die helpen bij toegankelijkheid:

- Gebruik altijd een alt-beschrijving bij een afbeelding. De alt-tekst wordt gebruikt, als afbeeldingen niet kunnen worden getoond of gezien (dat geldt dus ook voor zoekmachines). Als je iets wilt laten zien, als je over de afbeelding hovert, gebruik daar dan het title-attribuut voor, niet de alt-beschrijving.  
Als een afbeelding alleen maar voor de sier wordt gebruikt, zet je daarbij `alt=""`, om aan te geven dat de afbeelding niet belangrijk is voor het begrijpen van de tekst of zo. Dat kun je ook doen, als uit de tekst bij de afbeelding al duidelijk wordt, wat de afbeelding is. Een alt-tekst zou dan 'n beetje dubbelop zijn.
- Als uit de tekst bij een link niet duidelijk blijkt, waar de link naartoe leidt, gebruik dan een title bij de link. Een tekst als 'pagina met externe links' is waarschijnlijk duidelijk genoeg, een tekst als alleen 'links' mogelijk niet. Een duidelijke zwart-witregel is niet te geven, omdat dit ook van tekst e.d. in de omgeving van de link afhangt.
- 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.

Voor mij redenen om accesskeys (vrijwel) niet te gebruiken. Ik kan me wel voorstellen dat ze, op sites die gericht zijn op 'n specifieke groep gebruikers, nog enig nut kunnen hebben. Maar voor algemene sites zou ik zeggen: normaal genomen niet gebruiken.

- Met behulp van de Tab-toets (of op 'n soortgelijke manier) kun je in de meeste browsers door links, invoervelden, e.d. lopen. Elke tab brengt je één link, invoerveld, e.d. verder, Shift+Tab één plaats terug. Met behulp van het attribuut `tabindex` kun je de volgorde aangeven, waarin de Tab-toets werkt. Zonder `tabindex` wordt de volgorde van de html aangehouden bij gebruik van de Tab-toets, maar soms is een andere volgorde logischer. In principe is het beter, als `tabindex` niet nodig is, maar gewoon de volgorde van de html wordt aangehouden. Bij verkeerd gebruik kan `tabindex` heel verwarrend zijn. Het is niet bedoeld om van de pagina een hindernisbaan voor kangoeroes te maken, waarop van beneden via links over rechts naar boven wordt gesprongen.
- Als, zoals hierboven beschreven, een gebruiker van de Tab-toets bij een link, invoerveld, e.d. is aangekomen, wordt dit aangegeven door de link, invoerveld, e.d. extra te markeren met een kadertje. Dat kadertje mag je alleen weghalen, als op een andere manier wordt duidelijk gemaakt, welk element 'focus' heeft. Een gebruiker van de Tab-toets kan anders niet zien, waar zij of hij zit, en welk element gaat reageren op bijvoorbeeld een Enter.
- In het verleden werd vaak aangeraden de volgorde van de code aan te passen. Een menu bijvoorbeeld kon in de html onderaan worden gezet, terwijl het op het scherm met behulp van css bovenaan werd gezet. Inmiddels zijn schermlezers e.d. zo sterk verbeterd dat dit niet meer wordt aangeraden. De volgorde in de html kan tegenwoordig beter hetzelfde zijn als die op het scherm, omdat het anders juist verwarrend kan werken.
- Een zogenaamde skip-link is wel vaak nog zinvol. Dat is een link die je buiten het scherm parkeert met behulp van css, zodat hij normaal genomen niet te zien is. Zo'n link is wel gewoon zichtbaar in speciale programma's zoals tekstbrowsers en schermlezers, want die kijken gewoon naar wat er in de broncode staat.  
Een skip-link staat boven menu, header, e.d. en linkt naar de eigenlijke inhoud van de pagina, zodat mensen met één toetsaanslag naar de eigenlijke inhoud van de pagina kunnen gaan.  
Een skip-link is vooral nuttig voor gebruikers van de Tab-toets. Zodra de normaal genomen onzichtbare link door het indrukken van de Tab-toets focus krijgt, kun je hem op het scherm plaatsen, waardoor hij zichtbaar wordt. Bij een volgende tab wordt hij dan weer buiten het scherm geplaatst en is dus niet meer zichtbaar, zodat de lay-out niet wordt verstoord.  
Op pagina's en in voorbeelden waar dat nuttig is, wordt op deze site een skip-link gebruikt. (Althans: nog niet in alle voorbeelden die daarvoor in aanmerking komen, zit een skip-link. Maar geleidelijk aan worden dat er steeds meer.)
- Van oorsprong is html een taal om wetenschappelijke documenten weer te geven, pas later is hij gebruikt voor lay-out. Maar daar is hij dus eigenlijk nooit voor bedoeld geweest. Het gebruiken van html voor lay-out leidt tot enorme problemen voor gehandicapten en tot een lage plaats in zoekmachines.  
De html hoort alleen inhoud te bevatten, lay-out doe je met behulp van css. Die css moet in een externe stylesheet staan of, als hij alleen voor één bepaalde pagina van toepassing is, in de `<head>` van die pagina.
- Breng een logische structuur aan in je document. Gebruik een `<h1>` voor de belangrijkste kop, een `<h2>` voor een subkop, enz. Schermlezers e.d. kunnen van kop naar kop springen. En een zoekmachine gaat ervan uit dat `<h1>` belangrijke tekst bevat.  
Dit geldt voor al dit soort structuurbepalende tags.  
Als een `<h1>` te grote letters geeft, maak daar dan met behulp van je css 'n kleinere letter van, maar blijf die `<h1>` gewoon gebruiken. Op dezelfde manier kun je al dit soort dingen oplossen.
- `<table>` is fantastisch, maar alleen als die wordt gebruikt om een echte tabel weer te geven, niet als hij voor opmaak wordt misbruikt. In het verleden is dat op grote schaal gebeurd bij

