Reader
Development
Leerjaar 1

Naam:

Webdevelopment

jQuery 3.x

TECHNOVA
COLLEGE

Inhoudsopgave

Inleiding 4
Voordelen van jQuery 4
Planning 6
Inleiding tot JavaScript 9
Events en Actions 10
Syntax regels 10
JavaScript debugger 11
Variant 11
Soorten variabelen 12
Datatype controleren 13
Operatoren 13
Voorwaardelijke instructies 15
Lussen 20
Functies 23
Functies in jQuery 26
Scope (zichtbaarheid) van variabelen 26
Objecten 28
Ingebouwde JavaScriptobjecten 33
jQuery basis 45
jQuery integreren 45
$(document).ready() 47
Unobtrusive JavaScript 48
jQuery documentatie (API) 49
Afgeschafte (deprecated) methodes en selectors 52
jQuery cheat sheet 53
jQuery in combinatie met andere JavaScriptbibliotheken 53
Elementen selecteren 55
Basis selectors 56
Hiërarchische selectors 56
Basisfilters 58
Inhoudsfilters 59
Attribuut selectors 60
Childfilters 60
Formulierfilters 61

Zichtbaarheidsfilters ... 62
Toepassing 1: gemeentelijst filteren (basisversie) ... 62
Toepassing 2: openingsuren markeren ... 63
Selectie verfijnen (traversing) ... 66
Hiërarchische selectors ... 66
Filter methodes ... 73
Element looping: each() ... 79
Selecties in cascade: end() ... 81
Snelheidstest ... 82
Toepassing 1: gemeentelijst filteren (uitgebreide versie) ... 82
Toepassing 2: rating ... 84
Events ... 88
Documentevents ... 89
Gebruikersevents ... 91
Event bubbling ... 94
Event methodes en properties ... 95
Event properties ... 96
Hover-event: methode met een dubbele functie ... 97
De methode on() ... 98
Eén selector en meerdere events met dezelfde functie ... 98
Eén selector en meerdere events met verschillende functies ... 99
Event handlers verwijderen ... 101
Toepassing 1: reactietest ... 101
Animaties en effecten ... 104
Basiseffecten ... 104
Aangepaste animaties ... 109
Versnellingsparameter ... 114
Toepassing 1: vloeiend scrollen ... 115
Toepassing 2: tabbladen ... 118
DOM manipulatie ... 121
HTML-attributen ... 121
Object dimensies ... 124
Elementen toevoegen ... 127
Buiten de selector ... 129
Rond de selector ... 130
before() vs insertBefore() ... 132

Elementen verwijderen 134
Elementen vervangen 135
Elementen klonen 136
Toepassing 1: Lightbox 139
Inleiding tot AJAX 141
Historie 142
Wat is XML? 143
Wat is JSON? 144
Requests filteren met GET en POST 145
Zes soorten AJAX requests 147
Same origin policy 148
Cross-site scripting 149
AJAX zonder server-side scripting 150
Load() 150
$.getScript() 156
$.getJSON() 157
$.get() en $.post() 163
$.ajax() 168
Externe gegevens ophalen via JSONP 169
AJAX met server-side scripting 177
Master/detailrelatie met load() 179
Master/detailrelatie met $.get() 183
Contactformulier verzenden met $.post() 183
Cross-site scripting (XSS) 184

Inleiding

Interactieve websites bestaan al lang niet meer uitsluitend uit HTML en CSS. Deze technologieën zorgen respectievelijk voor de inhoud en de opmaak van de pagina, maar om het geheel dynamisch te maken, gebruikt men JavaScript.

JavaScript is een client-side scripttaal. Het script wordt dus niet op een webserver uitgevoerd, maar rechtstreeks in de browser. Alle recente browsers ondersteunen JavaScript. Moderne websites maken gretig gebruik van deze technologie om de gebruikerservaring te verbeteren.

Hoewel JavaScript een gestandaardiseerde scripttaal is, geldt dit niet altijd voor de verschillende browsers. Zo hanteren Firefox, Safari en Internet Explorer een totaal verschillend objectmodel.

Om een pagina op al deze browsers eenzelfde gebruikerservaring te geven, zal u vaak per browser specifieke instructies moeten schrijven. Dit maakt het programmeren in pure JavaScriptcode vaak een tijdrovende en frustrerende bezigheid.

Gelukkig zijn er verschillende JavaScriptbibliotheken, ook wel JavaScriptframeworks genoemd, die ons hierbij kunnen helpen. Zo hoeven we ons script maar eenmaal te programmeren en het framework transformeert het script naar de juiste instructies voor de juiste browser.

jQuery is veruit het populairste JavaScriptframework en wordt op heel wat grote websites gebruikt. Denk hierbij maar aan Twitter, Amazon, Microsoft, enz...

Voordelen van jQuery

De belangrijkste eigenschappen van jQuery kunnen we als volgt samenvatten:

- **jQuery: The Write Less, Do More, JavaScript Library**.
  De slogan van jQuery is duidelijk: met enkele lijntjes code kan u zeer krachtige en uitgebreide instructies schrijven.
- **Crossbrowser compatibel**. Schrijf één instructie voor een verscheidenheid aan browsers en browserversies. Zelfs oude browsers, zoals Internet Explorer 6 worden nog vrij goed ondersteund.
- **Zeer uitgebreide core-bibliotheek**. De basisbibliotheek (de core) bevat ontzettend veel kant-en-klare methodes. Zo kan u bijvoorbeeld in één instructie alle elementen met de class `.abc` selecteren. In pure JavaScriptcode is dit niet zo evident en zal u een ingewikkelde functie moeten schrijven om hetzelfde resultaat te bekomen.
- **Elementen selecteren gebeurt volgens de CSS-notatie**. Een instructie bestaat steeds uit twee of meer delen. Het eerste deel, de selector, selecteert een element. Alle volgende delen beschrijven de bewerking die op de selector wordt uitgevoerd. De selectie gebeurt volgens de CSS-notatie. Al uw kennis over CSS-selectors kan u dus dadelijk verwerken in een jQuery statement. Een voorbeeld: `$('img').hide();` Het eerste deel van de instructie, `$('img')` selecteert alle afbeeldingen op de pagina. Het tweede deel, `hide()` verbergt alle afbeeldingen.
- **De inhoud en de lay-out van een webpagina wijzigen (DOM manipulatie)**. Met jQuery kan u zeer eenvoudig elementen binnen de pagina manipuleren. U kan elementen verwijderen, verplaatsen, wissen en zelfs de lay-out wijzigen.
- **Reageren op acties van de gebruiker (events).** Zodra de gebruiker een event activeert (click, change, resize, ...) wordt een bepaalde actie uitgevoerd. Deze events zijn reeds jaren door alle browsers ondersteund. Meer recente events zoals dblclick en focusout worden maar

door enkele recente browsers herkend. jQuery zorgt ervoor dat deze events ook door oudere browsers begrepen worden.

- **Animaties**. jQuery bevat een zeer uitgebreide bibliotheek van animaties en effecten.
- **Informatie van de server ophalen zonder de volledige pagina te vernieuwen (AJAX)**. Eén van de belangrijkste toepassingen voor een goede gebruikerservaring is het dynamisch wijzigen van pagina-inhoud zonder de volledige pagina te vernieuwen. Deze techniek, ook wel AJAX genoemd, zit volledig in jQuery geïntegreerd.
- **jQuery is uitbreidbaar**. Naast de basisbibliotheek bestaan er tal van uitbreidingen of plug-ins die u vrij kan gebruiken.

Planning

Webdevelopment jQuery 3.x is verdeeld in 2 blokken van 9 weken. Hieronder tref je de voorlopige planning voor de komende blokken aan.

Blok 3

Lesweek	Inhoud van de lessen	Datum
1	**Introductie JavaScript** • Verschil Java & JavaScript • JavaScript programmeren	**21 februari 2020**
	Voorjaarsvakantie	**27 februari 2020**
2	**Introductie JavaScript** • Events en Actions • Syntax regels • JavaScript debugger • Variabelen • Operatoren • Voorwaardelijke instructies • Lussen • Functies • Functies in jQuery • Scope (zichtbaarheid) van variabelen • Objecten • Ingebouwde JavaScriptobjecten	**06 maart 2020**
3	**Introductie jQuery** • Unobtrusive JavaScript • jQuery documentatie (API) • Afgeschafte (deprecated) methodes en selectors • jQuery cheat sheet • jQuery in combinatie met andere JavaScriptbibliotheken	**13 maart 2020**
4	**Elementen selecteren** • Basis selectors • Hiërarchische selectors • Basisfilters • Inhoudsfilters	**20 maart 2020**
5	**Elementen selecteren** • Attribuut selectors • Childfilters • Formulierfilters • Zichtbaarheidsfilters	**27 maart 2020**
6	**Selectie verfijnen (traversing)** • Hiërarchische selectors • Filter methodes • Element looping: each() • Selecties in cascade: end() • Snelheidstest	**03 april 2019**
7	**Goede vrijdag**	**10 april 2020**
8	**Events** • Documentevents • Gebruikersevents • Event bubbling • Event methodes en properties • Hover-event: methode met een dubbele functie	**17 april 2020**

	• De methode on() • Event handlers verwijderen	
9	**Toets** **Animaties en effecten** • Basiseffecten • Aangepaste animaties • Animatie pauzeren: delay() • Animatie beëindigen: finish() • Animatie afbreken: stop() • Herhaling: Basiseffecten • Herhaling: Aangepaste animaties • Versnellingsparameter	**24 april 2020**
	Meivakantie	**1 mei 2020**
10	**Examenweek**	**8 mei 2020**

Blok 4

Lesweek	Inhoud van de lessen	Datum
1	**DOM manipulatie** • HTML-attributen • Object dimensies • Elementen toevoegen • Elementen verwijderen	**15 mei 2020**
2	**Hemelvaartsweekend**	**22 mei 2020**
3	**DOM manipulatie** • Elementen vervangen • Elementen klonen	**29 mei 2020**
4	**Inleiding tot AJAX** • Historiek • Wat is XML? • Wat is JSON?	**05 juni 2020**
5	**Inleiding tot AJAX** • Requests filteren met GET en POST • Zes soorten AJAX requests • Same origin policy • Cross-site scripting	**19 juni 2020**
6	**AJAX zonder server-side scripting** • Load() • $.getScript() • $.getJSON() • $.get() en $.post() • $.ajax() • Externe gegevens ophalen via JSONP	**26 juni 2020**
7	**AJAX zonder server-side scripting** • Herhaling: Load() • Herhaling: $.getScript() • Herhaling: $.getJSON() • Herhaling: $.get() en $.post() • Herhaling: $.ajax() • Externe gegevens ophalen via JSONP	**21 juni 2020**
8	**AJAX met server-side scripting** • Master/detailrelatie met load()	**3 juli 2020**

	• Master/detailrelatie met $.get() • Contactformulier verzenden met $.post() • Cross-site scripting (XSS)	
9	**Toets**	**10 juli 2019**
10	**Examenweek**	**12 juli 2019**

Inleiding tot JavaScript

Met HTML kan je een webpagina structureren. **C**ascading **S**tyle **S**heets zorgen voor de lay-out. Bijgevolg zijn pagina's die enkel gebruik maken van HTML en CSS zeer statisch. JavaScript zorgt voor de interactiviteit op een webpagina.

JavaScript is een scripttaal (een lichtgewicht programmeertaal) die door zowat alle browsers wordt ondersteund.

In dit hoofdstuk maak je kennis met een aantal basisfuncties en controlestructuren binnen JavaScript. We gaan na hoe we variabelen gebruiken en welke voorwaardelijke en conditionele instructies (if-else, select-case, ...) JavaScript ons biedt.

Deze documentatie heeft niet de intentie alle mogelijkheden die JavaScript ons biedt te belichten, maar enkel een goede basis te leggen. Na deze introductie zal je de werking van jQuery beter begrijpen en zodoende nog krachtigere scripts kunnen schrijven. Voor meer informatie over JavaScript raadpleeg jebest één van de talrijke websites of handboeken waarin JavaScript volledig besproken wordt. Bijvoorbeeld:

- http://www.w3schools.com/js/
- http://www.w3schools.com/jsref/
- https://legacy.gitbook.com/book/gitbookio/javascript/details

JavaScript en **Java** zijn twee totaal verschillende dingen.

In Java schrijf je volledige applicaties. Na compilatie kunnen Java-applicaties zelfstandig werken, maar de applicatie kan eveneens worden opgenomen binnen een webpagina.

JavaScript daarentegen is geen gecompileerde applicatie, maar een client-side scripttaal.

Dit wil zeggen dat de code in de webpagina verweven wordt. Een browser zal de code verwerken op het moment dat het script wordt uitgevoerd. Eventuele fouten komen bijgevolg pas boven bij het uitvoeren van het script. Bij een fout wordt enkel het script gestopt. De rest van de pagina blijft gewoon zichtbaar.

Op elke pagina die gebruik maakt van JavaScript kan je het script zelf bekijken in de broncode van de pagina. Een gebruiker kan dus op elk moment jouw script "lenen" en in zijn eigen project verwerken.

Events en Actions

In JavaScript kan een Action gekoppeld worden aan een bepaald Event.

Event

Events zijn gebeurtenissen of handelingen die een gebruiker uitvoert op een pagina. Je kan bijvoorbeeld op een link klikken (`onClick`), een pagina openen (`onLoad`), een browservenster vergroten of verkleinen (`onResize`), een formulier verzenden (`onSubmit`), enz ...

Al deze handelingen worden door de browser geregistreerd.

Action

Een event kan ervoor zorgen dat een bepaald JavaScript (een action) wordt uitgevoerd.

Syntax regels

JavaScript binnen een HTML-pagina wordt altijd omsloten door een script-tag.

- Zoals `<script>.....</script>`
- Net zoals CSS kan je JavaScriptcode herbruiken over meerdere pagina's. Bewaar het script in een tekstbestand met de extensie .js. Het attribuut src verwijst naar de scriptpagina. `<script src="mijnScriptDoc.js"></script>`
- Plaats, indien mogelijk, het script bij voorkeur onderaan in de html-pagina, bij voorkeur vlak voor het sluiten van de body-tag. Dit is echter niet verplicht.
- Afzonderlijke woorden worden door één of meerdere spaties gescheiden.
- Een opdracht of statement eindigt steeds met een puntkomma.
- De meeste browsers kunnen de opdracht ook uitvoeren als je de puntkomma weglaat.
- Variabelen worden aangemaakt met het kernwoord `var` of `let`.
- Variabelen beginnen steeds met een letter, een underscore _ of een dollarteken $, maar mogen nooit beginnen met een cijfer.

```
var getal = 2;
var tekst = "dit is een string";
```

Strings kunnen zowel tussen enkele als tussen dubbele aanhalingstekens staan. Enkele aanhalingstekens binnen dubbele aanhalingstekens (of omgekeerd) zijn eveneens toegestaan.

```
var string1 = "dit is string1";
var string2 = 'dit is string2';
var string3 = "Dit is ook een 'correcte' string";
var string3 = 'Dit is ook een "correcte" string';
var string4 = "Dit is een "foutieve" string"; // FOUT!!!
```

Gebruik een backslash als escape karakter. Het karakter na wordt dan speciaal geïnterpreteerd.

```
var string4 = "Ook dit is een \"goede\" string";
\n          //new line (nieuwe lijn)
\r          //return (naar begin regel)
\t          //tab
\b          //backspace
```

- JavaScript is hoofdlettergevoelig!

- Zowel variabelen, functies als objecten zijn hoofdlettergevoelig.
- Becommentarieer jouw script.
- Commentaar op één lijn wordt voorafgegaan door `//`.
- Commentaar over meerdere lijnen plaats je tussen `/*` en `*/`.

JavaScript debugger

Omdat fouten in JavaScript niet altijd dadelijk zichtbaar zijn, is het belangrijk dat je deze op één of andere manier toch kan opsporen. Alle recente browsers hebben daarom een JavaScript debugger ingebouwd. De ingebouwde console geeft meer uitleg over de soort fout en het lijnnummer waar je moet zoeken.

- **Chrome en Safari (F12)**

  Klik met de rechtermuistoets op de pagina en kies Inspecteren uit het snelmenu.
  Klik vervolgens op het probleem-icoon.

- **Firefox (F12)**

  Open menu > Webontwikkelaar > Webconsole.

- **Internet Explorer 9+ (F12)**

  Open menu > Extra > Ontwikkelhulpprogramma's.
  Klik vervolgens op Console of op Script.

Als jouw browser foutopsporing ondersteunt, kun je `console.log ()` gebruiken om bijvoorbeeld tussentijdse JavaScript-waarden weer te geven in het venster Foutopsporing.

> Open **jsIntro/console.html** en bekijk het resultaat in de console.

```
var a = 5;
var b = 6;
var som = a + b;
console.log('a = ', a);             // witte achtergrond in de
console
console.warn('a + b = ', som);      // gele achtergrond in de console
console.error('Oeps ...');          // rode achtergrond in de console
```

Variant

Variabelen zijn kleine containers waarin u informatie kan bewaren. Je kunt de waarde van een variabele gebruiken door de naam op te geven. In tegenstelling tot de meeste programmeertalen kan een variabele eender welk datatype bevatten, zonder dat je het datatype op voorhand meedeelt. Elke variabele is van het type variant. Een variant is een variabele waarvan het datatype kan wijzigen in functie van de bewerking die erop wordt uitgevoerd.

```
// Voorbeeld variant
var a = 7;       // a is een getal
a = "welkom";    // a is vanaf nu een string
```

Bij het declareren van de variabele a bevat a een waarde van het datatype getal. Op de tweede lijn verandert het datatype naar een string.

Open **jsIntro/variabelen.html**.

Binnen de script-tag worden elf variabelen aangemaakt. Merk nogmaals op dat elk statement wordt afgesloten door een puntkomma.

```
// De verschillende variabelen worden hier gedeclareerd
var getal1 = 17;
var getal2 = 3;
var string1 = 'dag';
var string2 = 'Jan';
var string3 = '17';
var string4 = '3';
var bool1 = true;
var bool2 = false;
var ar1 = ['ma','di','wo'];
var obj1 = new Object;
var var1;
var var2 = null;
```

Soorten variabelen

De belangrijkste types van variabelen zijn strings, getallen, booleans en objecten.

- **Getallen** worden principieel nooit omsloten door aanhalingstekens.
  Bv: 5, 3.14, 34e-2 (= 0.34), ...
- **Strings** worden altijd omsloten door enkele of dubbele aanhalingstekens.
  Bv: `"tekst”`, `'tekst'`, `"5.4"`, ...
- **Booleaanse waarden** kennen twee toestanden: `true` (waar) of `false` (onwaar).
- **Objecten** komen later in dit hoofdstuk uitgebreid aan bod.
  Verder kent JavaScript nog `null` en `undefined`.
  `undefined` indien de variabele gedeclareerd is, maar nog geen waarde heeft.
  `null` staat voor een lege of onbestaande waarde. Door een variabele op `null` te zetten, wordt de variabele vernietigd.

Getallen plaatst men per definitie nooit tussen aanhalingstekens. Doe je dit toch, dan heeft dit soms invloed op het resultaat. Afhankelijk van de bewerking worden de variabelen immers automatisch omgezet naar het juiste datatype. Het +teken wordt bijvoorbeeld gebruikt om getallen op te tellen, maar ook om strings met elkaar te verbinden.

Let op: formuliervelden geven ALTIJD een STRING terug. Afhankelijk van de bewerking die volgt, kan het dus zijn dat je de uitgelezen waarde via `parseInt()` of `parseFloat()` eerst moet omzetten naar een getal.

Datatype controleren

Met `typeof()` kan je het datatype controleren. Voor bovenstaande variabelen geeft dit:

```
typeof(getal1)   =   "number"
typeof(getal2)   =   "number"
typeof(string1)  =   "string"
typeof(string2)  =   "string"
typeof(string3)  =   "string"
typeof(string4)  =   "string"
typeof(ar1)      =   "object"    //een array is een object!
typeof(obj1)     =   "object"
typeof(bool1)    =   "boolean"
typeof(bool2)    =   "boolean"
typeof(var1)     =   "undefined"
```

Merk op dat een array geen eigen datatype heeft. Een array is in JavaScript een object. Hierover later meer.

Merk op dat een array geen eigen datatype heeft. Een array is in JavaScript een object. Hierover later meer.

Operatoren

Je kunt de belangrijkste operatoren onderverdelen in vier groepen, namelijk rekenkundige, vergelijkende, logische en string operatoren.

Rekenkundige operatoren

```
*        // vermenigvuldiging
/        // deling
%        // modulo of rest van een deling   (13%5 = 3)
+        // optelling
-        // aftrekking
++       // increment   (a++ is identiek aan a = a +1 en aan a += 1)
--       // decrement   (a-- is identiek aan a = a -1 en aan a -= 1)
```

```
getal1   = 17
getal2   = 3
string3  = "17"
string4  = "3"
getal1 + getal2 = 20
getal1 - getal2 = 14
getal1 * getal2 = 51
getal1 / getal2 = 5.666666666666667
getal1 % getal2 = 2
getal2 * string3     = 51        // getal * string = getal!!!
getal2 + string3     = "317"     // getal + string = string!!!
string3 + string4    = "173"     // string + string = string!!!
getal1 / 0           = Infinity
++getal2 = 4   // pre-increment: eerst + één, dan tonen
getal2++ = 4   // post-increment: eerst tonen, dan + één
getal2   = 5   // vorige increment nu pas zichtbaar
--getal2 = 4   // pre-decrement: eerst - één, dan tonen
```

```
getal2-- = 4   // post-decrement: eerst tonen, dan - één
getal2   = 3   // vorige decrement nu pas zichtbaar
```

Vergelijkende operatoren

```
==      // gelijk aan  (5 == "5" is True)
===     // gelijk aan en van hetzelfde datatype (5 === "5" is False)
<       // kleiner dan
>       // groter dan
<=      // kleiner of gelijk aan
>=      // groter of gelijk aan
!=      // niet gelijk aan
!==     // niet gelijk aan en niet van hetzelfde datatype
```

Het resultaat van een vergelijkende bewerking geeft altijd een booleaanse waarde `true` (waar) of `false` (onwaar).

```
getal1   = 17
getal2   = 3
string3  = "17"
string4  = "3"
(getal1 == getal2)   = false
!(getal1 == getal2)  = true
(getal1 < getal2)    = false
(getal1 > getal2)    = true
(getal1 <= getal2)   = false
(getal1 >= getal2)   = true
(getal1 != getal2)   = true
(getal1 == string3)  = true      // gelijk aan
(getal1 === string3) = false     // gelijk aan EN van hetzelfde
datatype
(string3 < string4)  = true      // string '17' < string '3' !!!
(parseInt(string3) < parseInt(string4)) = false   // getal 17 >
gatal 3
```

Let vooral op de drie laatste voorbeelden. De vergelijkingen `<`, `<=`, `>` en `>=` op twee strings geeft niet altijd het verwachte resultaat. Zo is string `'17'` kleiner dan string `'3'`!

De vergelijking start bij het eerste karakter van elke string en, inderdaad, `'1'` is kleiner dan `'3'`.

Indien we later gegevens uit een formulier opvragen, krijgen we steeds een string terug, ook al vullen we een getal in. Indien we hierop één van bovenstaande vergelijkingen uitvoeren, moeten we de string eerst omzetten naar een getal, en dan pas de vergelijking uitvoeren. Een string converteren naar een getal kan met de JavaScriptmethode `parseInt()` voor het omzetten naar een geheel getal of met `parseFloat()` voor het omzetten naar een gebroken getal.

```
parseInt('3')                // = 3
parseInt('54.74')            // = 54
parseInt('14a7')             // = 14
parseInt('de78')             // = NAN (Not A Number)
parseFloat('3')              // = 3
parseFloat('54.74')          // = 54.74
parseFloat(' 30 40 50 ')     // = 30
```

Logische operatoren

```
!        // ontkenning
&&       // en-functie
||       // of-functie
```

Het resultaat van een logische bewerking geeft eveneens een booleaanse waarde `true` (waar) of `false` (onwaar).

De en-functie is waar indien alle operatoren binnen de functie waar zijn. De of-functie is waar indien minstens één operand binnen de functie waar is.

De ontkenning is waar indien de operand onwaar is.

```
getal1 = 17
getal2 = 3
(getal1 == getal2)                = false
!(getal1 == getal2)               = true
(getal1 == 17 && getal2 < 2)      = false
(getal1 == 17 || getal2 < 2)      = true
```

Logische en vergelijkende operatoren worden hoofdzakelijk gebruikt binnen voorwaardelijke of conditionele instructies en lussen.

String operatoren

```
+        // samenvoegen van twee strings
+=       // voegt het rechtse argument aan het linkse toe.
         // (string1 += string2 is identiek aan string1 = string1 +
string2)
string1 = "dag"
string2 = "Jan"
string1 + string2         = "dagJan"
string1 + " " + string2   = "dag Jan"
string1 += string2        = "dagJan"    // string1 = string1 + string2
string1                   = "dagJan"
```

Voorwaardelijke instructies

Net als variabelen zijn voorwaardelijke instructies één van de belangrijkste onderdelen van een programmeertaal. Hiermee bepaal je wat er moet gebeuren (welke statements worden uitgevoerd) in functie van de toestand van een variabele.

If-else

De instructie die je het meest gaat gebruiken is de if-else structuur.

Indien een bepaalde voorwaarde waar is, voer dan alle instructies uit binnen het if-blok. Indien de voorwaarde onwaar is, voer dan de instructies binnen het else-blok uit. Enkele voorbeelden:

```
// if met één voorwaarde
if (voorwaarde1) {
    // voer instructie uit indien voorwaarde1 waar is
}
```

```
// if-else met één voorwaarde
if (voorwaarde1) {
```

```
    // voer instructie1 uit indien voorwaarde1 waar is
} else {
    // zoniet voer instructie2 uit
}
```

```
// if-else met meerdere voorwaarden
if (voorwaarde1) {
    // voer instructie1 uit indien voorwaarde1 waar is.
} elseif (voorwaarde2) {
    // voer instructie2 uit indien voorwaarde2 waar is.
} else {
    // zoniet voer instructie3 uit
}
```

Open **jsIntro/if_else_1.html**.
Bekijk het resultaat in een browser.

If-Else

Voorbeeld 1: vaste tekst

getal1 is groter dan 5

Voorbeeld 2: variabele tekst

getal1 heeft de waarde 8 en is groter dan 5

Voorbeeld 3: verkorte if-else

8 is een even getal

Op deze pagina vind je drie script-tags binnen de body-tag. Het commando `document.write(...)` schrijft op de plaats waar het script wordt aangeroepen een tekst, of beter gezegd, HTML-code. Vandaar dat het ook vrij logisch is dat we de script-tags voorlopig plaatsen waar we de uitvoer willen. Dadelijk gaan we alle commando's onderbrengen in een functie en kunnen we het script wel verhuizen naar onderaan de body-tag.

Laat ons de drie scripts op deze pagina van naderbij bekijken.

```
// voorbeeld 1: vaste tekst
var getal1 = 8;
if(getal1 > 5) {
    document.write('<p>getal1 is groter dan 5</p>');
} else {
    document.write('<p>getal1 is kleiner dan of gelijk aan 5</p>');
}
```

Op de eerste lijn stelt u de variabele getal1 gelijk aan 8. Daarna volgt de vergelijking. Is getal1 groter dan 5, toon dan de tekst getal1 is groter dan 5.

Is getal1 kleiner of gelijk aan 5, toon dan de tekst getal1 is kleiner dan of gelijk aan 5.

```
// voorbeeld 2: variabele tekst
if(getal1 > 5) {
    document.write('<p>getal1 heeft de waarde ' + getal1 +
    ' en is groter dan 5</p>');
} else {
    document.write('<p>getal1 heeft de waarde ' + getal1 +
    ' en is kleiner dan of gelijk aan 5</p>');
}
```

Merk op dat de tweede script-tag nog steeds gebruik maakt van de variabele getal1 uit het eerste script-blok. Ook hier maken we net dezelfde vergelijking, maar wordt de string die we uitschrijven halfweg onderbroken en voegen we de werkelijke waarde van getal1 toe.

Dus eerst schrijven we een vaste string, daarna de echte waarde van getal1 en daarachter weer een vaste string.

Let op dat je de deelstrings steeds mooi afsluit met aanhalingstekens.

Bestaat het if- en het else-statement maar uit één commando, dan kan u de structuur compacter schrijven.

```
// normale schrijfwijze
if (voorwaarde) {
    // instructie indien waar
} else {
    // instructie indien niet waar
}
// verkorte schrijfwijze
voorwaarde ? instructie indien waar : instructie indien niet waar ;
```

In het derde voorbeeld gaan we na of getal1 een even of een oneven getal is. Deel het getal door twee en bekijk de restwaarde. Bijvoorbeeld: 8%2 is 0. Dus 0 staat voor een even getal.

```
// Voorbeeld 3: verkorte if-else
var rest = (getal1%2 == 0) ? 'even' : 'oneven';
document.write('<p>' + getal1 + ' is een ' + rest + ' getal</p>');
```

Wijzig de waarde van **getal1** en bekijk het resultaat opnieuw in een browser.

Net zoals we CSS (het visuele) best scheiden van de HTML, is het ook raadzaam om alle scripts volledig te scheiden van HTML. Voor CSS maken we hierbij gebruik van de style-tag waarbinnen alle vormelementen worden opgenomen. Deze style-tag plaatsen we binnen de head-tag, of nog beter, in een extern document.

Ook voor JavaScript trachten we de script-tags zoveel mogelijk bij elkaar te houden. We plaatsen de script-tag dus ook beter onderaan in de body-tag of, indien je het script herbruikt op meerdere pagina's, in een extern js-document.

Maar hoe moet het dan met onze `document.write(...)` functie uit voorgaande voorbeelden?

Met jQuery is dit makkelijk op te lossen. Daarom een extra if-else oefening waarbij we al een tipje van de jQuery-sluier oplichten.

> Open **jsIntro/if_else_2.html**.
> Vul een getal in.
> Bekijk de broncode.

```
<!-- HTML-code -->
<label for="getal">Getal: </label>
<div class="input">
    <input type="number" id="getal">
</div>
<div class="message-bar background-success" id="resultaat"></div>
```

```
// jQuery script onderaan de pagina
$('#getal').keyup(function () {
    var getal = parseInt($(this).val());
    var rest = (getal % 2 == 0) ? 'even' : 'oneven';
    $('#resultaat').html(getal + ' is een <b>' + rest + '</b>
getal.');
});
```

De elementen die voor ons belangrijk zijn, zijn het tekstveld met id #getal en de div-tag met id #resultaat.

Dan naar het script. Alle jQuery statements beginnen met een dollarteken en de verschillende acties worden met de puntnotatie aan elkaar gelinkt. We krijgen als het ware een ketting van acties.

Laat ons de ketting even verduidelijken.

- `$('#getal');` Het eerste deel in de ketting is steeds een selector. In dit voorbeeld wordt het element met id `#getal` geselecteerd. Merk vooral de analogie met een CSS-selector op!
- `$('#getal').keyup(function(){...});` Na de selector volgt het event waar we naar kijken. In dit voorbeeld het keyup-event. Telkens je een toets los laat, wordt dit event opnieuw getriggerd. Aan het keyup-event koppelen we een functie (een actie).
- `var getal = $(this).val();` Binnen de selector `$(this)` verwijst het kernwoord `this` naar het huidige object, het tekstveld dus. De waarde (de value) van het tekstveld

halen we op met `val()`. De ingevulde waarde wordt vervolgens aan de variabele getal gekoppeld.

- `var rest = (getal%2 == 0) ? 'even' : 'oneven';` Met de verkorte if-else constructie bepalen we of het getal even of oneven is. Het resultaat (de string even of oneven) komt in de variabele rest terecht.
- `$('#resultaat').html(getal + ' is een ' + rest + ' getal.');` Selecteer het element met id #resultaat (de div-tag dus). Plaats vervolgens het resultaat van `getal + ' is een ' + rest + ' getal.'` als HTML-code binnen de div-tag.

> In jQuery vervangt `html(...)` de JavaScriptmethode `document.write(...)`.
> Met de methode `html(...)` kan u zowel tekst als HTML-code aan de pagina toevoegen.
> Met een gelijkaardige methode `text(...)` kan u enkel tekst toevoegen.

- `$('#div1').html('Dit is <em>HTML</em>.');` geeft als resultaat: Dit is HTML.
- `$('#div1').text('Dit is <em>TEXT</em>.');` geeft als resultaat: Dit is `<em>`TEXT`</em>`.

Switch

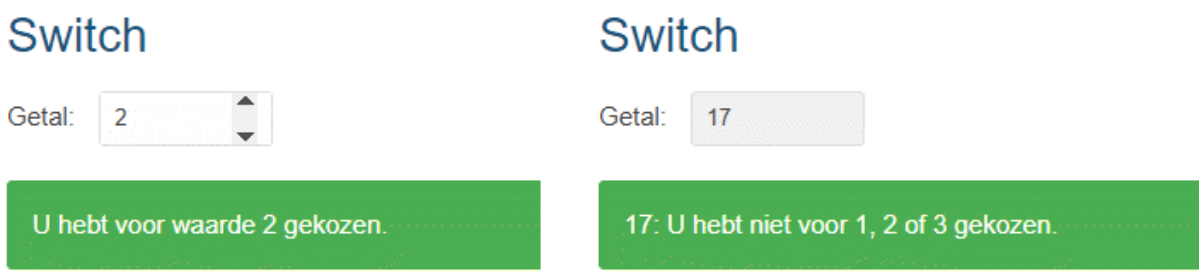
Met een switch test j de waarde van een variabele in functie van een aantal voorgedefinieerde waardes.

> Open **jsIntro/switch.html**.
> Vul een getal in.
> Bekijk de broncode.

```
<!-- HTML-code -->
<label for="getal">Getal: </label>
<div class="input">
    <input type="number" id="getal">
</div>
<div class="message-bar background-success" id="resultaat"></div>
```

```
// het script
$('#getal').keyup(function(){
    var getal = parseInt($(this).val());
    var txt;
    switch(getal) {
        case 1:
            txt = "U hebt voor waarde 1 gekozen.";
            break;
        case 2:
            txt = "U hebt voor waarde 2 gekozen.";
            break;
        case 3:
            txt = "U hebt voor waarde 3 gekozen.";
            break;
        default:
            txt = getal + ": U hebt niet voor 1, 2 of 3 gekozen.";
```

```
            break;
    }
    $('#resultaat').html(txt);
});
```



Net zoals in voorgaande oefening wordt het getal via jQuery opgehaald en in de variabele getal geplaatst. getal wordt vergeleken met de eerste case voorwaarde. Indien beide waardes overeenstemmen, worden de bijbehorende statements uitgevoerd. break zorgt ervoor dat men niet in een volgende case terechtkomt. Indien de voorwaarde niet waar is, gaan we verder naar de volgende case enz...

Indien geen enkele voorwaarde voldoet, wordt het default statement uitgevoerd.

Let op dat u de break niet vergeet. Anders loopt het programma altijd verder in de volgende case en bekomt u een foutief resultaat.

Lussen

For

```
// for-lus
for (van; tot; teller verhogen){
    // herhaalt n-maal dezelfde instructie.
}
```

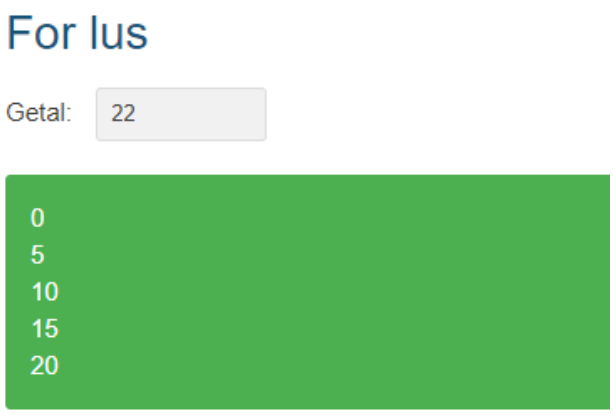
Open **jsIntro/for.html**.
Vul een getal in en klik vervolgens op de tabtoets (= blur-event).
Bekijk de broncode.

```
// Tellen met For-lus
$('#getal').blur(function(){
    var getal = $(this).val();
    var txt = '';
    for(var i = 0; i<= getal ; i++) {
        txt += i + '<br>';
    }
    $('#resultaat').html(txt);
});
```

Het getal wordt in de variabele getal geplaatst. De tweede variabele txt is een lege string. Binnen de for lus wordt een variabele i aangemaakt en krijgt i de waarde 0. Vervolgens wordt de lus uitgevoerd (i<=getal). Op de linkse afbeelding is de stapgrootte 1 (i wordt telkens met 1 verhoogd via i++).

Pas de code als volgt aan. `for(var i = 0; i <= getal ; i++)` wordt `for(var i = 0; i <= getal ; i += 5)`
Test het resultaat opnieuw in een browser.

De teller wordt nu met een stapgrootte van 5 verhoogd (afbeelding rechts).



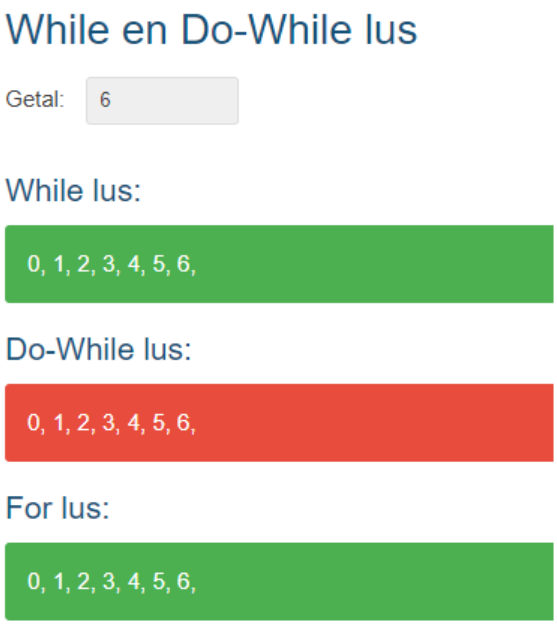
In een lus wordt de declaratie van de teller vaak weggelaten. Beide oplossingen zijn mogelijk: `for( i = 0; i <= tot ; i++)` en `for(var i = 0; i <= tot ; i++)`

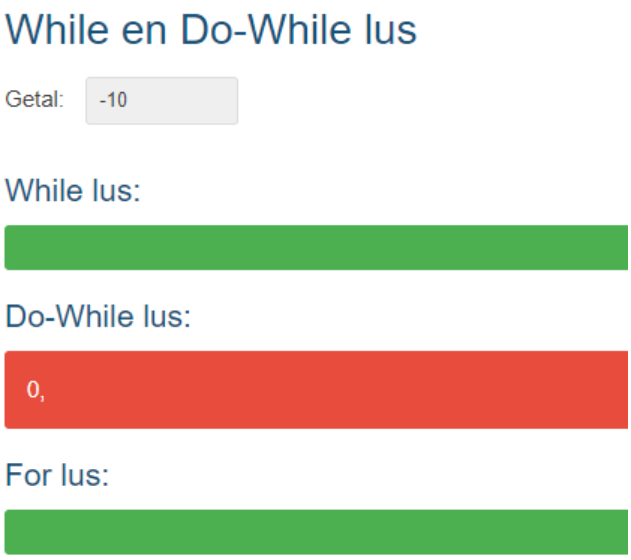
While

Deze lus wordt vaak gebruikt in situaties waarbij het aantal stappen op voorhand onbekend is. De lus stopt op het moment dat de voorwaarde false (onwaar) is.

```
while (voorwaarde) {
    // voer instructie uit en verhoog de teller
}
```

Open **jsIntro/while.html**.
Vul een getal in en klik vervolgens op de tabtoets.
Bekijk de broncode.






```
$('#getal').blur(function(){
    var van = 0;
    var tot = $(this).val();
    var txt= '';
    var i = van;
    while(i <= tot) {
        txt += i + ', ';
        i++;    // teller verhogen!
    }
    $('#resultaatWhile').html(txt);
});
```

Het resultaat van dit voorbeeld is een teller van van tot tot. De while-lus loopt zolang i kleiner dan of gelijk is aan de waarde van tot.

Vergeet binnen een while-lus vooral niet de waarde van de teller met één te verhogen (`i++`), anders blijft de lus oneindig lopen!

In bovenstaand voorbeeld is de telcyclus wel op voorhand gekend en kan u eventueel een for-lus gebruiken. Je hoeft de teller dan ook niet binnen de lus te verhogen.

```
// zelfde bewerking met for-lus
var txt= '';
var i = van;
for(i=0; i<= tot ; i++) {
    txt += i + ', ';
}
$('#resultaatFor').html(txt);
```

Do-while

Deze lus is bijna identiek aan de while-lus. Een while-lus wordt enkel uitgevoerd indien de voorwaarde `true` is. Indien de startvoorwaarde al `false` is, wordt de lus helemaal niet uitgevoerd.

```
do {
   // voer instructie uit en verhoog de teller
```

```
} while (voorwaarde);
```

In tegenstelling tot een while-lus, wordt een do-while-lus altijd minstens één maal uitgevoerd, ook al is de beginvoorwaarde `false`. De test gebeurt immers helemaal op het einde, nadat de bewerking reeds is uitgevoerd.

```
// zelfde bewerking als while, maar nu met een do-while lus
var txt= '';
var i = van;
do {
    txt += i + ', ';
    i++;    // teller verhogen
} while (i <= tot);
$('#resultaatDoWhile').html(txt);
```

Functies

Net zoals in de meeste programmeertalen kan je in JavaScript code in functies onderbrengen. Een functie is een programmablok waarbinnen alle instructies gezamenlijk worden uitgevoerd.

Een functie kan u vanaf meerdere plaatsen in het document oproepen. Meestal gebeurt dit vanuit een event (`click, blur`, ...). Desgewenst kan een functie het resultaat retourneren aan de oproeper.

Indien u een bepaalde functie regelmatig oproept vanuit meerdere pagina's binnen de site, kan je deze best in een apart js-bestand onderbrengen. Op deze manier hoeft u de functie maar op één plaats te onderhouden indien er wijzigingen moeten aangebracht worden.

Een functie begint met het sleutelwoord `function` gevolgd door de naam van de functie met op het einde ronde haakjes. De instructies binnen de functie worden omsloten door accolades.

Een functie kan optioneel één of meerdere parameters of argumenten ontvangen van de instructie die de functie oproept. Deze argumenten staan tussen de haakjes en worden gescheiden door een komma.

```
function functieNaam(param1,param2,...) {
    // voer instructies uit
    // return = resultaat dat wordt teruggegeven
}
```

Door de naam van de functie op een event te plaatsen of rechtstreeks binnen de script-tag, wordt de desbetreffende functie uitgevoerd. Onderstaande functie wordt dadelijk uitgevoerd zodra de script-tag is ingelezen.

```
<body>
    ...
    <script>
        functieNaam(param1,param2,...);
    </script>
    ...
</body>
```

In onderstaand voorbeeld wordt de functie uitgevoerd bij het click-event op een link en op het blur-event van een input veld.

```
<body>
    ...
    <a href="#" onClick="functieNaam(param1,param2,...)">...</a>
    <input ... onBlur="functieNaam(param1,param2,...)">
    ...
</body>
```

Open **jsIntro/functies.html**.
Bekijk de broncode en test het resultaat in een browser.