gebrek aan andere mogelijkheden. Een tabel is, als je niet heel erg goed oplet, volstrekt ontoegankelijk voor gehandicapten en zoekmachines. Het lezen van een tabel is ongeveer te vergelijken met het lezen van een krant van links naar rechts: niet per kolom, maar per regel. Dat gaat dus alleen maar goed bij een echte tabel, zoals een spreadsheet. In alle andere gevallen garandeert 'n tabel 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 of SSI (Server Side Includes). 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 beter PHP dan SSI gebruiken, omdat SSI min of meer aan het uitsterven is en PHP veel meer mogelijkheden heeft. Op deze site wordt in enkele voorbeelden nog SSI gebruikt, maar zodra die worden bijgewerkt, gaat dat vervangen worden door PHP.)

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

- 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.

- 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. Het zelfde geldt voor die fantastisch mooie flash-pagina's, als daarbij geen voorzieningen voor dit soort programma's zijn aangebracht.

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](http://invisible-island.net/lynx).

Ook kun je in Windows het gratis programma WebbIE installeren. WebbIE laat de pagina zien, zoals een tekstbrowser e.d. hem zien. WebbIE is te downloaden vanaf [www.webbie.org.uk](http://www.webbie.org.uk).

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

[lowvision.support](http://lowvision.support)

Laat zien hoe een kleurenblinde de site ziet. Engelstalig.

[wave.webaim.org](http://wave.webaim.org)

Deze laat grafisch zien, hoe de toegankelijkheid is. Engelstalig.

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

### Specifiek voor dit voorbeeld

- Zonder JavaScript reageert op iOS in Safari, UC browser en Chrome for iOS `div#vooruit` niet op een aanraking, waardoor de animatie niet kan worden afgespeeld. Met behulp van een klein stukje [JavaScript](#) wordt dit opgelost. Omdat

een iPhone en een iPad zonder JavaScript nogal onbeholpen zijn, zal JavaScript niet vaak uitstaan. Maar het is wel één van de redenen, waarom je nooit belangrijke informatie in een animatie moet verbergen, als die niet ook op een andere manier toegankelijk is.

- Schermlezers hebben niets aan een animatie, die lezen alleen 'Vooruit Terug' voor. Daarom is voor schermlezers met behulp van de [WAI-ARIA-code](#) `aria-label` een korte omschrijving van de bedoeling van de `<div>`'s toegevoegd. Ook om deze reden moet je trouwens nooit belangrijke informatie in een animatie verbergen, als die niet ook op een andere manier toegankelijk is.
- De tekstbrowser Lynx laat alleen 'Vooruit Terug' zien. Ook WebbIE laat alleen 'Vooruit Terug' zien en negeert `aria-label`.
- Zonder css is het resultaat ook niet heel indrukwekkend: op een verder volledig leeg scherm verschijnt de tekst 'Vooruit Terug'. Wat, op een verder volkomen wit scherm, een ietwat dreigende indruk oplevert. Alsof je in je eentje in 'n steegje tussen twee pelotons Mobiele Eenheid zit...
- Voor gebruikers van de [Tab-toets](#) is `tabindex="0"` aangebracht in `div#vooruit` en `div#terug`, zodat de animatie ook kan worden afgespeeld met gebruik van de Tab-toets.

## Getest in

Laatst gecontroleerd op 7 maart 2017.

Onder dit kopje staat alleen maar, hoe en waarin is getest. Eventuele problemen, ook die met betrekking tot zoomen en lettergroottes, staan 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!

Eventuele opmerkingen over de toegankelijkheid specifiek voor dit voorbeeld staan hierboven bij Toegankelijkheid en zoekmachines onder het kopje [Specifiek voor dit voorbeeld](#).

Dit voorbeeld is getest op de volgende systemen:

- Windows 7 (1280 x 1024 px, resolution: 96 dpi):  
Firefox, UC Browser, Google Chrome en Internet Explorer 11, in grotere en kleinere browservensters.
- Windows 8.1 (1366 x 768 px, resolution: 96 dpi):  
Bureaublad-versie: Firefox, UC Browser, Google Chrome en Internet Explorer 11, in grotere en kleinere browservensters.  
Startscherm-versie: Internet Explorer 11.
- Windows 10 (1600 x 900 px, resolution: 96 dpi):  
Firefox, UC Browser, Google Chrome, Internet Explorer 11, Edge, in grotere en kleinere browservensters.
- OS X 10.11.6 ('El Capitan') (1680 x 1050 px, resolution: 96: dpi, device-pixel-ratio: 1):  
Firefox, Safari en Google Chrome, in grotere en kleinere browservensters.
- Linux (Kubuntu 14.04 LTS, 'Trusty Tahr') (1280 x 1024 px, resolution: 96 dpi):  
Firefox en Google Chrome, in grotere en kleinere browservensters.
- Windows Phone 8.1 (800 x 480 px, resolution: 136 dpi):  
Internet Explorer en UC browser (portret en landschap).
- Windows 10 Mobile (1280 x 720 px, resolution: 192 dpi):  
Edge (portret en landschap).
- iPad met iOS 9.3.5 (1024 x 768 px, device-pixel-ratio: 1):  
Safari, Chrome for iOS, UC Browser, Firefox (alle portret en landschap).  
Opera Mini (turbo mode) portret en landschap.
- iPad met iOS 10.2.1 (2048 x 1536 px, device-pixel-ratio: 2 ('retina')):