Functie zonder parameters

In het eerste voorbeeld wordt de functie toonKnop1() opgeroepen door op de button met id #knop1 te klikken (click-event). Binnen de functie wordt het alertvenster (pop-upvenster) getoond met de boodschap U hebt op knop1 geklikt.

```
<body>
    ...
    <button class="button" id="knop1" onClick="toonKnop1()">knop
1</button>
    ...
    <script>
        ...
        function toonKnop1() {
            alert("U hebt op knop1 geklikt.");
        }
        ...
    </script>
</body>
```

Functie met parameters

Stel dat we vanuit meerdere knoppen een pop-upvenster willen activeren, maar telkens met een andere boodschap. Dan kunnen we hier één functie voor schrijven en sturen we een parameter mee. Deze parameter bepaalt wat uiteindelijk de boodschap wordt.

```
<body>
    ...
    <button class="button" id="knop2" onClick="toonKnopX('knop
2')">knop 2</button>
    <button class="button" id="knop3" onClick="toonKnopX('knop
3')">knop 3</button>
    ...
    <script>
        ...
        function toonKnopX(knop) {
            alert("U hebt op " + knop + " geklikt.");
        }
        ...
    </script>
</body>
```

Afhankelijk van de geklikte knop wordt de waarde knop2 of knop3 doorgestuurd naar de functie toonKnopX. Deze waarde wordt in de functie als variabele knop doorgegeven en als dusdanig in de alert-string verwerkt.

Functie met return-waarde

In de twee eerste functies gebeurt alles binnen de functie. Soms is het ook interessant dat de functie een aantal bewerkingen uitvoert en het resultaat teruggeeft aan de oproeper. In het volgend voorbeeld gaan we enkele malen de faculteit van een getal berekenen. (Faculteit van 5 = 1x2x3x4x5 = 120)

```
<body>
    <script>
        function berekenFaculteit(getal) {
            resultaat = 1;
            for(i=1; i<=getal; i++) {
                resultaat = resultaat * i;
            }
            return "Faculteit van " + getal + " = " + resultaat;
        }
        ...
    </script>
    <p>
        <script>document.write(berekenFaculteit(2)); </script><br>
        <script>document.write(berekenFaculteit(5)); </script><br>
        <script>document.write(berekenFaculteit(10)); </script>
    </p>
    ...
</body>
```

Anonieme functie

Een anonieme functie is een functie zonder naam. De functie is meestal gekoppeld aan een variabele of rechtstreeks gekoppeld aan een event.

```
// functie met naam abc
function abc() {
    ...
}

// anonieme functie gekoppeld aan variabele abc
var abc = function() {
    ...
}
```

Er is in feite geen verschil tussen de eerste en de tweede schrijfwijze. In beide gevallen wordt de functie op dezelfde manier opgeroepen.

```
<body>
    ...
    <button class="button" id="knop4" onClick="drukKnop('knop
4')">knop 4</button>
    <button class="button" id="knop5" onClick="drukKnop('knop
5')">knop 5</button>
    ...
    <script>
```

```
        ...
        var drukKnop = function(knop) {
            alert("U hebt op " + knop + " geklikt.");
        }
        ...
    </script>
</body>
```

Functies in jQuery

Zodra je met jQuery aan de slag gaat, kan u niet buiten anonieme functies. Blader maar even terug naar het if-else voorbeeld uit dit hoofdstuk.

```
$('#getal').keyup(function(){
    // voer instructies uit
});
```

In jQuery zijn deze functies meestal niet gekoppeld aan een variabele (het kan wel), maar zijn ze rechtstreeks gekoppeld aan een event.

Open **jsIntro/functies_jQuery.html**.

Hier zie je het jQuery equivalent van de functie `toonKnopX(knop)` uit de vorige paragraaf.

```
<body>
    ...
    <button class="button" id="knop1">knop 1</button>
    <button class="button" id="knop2">knop 2</button>
    <button class="button" id="knop3">knop 3</button>
    ...
    <script>
    ...
    $('.button').click(function(){
        var knop = $(this).text();
        alert("U hebt op " + knop + " geklikt.");
    });
    ...
</script>
</body>
```

- **Lijn 9:** de functie wordt geactiveerd vanuit het click-event op alle objecten met als class .knop.
- **Lijn 10:** het kernwoord `$(this)` verwijst naar het object dat de functie oproept. De waarde of value `val()` van dit object wordt gekoppeld aan de variabele knop.
- **Lijn 11:** Vervolgens wordt het resultaat in een pop-upvenster getoond.

Merk nogmaals op hoe we in dit voorbeeld de code volledig van de HTML scheiden.

Scope (zichtbaarheid) van variabelen

Als een functie wordt aangeroepen, hebben de wijzigingen van variabelen die in de functie gedefinieerd worden geen invloed op de waarde van gelijknamige variabelen buiten de functie.

Open **jsIntro/scopeVars.html.**

Scope (zichtbaarheid) van variabelen

som = 0
som = 11
som = 11

```
var som = 0;
function maakSom(getal1, getal2){
    som = getal1 + getal2;
    return som;
}
$('#lijn1').html(som);
$('#lijn2').html(maakSom(5,6));
$('#lijn3').html(som);
```

- **Lijn 1:** bovenaan het script wordt een variabele som aangemaakt en op 0 gezet.
- **Lijn 6:** schrijf de waarde van som in de eerste paragraaf. som is 0.
- **Lijn 7:** de functie maakSom telt beide getallen op en plaatst het resultaat in de variabele som. som is dus 11.
- **Lijn 8:** schrijf de waarde van som in de derde paragraaf. som is nog altijd 11.

In dit voorbeeld merkt u dat de variabele som, die buiten de functie gedeclareerd is, ook binnen de functie bereikbaar wordt. Met andere woorden: de variabele som is een globale variabele.

Vervang binnen de functie
som = ... door
var som = ...

```
var som = 0;    // voeg het kernwoord var toe
function maakSom(getal1, getal2){
    var som = getal1 + getal2;
    return som;
}
$('#lijn1').html(som);
$('#lijn2').html(maakSom(5,6));
$('#lijn3').html(som);
```

Door het kernwoord var opnieuw te gebruiken binnen de functie, ga je een volledig nieuwe variabele definiëren. Dus som buiten de functie en som binnen de functie zijn niet dezelfde variabele.

- **Lijn 6:** schrijf de waarde van som in de eerste paragraaf. som is 0.
- **Lijn 7:** de functie maakSom telt beide getallen op en plaatst het resultaat in de variabele som. som is dus 11.
- **Lijn 8:** schrijf de waarde van som in de derde paragraaf.

Buiten de functie is som nog steeds 0.

De scope of zichtbaarheid van een variabele beperkt zich tot het blok waarin deze gedefinieerd wordt. Met andere woorden:

- Alle variabelen gedefinieerd binnen een functie zijn niet zichtbaar buiten de functie.
- Alle variabelen gedefinieerd buiten een functie zijn wel zichtbaar binnen alle functies, tenzij een gelijknamige variabele binnen de functie wordt gedefinieerd.

Objecten

Als laatste gaan we nog een zeer belangrijk onderdeel van JavaScript behandelen, namelijk objecten. Bijna alles binnen JavaScript, en ook binnen jQuery, is gebaseerd op objecten.

JavaScript is immers een **OOP** (**O**bject-**O**riented **P**rogramming) taal.

Het is de bedoeling dat u wat meer inzicht krijgt in het begrip object en de bijbehorende termen zoals properties, methods, constructor, enz... zodat u later de jQuery documentatie beter zal begrijpen.

Wat is een object?

Een object is een voorwerp met eigenschappen en functies. Neem bijvoorbeeld het object auto. Een auto heeft als eigenschappen het vermogen, de kleur, ... en als functies rijden, parkeren ....

- Eigenschappen van een object noemt men properties.
- Functies van een object noemt men methodes (methods).

Een object aanmaken

Je kan objecten op verschillende manieren definiëren.

- Via een object literal: door dadelijk waardes aan het object toe te kennen.
  `{eigenschap1: waarde1, eigenschap2: waarde2, ...}`
- Via een object() constructor:
  `var obj = new Object();`
- Via een constructor functie:
  `var obj = function(){...};`

De twee laatste constructors genereren als het ware een `sjabloon` waarop we onze eigenlijke objecten kunnen baseren. Objecten afleiden van een sjabloon noemt men `prototyping`. Dit wil zeggen dat alle publieke variabelen binnen het sjabloon kunnen uitgelezen en aangepast worden en dat je alle interne functies kan uitvoeren.

Als voorbeeld gaan we een eenvoudige constructor functie bekijken.

Eerst definieer je het basissjabloon van bijvoorbeeld een auto. Daarna kan je een specifieke auto koppelen aan het type `Auto` en kan je vervolgens waardes toekennen en de interne functies (de methodes) uitvoeren.

```
// constructor functie (sjabloon) voor het auto object
var Auto = function(kleur,vermogen){
    this.kleur = kleur;
    this.vermogen = vermogen;
```

```
    this.rijden = function(){
        // rij acties
    };
    this.parkeren = function(){
        // parkeer acties
    }
}

// Het object ford
var ford = new Auto('grijs',190);
ford.rijden()        // laat de auto rijden
ford.parkeren()      // laat de auto parkeren
ford.kleur           // wat is de kleur?
ford.vermogen        // wat is het vermogen?
```

Binnen een constructor functie verwijst het kernwoord `this` naar het object dat erop gebaseerd wordt. Dus voor de ford wagen staat `this.kleur` voor `ford.kleur`. Je kan vervolgens een object koppelen aan deze functie door het kernwoord new voor de functievariabele te plaatsen.

Indien we het datatype van `Auto` en ford opvragen, dan geeft dit respectievelijk `function` en `object`. De functie op zich is dus nog geen object!

Variabelen gekoppeld aan constructor functies beginnen steeds met een hoofdletter. Dit is in principe niet verplicht, maar zo merk je snel het verschil tussen een gewone variabele en een constructor.

Ook de ingebouwde JavaScriptobjecten gebruiken deze notatie. Dadelijk gaan we enkele ingebouwde objecten bespreken, maar hier alvast een voorbeeld om een nieuw `Datum` object aan te maken.:

```
var vandaag = new Date();
```

Een constructor functie

Laat ons de theorie omzetten in een praktisch voorbeeld.

Open **jsIntro/constructorFunctie.html**

In dit voorbeeld gaan we een banner object ontwerpen. Onze banner heeft twee eigenschappen (twee properties), namelijk een achtergrondkleur `kleur` en een tag `tag` . Verder heeft de banner één actie (één methode), teken de banner `tekenBanner()`.

We starten dus met een constructor functie die als basis (als sjabloon) dient voor onze objecten.

```
// constructor functie (sjabloon) voor het banner object
var Banner = function(tag, kleur){
    this.tag = tag;
    this.bgKleur = kleur;
    this.tekenBanner = function(){
        $(this.tag).css('background-color', this.bgKleur);
    }
};
```

De methode `tekenBanner()` wordt ook hier weer met een jQuery statement uitgevoerd.

- `$(this.tag)` : selecteer de opgegeven selector.
- `.css('background-color', this.bgKleur)` : en pas de CSS-eigenschap `background-color` aan met de opgegeven kleur.

Vervolgens gaan we drie objecten aanmaken, gebaseerd op deze functie.

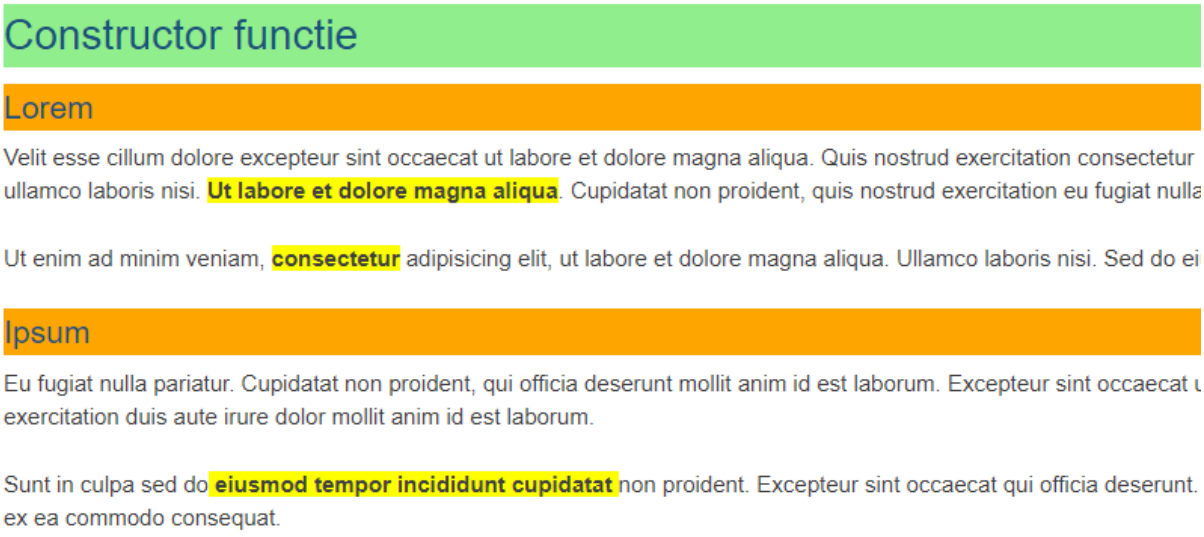
```
// banner 1
var banner1 = new Banner('h1', 'lightGreen');
banner1.tekenBanner();

// banner 2
var banner2 = new Banner('p strong', 'yellow');
banner2.tekenBanner();

// banner 3
var banner3 = new Banner();
banner3.tag = 'h2';
banner3.bgKleur = 'orange';
banner3.tekenBanner();
```

Merk op dat we een object op twee manieren kunnen aanmaken.

In banner1 en banner2 worden de eigenschappen rechtstreeks als parameters naar de functie gestuurd. In banner3 krijgt de functie geen parameters, maar worden de waarden aan de properties rechtstreeks toegekend.

Als we dadelijk enkele ingebouwde JavaScriptobjecten bekijken, zal u merken dat we hoofdzakelijk de tweede manier gebruiken.

Ziehier het resultaat:



Constructor zonder variabelen

Een constructor functie met default eigenschappen

De constructor functie die we net hebben aangemaakt heeft geen default waardes voor de verschillende eigenschappen. Dus telkens je een object aanmaakt, moet je ALLE eigenschappen doorgeven aan de functie. Vergeet je één eigenschap, dan zal de methode `tekenBanner()` niet over voldoende informatie beschikken en kan dus niet worden uitgevoerd. Een tweede probleem

stelt zich bij het rechtstreeks doorgeven van de parameters naar de functie. De volgorde ligt namelijk vast. Eerst komt de parameter `tag` en daarna de parameter `kleur`.

Je merkt al dat deze manier van werken niet echt handig is als het object bijvoorbeeld 10 eigenschappen bevat. Bijvoorbeeld:

```
var obj1 = new Obj(waarde1,waarde2,waarde3 ...,waarde10);
```

Daarom gaan we onze functie nu op een andere manier opstellen. De functie krijgt default waardes waardoor we niet alle parameters hoeven door te sturen en, belangrijker nog, we gaan de parameters niet meer apart (in de juiste volgorde) doorsturen. **Hoe lossen we dit op?**

We sturen maar één parameter door, maar wel volgens een speciale constructie. Namelijk in de vorm van een object. (Een object literal om precies te zijn.). Je hoeft dan nog enkel de parameters door te sturen waarvan we de default waarde wensen te overschrijven. Zelfs de volgorde is niet meer belangrijk. Stel dat we voor parameters 1, 3 en 5 default waarde wensen te overschrijven, dan wordt dit:

```
var obj1 = new Obj({par1:waarde1, par3:waarde3, par5:waarde5});
```

Let vooral op de syntax van een object literal. Het object is omsloten door accolades. De parameters en waardes worden van elkaar gescheiden door een dubbelepunt.

De parameter/waardeparen worden gescheiden door een komma.

In dit voorbeeld gaan we onze banner van daarnet van default waardes voorzien en gaan we nog wat meer CSS-eigenschappen wijzigen.

> Open **jsIntro/constructorFunctieDefault.html**.

Deze functie gebruikt zeven parameters. De parameter `tag` selecteert het element en de zes overige parameters wijzigen de CSS-eigenschappen (achtergrondkleur, tekstkleur, font, ...) van `tag`. De zeven eigenschappen krijgen nu een default waarde (sommige gewoon een lege string). De methode `tekenBanner()` verwerkt de eigenschappen opnieuw in de `css()` methode van jQuery.

De methode `css()` wordt later nog wel verder verduidelijkt, maar voorlopig volstaat deze syntax.

Eén CSS-eigenschap wijzigen:
`$(selector).css(parameter, waarde);`

Meerdere CSS-eigenschappen gelijktijdig wijzigen:
`$(selector).css({parameter1: waarde1, parameter2: waarde2, ...});`

Ook binnen jQuery gebruiken we vaak object literals om waardes door te geven!

Merk op dat er geen vaste parameters via `function()` worden doorgegeven en dat deze dus ook niet meer aan `this.xxx` worden gekoppeld.

```
// constructor functie voor het banner object
var Banner =  function(){
    this.tag = 'h2';
    this.bgKleur = 'lightGreen';
    this.kleur = 'red';
```

```
    this.font = '';
    this.fontSize = '';
    this.align = '';
    this.border = '';
    this.tekenBanner = function(){
        $(this.tag).css({
            'background-color': this.bgKleur,
            'color': this.kleur,
            'font-family': this.font,
            'font-size': this.fontSize,
            'text-align': this.align,
            'border': this.border
        });
    };
    // Deze for-lus hoeft u voorlopig nog niet te begrijpen.
    for(var i in arguments[0]) {
        this[i] = arguments[0][i];
    }
};
```

Verder op de pagina worden de drie objecten aangemaakt.

```
// banner 1
var banner1 = new Banner();
banner1.tekenBanner();

// banner 2
var banner2 = new Banner({tag:'p strong', bgKleur: 'yellow'});
banner2.tekenBanner();

// banner 3
var banner3 = new Banner();
banner3.tag = 'h1';
banner3.bgKleur = 'orange';
banner3.align = 'center';
banner3.border = 'solid 3px blue';
banner3.tekenBanner();
```

De eerste banner banner1, wordt afgeleid van de bannerfunctie en behoudt al zijn default waardes. Resultaat: maak binnen de h2-tag de tekst rood en de achtergrondkleur lichtgroen.

Voor de tweede banner wordt een object doorgestuurd, maar hoe lezen we deze waardes uit? Elke JavaScript functie heeft automatisch het object arguments in zich. De for-in lus op het einde van de functie doorloopt alle parameter/waardeparen die we doorgeven en koppelt de verschillende waardes aan de juiste eigenschap.

Voor de eigenschappen die we via banner2 doorsturen is dit het resultaat:

- `this.tag = 'p strong'`
- `this.bgKleur = 'yellow'`
- alle overige eigenschappen behouden hun default waardes.

Voor de derde banner geven we alle eigenschappen apart door. Ook deze manier van werken blijft dus geldig.

Ziehier het resultaat:

Constructor functie met default eigenschappen

Lorem

Velit esse cillum dolore excepteur sint occaecat ut labore et dolore magna aliqua. Quis nostrud exercitation consectetur adipisicing elit, eu fugiat nulla pariatur. Sunt in culpa qui officia deserunt ullamco laboris nisi. **Ut labore et dolore magna aliqua**. Cupidatat non proident, quis nostrud exercitation eu fugiat nulla pariatur.

Ut enim ad minim veniam, **consectetur** adipisicing elit, ut labore et dolore magna aliqua. Ullamco laboris nisi. Sed do eiusmod tempor incididunt lorem ipsum dolor sit amet, velit esse cillum dolore.

Ipsum

Eu fugiat nulla pariatur. Cupidatat non proident, qui officia deserunt mollit anim id est laborum. Excepteur sint occaecat ut enim ad minim veniam, sed do eiusmod tempor incididunt.Quis nostrud exercitation duis aute irure dolor mollit anim id est laborum.

Sunt in culpa sed do **eiusmod tempor incididunt cupidatat** non proident. Excepteur sint occaecat qui officia deserunt. Ullamco laboris nisi duis aute irure dolor ut enim ad minim veniam. Ut aliquip ex ea commodo consequat.

Constructor met variabelen

Ingebouwde JavaScriptobjecten

Nu we weten wat objecten zijn en hoe we zelf objecten kunnen aanmaken, zal ook de objectstructuur binnen JavaScript snel duidelijk worden. Het is niet de bedoeling van dit boek om alle JavaScriptobjecten in detail te bespreken, maar we gaan wel eens kijken welke types objecten er zijn en waar we de properties en methodes terug vinden. Ter illustratie gaan we enkele objecten aan de hand van een voorbeeld bespreken.

De eerste vraag is natuurlijk waar we een overzicht en beschrijving van alle objecten vinden. Persoonlijk gebruik ik vaak de referentiesite **w3schools.com**.

Deze site beschrijft kort en bondig de belangrijkste objecten geïllustreerd aan de hand van korte voorbeelden. De objecten die hier beschreven staan, worden door alle recente browsers ondersteund.

This section contains a complete JavaScript reference documentation.

JavaScript Reference

The references describe the properties and methods of all JavaScript objects, along with examples.

Array	Boolean	Date	Error	Global
JSON	Math	Number	Operators	RegExp
Statements	String			

Merk op dat we de objecten in drie categorieën kunnen opdelen.

JavaScriptobjecten: de ingebouwde JavaScriptobjecten.

Dit zijn voor ons de interessantste objecten omdat we deze nog vaak in onze toepassingen verwerken.

Browser objecten: objecten specifiek gebonden aan de browser.

Met het History object kunnen we bijvoorbeeld de historiek van het browserwindow (de back button) uitlezen en vanuit JavaScript naar de vorige pagina terugkeren.

HTLM DOM objecten: bevat informatie over alle objecten binnen de pagina.

Zo kunnen we bijvoorbeeld het aantal links op een pagina tellen of de waarde van een invulveld uitlezen en aanpassen. Deze objecten gaan we niet bespreken. : Alle DOM objecten zijn immers veel makkelijker vanuit jQuery te benaderen.

Laat ons van de twee eerste categorieën enkele objecten van naderbij bekijken.

Het JavaScriptobject: Date

Een datum en/of tijd verwerken binnen een script, gaat enkel via het `Date` object.

Date Object Properties

Property	Description
constructor	Returns the function that created the Date object's prototype
prototype	Allows you to add properties and methods to an object

Date Object Methods

Method	Description
getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)

Het Date-object

Het `Date` object kent maar twee properties, namelijk `constructor` en prototype.

De `constructor` kennen we al. Met het kernwoord `new` kunnen we een nieuwe datum aanmaken. De parameters bepalen welke datum we aan ons object toekennen. De `constructor` zonder parameters geeft de datum en tijd van dit moment. Met prototype kan u een object uitbreiden met nieuwe methodes en properties. Dit hoort echter niet echt thuis in een basiscursus en daarom gaan we hier ook niet verder op in.

Het `Date` object heeft heel wat methodes. Om delen van de datum op te vragen (zoals dag, maand, jaar, enz..) gebruiken we methodes die beginnen met `get...()`. Om delen van een datum aan te passen, gebruiken we de methodes beginnend met `set...()`.

> Open **jsIntro/date.html.**

```
// Datum voorbeeld 1
var vandaag = new Date();
var datum = vandaag.getDate() + '-' + (vandaag.getMonth() + 1 ) + '-
' +
    vandaag.getFullYear();
var uur = vandaag.getHours();
var minuten = vandaag.getMinutes();
var APm = '';

if (minuten < 10) {
    minuten = '0' + minuten;
}
if (uur <= 12) {
    APm = ' AM';
}else {
    uur = uur - 12;
    APm = ' PM';
}
$('#datum1').html('Het is vandaag: ' + datum + '<br>Het is momenteel
'
    + uur + ':' + minuten + ' ' + APm);

// Datum voorbeeld 2
var arDagen = ['zo','ma','di','wo','do','vr','za'];
var dag = vandaag.getDay();
$('#datum2').html('Het is vandaag: ' + arDagen[dag] + ' ' + datum);
```

In het eerste voorbeeld wordt eerst een nieuw datum object aangemaakt en gekoppeld aan de variabele vandaag. Uit dit object halen we achtereenvolgens:

- `var datum`
  (is een combinatie van dag, maand en jaar, gescheiden door een streepje.)
- `var uur`
- `var minuten`

`vandaag.getMonth()` geeft een getal tussen 0 (januari) en 11 (december). Om het maandnummer op een normale manier te tonen, tellen we bij `getMonth()` nog één bij.

Date object

Voorbeeld 1

Het is vandaag: 17-4-2018
Het is momenteel 5:52 PM

Voorbeeld 2

Het is vandaag: di 17-4-2018

Datum vandaag

De minuten gaan we altijd met twee cijfers tonen. Dus vijf na tien tonen we als 10:05. Indien het aantal minuten kleiner is dan 10, nemen we de minuten en plakken er de string 0 voor. Dit gebeurt in de eerste if functie.

De tweede if-else controleert of het nu voor- of namiddag is en plaats respectievelijk AM of PM in de variabele APm.

Uiteindelijk worden al deze gegevens samengevoegd tot één string en in de paragraaf met id #datum1 getoond.

In het tweede voorbeeld gaan we ook de dagnaam tonen. Deze gegevens zijn echter niet als dusdanig beschikbaar vanuit het `Date` object. Wat we wel kunnen opvragen, is het cijfer van de dag. `vandaag.getDay()` geeft een cijfer tussen 0 (zondag) en 6 (zaterdag). Om deze cijfers te vertalen in woorden, maken we gebruik van een array.

Een array is een variabele met een lijst van soortgelijke gegevens. Een reeks van dagnamen is bijgevolg ideaal om in een array opgenomen te worden. De gegevens in een array worden van elkaar gescheiden door een komma en in zijn geheel omsloten door vierkante haakjes. Onze array met dagnamen kan er dus als volgt uitzien:

```
var arDagen = ['zo','ma','di','wo','do','vr','za'];
```

Met `arDagen[index]` kunnen we één bepaald element uit de array ophalen. Let op: de index start vanaf 0. Enkele voorbeelden:

`arDagen[0]` geeft 'zo'.

`arDagen[3]` geeft 'wo'.

`arDagen[6]` geeft 'za'.

Of in ons voorbeeld:

- var arDagen = ['zo','ma','di','wo','do','vr','za'];
  var dag = vandaag.getDay();
  $('#datum2').html(... + arDagen[dag] + ...);
- Op zondag is variabele dag gelijk aan 0 en is arDagen[dag] gelijk aan 'zo'.
- Op woensdag is variabele dag gelijk aan 3 en is arDagen[dag] gelijk aan 'wo'.

- Op zaterdag is variabele dag gelijk aan 6 en is arDagen[dag] gelijk aan 'za'.

Het resultaat van deze bewerking plus de datum komt in de paragraaf met id #datum2 terecht.

Het JavaScriptobject: Math

In tegenstelling tot het `Date` object kent het `Math` object geen `constructor`. Je kan het object dus zonder het kernwoord `new` gebruiken. De properties bevatten enkele wiskundige constanten zoals `π`.

Bij de methodes vinden we hoofdzakelijk wiskundige functies zoals sinus, cosinus, getallen afronden en een willekeurig getal genereren.

Math Object Properties

Property	Description
E	Returns Euler's number (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.442)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14)
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)

Math Object Methods

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
acosh(x)	Returns the hyperbolic arccosine of x
asin(x)	Returns the arcsine of x, in radians

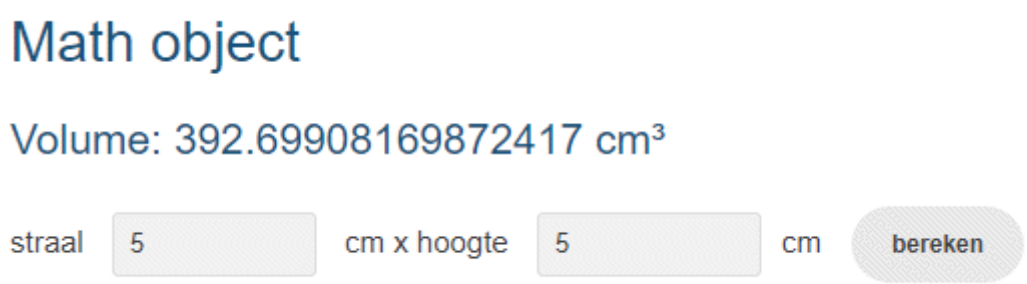
Het Math-object

Open **jsIntro/math.html.**

Het eerste voorbeeld berekent het volume van een cilinder.

```
// volume cilinder (zonder afronding)
$('#bereken').click(function(){
    var r = $('#straal').val();
    var h = $('#hoogte').val();
    var volume = r * r  * Math.PI * h;
    $('#resVol').html('Volume: ' + volume + ' cm&sup3;');
});
```

De straal en de hoogte zijn afkomstig van de twee invulvelden op het formulier. Het volume is gelijk aan straal * straal * π * hoogte.

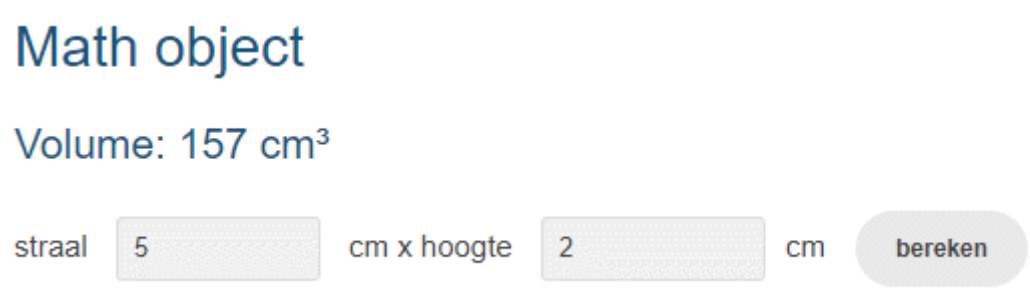


Volume cilinder

Het resultaat toont vrij veel cijfers na de komma.

> Rond het volume af met de methode `Math.round()`.
> Voeg één extra lijn (lijn 6) toe:

```
// volume cilinder (volume afronden)
$('#bereken').click(function(){
    var r = $('#straal').val();
    var h = $('#hoogte').val();
    var volume = r * r  * Math.PI * h;
    volume = Math.round(volume);          // op 0 cijfers afronden
    $('#resVol').html('Volume: ' + volume + ' cm&sup3;');
});
```

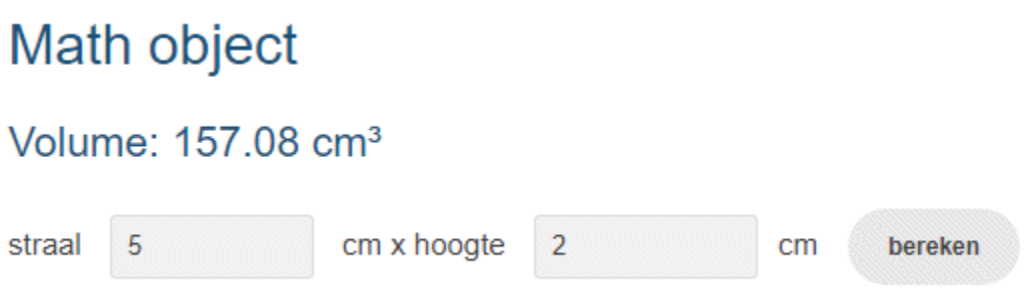


Volume cilinder (geheel getal)

De methode rondt het getal altijd af tot een geheel getal. Met een eenvoudig wiskundig trucje kunnen we het getal makkelijk afronden tot op bijvoorbeeld twee cijfers na de komma. Vermenigvuldig het volume eerst met 100. Rond het getal vervolgens af en deel dit resultaat daarna door 100.

> Rond op lijn 6 het volume op 2 cijfers na de komma af.

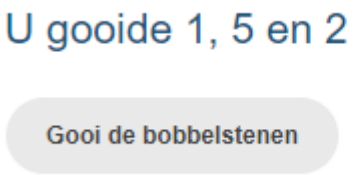
```
// volume cilinder (volume afronden)
$('#bereken').click(function(){
    var r = $('#straal').val();
    var h = $('#hoogte').val();
    var volume = r * r  * Math.PI * h;
    volume = Math.round(volume*100)/100;    // op 2 cijfers afronden
    $('#resVol').html('Volume: ' + volume + ' cm&sup3;');
});
```



Volume cilinder (2 cijfers na de komma)

Het tweede voorbeeld simuleert het gooien met drie dobbelstenen.

```
// dobbelstenen
$('#gooi').click(function(){
    var d1 = Math.floor(Math.random()*6) + 1;
    var d2 = Math.floor(Math.random()*6) + 1;
    var d3 = Math.floor(Math.random()*6) + 1;
    $('#resDobbel').html('U gooide ' + d1 + ', ' + d2 + ' en ' + d3
);
});
```



Dobbelsteen gooien

Voor een dobbelsteen maken we gebruik van de methode `Math.random()`. Deze methode geeft ons een willekeurig getal tussen 0 en 0,999...

Vermenigvuldig dit getal vervolgens met 6. Dit resulteert in een getal tussen 0 en 5,999... Vervolgens ronden we dit getal naar onder af met `Math.floor()`. Zo bekomen we een geheel getal tussen 0 en 5. Als laatste tellen we hier nog één bij.

JavaScript kent drie methodes om een getal af te ronden:

- `Math.round()`: rond vanaf x.5 het getal naar boven af of naar onder indien kleiner.
- `Math.floor()`: rond het getal steeds naar onder af.
- `Math.ceil()`: rond het getal steeds naar boven af.

Het JavaScriptobject: String

String Properties

Property	Description
constructor	Returns the string's constructor function
length	Returns the length of a string
prototype	Allows you to add properties and methods to an object

String Methods

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index

Het String-object

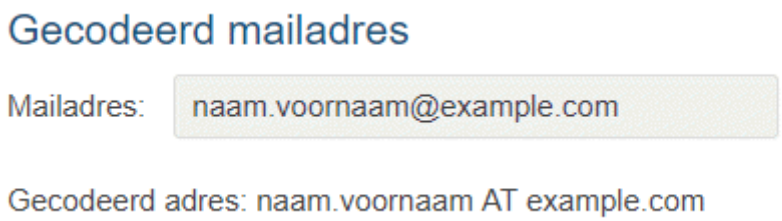
De `constructor` eigenschap van het `String` object gaan we in de praktijk zelden of nooit gebruiken. Zoals we reeds weten, kunnen we een string ook rechtstreeks aan een variabele koppelen. Gelukkig kunnen we alle methodes van het `String` object wel blijven gebruiken zonder dat we de string met het kernwoord `new` hoeven te definiëren.

Open **jsIntro/string.html**.

In dit voorbeeld gaan we een spamvrij mailadres genereren door @ te converteren naar het woord AT en de punt te vervangen door het woord DOT.

```
// email maskeren
$('#email').blur(function(){
    var resultaat = "";
    var mailadres = $(this).val();
    if(mailadres.length == 0) {
        resultaat = "Gelieve <mark>een mailadres</mark> in te
vullen";
    } else if (mailadres.search('@') == -1) {
```

```
        resultaat = "Dit is <mark>geen geldig mailadres</mark>";
    } else {
        mailadres = mailadres.replace('@', ' AT ');
        resultaat = 'Gecodeerd adres: ' + mailadres;
    }
    $('#resultaat').html(resultaat);
});
```



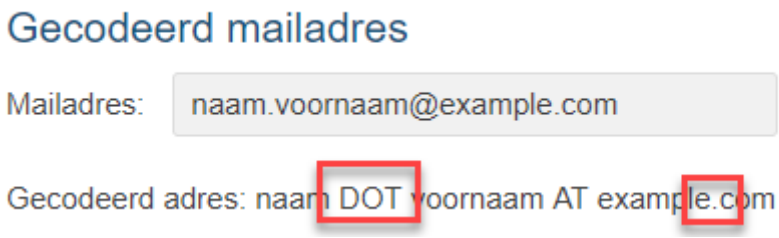
Gecodeerd mailadres

Het resultaat van het tekstveld komt in de variabele mailadres. Eerst wordt gekeken of het veld wel is ingevuld. Indien de string geen waarde bevat (`mailadres.length == 0`) wordt een boodschap in de p-tag met id #resultaat getoond.

Indien de string niet leeg is, wordt met de methode `search()` gecontroleerd of de string het @-symbool bevat. `search()` geeft de positie van het @-symbool weer. Indien het symbool ontbreekt, geeft de methode de waarde -1 terug. Ook hier wordt weer een gepaste boodschap getoond. Bevat de string een @-symbool, dan gaat de methode `replace()` het @-symbool vervangen door het woord AT.

> Pas de functie als volgt aan:
> `mailadres = mailadres.replace('@', ' AT ');` wordt: `mailadres = mailadres.replace('@', ' AT ').replace('.', ' DOT ');`

Door tweemaal de methode `replace()` toe te passen, kunnen we zowel het @-symbool als het puntsymbool in het mailadres vervangen.



Gecodeerd mailadres

Maar wat blijkt nu: enkel het eerste punt is vervangen. Het tweede punt nog niet.

De werkmethode die we nu gebruiken heeft nog wel enkele nadelen. Nu testen we immers uitsluitend op karakters, maar de structuur van een mailadres is net iets ingewikkelder.

Een mailadres is immers opgebouwd volgens een bepaald patroon. Zo moet een mailadres steeds eindigen op een punt, gevolgd door een extensie bestaande uit twee, drie of meerdere letters. Dit kunnen we onmogelijk testen via een string.

De methode `replace()` kan je nog op een tweede manier gebruiken. In plaats van te testen op een string, kan je ook testen op een reguliere expressie (kortweg regex). Een reguliere expressie is in feite een testpatroon.

Hoe je een regex samenstelt, vind je o.a. hier:
http://www.w3schools.com/jsref/jsref_obj_regexp.asp

Gelukkig hoef jeniet alle expressies zelf te ontwerpen. Zo bestaan er verschillende sites waar je de meest voorkomende expressies kan downloaden. Bijvoorbeeld via: http://www.regexlib.com/

Reguliere expressies laten we verder buiten beschouwing, maar in het tweede voorbeeld gaan we toch heel even kijken hoe we in een string niet één, maar alle gevonden karakters kunnen vervangen.

```
// regex
$('#vervang').click(function(){
    var jq = $('#section2').html();
    jq = jq.replace('JavaScript', 'jQuery');
    $('#section3').html(jq);
});
```

Door een druk op de knop, wordt de tekst uit section2 in de variabele jq geplaatst. Vervolgens wordt binnen de string het woord JavaScript vervangen door het woord jQuery. **(Let op: de woorden zijn hoofdlettergevoelig.)** Het resultaat wordt daarna in section3 geplaatst.

Zoals we uit voorgaand voorbeeld weten, wordt enkel het eerste woord vervangen. De vier andere woorden blijven onaangeroerd.

String replace met regex

Originele tekst:

JavaScript is een scripttaal die veel gebruikt wordt om webpagina's interactief te maken en webapplicaties te ontwikkelen. De syntaxis van JavaScript vertoont overeenkomsten met de programmeertaal Java.

Events vormen het hart van een JavaScript-pagina. Event-handling staat toe dat de JavaScript opdrachten op het juiste moment worden uitgevoerd, wanneer de juiste gebreurtenis heeft plaatsgevonden:
Een berekening wordt uitgevoerd wanneer een gebruiker op een knop klikt, een figuur verandert wanneer de gebruiker er met de muis over beweegt, een animatie wordt gestart wanneer de pagina geladen is ...

Een functie kan je in JavaScript gebruiken om een groep opdrachten te kunnen uitvoeren met een aangegeven naam. De opdrachten of statements die binnen de functie staan worden uitgevoerd wanneer de functie wordt aangeroepen.Een functie kan waarden ontvangen en retourneren naar de plaats van aanroep.

Vervang 'JavaScript' door 'jQuery'

Aangepaste tekst:

jQuery is een scripttaal die veel gebruikt wordt om webpagina's interactief te maken en webapplicaties te ontwikkelen. De syntaxis van JavaScript vertoont overeenkomsten met de programmeertaal Java.

Events vormen het hart van een JavaScript-pagina. Event-handling staat toe dat de JavaScript opdrachten op het juiste moment worden uitgevoerd, wanneer de juiste gebreurtenis heeft plaatsgevonden:
Een berekening wordt uitgevoerd wanneer een gebruiker op een knop klikt, een figuur verandert wanneer de gebruiker er met de muis over beweegt, een animatie wordt gestart wanneer de pagina geladen is ...

Een functie kan je in JavaScript gebruiken om een groep opdrachten te kunnen uitvoeren met een aangegeven naam. De opdrachten of statements die binnen de functie staan worden uitgevoerd wanneer de functie wordt aangeroepen.Een functie kan waarden ontvangen en retourneren naar de plaats van aanroep.

In plaats van te zoeken naar de string JavaScript, gaan we nu vergelijken met een regex. Dit is in onze toepassing vrij simpel. Vervang de aanhalingstekens door een slash. Dus `'JavaScript'` wordt `/JavaScript/`.

> Pas de code als volgt aan en test het resultaat in een browser.

```
// regex
$('#vervang').click(function(){
    var jq = $('#section2').html();
    // vervang '' door /JavaScript/
    jq = jq.replace(/JavaScript/, 'jQuery');
    $('#section3').html(jq);
});
```

Voorlopig merk je nog geen verschil met de vorige oplossing. We hebben in de expressie ook nog niet aangegeven dat we naar alle woorden op zoek zijn. Om te zoeken naar alle overeenkomstige woorden, voeg je op het einde van de expressie de kleine letter g toe.

> Pas de code als volgt aan en test het resultaat in een browser.

```
// regex
$('#vervang').click(function(){
    var jq = $('#section2').html();
    // voeg achter /JavaScript/ de letter g toe
    jq = jq.replace(/JavaScript/g, 'jQuery');
    $('#section3').html(jq);
});
```

String replace met regex

Originele tekst:

JavaScript is een scripttaal die veel gebruikt wordt om webpagina's interactief te maken en webapplicaties te ontwikkelen. De syntaxis van JavaScript vertoont overeenkomsten met de programmeertaal Java.

Events vormen het hart van een JavaScript-pagina. Event-handling staat toe dat de JavaScript opdrachten op het juiste moment worden uitgevoerd, wanneer de juiste gebreurtenis heeft plaatsgevonden:
Een berekening wordt uitgevoerd wanneer een gebruiker op een knop klikt, een figuur verandert wanneer de gebruiker er met de muis over beweegt, een animatie wordt gestart wanneer de pagina geladen is ...

Een functie kan je in JavaScript gebruiken om een groep opdrachten te kunnen uitvoeren met een aangegeven naam. De opdrachten of statements die binnen de functie staan worden uitgevoerd wanneer de functie wordt aangeroepen.Een functie kan waarden ontvangen en retourneren naar de plaats van aanroep.

Vervang 'JavaScript' door 'jQuery'

Aangepaste tekst:

jQuery is een scripttaal die veel gebruikt wordt om webpagina's interactief te maken en webapplicaties te ontwikkelen. De syntaxis van jQuery vertoont overeenkomsten met de programmeertaal Java.