- Safari, Chrome for iOS, UC browser, Firefox (alle portret en landschap).
- Opera Mini (turbo mode) portret en landschap.
- Android 4.1.2 ('Jelly Bean') (800 x 480 px, resolution: 144 dpi):  
Chrome, Android browser, UC Browser en Firefox (alle portret en landschap).
- Opera Mini (besparingen uitgeschakeld) portret en landschap.
- Android 4.4.2 ('Kitkat') (1280 x 800 px, resolution: 96 dpi):  
Android browser, UC Browser, Firefox en Chrome (alle portret en landschap).
- Opera Mini (besparingen uitgeschakeld) portret en landschap.
- Android 4.4.2 ('Kitkat') (2560 x 1600 px, resolution: 192 dpi):  
Android browser, UC Browser, Firefox en Chrome (alle portret en landschap).
- Opera Mini (besparingen uitgeschakeld) portret en landschap.
- Android 5.0.1 ('Lollipop') (1920 x 1200 px, resolution: 144 dpi):  
Dolphin, UC Browser, Firefox en Chrome (alle portret en landschap).
- Opera Mini (besparingen uitgeschakeld) portret en landschap.
- Android 6.0.1 ('Marshmallow') (1920 x 1200 px, resolution: 144 dpi):  
Dolphin, Samsung Internet, UC Browser, Firefox en Chrome (alle portret en landschap).
- Opera Mini (besparingen uitgeschakeld) 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 resoluties groter dan 800x600 is ook in- en uitzoomen en – voor zover de browser dat kan – een kleinere en grotere letter 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 de iPad, Android, Windows Phone en Windows 10 Mobile, waar een touchscreen is gebruikt. Op Windows 8.1 en 10 is getest met een touchscreen, met een combinatie van toetsenbord en touchpad, en met een combinatie van toetsenbord en muis.

Als dat relevant is, is op de desktop ook getest, als JavaScript uitstaat. Eventuele problemen staan hierboven bij Toegankelijkheid en zoekmachines onder het kopje [Specifiek voor dit voorbeeld](#). (Op iOS, Android, Windows Phone en Windows 10 Mobile is niet getest zonder JavaScript, omdat je JavaScript in een toenemend aantal mobiele browsers niet uit kunt zetten. Bovendien is een mobiel apparaat zonder JavaScript niet veel meer dan een duur en groot uitgevallen horloge.)

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

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 7.

NVDA is een schermlezer, zoals die door blinden wordt gebruikt. Er is getest in combinatie met Firefox op Windows 7 en Firefox op Windows 10.

TalkBack is een in Android ingebouwde schermlezer. Er is getest in combinatie met Chrome.

VoiceOver is een in iOS en OS X ingebouwde schermlezer. Er is getest in combinatie met Safari op iOS en OS X.

ChromeVox is een schermlezer in de vorm van een extensie bij Google Chrome. Er is getest op een systeem met Kubuntu Linux.

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 opmerkingen over de toegankelijkheid specifiek voor dit voorbeeld staan hierboven bij Toegankelijkheid en zoekmachines onder het kopje [Specifiek voor dit voorbeeld](#).

Alleen op de hierboven genoemde systemen en browsers is getest. Er is dus niet getest op bijvoorbeeld 'n Blackberry. De kans is (heel erg) groot dat dit voorbeeld niet (volledig) werkt op niet-geteste systemen en apparaten. Om het wel (volledig) werkend te krijgen, zul je vaak (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 Android browser 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 eigenlijk geen beginnen aan.

De html is gevalideerd met de validator van w3c, de css ook. 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, omdat er anders 'n redelijke kans is 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!)

### **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.

Er zijn geen bekende problemen. Dat is uiteraard hartstikke mooi, alleen jammer dat nu ook de, in alle bescheidenheid, buitengewoon briljante oplossingen bij gebrek aan problemen ontbreken.

### **Wijzigingen**

Alleen grotere wijzigingen worden hier vermeld, geen dingen als een link die is geüpdatet.  
7 maart 2017:

Nieuw opgenomen.

### **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](http://www.css-voorbeelden.nl) en medewerkers daarvan op geen enkele manier verantwoordelijk.

\* Dit voorbeeld (en de bijbehorende uitleg e.d.) wordt 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](http://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](http://www.css-voorbeelden.nl) wordt trouwens altijd op prijs gesteld.

animatie-118-dl.html: de pagina met het voorbeeld.

animatie-118.pdf: de uitleg (aangepast aan de inhoud van de download).

animatie-118-inhoud-download-en-licenties.txt: een kopie van de tekst onder dit kopje  
(Inhoud van de download en licenties).

118-css-dl:

animatie-118-dl.css: stylesheet voor animatie-118-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 rood gekleurd. 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](#).)

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 `doctype`s vind je op [www.w3.org/QA/2002-04/valid-dtd-list](http://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">`

De toevoeging `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 (`&auml;` e.d.) te gebruiken, maar kun je bijvoorbeeld gewoon `ä` gebruiken.

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

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

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

Mobiele apparaten variëren enorm in breedte. En dat is een probleem. Sites waren, in ieder geval tot voor kort, 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 álle sites zijn omgebouwd voor smartphones, tablets, enz., maar dan moet je waarschijnlijk heel erg lang wachten.