Events vormen het hart van een jQuery-pagina. Event-handling staat toe dat de jQuery opdrachten op het juiste moment worden uitgevoerd, wanneer de juiste gebreurtenis heeft plaatsgevonden:
Een berekening wordt uitgevoerd wanneer een gebruiker op een knop klikt, een figuur verandert wanneer de gebruiker er met de muis over beweegt, een animatie wordt gestart wanneer de pagina geladen is ...

Een functie kan je in jQuery gebruiken om een groep opdrachten te kunnen uitvoeren met een aangegeven naam. De opdrachten of statements die binnen de functie staan worden uitgevoerd wanneer de functie wordt aangeroepen.Een functie kan waarden ontvangen en retourneren naar de plaats van aanroep.

Het browser object: History

Als laatste nog een voorbeeld van één van de vijf browser objecten. In tegenstelling tot de eerder geziene objecten kan u deze objecten niet zelf aanmaken met een `constructor`. De objecten zijn immers al aangemaakt vanuit de browser. Net als andere objecten hebben de browser objecten ook properties en methodes.

Neem bijvoorbeeld het `History` object. Met het `History` object kan je de geschiedenis van het browservenster ondervragen.

Open **jsIntro/history.html.**

Zoals je in onderstaande afbeelding merkt, heb ik deze pagina niet rechtstreeks in een nieuw venster geopend, maar zit er nog een hele geschiedenis voor. Achter de terugknop van de browser vind je de verschillende pagina's die dit browservenster heeft doorlopen.

```
// History van de pagina
$('h2').html('History.length = ' + history.length);

// Ga naar vorige pagina
$('#h1').click(function(){
    history.back();
});

// Ga naar eerste pagina
$('#h2').click(function(){
    history.go( - (history.length - 1));
```

```
});
```

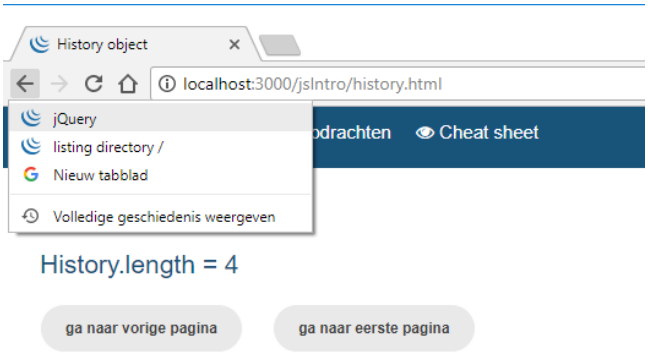
De eigenschap `history.length` geeft het aantal pagina's in de historiek weer. In mijn voorbeeld sta ik momenteel op het vijfde venster.

Verder heeft het history object drie methodes:

- `history.back()`: ga één stap terug in de historiek.
- `history.forward()`: ga één stap verder in de historiek.
- `history.go(n)`: ga n stappen verder (positief getal) of terug (negatief getal) in de historiek.

Om terug te keren naar de eerste pagina uit de historiek, neem kede lengte van de historiek min één en maakt u dit getal negatief.

In onderstaand voorbeeld wordt dit `history.go(-3)`.



Browser history

jQuery basis

In het vorige hoofdstuk zijn we al even in aanraking gekomen met jQuery. Twee zeer belangrijke eigenschappen hebben we met opzet nog niet besproken. Als eerste moeten we steeds het jQuery framework integreren op de pagina. Zonder dit framework zal de pagina niet functioneren. Verder staat alle jQuery functionaliteit binnen een speciale jQuery functie.

```
<script src="pad/jquery.x.x.js"></script>
<script>
$(function(){
    // jQuery functionaliteit
});
</script>
```

jQuery integreren

Voor u gebruik kan maken van het jQuery framework, moet u eerst een verwijzing (een link) in uw webpagina's integreren. Ofwel kan u de bibliotheek downloaden, ofwel maakt u gebruik van de online bibliotheek van Google, Microsoft of jQuery.

Framework downloaden

Op de website http://jquery.com/ kan u de laatste release van het jQuery Framework in twee versies downloaden:

- **Productieversie** (jquery-x.x.x.min.js) Dit is het kleinste bestand en daarom ideaal om in uw site te verwerken. In deze versie zijn alle spaties uit de code verwijderd en is het bestand ook nog eens gecomprimeerd.
- **Ontwikkelversie** (jquery-x.x.x.js) Dit bestand is niet gecomprimeerd en is bijgevolg veel groter. De code in dit bestand is mooi geformatteerd en zodoende makkelijk "leesbaar" voor de programmeur.

Download steeds de productieversie naar de website en leg net voor </body> of binnen de head-tag een verwijzing naar het script. Let op: alle volgende scripts die u zelf schrijft en die gebruik maken van jQuery, moet u achter het basisscript toevoegen.

> Download steeds de **productieversie** naar de website en leg **net voor** `</body>` of binnen de head-tag een verwijzing naar het script. Let op: alle volgende scripts die u zelf schrijft en die gebruik maken van jQuery, moet u achter het basisscript toevoegen.

Optie 1: JavaScript bovenaan de pagina (niet ideaal)

```
<!DOCTYPE HTML>
<html>
<head>
     <meta charset="utf-8">
     <title>jQuery</title>
     <script src="../jQuery/jquery-x.x.x.min.js"></script>
     <script src="mijnSiteScript.js"></script>
     <script>
        // mijn lokaal script
     </script>
</head>
```

Optie 2: JavaScript onderaan de pagina (beste optie)

```
<!DOCTYPE HTML>
<html>
<head>
     ...
</head>
<body>
     ...
     <script src="../jQuery/jquery-x.x.x.min.js"></script>
     <script src="mijnSiteScript.js"></script>
     <script>
        // mijn lokaal script
     </script>
</body>
```

Sommige ontwikkelaars downloaden het script naar hun eigen site, maar u kan ook linken naar een online script bij **Google**, **cdnjs** of op de **jQuery** website.

Framework linken via CDN

Zowel **Google** als **cdnjs** hosten een aantal open-source bibliotheken waar u rechtstreeks naar kan verwijzen. Dit noemt men een **C**ontent **D**elivery **N**etwork (CDN). Door rechtstreeks naar een CDN te

linken, bespaart u wat bandbreedte op uw eigen netwerk. Een gebruiker downloadt het bestand immers niet vanaf uw server, maar rechtstreeks vanaf de externe CDN-server. De kans is groot dat een bezoeker reeds vanaf een andere website het script heeft ingelezen, zodat het script reeds in het cachegeheugen van de browser aanwezig is. Hierdoor reageert uw pagina ook iets sneller.

```
<script src="externe CDN link naar de jQuery bibliotheek"></script>
<script src="mijnSiteScript.js"></script>
<script>
    // mijn lokaal script hier
</script>
```

CDN-servers voor jQuery:

- jQuery: http://jquery.com/download/
- Google: https://developers.google.com/speed/libraries/
- Cdnjs: https://cdnjs.com/libraries/jquery/

$(document).ready()

Scripts die dadelijk worden uitgevoerd, plaatst men in een JavaScripttoepassing steeds in de `window.onload` methode. Zodra de pagina VOLLEDIG geladen is, wordt het script dan uitgevoerd. Volledig geladen wil zeggen dat zowel de HTML, de JavaScriptcode als alle externe media zoals foto's en filmpjes, volledig in het geheugen zijn geladen. Zoals u dadelijk zal merken zorgt het laden van grote bestanden voor een kleine vertraging die de gebruiksvriendelijkheid niet ten goede komt.

jQuery biedt hiervoor een alternatief. De methode `$(document).ready()` wordt uitgevoerd NET VOOR de externe media worden ingeladen. Ook al zit er tussen beide methodes vaak maar een fractie van een seconde, de jQuery methode zorgt bijgevolg voor een betere gebruikerservaring. U kan de vrij lange notatie trouwens nog sterk inkorten.

```
// lange notatie
$(document).ready(function(){
     ...
});
// korte notatie
$(function(){
      ...
});
```

Open **jqIntegratie/youTube_1.htm**
Open **jqIntegratie/youTube_2.html**

Beide pagina's bevatten een gelijkaardige functionaliteit. De container #movie bevat een YouTube filmpje. Bij het laden van de pagina wordt de container onzichtbaar gemaakt. Door op de link te klikken, komt het filmpje tevoorschijn.

In het eerste document wordt de container vanuit een standaard JavaScript onzichtbaar gemaakt. In het tweede document gebeurt dit via een jQueryfunctie.

De code voor **youTube_1.html**:

```
<script>
window.onload = function() {
    document.getElementById('movie').style.display = 'none';
}
function toonMovie(){
    document.getElementById('movie').style.display = 'block';
}
</script>
...
<p><a href="#" onclick="toonMovie();">Toon movie.</a></p>
<section id="movie"> ... </section>
```

De code voor **youTube_2.html**:

```
<script>
$(function() {
    $('#movie').hide();          // verberg selectie met hide()
    $('#movieLink').click(function(){
        $('#movie').show();      // toon selectie met show()
    });
});
</script>
...
<p><a href="#" id="movieLink">Toon movie.</a></p>
<section id="movie"> ... </section>
```

Open beide pagina's in een browser
Vernieuw de pagina's enkele malen en vergelijk

Op youTube_1.html verschijnt kortstondig een lege container voor deze zich sluit. `window.onload` komt pas in actie zodra de externe movie VOLLEDIG is ingeladen. De tweede pagina, youTube_2.html, opent dadelijk met een gesloten container. `$(function() { ... }` komt reeds in actie VOOR de externe movie is ingeladen.

Unobtrusive JavaScript

In een goede webpagina wordt de structuur (de HTML-code) volledig gescheiden van de opmaak (CSS). Met deze techniek bent u zeker vertrouwd.

Ook de JavaScripts worden best volledig gescheiden van de HTML-structuur. Dit heeft als voordeel dat we de code makkelijk kunnen onderhouden en hergebruiken.

Zetten we de vergelijking met CSS nog even op een rijtje:

- **Inline CSS en Obtrusive JavaScript:**
  Inline CSS: `<span style="color: sienna;">...`
  Obtrusive JavaScript: `<a onclick="alert('Link 1 geklikt')"...`
  Stijleigenschappen of JavaScript rechtstreeks op de tag zijn niet efficiënt en volledig af te raden. Aanpassingen zijn vaak tijdrovend.
- **Interne CSS en Intern unobtrusive JavaScript:**
  Interne CSS: style-tag in de head van het document.
  Intern unobtrusive JavaScript: script-tag in de head-sectie of in de body-tag van het document.

Stijleigenschappen of JavaScript binnen de head-sectie of in de body-tag van het document zijn enkel bruikbaar binnen het document zelf.

- **Externe CSS - Extern unobtrusive JavaScript:**
  Externe CSS: link naar een extern .css-bestand.
  Extern unobtrusive JavaScript: link naar een extern .js-bestand.
  Stijleigenschappen of JavaScript in een extern document zijn bruikbaar binnen de ganse website.

Met jQuery trachten we zo veel mogelijk de scripts volledig te scheiden van de HTML-code. Voorgaande oefeningen zijn hiervan een goed voorbeeld. Op youTube_1.html wordt het openen van de container rechtstreeks gekoppeld aan de a-tag `(<a onclick="toonMovie();").`

De tweede pagina youTube_2.html heeft geen onclick-event meer. In het script zelf wordt het click-event gekoppeld aan het id #movieLink.

> Alle functies die we schrijven MOETEN steeds binnen `$(function() { ... }` staan. jQueryscripts buiten deze methode werken niet.

Onderstaande code is NIET correct.

Lijn 4, 5 en 6 bevinden zich buiten `$(function() { ... }` en worden dus niet door jQuery herkend!

```
$(function() {
    $('#movie').hide();                // correct
});
$('#movieLink').click(function(){    // fout!
    $('#movie').show();
});
```

Onderstaande code is correct:

```
$(function() {
    $('#movie').hide();
    $('#movieLink').click(function(){
        $('#movie').show();
    });
});
```

> Let ook op met globale en lokale variabelen. Een variabele aangemaakt binnen `$(function() { ... });` is niet zichtbaar buiten deze functie. De variabele is echter wel zichtbaar binnen geneste functies.

jQuery documentatie (API)

De website http://api.jquery.com/ bevat een schat aan informatie. In de API reference zijn alle eigenschappen en methodes mooi gerangschikt volgens thema. Maar hoe moet je de documentatie nu begrijpen?

Neem bijvoorbeeld de methode `css()` uit de categorie CSS. Bovenaan de pagina vind je een korte omschrijving van de mogelijkheden. Verderop worden de verschillende mogelijkheden in detail besproken en geïllustreerd aan de hand van eenvoudige voorbeelden.

.css()

Categories: CSS | Manipulation > Style Properties

Get the value of a computed style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

Contents:

- .css(propertyName)
  - .css(propertyName)
  - .css(propertyNames)
- .css(propertyName, value)
  - .css(propertyName, value)
  - .css(propertyName, function)
  - .css(properties)

css-methode van jQuery

Deze methode, net zoals vele andere methodes uit de bibliotheek, kan u op verschillende manieren gebruiken.

- **Getter** (haal de waarde van een CSS-eigenschap op).
  `.css(propertyName)`
  `.css(propertyNames)`

- **Setter** (geef de CSS-eigenschap een nieuwe waarde).
  `.css(propertyName,value)`
  `.css(propertyName,function)`
  `.css(properties)`

Open **jqIntegratie/api.html.**

Laat ons bovenstaande methodes even verduidelijken.

getter vs. setter

Getter

```
$(function() {
    $('#getter').click(function(){
        alert($('#sect2').css('width'));
    });
});
```

Met de eerste functie (een getter) kan u een waarde ophalen. Met de methode `css()` kan u één of meerdere css-eigenschappen gelijktijdig opvragen.

Op lijn 3 wordt via `$('#sect2').css('width')` de breedte van de container #sect2 opgevraagd en in een alertbox getoond.

Setter met één parameter

```
$(function() {
    $('#setter1').click(function(){
        $('#sect2').css('width', '400px');
    });
});
```

Op lijn 3 wordt één css-eigenschap, de breedte van #sect2, aangepast via een setter. De eigenschap en de bijhoorende waarde worden altijd gescheiden door een komma: `'naam','waarde'`. Zowel de naam als de waarde worden als een string doorgeven. Vergeet dus de aanhalingstekens niet!

Setter met één of meer parameters

```
$(function() {
    $('#setter2').click(function(){
        $('#sect2').css({
            'box-shadow': '5px 5px 3px grey',
```

```
            'color': 'green',
            'width': '700px'
        });
    });
});
```

U kan meerdere eigenschappen gelijktijdig aanpassen (lijn 4, 5 en 6) door gebruik te maken van een map of object literal.

> In een object literal worden de waardes altijd omsloten door aanhalingstekens. Voor de namen hoeft dit niet. Indien u de aanhalingstekens weglaat, moet u de namen wel in camelCasenotatie schrijven!

Beide schrijfwijzen zijn dus correct:

Goed: `.css({'box-shadow':'5px 5px 3px grey', 'color':'green', ...})`

Goed: `.css({boxShadow:'5px 5px 3px grey', color: 'green', ...})`

Fout: `.css({box-shadow:'5px 5px 3px grey', color: 'green', ...})`

Setter met een anonieme functie

```
$(function() {
    $('#setter3').click(function(){
        $('#sect2').css({
            fontSize: function(){
                return parseFloat($('#sect2').css('font-size')) *
1.2;
            }
        });
    });
});
```

In het laatste voorbeeld op de pagina gaan we op lijn 5 de lettergrootte bij elke klik vergroten met een factor 1,2. Hiervoor hebben we natuurlijk een functie nodig die de huidige waarde uitleest en vervolgens de nieuwe waarde berekent.

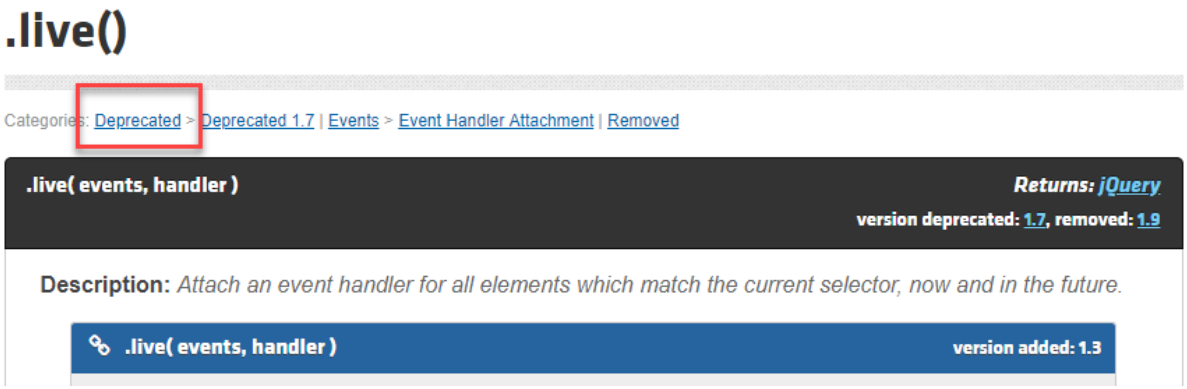
Zoals u merkt aan de bovenstaande syntax, wordt `fontSize` nu niet gekoppeld aan een vaste waarde, maar wordt de waarde vervangen door een functie waarbinnen de vermenigvuldiging plaatsvindt.

In feite is deze schrijfwijze eveneens een object literal en mogen we de CSS-eigenschap met of zonder aanhalingstekens schrijven. Let ook hier op de camelCase schrijfwijze indien u de aanhalingstekens weglaat.

Afgeschafte (deprecated) methodes en selectors

Verschillende versies van jQuery volgen elkaar in een razendsnel tempo op. Meestal komen er bij elke versies enkele nieuwe methodes bij, maar er verdwijnen er ook. Vaak gaat het dan over verouderde methodes, methodes die weinig of niet gebruikt worden of methodes waarvoor een

beter alternatief beschikbaar is. In de API worden deze methodes aangeduid met de term deprecated.



Deprecated methodes

Voorlopig zijn de meeste van deze methodes en selectors nog in de bibliotheek aanwezig, maar de kans is groot dat deze in een toekomstige versie volledig zullen verdwijnen. Het is dus niet aan te raden om deze verouderde methodes in een nieuw ontwerp te verwerken. In deze cursus komen deprecated methodes dan ook niet aan bod.

Een volledig overzicht van alle afgekeurde methodes en selectors vindt u hier: http://api.jquery.com/category/deprecated/

Overzicht depricated methodes en selectors

jQuery cheat sheet

Zodra je wat beter vertrouwd bent met jQuery, hebt u waarschijnlijk al voldoende aan een korte omschrijving van de verschillende methodes en eigenschappen. Op het net zijn ondertussen ook al verschillende spiekbriefjes (cheat sheets) beschikbaar die zeker nuttig zijn tijdens het programmeren.

- Online cheat sheet: http://oscarotero.com/jquery/

jQuery in combinatie met andere JavaScriptbibliotheken

Misschien kom je hier nooit mee in aanraking, maar wat kan er gebeuren indien u meerdere JavaScriptbibliotheken in één project verwerkt?

Zoals je ondertussen weet, gebruikt jQuery default het dollarteken als alias voor het woord jQuery. Bevat jouww project ook nog een andere JavaScriptbibliotheek die het dollarteken als alias gebruikt (bijvoorbeeld het Prototype framework), dan komen beide bibliotheken natuurlijk in conflict met elkaar.

Via de methode `noConflict()` kan u een nieuwe alias creëren voor jQuery.

```
var $jq = jQuery.noConflict();
// gebruik vanaf nu $jq i.p.v. $
$jq(function() {
    $jq('#movie').hide();
    $jq('#movieLink').click(function(){
        $jq('#movie').show();
    });
});
```

Vanaf nu gebruik je enkel nog de alias `$jq` voor jQueryfuncties en blijft `$` voorbehouden voor de Prototypefuncties.

Elementen selecteren

Een jQuery statement bestaat steeds uit twee of meer delen. Het eerste deel is altijd een selector. De volgende delen beschrijven de acties die u op de selector uitvoert.

Een voorbeeld:

```
$('p.foo').addClass('bar').show('slow');
```

- `$('p.foo')` : selecteer alle p-tags met het class-attribuut `.foo`.
- `.addClass('bar')` : voeg aan deze selectie de class `.bar` toe.
- `.show('slow');` en maak de selectie zichtbaar.

Een goede selectie maken is bijgevolg cruciaal om een krachtig script uit te voeren.

JavaScript kent een zeer beperkte set aan selectors.

Je kan bijvoorbeeld een bepaald id (`document.getElementById(id)`) of een bepaalde tag opvragen (`document.getElementByTagName(tag)`), maar wat als u de derde rij in een tabel wilt ophalen of alle links met een externe URL? Dit is onmogelijk met één commando.

Dit hoofdstuk beschrijft de belangrijkste selectiemethodes binnen jQuery. De syntax is eenvoudig. Deze volgt immers de selectiemethodes van CSS3, aangevuld met enkele jQuery-specifieke selectiemethodes.

De voorbeelden die volgen, kan u uittesten op het oefenbestand **selectorTest.html**.

> Open **selectors/selectorTest.html**.
> De voorbeelden die volgen, kan je uittesten op dit oefenbestand.

Geef in het tekstveld een selectie in (bv: `p#paragraaf1`). Het script op de pagina gaat op zoek naar de selectie en voegt aan de selectie de class `.highLight` toe. Deze class tekent een groen kader rond de selector, maakt de achtergrond geel en kleurt de tekst rood.

Basis selectors

Een overzicht van de verschillende basis selectors vind je hier:

http://api.jquery.com/category/selectors/basic-css-selectors/

Tags, classes en id's

Volgende voorbeelden gebruikt u waarschijnlijk dagelijks in CSS:

```
$('a')                  // selecteer alle a-tags
$('a, tr')              // selecteer alle a-tags en alle tr-tags
$('#paragraaf1')        // selecteer de tag met id #paragraaf1
$('.sublist')           // selecteer alle tags met class .sublist
```

Universele selector

Het sterretje is, net zoals in CSS, de universele selector voor alle tags.

```
$('*')                  // selecteer alle tags
```

Hiërarchische selectors

Zie: http://api.jquery.com/category/selectors/hierarchy-selectors/

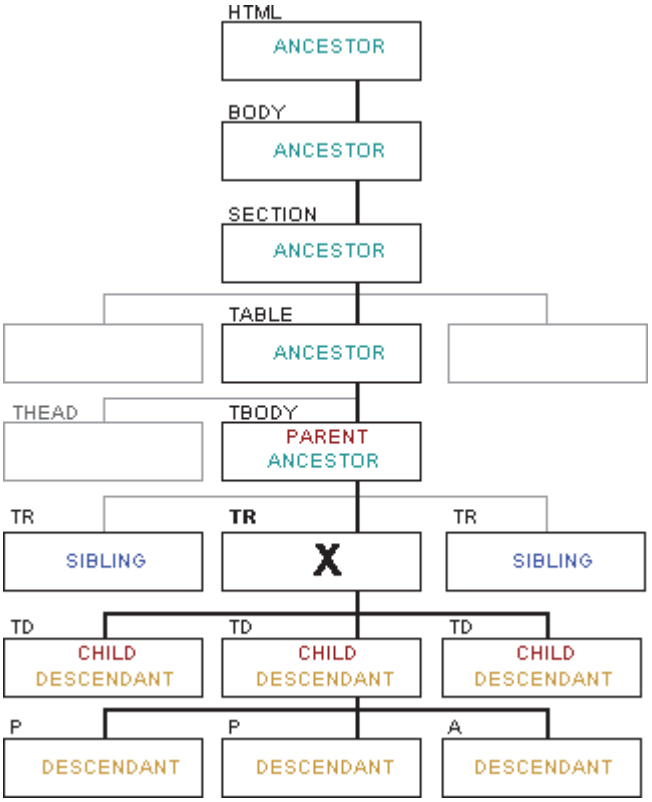
Hiërarchische selectors kan u best vergelijken met een stamboom. Bekijk in onderstaande code hoe de tags in elkaar genest zijn.

```
<table class="table">
    <thead>
        <tr>
            <th>Opleiding:</th>
            <th>Datum:</th>
            <th>Lokaal:</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Dreamweaver</td>
            <td>15 december</td>
            <td>PC1</td>
        </tr>
    </tbody>
</table>
```

De hiërarchische selectors zijn:

- **Parent en child**
  Parent en child zijn geneste tags op het eerste niveau.
  De thead-tag en de tbody-tag staan rechtstreeks onder de table-tag en gedragen zich in een parent-child relatie.
  table is een parent van thead of
  thead is een child van table.
- **Siblings**
  Siblings (broers of zussen) zijn elementen die op gelijk niveau staan.
  De drie td-tags binnen een tr-tag zijn siblings van elkaar.

- **Ancestor (voorouder)**
  Een ancestor kan de parent zijn, maar ook de parent van de parent, of nog een verdere relatie.
  th heeft als parent tr.
  th heeft als ancestor tr, maar ook thead en table.
- **Descendant (nakomeling)**
  Een descendant kan een child zijn, maar ook een child van een child of nog een verdere relatie.
  table heeft als child thead en tbody.
  table heeft als descendant thead, tbody, maar ook tr, th en td.



Controleer onderstaande selecties op het oefenbestand.
Je mag enkel het deel tussen de aanhalingstekens invullen.
Bijvoorbeeld: table tr en niet $('table tr').

Descendant selector: $('ancestor descendant')

```
$('table tr')     // alle tr-tags binnen een tabel
$('table td')     // alle td-tags binnen een tabel
$('table td a')   // enkel de links in td-tags van een tabel
$('ul *')         // alle tags binnen een ul-tag
```

Child selector: $('parent > child')

```
$('table > tbody')        // de tbody-tags die een child zijn van een
tabel
$('table > td')           // de td-tags die een child zijn van een
tabel (*)
```

```
$('table > tbody > tr')  // alle tabelrijen die een child zijn van
tbody-tags die op hun beurt een child zijn van table
```

(*) Merk op dat in ons voorbeeld geen enkele td-tag rechtstreeks onder de table-tag staat. Er wordt dus niets geselecteerd!

Sibling selector (alle volgende) : $('prev ~ sibling')

```
$('thead ~ tbody')   // alle tbody-tags die op hetzelfde niveau
staan van en volgen op een thead-tag
$('li ~ li')         // alle li-tags die op hetzelfde niveau staan
van en volgen op een li-tag
$('h3 ~ p')          // alle p-tags die op hetzelfde niveau staan
van en volgen op een h3-tag
```

Sibling selector (dadelijk aangrenzende): $('prev + next')

```
$('thead + tbody')   // alle tbody-tags die op hetzelfde niveau
staan van en DADELIJK volgen op een thead-tag.
$('li + li')         // alle li-tags die op hetzelfde niveau staan
van en DADELIJK volgen op een li-tag
$('h3 + p')          // alle p-tags die op hetzelfde niveau staan
van en DADELIJK volgen op een h3-tag
```

In het laatste voorbeeld worden paragraaf2 en paragraaf3 niet geselecteerd.

Basisfilters

Zie : http://api.jquery.com/category/selectors/basic-filter-selectors/

De selectors die we tot hiertoe hebben besproken, selecteren telkens een volledig element.

`$('tr')` selecteert bijvoorbeeld alle tr-tags in een tabel.

Stel dat we nu enkel de even of oneven rijen in een tabel wensen te selecteren, dan doen we dit aan de hand van filters. Een filter is als het ware een subselectie binnen een hoofdselectie. Alle filters worden met een dubbelpunt als prefix aangeduid. Voor de oneven rijen (rijen met index 1, 3, ...) in een tabel wordt dit: `$('tr:odd')`.

Bij sommige filters moet je eveneens een index meegeven. Het eerste element heeft de index 0. Om bijvoorbeeld de derde rij binnen een tabel te selecteren, geeft u als index de waarde 2 mee `$('tr:eq(2)')`.

Subselecties binnen een reeks

```
$('table tr:even')   // alle even tr-tags binnen een tabel
$('table tr:odd')    // alle oneven tr-tags binnen een tabel
$('img:first')       // de eerste afbeelding op de pagina
$('img:last')        // de laatste afbeelding op de pagina
```

```
$('img:eq(2)')       // de DERDE (niet tweede) afbeelding op de
pagina
$('img:lt(2)')       // alle afbeeldingen voor de derde afbeelding
$('img:gt(2)')       // alle afbeeldingen na de derde afbeelding
```

Inverse subselecties binnen een reeks met :not()

```
$('img:not(:eq(2))')  // alle afbeeldingen, behalve de derde
afbeelding
```

Overige subselecties

(De drie laatste selectors kan je niet op het oefenbestand testen.)

```
$('*:header')          // alle hx-tags (h1 t/m h6)
$(':header')           // alle hx-tags (h1 t/m h6)
$('*:focus')           // het element dat op dit moment geselecteerd
is
$(':focus')            // het element dat op dit moment geselecteerd
is
$('div:animated')      // alle div-tags die op dit moment bewegen
```

Inhoudsfilters

Zie : http://api.jquery.com/category/selectors/content-filter-selector/

Deze filters testen op de inhoud (tekst of html) binnen een bepaald element.

Het element bevat een bepaalde tekst :contains()

```
$('td:contains("PC")')            // alle cellen die de tekenreeks PC
bevatten
$('td:contains(PC)')
$('td:not(:contains("PC"))')      // alle cellen die de tekenreeks PC
niet bevatten
$('td:not(:contains(PC))')
```

De tekst is hoofdlettergevoelig, maar de aanhalingstekens zijn niet verplicht.

`contains("PC")` is **NIET** hetzelfde als `contains("pc")`.

`contains("PC")` is **WEL** hetzelfde als `contains(PC)`.

Het element bevat een bepaald element :has()

```
$('td:has(a)')                    // alle cellen die een link
bevatten
$('td:not(:has(a))')              // alle cellen die geen link
bevatten
```

Lege (:empty) of niet lege (:parent) elementen

```
$('td:empty')                     // alle lege cellen binnen een
tabel
$('td:not(:empty)')               // alle cellen die niet leeg zijn
$('td:parent')                    // alle cellen die minstens één
child hebben
```

Merk op dat de 2 laatste selectors hetzelfde resultaat geven. De :parent-filter selecteert alle elementen die minstens één child hebben (in de vorm van een element of tekst!).

Attribuut selectors

Vanuit CSS kan u elementen selecteren op basis van hun attribuut. Ook deze eigenschap werd door jQuery overgenomen. Bekijk de broncode van het oefenbestand. Neem bijvoorbeeld een afbeelding. Alle afbeeldingen hebben natuurlijk een src-attribuut en een alt-attribuut. Sommige afbeeldingen hebben ook het title-attribuut.

Zie : http://api.jquery.com/category/selectors/attribute-selectors/

Attribuut bestaat [name]

```
$('img[src]')              // alle afbeeldingen met het src-attribuut
$('img[title]')            // alle afbeeldingen met het title-
attribuut
$('a[target]')             // alle links met het target-attribuut
$('a:not([target])')       // alle links zonder target-attribuut
```

We kunnen een selectie nog verder verfijnen door de waarde of inhoud van het attribuut te controleren.

Attribuut is gelijk aan [name="value"]

$('a[target="_blank"]') // alle links met target="_blank" of zonder opgegeven target

Attribuut is niet gelijk aan [name!="value"]

```
$('a[target!="_blank"]')      // alle links met een target
(verschillend van "_blank")
```

Attribuut begint met [name^="value"]

```
$('a[href^="http://"]')       // alle externe html-links
```

Attribuut eindigt op [name$="value"]

```
$('a[href$=".com"]')          // alle links die eindigen op .com
```

Attribuut bevat[name*="value"]

```
$('a[href*="wiki"]')          // alle links die de tekenreeks "wiki"
bevatten
```

Meervoudige attribuutfilters [filter1][filter2]

```
$('a[href^="http://"][title]')   // alle externe links met een
title-attribuut
```

Childfilters

Een childfilter selecteert het n-de element binnen het parent element. Zie : http://api.jquery.com/category/selectors/child-filter-selectors/

Eerste child element :first-child

```
$('p b:first-child')   // de eerste b-tag in ELKE paragraaf
$('h3:first-child')    // alle h3-tags die een eerste child zijn van
een (willekeurige) parent
```

Laatste child element :last-child

```
$('p b:last-child')   // de laatste b-tag in ELKE paragraaf
```

```
$('p:last-child')      // alle paragrafen die een laatste child zijn
van een (willekeurige) parent
```

Eerste sibling element binnen dezelfde parent :first-of-type

```
$('a:first-of-type')          // de eerste a-tag binnen dezelfde
parent
```

Laatste sibling element binnen dezelfde parent :last-of-type

```
$('a:last-of-type')           //de laatste a-tag binnen dezelfde
parent
```

n-de child elementen

De index start hier vanaf 1 en niet vanaf 0!
De letter n staat voor een geheel getal (0, 1, 2, 3, ...).

```
$('p b:nth-child(2)')            // de tweede b-tag in elke paragraaf
$('p b:nth-child(3n)')           // b-tags 3, 6, 9, ... in elke
paragraaf
$('p b:nth-child(even)')         // alle even b-tags in elke
paragraaf
$('p b:nth-child(2n)')           // alle even b-tags in elke
paragraaf
$('p b:nth-child(odd)')          // alle oneven b-tags in elke
paragraaf
$('p b:nth-child(2n+1)')         // alle oneven b-tags in elke
paragraaf
$('p b:nth-last-child(2)')       // de voorlaatste b-tag in elke
paragraaf
$('li:nth-of-type(even)')        // alle even li-tags binnen dezelfde
parent
$('a:nth-last-of-type(2)')       // de voorlaatste a-tag binnen
dezelfde parent
```

Het enige child element :only-child()

```
$('p b:only-child()')    // de b-tag, maar ENKEL indien dit HET ENIGE
child element binnen de p-tag is
```

Het enige child element van dat type binnen dezelfde parent :only-of-type()

```
$('b:only-of-type()')    // de b-tag, maar ENKEL indien dit DE ENIGE
b-tag is binnen dezelfde parent
```

Formulierfilters

Zie : http://api.jquery.com/category/selectors/form-selectors/

Filtert formulierelementen van een bepaald type of met een bepaald attribuut.

De type-selector (button, submit, reset, checkbox, ...)

```
$('input')                    // ALLE input-tags.
$('input[type="button"]')     // alle input-tags met type="button"
$('input[type="submit"]')     // alle input-tags met type="submit"
$('input[type="reset"]')      // alle input-tags met type="reset"
```

Overige attributen (:checked, :disabled, :enabled en :selected)

```
$(':checked')     // checkboxen en radiobuttons met een checked-
attribuut
                  // ':checked' werkt NIET op een keuzelijst!
$(':disabled')    // alle elementen met een disabled-attribuut
$(':enabled')     // alle elementen die niet disabled staan
$(':selected')    // het gekozen item uit een keuzelijst
                  // ':selected' werkt NIET op checkboxen en
radiobuttons!
```

Met :checked en :selected selecteert u enkel het element, maar haalt u de gekozen waarde nog niet op. De methodes om deze waardes uit te lezen, komen later aan bod.

> Met `:checked` en `:selected` selecteer je enkel het element, maar haal je de gekozen waarde nog niet op. De methodes om deze waardes uit te lezen, komen later aan bod.

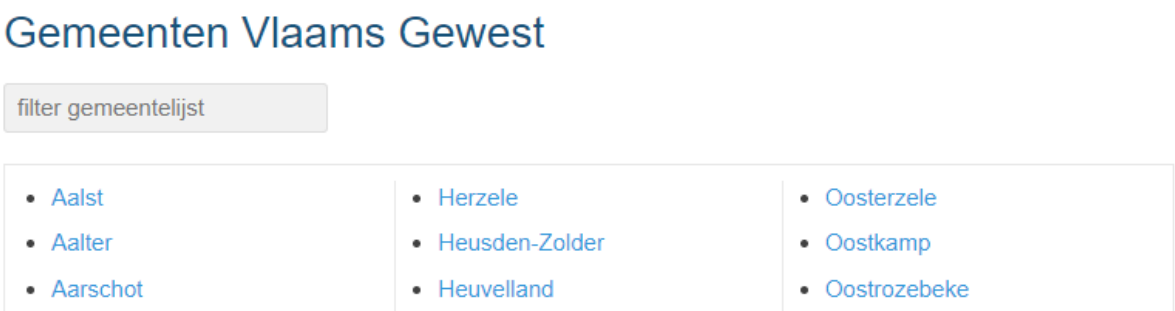
Zichtbaarheidsfilters

Zie : http://api.jquery.com/category/selectors/visibility-filter-selectors/

```
$('p:visible')        // alle zichtbare p-tags
$('p:hidden')         // alle onzichtbare p-tags
```

Toepassing 1: gemeentelijst filteren (basisversie)

> Open **selectors/toep_gemeentelijst_1.html**



Gemeentelijst filteren

De pagina bevat een lijst van alle Vlaamse gemeenten.
Het zoekveld bovenaan de pagina filtert de lijst met gemeentenamen.

```
<body>
...
<input type="text" id="filter" ... ">
...
<ul>
    <li><a ... >Aalst</a></li>
    <li><a ... >Aalter</a></li>
    <li><a ... >Aarschot</a></li>
```

```
    <li><a ... >Aartselaar</a></li>
</ul>
...
</body>
```

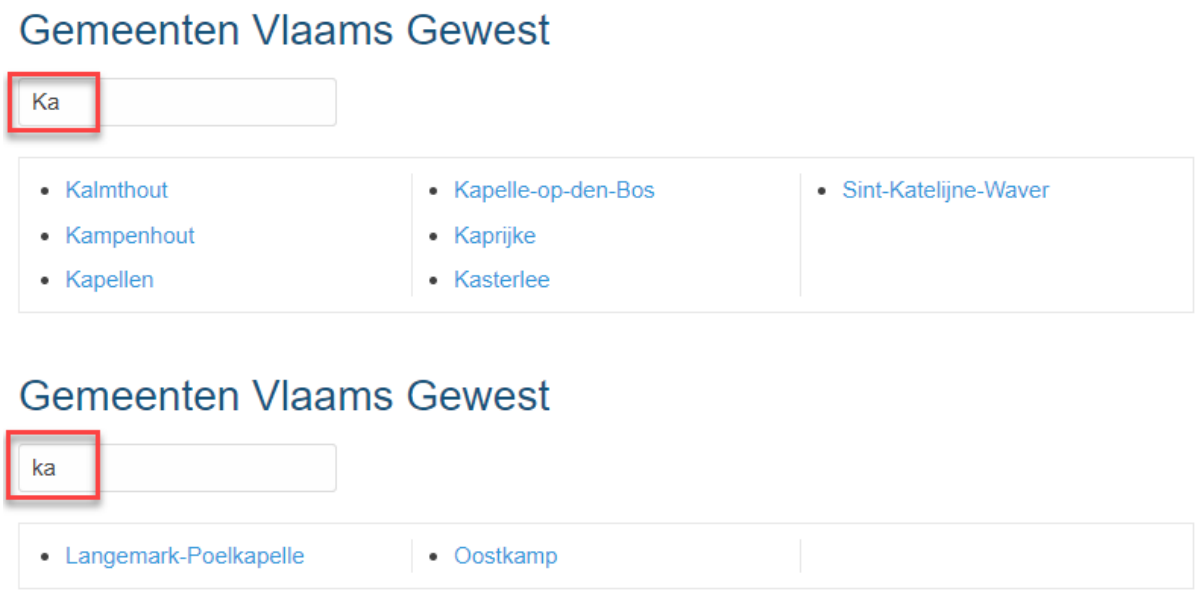
Pas het script als volgt aan.

```
$(function() {
    $('#filter').keyup(function(){
        var filter = $(this).val();
        $('main li').hide();
        $('main li:contains(' + filter + ')').show();
    });
});
```

Telkens u in het tekstveld een letter intypt, wordt het event `keyup()` geactiveerd. De waarde uit het tekstveld komt in de variabele filter terecht.
Indien je bijvoorbeeld ka invult, wordt getest of dit woord binnen een li-tag voorkomt. De selectoren op lijn 4 en 5 bepalen of de li-tag zichtbaar of onzichtbaar wordt.

Merk op dat op lijn 5 de variabele dynamisch in de selector wordt verwerkt. Voor het zoekwoord ka wordt dit: `$('li:contains(ka)').show();`

Zoeken op Ka of op ka een toont een totaal verschillend resultaat. De zoekfunctie is namelijk hoofdlettergevoelig. Met de selecties uit dit hoofdstuk kunnen we dit nog niet oplossen. In het volgend hoofdstuk gaan we dit probleem wegwerken door de selecties verder te verfijnen.



Filter op Ka (boven) en ka (onder)

Toepassing 2: openingsuren markeren

Open selectors/toep_openingsuren.html.

Openingsuren stadhuis Geel

Geef in elke tabel de rij van vandaag een opvallende achtergrondkleur.
Zie voorbeeld: http://www.openingsuren.com/detail.php?edit=5031593

Van september t.e.m. juni

Dag	Voormiddag	Namiddag	Avond
Zondag	Gesloten	Gesloten	Gesloten
Maandag	9u – 12u	13.30u – 16u	Gesloten
Dinsdag	9u – 12u	13.30u – 16u	17.30u – 19.30u
Woensdag	9u – 12u	13.30u – 16u	Gesloten
Donderdag	9u – 12u	13.30u – 16u	Gesloten
Vrijdag	9u – 12u	Gesloten	Gesloten
Zaterdag	Gesloten	Gesloten	Gesloten

In juli en augustus

Dag	Voormiddag	Namiddag	Avond
Zondag	Gesloten	Gesloten	Gesloten
Maandag	9u – 12u	Gesloten	Gesloten
Dinsdag	9u – 12u	Gesloten	17.30u – 19.30u
Woensdag	9u – 12u	Gesloten	Gesloten
Donderdag	9u – 12u	Gesloten	Gesloten
Vrijdag	9u – 12u	Gesloten	Gesloten
Zaterdag	Gesloten	Gesloten	Gesloten

Openingsuren op dinsdag

De pagina bevat twee tabellen waarop we de openingsuren van vandaag markeren. Elke weekdag komt overeen met één rij binnen de tbody-tag. Merk op dat we de tabel ook starten met zondag. Dit is een bewuste keuze omdat we het rijnummer dan makkelijk kunnen koppelen aan de weekdag van het Date-object.

```
<table class="table">
    <thead>...</thead>
    <tbody>
        <tr>
            <td><b>Zondag</b></td>
            <td>Gesloten</td>
            <td>Gesloten</td>
            <td>Gesloten</td>
        </tr>
        ...
    </tbody>
</table>
```

Voeg volgende code toe.

```
$(function() {
    var vandaag = new Date();
    var dag = vandaag.getDay();
    $('tbody tr:nth-child(' + (dag + 1) + ')').css('background-
color', 'greenYellow');
});
```

Op lijn 3 halen we de weekdag op. (zondag = 0, maandag = 1, ...). Vervolgens gaan we op lijn 4 dit getal met één verhogen om zo de juiste rij binnen tbody te markeren met een groen-gele achtergrond.

> Merk op dat de pagina twee tabellen bevat en dat we daarom `tr:nth-child()` gebruiken en niet `tr:eq()`. `tr:eq()` zou enkel een rij in de eerste tabel markeren.