Mobiele browsers gokken erop dat een pagina een bepaalde breedte heeft. Safari voor mobiel bijvoorbeeld gaat ervan uit dat een pagina 980 px breed is. De pagina wordt vervolgens zoveel versmald dat hij binnen het venster van het apparaat past. Op een iPhone wordt de pagina dus veel smaller dan op een iPad. Vervolgens kan de gebruiker inzoomen op het deel van de pagina dat hij of zij 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 is de animatie slechts 300 px breed, en dat past (vrijwel) overal. Maar die stomme mobiele browser weet dat niet, dus die gaat ervan uit dat ook de al aangepaste pagina 980 px breed is, en verkleint die dan. Dat is ongeveer even behulpzaam als de gediensstige 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. Een iPad in portretstand bijvoorbeeld is 768 px breed. 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. Voor een iPad in portretstand dus 768 px.

Simpeler gezegd: je zegt tegen het mobiele apparaat dat de pagina geen vaste breedte heeft, en dat het dus niet nodig is om de weergave aan te passen.

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, want de pagina past zich aan het apparaat aan. 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="118-css-dl/animatie-118-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 een extern stylesheet is, 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.)

```
<div id="vooruit" tabindex="0" aria-label="Animatie speelt vooruit
als div wordt aangeraakt of -geklikt, focus heeft of wordt
gehoverd">Vooruit</div>
```

div#vooruit, de <div> met de animatie.

Om te zorgen dat de animatie ook afspeelt voor gebruikers van de Tab-toets, is tabindex="0" aan de <div> toegevoegd. Meer hierover is te vinden bij [Tabindex](#).

Voor gebruikers van een schermlezer is een aria-label met tekst toegevoegd, zodat meer wordt voorgelezen dan alleen het raadselachtige woord 'Vooruit'. Meer hierover is te vinden bij [WAI-ARIA-codes](#).

```
<div id="terug" tabindex="0" aria-label="Animatie speelt achteruit
als div wordt aangeraakt of -geklikt">Terug</div>
```

Om te zorgen dat de animatie ook achterwaarts afspeelt voor gebruikers van de Tab-toets, is tabindex="0" aan de <div> toegevoegd. Meer hierover is te vinden bij [Tabindex](#).

Voor gebruikers van een schermlezer is een aria-label met tekst toegevoegd, zodat meer wordt voorgelezen dan alleen het raadselachtige woord 'Terug'. Meer hierover is te vinden bij [WAI-ARIA-codes](#).

## 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 rood gekleurd. 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, garandeer ik je hele grote problemen, omdat het volstrekt onoverzichtelijk is. Beter is het om de css netjes in te laten springen:

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

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

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

Dat comprimeren kun je met de hand doen, maar er bestaan ook hulpmiddelen voor. Op de pagina met [links](#) kun je onder Gereedschap → Snelheid, testen, gzip, comprimeren 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.)

```
/* animatie-118-dl.css */
```

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

body

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

```
background: #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

voorggrondkleur, 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 voorggrondkleur geen van beide.

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

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

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

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

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

```
margin: 0; padding: 0;
```

Slim om te doen vanwege verschillen tussen browsers.

## main

Alle <main>'s. Dat is er maar eentje. De belangrijkste inhoud van de pagina staat hierin. In dit voorbeeld zijn dat alleen de twee <div>'s (waarvan eentje met de animatie).

```
display: block;
```

Oudere browsers kennen <main> niet. Onbekende elementen worden standaard als inline-element weergegeven. Daarom wordt hier expliciet gezegd dat dit een blok-element is.