Selectie verfijnen (traversing)

Een jQuery selector is al zeer krachtig, maar soms volstaat één selector niet en gaan we de selectie verder moeten verfijnen. We beginnen met enkele voorbeelden die we verder in dit hoofdstuk uitgebreid bespreken:

Stel dat we het eerste en het laatste item in een lijst willen kleuren. Dit kan perfect met de selectiemethodes uit voorgaand hoofdstuk. Maar wat als de pagina meerdere lijsten bevat waarvan we telkens het eerste en laatste item willen kleuren?
Dit kan reeds met `$('ul li:first-child, ul li:last-child')` maar in dit hoofdstuk gaan we op zoek naar een beter alternatief.

Een tweede voorbeeld: de pagina bevat een ratingsysteem met sterretjes. Als we één van de sterretjes aanklikken moeten deze ster plus alle voorgaande sterren van kleur veranderen.

Met de methodes die we in dit hoofdstuk bespreken (traversing methodes) is dit perfect mogelijk. In de voorbeelden beperken we ons tot een set van methodes die we in de volgende hoofdstukken nog regelmatig tegenkomen.

Een overzicht van de verschillende traversing methodes vindt u hier:
http://api.jquery.com/category/traversing/

In de meeste voorbeelden van dit hoofdstuk wordt door het click-event de class .highLight toegevoegd aan een bepaalde selectie.

Bijvoorbeeld:

```
// Knop1
$('#first').click(function(){
    $('*').removeClass('highLight');
    $('li').first().addClass('highLight');
});
```

De eerste lijn binnen het click-event wist op elk element de class .highLight. De tweede lijn maakt een nieuwe selectie waarop de class .highLight wordt toegepast.

Om ons volledig te concentreren op de selectiemethodes, tonen we in de voorbeelden enkel de relevante code. Kijk steeds in de broncode voor het volledig script.

```
$('li').first().addClass('highLight');     // Knop1
```

Hiërarchische selectors

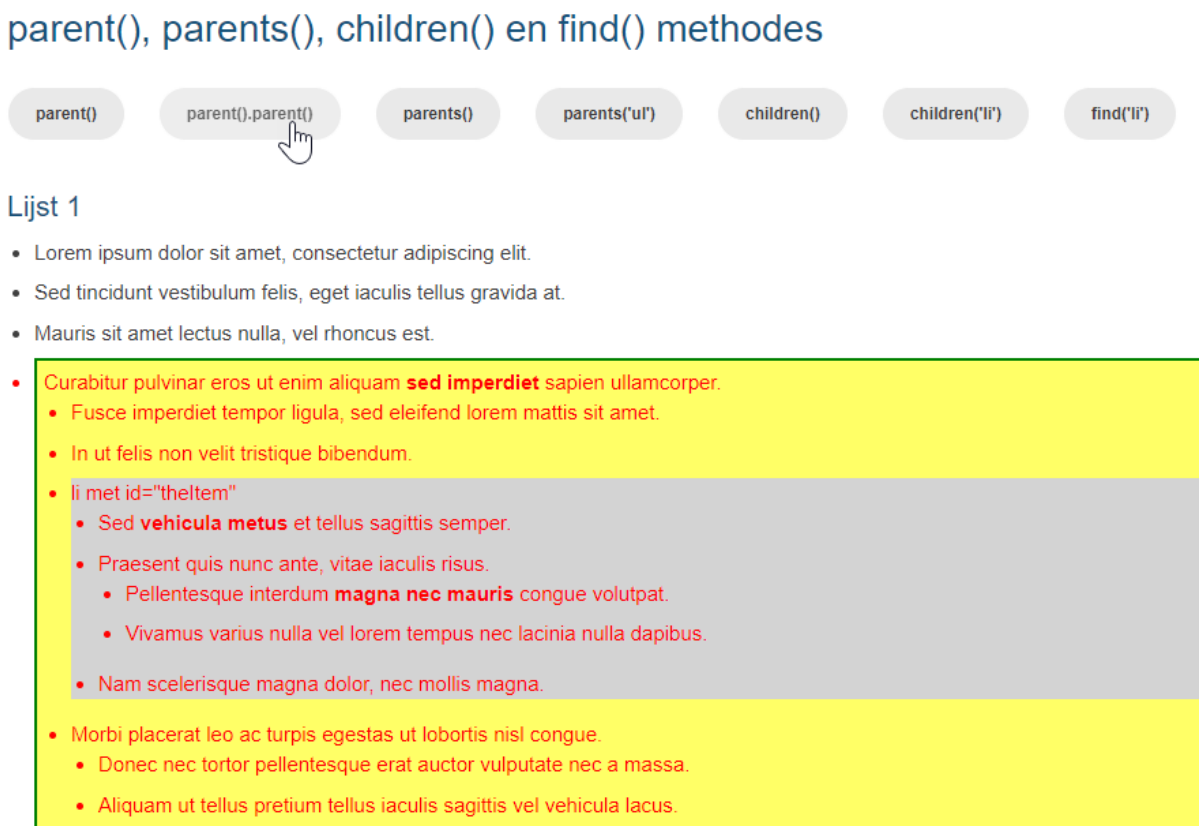
Zie: http://api.jquery.com/category/traversing/tree-traversal/

Hiërarchische subselecties zijn uiterst handig om op een snelle en elegante manier elementen binnen de boomstructuur te benaderen. Deze methodes zal u verder in dit boek nog regelmatig tegenkomen.

Parent(), parents(), children() en find()

Selecteer hoger of lager gelegen elementen in de boomstructuur.

Open **selectors/parent.html.**



Hiërarchische selectors

```
$('#theItem').parent().addClass('highLight');              // knop1
$('#theItem').parent().parent().addClass('highLight');     // knop2
$('#theItem').parents().addClass('highLight');             // knop3
$('#theItem').parents('ul').addClass('highLight');         // knop4
$('#theItem').children().addClass('highLight');            // knop5
$('#theItem').children('li').addClass('highLight');        // knop6
$('#theItem').find('li').addClass('highLight');            // knop7
```

In deze voorbeelden vertrekt de hoofdselectie telkens van de li-tag met id #theItem.

- `$('#theItem').parent()`
  Selecteer het parent element van de li-tag. In dit voorbeeld is dit de ul-tag rond de li-tag met id #theItem.
- `$('#theItem').parent().parent()`
  Ga twee stappen hoger in de boomstructuur. Dit is de li-tag waarbinnen zich de ul-tag uit vorige selectie bevindt.
- `$('#theItem').parents()`
  Selecteer alle ancestor elementen van #theItem.
- `$('#theItem').parents('ul')`
  Selecteer alle ul-tags die een ancestor van #theItem zijn.
- `$('#theItem').children()`

Selecteer alle rechtstreekse children van #theItem. In dit voorbeeld bevat #theItem maar één child element, namelijk een ul-tag. Merk op dat we maar één niveau dalen. De onderliggende li-tags worden niet geselecteerd.

- `$('#theItem').children('li')`
  Hier wordt niets geselecteerd omdat er geen li-tag rechtstreeks onder #theItem staat. Alle onderliggende li-tags zitten immers nog een niveau dieper in een ul-tag. De li-tags zijn geen children, maar wel descendants.
- `$('#theItem').find('li')`
  Selecteer alle li-tags die een descendant van #theItem zijn.

> `children()` zoekt enkel binnen childelementen. find() gaat daarentegen dieper en zoekt zowel naar childelementen als naar descendants.

children() zoekt enkel binnen childelementen. `find()` gaat daarentegen dieper en zoekt zowel naar childelementen als naar descendants.

Next(), nextAll() en nextUntil()

Selecteer elementen op een gelijk niveau (siblings) die volgen na de hoofdselectie.

> Open **selectors/next.html.**

Next(), nextAll() en nextUntil() methodes

next() nextAll() nextUntil() nextUntil('#end')

Lijst 1

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Sed tincidunt vestibulum felis, eget iaculis tellus gravida at.
- Mauris sit amet lectus nulla, vel rhoncus est.
- Curabitur pulvinar eros ut enim aliquam sed imperdiet sapien ullamcorper.
  - Fusce imperdiet tempor ligula, sed eleifend lorem mattis sit amet.
  - --- li met id="start" ---
  - Fusce nec risus vitae massa eleifend rutrum.
  - Vestibulum ac odio nibh, aliquam consequat lectus.
  - Nunc et turpis orci, sit amet imperdiet risus.
  - --- li met id="theItem" ---
  - Sed vehicula metus et tellus sagittis semper.
  - Praesent quis nunc ante, vitae iaculis risus.
    - Pellentesque interdum magna nec mauris congue volutpat.
    - Vivamus varius nulla vel lorem tempus nec lacinia nulla dapibus.
  - Namscelerisque magna dolor, nec mollis magna.
  - Morbi placerat leo ac turpis egestas ut lobortis nisl congue.
  - --- li met id="end" ---
  - Aliquam ut tellus pretium tellus iaculis sagittis vel vehicula lacus.

```
$('#theItem').next().addClass('highLight');              // knop1
$('#theItem').nextAll().addClass('highLight');           // knop2
$('#theItem').nextUntil().addClass('highLight');         // knop3
$('#theItem').nextUntil('#end').addClass('highLight');   // knop4
$('#theItem').next()
```

Selecteer het eerstvolgende element uit de lijst.

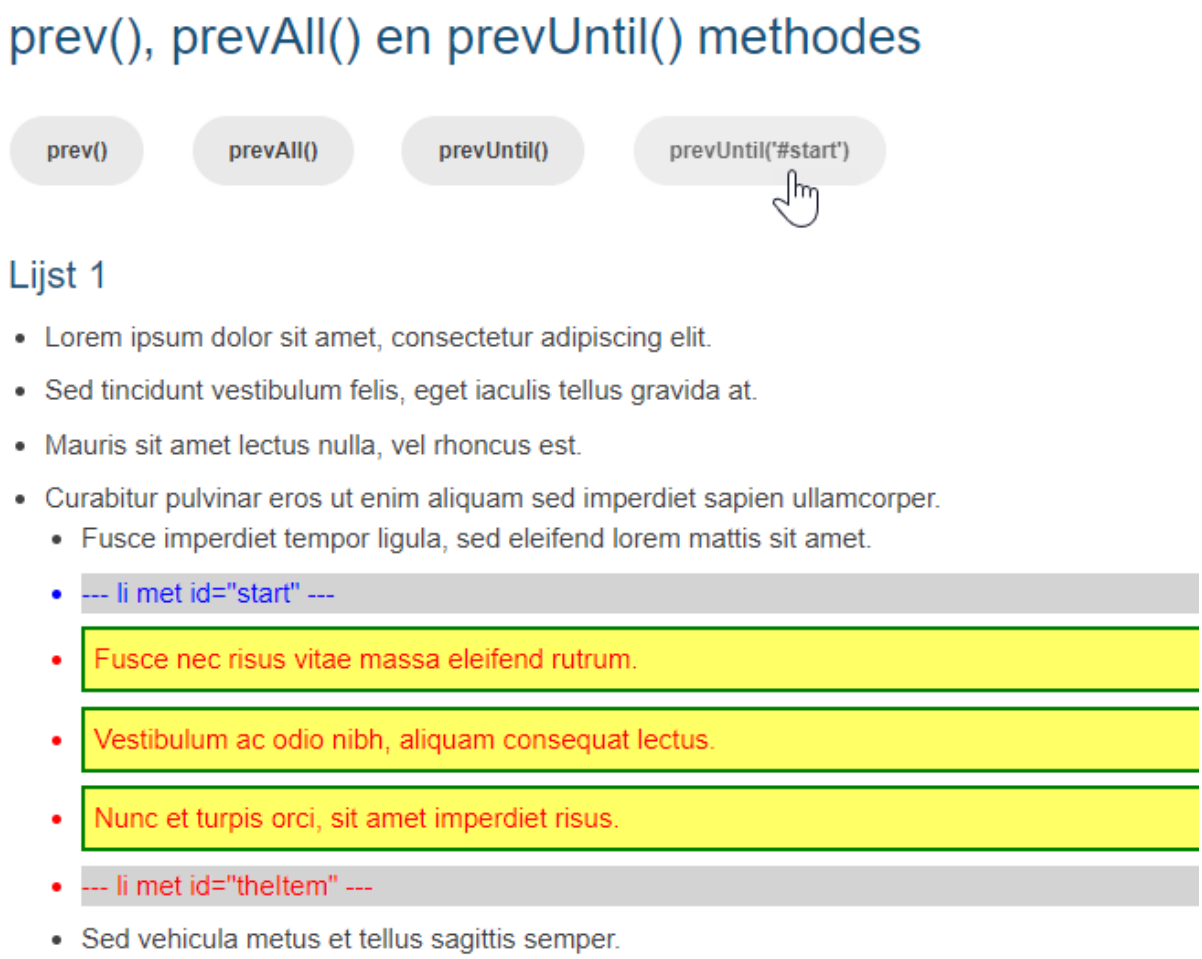
- `$('#theItem').nextAll()`
  Selecteer alle volgende elementen uit de lijst. Let op: enkel elementen op gelijk niveau met #theItem worden geselecteerd.
- `$('#theItem').nextUntil()`
  Zonder parameter geven nextAll() en nextUntil() hetzelfde resultaat.

- `$('#theItem').nextUntil('#end')`
  Selecteer alle volgende elementen uit de lijst, maar stop net voor het element met id #end.

Prev(), prevAll() en prevUntil()

Selecteer elementen op een gelijk niveau, voorafgaand aan de hoofdselectie.

Open **selectors/prev.html.**




```
$('#theItem').prev().addClass('highLight');                    //
knop1
$('#theItem').prevAll().addClass('highLight');                 //
knop2
$('#theItem').prevUntil().addClass('highLight');               //
knop3
$('#theItem').prevUntil('#start').addClass('highLight');       //
knop4
$('#theItem').prev()
```

Selecteer het voorgaande element uit de lijst.

- `$('#theItem').prevAll()`
  Selecteer alle voorgaande elementen uit de lijst.

- `$('#theItem').prevUntil()`
  Zonder parameter geven prevAll() en prevUntil() hetzelfde resultaat.
- `$('#theItem').prevUntil('#start')`
  Selecteer alle voorgaande elementen uit de lijst, maar stop net voor het element met id #start.

Siblings()

Selecteer elementen, zowel voorafgaand als volgend, op een gelijk niveau met de hoofdselectie.

De hoofdselectie zelf is nooit inbegrepen.

> Open **selectors/siblings.html.**

```
$('#theItem').siblings().addClass('highLight');                    //
knop1
$('#theItem').siblings('.grey').addClass('highLight');             //
knop2
$('#theItem').siblings().not('.grey').addClass('highLight');       //
knop3
$('#theItem').siblings()
```

Selecteer alle elementen uit de lijst.

$('#theItem').siblings('.grey')
Selecteer alle elementen met de class .grey uit de lijst.

$('#theItem').siblings().not('.grey')
Selecteer alle elementen zonder de class .grey uit de lijst.

Closest()

Selecteer, vanaf de hoofdselectie naar de body-tag toe, het eerste element dat voldoet aan de voorwaarde. De subselectie gebeurt enkel in parent of ancestors, niet op siblings, children of descendants.

> Open **selectors/closest.html.**

closest() methode

closest('li') closest('ul') closest('.grey')

Lijst 1

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Sed tincidunt vestibulum felis, eget iaculis tellus gravida at.
- Mauris sit amet lectus nulla, vel rhoncus est.
- Curabitur pulvinar eros ut enim aliquam sed imperdiet sapien ullamcorper.
- Fusce imperdiet tempor ligula, sed eleifend lorem mattis sit amet.
- --- li met class="grey" ---
- Fusce nec risus vitae massa eleifend rutrum.
- --- li met class="grey" ---
- Vestibulum ac odio nibh, aliquam consequat lectus.
- --- li met id="theItem" ---
- Sed vehicula metus et tellus sagittis semper.
- Praesent quis nunc ante, vitae iaculis risus.
  - Pellentesque interdum magna nec mauris congue volutpat.
  - Vivamus varius nulla vel lorem tempus nec lacinia nulla dapibus.
- Namscelerisque magna dolor, nec mollis magna.
- Morbi placerat leo ac turpis egestas ut lobortis nisl congue.
- -- -li met class="grey" ---
- Aliquam ut tellus pretium tellus iaculis sagittis vel vehicula lacus.

```
$('#theItem').closest('li').addClass('highLight');        // knop1
$('#theItem').closest('ul').addClass('highLight');        // knop2
$('#theItem').closest('.grey').addClass('highLight');     // knop3
$('#theItem').closest('li')
```

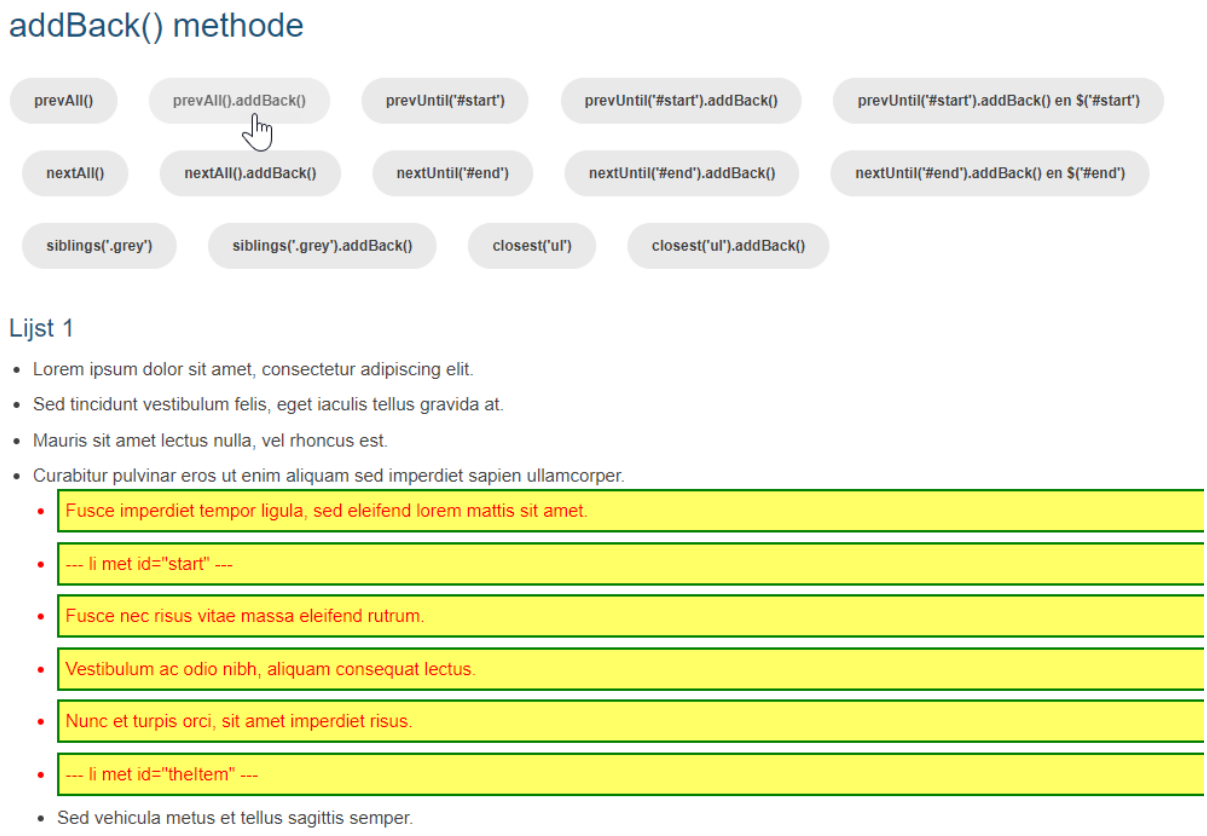
Het id #theItem is zelf een li-tag, dus de subselectie is gelijk aan de hoofdselectie.

- `$('#theItem').closest('ul')`
  Selecteer de eerste ul-tag net boven #theItem.
- `$('#theItem').closest('.grey')`
  Er wordt niets geselecteerd. Er bevindt zich immers geen enkel element met class .grey op een ancestor element van #theItem.

AddBack()

Voor de meeste hiërarchische selectors die we net hebben besproken, geldt dat de hoofdselectie zelf nooit is inbegrepen. Om de hoofdselectie ook toe te voegen, volstaat het addBack() achter de subselectie toe te voegen.

> Open **selectors/addBack.html.**
> Bekijk de broncode van de verschillende voorbeelden.




```
// zonder hoofdselectie
$('#theItem').prevAll().addClass('highLight');
// met hoofdselectie
$('#theItem').prevAll().addBack().addClass('highLight');
```

Filter methodes

Zie : http://api.jquery.com/category/traversing/filtering/

Enkele selectiefilters uit voorgaand hoofdstuk zijn eveneens beschikbaar als een methode. Dit is onder andere handig om verschillende lijnen code te groeperen tot één instructie.

Een ketting ziet er als volgt uit: Het eerste deel is zoals altijd de hoofdselectie. Daarna volgt de subselectie binnen de hoofdselectie. Alle volgende delen hebben betrekking op de subselectie. Een voorbeeld:

```
$('li').first().addClass('highLight');
```

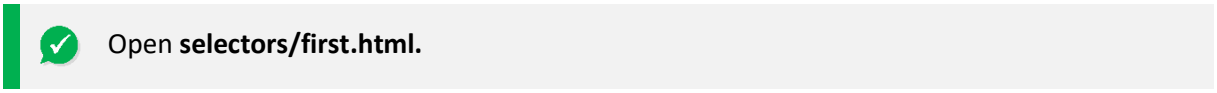
- `$('li')`
  Selecteer alle li-tags (hoofdselectie).

- `.first()`
  Selecteer van deze li-tags het eerste element (subselectie).