```
width: 300px;
```

Breedte.

De breedte wordt hier tot de essentiële css gerekend. Een blok-element zoals <main> wordt normaal genomen even breed als z'n ouder. Die ouder is hier <body>, ook een blok-element, dat normaal genomen ook weer even breed wordt als z'n ouder <html>. <html> is het buitenste element en wordt daarom normaal genomen even breed als het venster van de browser.

Zonder beperking zou <main> hierdoor even breed als het venster van de browser worden. En daarmee ook de erin zittende <div> met de animatie, want ook een <div> is een blok-element. Dat zou een nogal woeste animatie opleveren die, naar valt te vrezen, bij sommige mensen tot een acute verlaging van het levensgeluk gaat leiden, omdat ze zich 'n ongeluk schrikken. Daarom wordt hier paal en perk aan de breedte van de animatie gesteld.

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

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

Bovenaan kleine afstand tussen bovenkant van het browservenster en de bovenste <div>. De marge onderaan is overbodig, maar kan hier verder geen kwaad, dus die mag blijven staan. (Met margin: 50px auto 0; zou je hem heel eenvoudig kunnen weghalen.)

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

breed als het venster van de browser. Hierdoor staat <main>, en daarmee ook de erin zittende <div>'s, altijd horizontaal gecentreerd binnen het venster, ongeacht de breedte van het venster.

Deze manier van horizontaal centreren van een blok-element werkt alleen, als het te centreren element een breedte heeft.

## #vooruit

Het element met id="vooruit". De bovenste <div>, de <div> met de animatie.

`background-color: white;`

Witte achtergrond.

`color: blue;`

Blauwe voorgrondkleur. Dit is ook de kleur van de tekst.

`height: 280px;`

Hoogte.

`text-align: center;`

Tekst horizontaal centreren.

`text-decoration: underline;`

Tekst onderstrepen. Hierdoor lijkt het op een link, wat hopelijk uitnodigt tot aanraken of -klikken.

`border: black solid 1px;`

Zwart randje.

`border-radius: 15px 15px 0 0;`

Links- en rechtsboven ronde hoek.

`padding-top: 20px;`

Kleine afstand tussen bovenkant van en tekst in de <div>

`-webkit-transform: rotate(0deg); transform: rotate(0deg);`

Hier staat in feite twee keer hetzelfde: `transform: rotate(0deg);`. Waarom dat zo is, staat bij [De voorvoegsels -moz-, -ms- en -webkit-](#).

Deze regel is alleen nodig voor UC browser, Opera Mini, Dolphin en Android browser op oudere versies van Android. Bij [#vooruit:focus](#), [#vooruit:hover](#) wordt de animatie in voorwaartse richting afgespeeld, als over de bovenste <div> wordt gehoverd, als deze focus heeft of als deze wordt aangeraakt of -geklikt.

Onderdeel van deze animatie is het volledig in de rondte draaien van de <div> met behulp van `transform: rotate(360deg);`.

Standaard is de <div> (uiteraard) niet gedraaid. Dus zou je dit eigenlijk niet op hoeven geven. Maar de genoemde browsers draaien de animatie niet achterwaarts af, als niet ook de beginstand wordt opgegeven, ook al is dit – zoals hier – de standaardwaarde.

`background-color` en `color`, de twee andere onderdelen van de animatie, hebben hierboven al een waarde gekregen, dus daarvoor hoeft niet apart iets te worden opgegeven.

`-webkit-transition: 5s; transition: 5s;`

Hier staat in feite twee keer hetzelfde: `transition: 5s;`. Waarom dat zo is, staat bij [De voorvoegsels -moz-, -ms- en -webkit-](#).

Bij [#vooruit:focus](#), [#vooruit:hover](#) wordt de animatie bij `div#vooruit` in voorwaartse richting afgespeeld, als over `div#vooruit` wordt gehoverd, als deze focus heeft of als de <div> wordt aangeraakt of -geklikt. Daar wordt met `transition: 0.5s;` aangegeven dat de animatie 0,5 seconde moet duren.

Door hier bij `#vooruit` een andere tijd op te geven, vijf seconden, duurt de animatie in achterwaartse richting vijf seconden. Bij hoveren, focus, aanraken of klikken duurt de animatie in voorwaartse richting 0,5 seconde, en zodra hoveren, focus, aanraken of klikken niet meer het geval is, duurt de animatie in terugwaartse richting 5 seconden.

Op deze manier kan de animatie in beide richting in een verschillend tempo plaatsvinden.

Bij `transition` staat maar één waarde: 5s. Als er maar één waarde wordt opgegeven, bepaalt die over hoeveel tijd de verandering wordt uitgesmeerd. Dat is hier vijf seconden. Een eventuele tweede waarde, die hier ontbreekt, geeft een eventueel uitstel aan het begin van de verandering aan.

Het is ook mogelijk om het verloop van de verandering aan te geven met behulp van sleutelwoorden of getallen. Hiermee kun je bijvoorbeeld aangeven dat de verandering aan het eind snel of juist langzaam moet gaan.

Deze andere waarden worden hier niet gebruikt, maar daar geldt precies hetzelfde voor. Als je in deze regel bijvoorbeeld een vertraging van 1 seconde opgeeft, maar niet bij de transition bij [#vooruit:focus](#), [#vooruit:hover](#), dan begint de animatie in voorwaartse richting zonder vertraging. Maar bij achterwaarts afspelen is er dan een vertraging van 1 seconde, voordat de animatie begint af te spelen.

`transition` brengt een geheel eigen risico met zich mee. In de specificatie staat, welke eigenschappen met behulp van `transition` kunnen veranderen. Je kunt eventueel bij `transition` opgeven, voor welke eigenschappen het geldt. Als je bijvoorbeeld `top` en `left` beide wilt veranderen bij `:checked`, kun je bijvoorbeeld aangeven dat de geleidelijke verandering met behulp van `transition` alleen voor `top` geldt. `left` verandert dan gewoon in één keer.

Als je niet opgeeft, voor welke eigenschappen `transition` geldt, geldt het voor alle eigenschappen die bij `:hover`, `:focus`, `:checked`, e.d. worden veranderd.

Als daar 'n eigenschap bij zit die op dit moment niet door `transition` kan worden vertraagd, verandert die gewoon in één keer. Maar als de specificatie of een van de browsermakers de geest krijgt en plotsklaps die eigenschap ook onder `transition` laat vallen, kan dat tot heel onverwachte resultaten leiden.

Stel dat je een bepaalde eigenschap heel traag wilt veranderen bij `:hover` en een andere eigenschap flitsend snel. Mogelijk wordt dat flitsend snel dan opeens ook heel traag, als `transition` in de toekomst die eigenschap ook gaat ondersteunen.

Daarom moet je absoluut zeker weten dat ook in de toekomst geen problemen kunnen ontstaan. Bij twijfel: geef aan welke eigenschap(pen) `transition` mag aansturen.

Als alle eigenschappen mogen worden vertraagd door `transition`, is er geen enkel risico voor de toekomst. Dat is hier het geval. Bij [#vooruit:focus](#), [#vooruit:hover](#) worden alleen `background-color`, `color` en `transform: rotate()` veranderd. Verder verandert er niets. In dit geval mag dus alles vertraagd worden veranderd en is er geen risico voor de toekomst. Als dat niet zo zou zijn, moet je opgeven, voor welke eigenschappen `transition` geldt.

Je kunt `transition` beperken tot bepaalde eigenschappen door die eigenschappen te noemen, gevolgd door tijdsduur en eventueel vertraging:

```
transition: width 0.5s;
```

Nu geldt `transition` alleen voor `width`, ongeacht wat er in de toekomst mogelijk nog zou kunnen veranderen. Maar hier is dat dus niet nodig, omdat bij [`#vooruit:focus`](#), [`#vooruit:hover`](#) alleen `background-color`, `color` en `transform: rotate()` worden veranderd.

Ook hier geldt weer dat je bij de voorwaartse animatie andere eigenschappen kunt opgeven dan bij de achterwaartse animatie. Die eigenschappen worden dan in de ene richting langzaam veranderd, in de andere richting worden ze zonder vertraging in één keer veranderd.

```
will-change: background-color, color, transform;
```

Dit is een tamelijk nieuwe eigenschap, die nog niet in alle browsers wordt ondersteund. In de browsers die het al ondersteunen is het een aankondiging dat bepaalde eigenschappen binnenkort gewijzigd gaan worden. Achter `will-change` worden de eigenschappen die gaan wijzigen opgegeven, gescheiden door een komma. De browser kan de wijzigingen hierdoor als het ware al voorbereiden, voordat ze feitelijk plaatsvinden. Waardoor de feitelijke verandering later (veel) sneller plaats kan vinden.

Daarbij moet je 'sneller' niet lezen als 'snel' in de betekenis van kilometers per uur, maar in het aantal frames dat wordt weergegeven. Als een element beweegt, wordt in werkelijkheid het element razendsnel steeds opnieuw getekend op een iets andere plaats. De draaiende `div#vooruit` draait helemaal niet, want dan zou het hier vermoedelijk schadeclaims gaan regenen vanwege vernielde beeldschermen. De `<div>` wordt steeds opnieuw getekend op een iets andere plaats, steeds in een nieuw frame. Dit geeft de illusie van draaien.

Als het tempo van nieuwe frames onder de zestig per seconde komt, wordt voor mensen zichtbaar, wat er eigenlijk gebeurt. De `<div>` draait niet meer soepel, maar begint schokkerig te draaien, omdat de afzonderlijke tekeningen zichtbaar worden.

Als `will-change` zo fantastisch is, waarom dan niet altijd alles achter `will-change` zetten? Dat kan makkelijk met iets als `* {will-change: top, right, bottom, left, width, transform, opacity; (...);}`. (De `*`, 'universal selector', selecteert álles. Meestal is het een bijzonder slecht idee deze te gebruiken.)

Het voorbereiden van de verandering kost echter capaciteit. Als `will-change` te vaak wordt gebruikt, kan het daardoor zelfs tot een vertraging leiden. In dit geval kan het worden gebruikt, omdat redelijk zeker is dat de `<div>` geanimeerd gaat worden. Dat is immers het enige nut van deze hele pagina. Als dat niet redelijk zeker zou zijn, kun je `will-change` beter niet gebruiken. Een ander goede plaats om `will-change` te gebruiken zou bijvoorbeeld een menu zijn dat opent bij aanraken, klikken of hoveren, als dat menu waarschijnlijk geopend gaat worden.

Normaal genomen wordt in de voorbeelden op deze site vaak een zogenaamde 'shorthand' gebruikt. Met bijvoorbeeld `border` worden alle borders in één keer aangemaakt en hoef je niet apart `border-top-style`, `border-top-width`, `border-top-color`, `border-right-style`, enz. te gebruiken: twaalf aparte regels.

Een shorthand heeft echter ook een nadeel. Als je `background: white;` opgeeft, zoals hierboven bij `div#vooruit` is gedaan, wordt de achtergrondkleur wit. Als je dat opgeeft met `background-color: white;` wordt echt alleen de achtergrondkleur verandert. Maar als je het opgeeft met de shorthand `background`,

wordt niet alleen de achtergrondkleur opgegeven. De niet gebruikte eigenschappen van background worden dan teruggezet naar de standaardwaarden.

Als je `background: white;` opgeeft, geef je in werkelijkheid het volgende op:

```
background-image: none; background-position: 0% 0%;  
background-size: auto auto; background-repeat:  
repeat; background-origin: padding-box;  
background-clip: border-box; background-  
attachment: scroll; background-color: white;
```

(De standaardwaarde voor `background-color` is `transparent`.)

Er is een grote kans dat er in de toekomst nog andere eigenschappen worden toegevoegd aan een shorthand, dus deze lijst wordt waarschijnlijk nog langer.

Als je `background: white;` gebruikt, geef je dus in werkelijkheid de hele bovenstaande riedel op. `will-change` gaat er dan vanuit, dat ál deze eigenschappen worden veranderd. Daarom wordt in dit geval overal `background-color` gebruikt in plaats van `background`: hier iets boven bij `background-color` en `will-change`, en hier gelijk onder bij `#vooruit:focus`, `#vooruit:hover`.

`#vooruit:focus`, `#vooruit:hover`

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.

```
#vooruit {background-color: white; color: blue; width: 300px; height: 280px; text-align: center;  
text-decoration: underline; border: black solid 1px; border-radius: 15px 15px 0 0; padding-top: 20px;  
-webkit-transform: rotate(0deg); transform: rotate(0deg); -webkit-transition: 5s; transition: 5s;  
will-change: background-color, color, transform;}
```

Het element met `id="vooruit"`, maar alleen als dit [focus](#) heeft, of als erover wordt gehoverd. Dit is de `<div>` die geanimeerd wordt. Hoveren of focus worden bereikt door de `<div>` aan te raken, erop te klikken, erover te hovern of door er met de [Tab-toets](#) naartoe te gaan. Als `div#vooruit` op één van deze manieren wordt geactiveerd, valt de `<div>` onder deze selector.

Hieronder wordt een aantal eerder bij [#vooruit](#) opgegeven eigenschappen veranderd, als `div#vooruit` wordt geactiveerd. Omdat hieronder ook `transition` wordt gebruikt, veranderen de eigenschappen niet bliksemsnel, maar geleidelijk aan.

`background-color: black;`

Zwarte achtergrondkleur.

`color: white;`

Witte voorgrondkleur. Dit is ook de kleur van de tekst.

`-webkit-transform: rotate(360deg); transform: rotate(360deg);`

Hier staat in feite twee keer hetzelfde: `transform: rotate(360deg);`.

Waarom dat zo is, staat bij [De voorvoegsels -moz-, -ms- en -webkit-](#).

360 graden is precies één volledige draai. De hele `<div>` één keer volledig ronddraaien.

`-webkit-transition: .5s; transition: .5s;`

Hier staat in feite twee keer hetzelfde: `transition: .5s;`. Waarom dat zo is, staat bij [De voorvoegsels -moz-, -ms- en -webkit-](#).

De hierboven opgegeven veranderingen van `background-color`, `color` en `transform: transition()` duren een halve seconde. Een uitgebreider verhaal over `transition` is te vinden bij [-webkit-transition: 5s; transition: 5s;](#).

#terug

Het element met id="terug". De onderste <div>.

background: white;

Witte achtergrondkleur.

color: blue;

Blauwe voorgrondkleur. Dit is ook de kleur van de tekst.

line-height: 3em;

Regelhoogte.

Tekst wordt automatisch in het midden van de regelhoogte neergezet. Omdat verder geen hoogte aan de <div> is gegeven, is de regelhoogte gelijk de hoogte voor de <div> zelf. Hierdoor komt de tekst netjes in het midden van de <div> te taan.

Als eenheid wordt de relatieve eenheid em gebruikt, omdat bij een absolute eenheid als px de regelhoogte niet mee verandert met de lettergrootte.

De regelhoogte wordt hier tot de essentiële css gerekend, omdat de <div> anders zo laag is, dat deze nauwelijks is aan te raken.

text-align: center;

Tekst horizontaal centreren.

text-decoration: underline;

Onderstrepen.

margin-top: -1px;

De <div> 1 px naar boven zetten.

Hier gelijk onder wordt een zwarte border aan div#terug gegeven. Bij [#vooruit](#) is ook aan de boven div#terug zittende div#vooruit een border gegeven. De onderkant van div#vooruit raakt de bovenkant van div#terug, waardoor op die plaats een border van 2 px breed ontstaat. Door de onderste <div> 1 px naar boven te verplaatsen, vallen de twee borders over elkaar heen en is ook hier de border slechts 1 px breed.

Normaal genomen zou je de border aan de onderkant van div#vooruit of aan de bovenkant van div#terug weghalen, maar dat is hier niet mooi. Als

div#vooruit gaat draaien, draait de border om div#vooruit mee. Hierdoor zou dan aan één kant tijdens het draaien de border ontbreken.

De border aan de bovenkant van div#terug weghalen is ook geen goed idee, want dan zou tijdens het draaien van div#vooruit de border aan de bovenkant van div#terug zichtbaar missen.

Met deze oplossing zijn beide borders nog steeds aanwezig. Alleen zie je dat niet, omdat ze over elkaar heen staan. Pas als div#vooruit gaat draaien, worden beide borders apart zichtbaar.

border: black solid 1px;

Zwart randje.

border-radius: 0 0 15px 15px;

Rechts- en linksonder ronde hoeken.

## JavaScript

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 rood gekleurd. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles in een gewone letter vanwege de leesbaarheid.) Bij de uitleg van deze code zijn allerlei haakjes e.d. grotendeels weggelaten, want dat voert hier te ver. Als je je in dat soort dingen wilt verdiepen, kun je beter naar sites gaan die meer voor JavaScript zijn bedoeld.

```
<script>
document.getElementById("vooruit").addEventListener("touchstart",
    function () {return null; });
document.getElementById("terug").addEventListener("touchstart",
    function () {return null; });
</script>
```

Dit stukje JavaScript is nodig voor Safari, Chrome for iOS en UC browser op iOS. Zonder dit JavaScript wordt de animatie van `div#vooruit` niet gestart, omdat iOS [.hover](#) negeert. Met behulp van dit script wordt op hoveren gereageerd, alsof het een aanraking is, waardoor de animatie bij aanraking van `div#vooruit` ook hier wordt afgespeeld.

`<script>` en `</script>`

Dit is gewone html: het zijn de openings- en sluittag voor het script. In html5 is de toevoeging `type="text/javascript"` niet meer nodig.

```
document.getElementById("vooruit").addEventListener("touchstart",
    function () {return null; });
```

Deze regel zorgt ervoor dat `div#vooruit` gaat reageren op een aanraking.

`document.getElementById("vooruit")`: het middelste stukje

`getElementById` is een zogenaamde 'functie'. Een functie is een stukje in de browser ingebakken code, waarmee je iets kunt doen. Deze functie is te vinden in het object `document`, vandaar dat `document` ervoor staat. (Een object is een bij elkaar horende verzameling van functies en andere code. Een van die objecten heeft de naam 'document'.)

Met de functie `getElementById()` kan JavaScript een element met een bepaalde id opzoeken. De id waarnaar wordt gezocht, staat tussen haakjes:

```
getElementById("vooruit")
```

Er wordt gezocht naar het element met `id="vooruit"`, oftewel: `div#vooruit`.

`addEventListener`: dit is wat er gedaan moet worden met `div#vooruit`: er wordt

een 'eventlistener' aan toegevoegd. Een eventlistener is een ding dat luistert, of er iets gebeurt. Zo'n gebeurtenis kan een aanraking van het scherm zijn, een liefdevolle aai van de muis, een ram op het toetsenbord, van alles.

`"touchstart"`: in dit geval wordt naar het begin van een aanraking van het touchscreen geluisterd. Omdat de eventlistener aan `div#vooruit` is gekoppeld, wordt alleen een aanraking van `div#vooruit` geregistreerd. Je kunt een eventlistener aan elk element koppelen, en zelfs aan het hele browservenster, maar het gaat hier alleen om `div#vooruit`, want alleen die moet reageren op een aanraking.

Dat is nog niet zo interessant, het wordt pas echt opwindend wat er ná die eerste, inleidende aanraking gaat gebeuren. Vinden zij elkaar, of wordt het moord- en doodslag?

```
function () {return null; };
```

 helaas, ze vinden elkaar niet én het wordt geen moord- en doodslag.

`function()` is gewoon een soort aankondiging dat er iets gaat gebeuren. Erachter tussen de accolades staat, wat er gaat gebeuren: `return null;`. Doe niets.

Helemaal niets. Vanavond niet, schat, ik heb hoofdpijn. Of, afhankelijk van opvoeding en omgeving: blijf met je gore rotpoten van me af, viespeuk.

In feite gebeurt er dus helemaal niets. Maar dit stukje JavaScript zorgt er wel voor dat `div#vooruit` nu ook in de drie genoemde browsers op iOS reageert op een aanraking en de animatie gaat afspelen.

```
document.getElementById("terug").addEventListener("touchstart",  
function () {return null; });
```

De tweede regel uit het script. Deze is precies hetzelfde als de eerste regel, dus de beschrijving van deze regel is gelijk hierboven te vinden bij de eerste regel. Het enige verschil is dat deze eventlistener aan `div#terug` wordt gekoppeld in plaats van aan `div#vooruit`.

Met de eerste regel van het script is `div#vooruit` gevoelig gemaakt voor aanraking, waardoor de animatie bij aanraking van `div#vooruit` wordt afgespeeld. Op iOS blijft hoveren over een element echter voortduren, tot er over een ander element wordt gehoverd (een ander element wordt aangeraakt). Oftewel: de animatie speelt wel vooruit, maar achteruit afspelen gebeurt pas, als een ander element wordt aangeraakt. Maar dan moet er wel een ander element zijn.

Dat is precies waarvoor `div#terug` is bedoeld, alleen werkt ook deze `<div>` niet in de genoemde browsers op iOS. Daarom moet ook deze `<div>` gevoelig worden gemaakt voor aanraken.

## 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 rood 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="Animatie speelt vooruit  
    sneller af dan achteruit - voorbeeld. Gebruikt (vrijwel)  
    alleen html en css.">
```

```

<title>Animatie speelt vooruit sneller af dan achteruit -
        voorbeeld</title>
<link rel="stylesheet" href="118-css-dl/animatie-118-dl.css">
</head>
<body>
<main>
    <div id="vooruit" tabindex="0" aria-label="Animatie speelt
        vooruit als div wordt aangeraakt of -geklikt, focus heeft
        of wordt gehoverd">Vooruit</div>
    <div id="terug" tabindex="0" aria-label="Animatie speelt
        achteruit als div wordt aangeraakt of
        -geklikt">Terug</div>
</main>
<script>
document.getElementById("vooruit").addEventListener("touchstart",
    function () {return null; });
document.getElementById("terug").addEventListener("touchstart",
    function () {return null; });
</script>
</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 rood gekleurd. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles in een gewone letter vanwege de leesbaarheid.)

```

/* animatie-118-dl.css */

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

main {
    display: block;
    width: 300px;
    margin: 50px auto;
}

```

```
#vooruit {
    background-color: white;
    color: blue;
    height: 280px;
    text-align: center;
    text-decoration: underline;
    border: black solid 1px;
    border-radius: 15px 15px 0 0;
    padding-top: 20px;
    -webkit-transform: rotate(0deg);
    transform: rotate(0deg);
    -webkit-transition: 5s;
    transition: 5s;
    will-change: background-color, color, transform;
}
```

```
#vooruit:focus, #vooruit:hover {
    background-color: black;
    color: white;
    -webkit-transform: rotate(360deg);
    transform: rotate(360deg);
    -webkit-transition: .5s;
    transition: .5s;
}
```

```
#terug {
    background: white;
    color: blue;
    line-height: 3em;
    text-align: center;
    text-decoration: underline;
    margin-top: -1px;
    border: black solid 1px;
    border-radius: 0 0 15px 15px;
}
```

## JavaScript

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 rood gekleurd. Alle niet-essentiële code is zwart. (In de inhoudsopgave staat alles in een gewone letter vanwege de leesbaarheid.) Bij de uitleg van deze code zijn allerlei haakjes e.d. grotendeels weggelaten, want dat voert hier te ver. Als je je in dat soort dingen wilt verdiepen, kun je beter naar sites gaan die meer voor JavaScript zijn bedoeld.

```
<script>
document.getElementById("vooruit").addEventListener("touchstart",
    function () {return null; });
document.getElementById("terug").addEventListener("touchstart",
    function () {return null; });
</script>
```