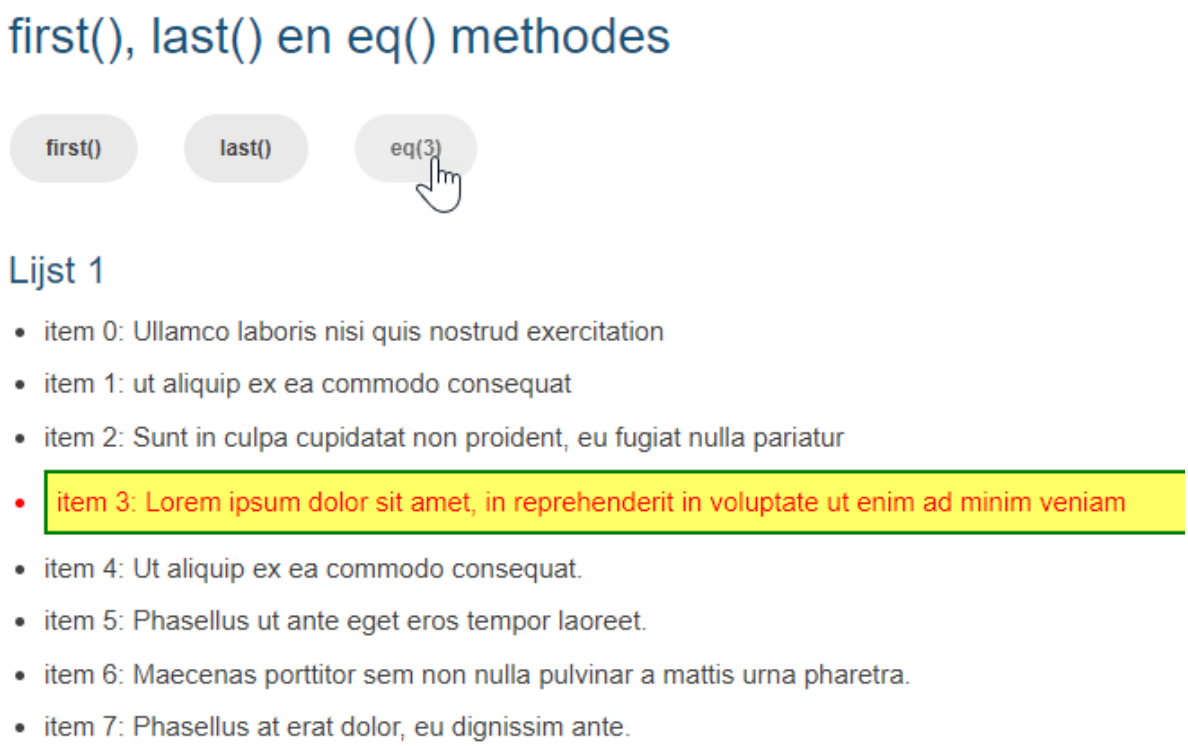
- `.addClass('highLight')`
  Ken aan de eerste li-tag op de pagina de class .highLight toe.

First(), last() en eq()

Deze methodes filteren respectievelijk het eerste, het laatste of het n-de element uit de hoofdselectie.



Open **selectors/first.html.**




```
$('main li').first().addClass('highLight');    // knop1
$('main li').last().addClass('highLight');     // knop2
$('main li').eq(3).addClass('highLight');      // knop3
```

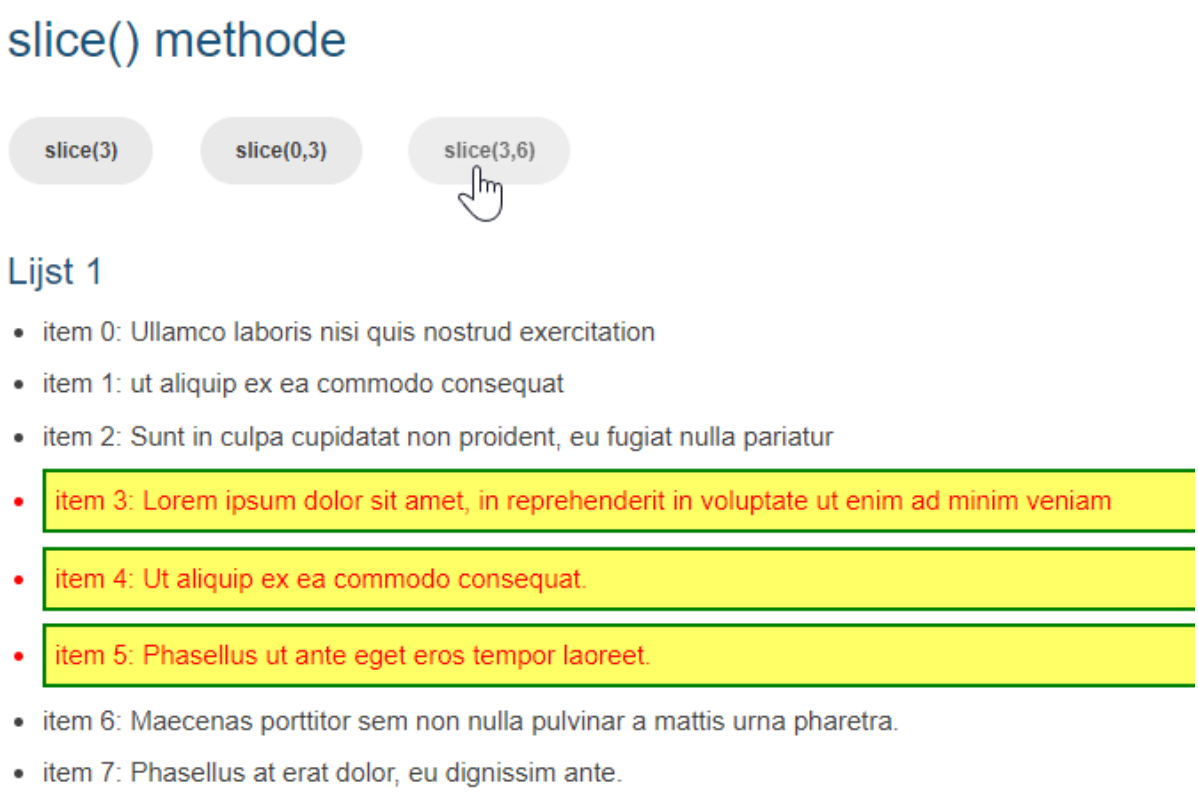
De pagina bevat één lijst. De verschillende knoppen selecteren respectievelijk het de eerste, de laatste en de vierde li-tag binnen de main-tag. (De index start vanaf 0!)

Merk op dat deze subselecties, net zoals de meeste filtermethodes in dit hoofdstuk, enkel kijken naar het aantal li-tags op de ganse pagina. Indien de pagina meerdere lijsten bevat, gebeurt de selectie niet individueel per lijst, maar over alle lijsten heen. Wilt u van elke lijst op de pagina het eerste element selecteren, dan moet u de selectie in een soort van lus onderbrengen. Hierover dadelijk meer.

Slice()

Selecteer een subset of een bepaalde range van elementen.

> Open **selectors/slice.html.**




```
$('main li').slice(3).addClass('highLight');       // knop1
$('main li').slice(0,3).addClass('highLight');     // knop2
$('main li').slice(3,6).addClass('highLight');     // knop3
```

De methode slice() heeft één of twee parameters.

- `slice(n)`
  Selecteer alle elementen beginnend vanaf element n.

- `slice(n,m)`
  Selecteer alle elementen vanaf element n tot net voor element m. Dus element m niet inbegrepen.

Filter()

Beperk de hoofdselectie tot de set die overeenstemt met de filtervoorwaarde. De filtervoorwaarde kan bestaan uit een selector, een element of zelfs uit een volledige functie.

> Open **selectors/filter.html.**

filter() methode

.filter(':first') .filter(':last') .filter(':eq(3)') .filter('#theItem') .filter(function(){...})

Lijst 1

- item 0: Ullamco laboris nisi quis nostrud exercitation
- item 1: ut aliquip ex ea commodo consequat
- item 2: Sunt in culpa cupidatat non proident, eu fugiat nulla pariatur
- item 3: Lorem ipsum dolor sit amet, in reprehenderit in voluptate ut enim ad minim veniam
- item 4: Ut aliquip ex ea commodo consequat.
- item 5: Phasellus ut ante eget eros tempor laoreet.
- item 6: Maecenas porttitor sem non nulla pulvinar a mattis urna pharetra.
- item 7: Phasellus at erat dolor, eu dignissim ante.

```
$('main li').filter(':first').addClass('highLight');      // knop1
$('main li').filter(':last').addClass('highLight');       // knop2
$('main li').filter(':eq(3)').addClass('highLight');      // knop3
$('main li').filter('#theItem').addClass('highLight');    // knop4
$('main li').filter(function(index){                      // knop5
    if ((index % 3) == 0) {
        return true;
    }
}).addClass('highLight');
```

Bij knop1 tot en met knop4 bevat de filter telkens een selector die we kennen uit voorgaand hoofdstuk. Knop5 bevat een functie waarbij we 0 plus elk veelvoud van 3 filteren. De parameter index refereert zoals steeds naar het actieve element.

Not()

De methode not() selecteert het inverse van filter().

Open **selectors/not.html.**

not() methode

.not(':first') .not(':last') .not(':eq(3)') .not('#theItem') .not(function(){...})

Lijst 1

- item 0: Ullamco laboris nisi quis nostrud exercitation
- item 1: ut aliquip ex ea commodo consequat
- item 2: Sunt in culpa cupidatat non proident, eu fugiat nulla pariatur
- item 3: Lorem ipsum dolor sit amet, in reprehenderit in voluptate ut enim ad minim veniam
- item 4: Ut aliquip ex ea commodo consequat.
- item 5: Phasellus ut ante eget eros tempor laoreet.
- item 6: Maecenas porttitor sem non nulla pulvinar a mattis urna pharetra.
- item 7: Phasellus at erat dolor, eu dignissim ante.

```
$('main li').not(':first').addClass('highLight');      // knop1
$('main li').not(':last').addClass('highLight');       // knop2
$('main li').not(':eq(3)').addClass('highLight');      // knop3
$('main li').not('#theItem').addClass('highLight');    // knop4
$('main li').not(function(index){                      // knop5
    if ((index % 3) == 0) {
        return true;
    }
}).addClass('highLight');
```

Has()

Filtert alle elementen die een bepaald childelement bevatten.

Open **selectors/has.html.**

has() methode

has('a') has('b') hast('a[target="_blank"])

Lijst 1

- item 0: Ullamco laboris nisi quis nostrud exercitation
- item 1: ut aliquip ex ea commodo consequat
- item 2: Sunt in culpa cupidatat non proident, eu fugiat nulla pariatur
- item 3: Lorem ipsum dolor sit amet, in reprehenderit **in voluptate ut enim ad** minim veniam
- item 4: Ut aliquip ex ea commodo consequat.
- item 5: Phasellus **ut ante eget eros tempor** laoreet.
- item 6: Maecenas porttitor sem non nulla pulvinar a mattis urna pharetra.
- item 7: Phasellus at erat dolor, eu dignissim ante.

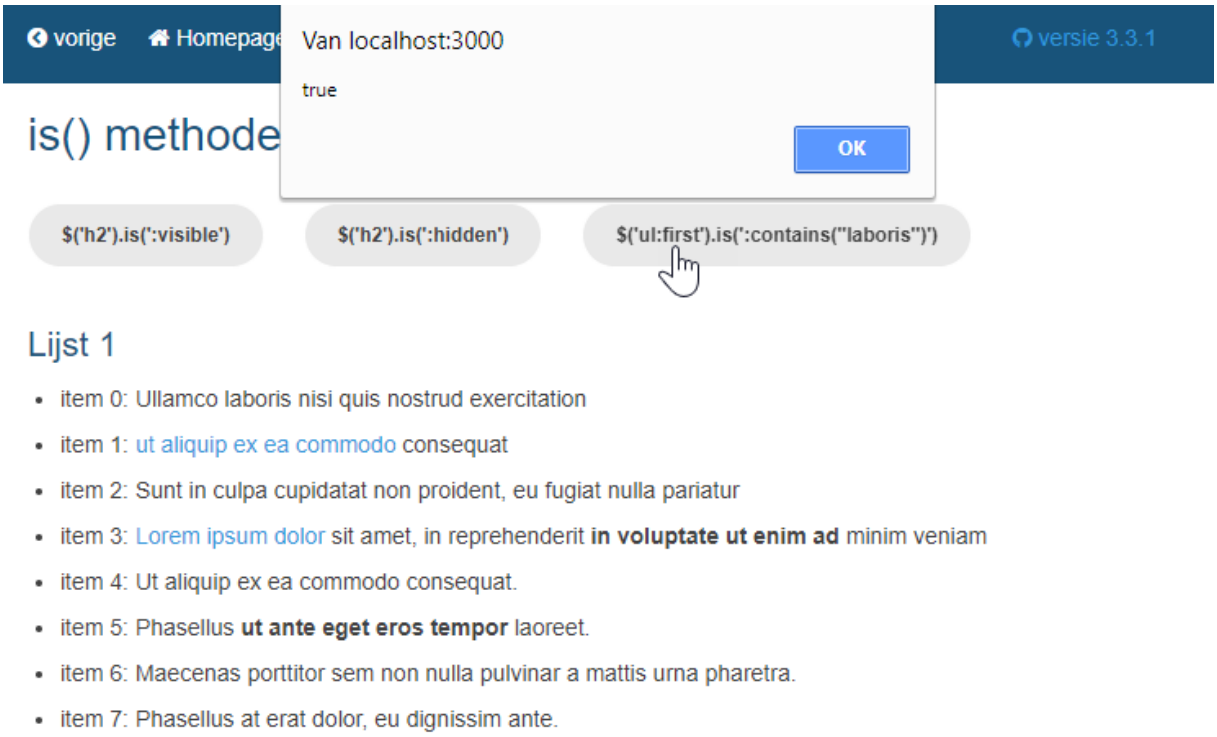
```
$('main li').has('a').addClass('highLight');                    //
knop1
$('main li').has('b').addClass('highLight');                    //
knop2
$('main li').has('a[target="_blank"]').addClass('highLight');   //
knop3
```

De verschillende knoppen selecteren respectievelijk alle li-tags die een a-tag bevatten, alle li-tags die een strong-tag bevatten en alle li-tags die een link met het attribuut `target="_blank"` bevatten.

Is()

Controleert of de hoofdselectie aan een bepaalde voorwaarde voldoet. De methode retourneert true of false.

Open **selectors/is.html.**




```
alert($('h2').is(':visible'));                          // knop1
alert($('h2').is(':hidden'));                           // knop2
alert($('main ul:first').is(':contains("laboris")'));   // knop3
```

- `$('h2').is(':visible')`
  Controleer of minstens één h2-tag zichtbaar is. In dit voorbeeld is het resultaat `true`.
- `$('h2').is(':hidden')`
  Controleer of minstens één h2-tag onzichtbaar is. In dit voorbeeld is het resultaat `false`.
- `$('ul:first').is(':contains("laboris")')`
  Controleer of de eerste ul-tag de tekenreeks laboris bevat. In dit voorbeeld is het resultaat `true`.

Element looping: each()

Zie: http://api.jquery.com/each/

> Open **selectors/each.html.**

each() methode

highLight (first en last) highLight (each)

Lijst 1

- Ullamco laboris nisi quis nostrud exercitation
- ut aliquip ex ea commodo consequat
- Sunt in culpa cupidatat non proident, eu fugiat nulla pariatur
- Lorem ipsum dolor sit amet, in reprehenderit in voluptate ut enim ad minim veniam
- Ut aliquip ex ea commodo consequat.

Lijst 2

- Ullamco laboris nisi quis nostrud exercitation
- ut aliquip ex ea commodo consequat
- Sunt in culpa cupidatat non proident, eu fugiat nulla pariatur
- Lorem ipsum dolor sit amet, in reprehenderit in voluptate ut enim ad minim veniam
- Ut aliquip ex ea commodo consequat.

Loop over elementen

```
// knop1
$('main li').first().addClass('highLight');
$('main li').last().addClass('highLight');
// knop2 (met lus)
$('main ul').each(function() {
    $(this).find('li').first().addClass('highLight');
    $(this).find('li').last().addClass('highLight');
});
// alternatief voor knop 2 (zonder lus)
// $('main ul li:first-child').addClass('highLight');
// $('main ul li:last-child').addClass('highLight');
```

De pagina bestaat nu uit drie lijsten. De link op knop1 bevat enkele methodes uit het eerste voorbeeld van dit hoofdstuk. De kleurwijziging geldt niet voor elke lijst, maar loopt deze over alle lijsten heen. Het eerste item van lijst 1 en het laatste item van lijst 3 krijgen de class .highLight.

Om elke lijst apart te kleuren, gaan we de hoofdselectie verplaatsen naar de ul-tags. Met de methode `each()` doorlopen we alle geselecteerde lijsten. Binnen `each()` verwijst het kernwoord `this` naar de actieve ul-tag. De methodes, volgend op `$(this)` zijn identiek aan de methodes achter knop1.

Selecties in cascade: end()

Zie: http://api.jquery.com/end/

Subselecties zijn ook handig om meerdere selecties op eenzelfde hoofdselectie te combineren tot één instructie. In onderstaand voorbeeld gaan we het eerste, het laatste en het vierde element in een lijst kleuren. Dit kennen we reeds uit het eerste voorbeeld van dit hoofdstuk.

```
$('main li').first().addClass('highLight');
$('main li').last().addClass('highLight');
$('main li').eq(3).addClass('highLight');
```

Hoe kunnen we de drie instructies nu herleiden tot één instructie?

Selecties in cascade

```
$('main li').first().addClass('highLight').end()      // knop1
.last().addClass('highLight').end()
.eq(3).addClass('highLight');
```

De hoofdselectie blijft de li-tag. Daarna volgt een eerste subselectie `first()`. Na het toekennen van de class komt de methode `end()`. Met deze methode keert u terug naar de hoofdselectie. Vanaf de hoofdselectie kan u dan een nieuwe subselectie voor het laatste element maken. Na het inkleuren opnieuw eindigen met `end()` en vervolgens het vierde element selecteren.

Heeft deze manier van werken voordelen? De code wordt wat compacter, maar het resultaat is uiteindelijk hetzelfde. In de praktijk kies je de werkwijze die je persoonlijk het beste ligt.

Snelheidstest

Het is je wellicht opgevallen dat we een aantal selecties op verschillende manieren kunnen uitschrijven. Bijvoorbeeld `$('li:first')` selecteert net hetzelfde als `$('li').first()` of `$('li').filter(':first')`. Het uiteindelijke resultaat is wel hetzelfde, maar hoe de selectie achter de schermen tot stand komt, is totaal verschillend.

> Open **selectors/speedTest.html** in een browser.

Snelheidstest

De snelheid waarmee de instructies worden uitgevoerd, is afhankelijk van de browser en van de computersnelheid. Om de snelheid beter met elkaar te kunnen vergelijken, gebeurt de meting door 10.000 identieke instructies na elkaar uit te voeren.

Uit de verschillende testen kunnen we afleiden dat `$('li').first()` sneller werkt dan `$('li:first')`. In de praktijk zal u waarschijnlijk nooit 10.000 instructies uitvoeren en kan u gerust één van de drie besproken selecties gebruiken.

Toepassing 1: gemeentelijst filteren (uitgebreide versie)

> Open **selectors/toep_gemeentelijst_2.html.**

De code in deze pagina is identiek aan de code uit voorgaand hoofdstuk. Om het probleem met hoofd- en kleine letters op te lossen, gaan we als volgt te werk: zet zowel de zoekterm als de gemeentenamen eerst om naar kleine letters en maak pas daarna de vergelijking.

Voeg bovenaan het script volgende code toe:

```
$(function() {
    // voeg data-g aan elke li-tag toe
    $('main li').each(function(){
        $(this).attr('data-g', $(this).text().toLowerCase());
    });
    ...
});
```

Eerst doorlopen we elke li-tag en wordt via `$(this).text().toLowerCase()` de gemeentenaam omgezet naar kleine letters. In plaats van de naam zelf te wijzigen, gaan we de nieuwe naam als attribuut toevoegen aan de li-tag. De methode `attr()` komt later nog uitgebreid aan bod, maar de syntax is eenvoudig `attr(attribuut, waarde)`. Voor de li-tag rond de gemeente Geel is dit het resultaat:

```
<li data-g="geel"><a ... >Geel</a></li>
```

Pas het reeds bestaande script op lijn 3 en lijn 5 aan zoals hieronder aangegeven.
Test het resultaat in een browser.

```
...
$('#filter').keyup(function(){
    var filter = $(this).val().toLowerCase();
    $('main li').hide();
    $('main li[data-g*="' + filter + '"]').show();
    teller();
});
...
```

De variabele filter bevat vanaf nu (lijn 3) de gemeentenaam in kleine letters. De oude methode `contains()` gaan we nu niet meer gebruiken. De aangepaste selector (lijn 5) controleert of het attribuut data-g de waarde van filter bevat.

Naast het zoekveld komt het aantal zichtbare gemeentenamen. Hiervoor gaan we een nieuwe functie `teller()` aanmaken. Bij elke filtering wordt deze functie opgeroepen.

Voeg de tellerfunctie toe:

```
// tellerfunctie
function teller() {
    var teller = $('main li:visible').length;
    $('#zichtbaar').text(teller);
}
// start de teller bij het openen van de pagina
teller();
```

De tellerfunctie zoekt alle zichtbare li-tags. Het aantal kan u opvragen met de (JavaScript) eigenschap `length`. Deze waarde wordt als tekst in de span-tag #zichtbaar getoond.

Door achter de tellerfunctie dadelijk `teller()` op te roepen, wordt bij het openen van de pagina de tellerfunctie dadelijk uitgevoerd.

> Test de pagina in een browser.
> Maak het tekstveld met de backtoets leeg.
> Pas het script aan.
> Test het resultaat in een browser.

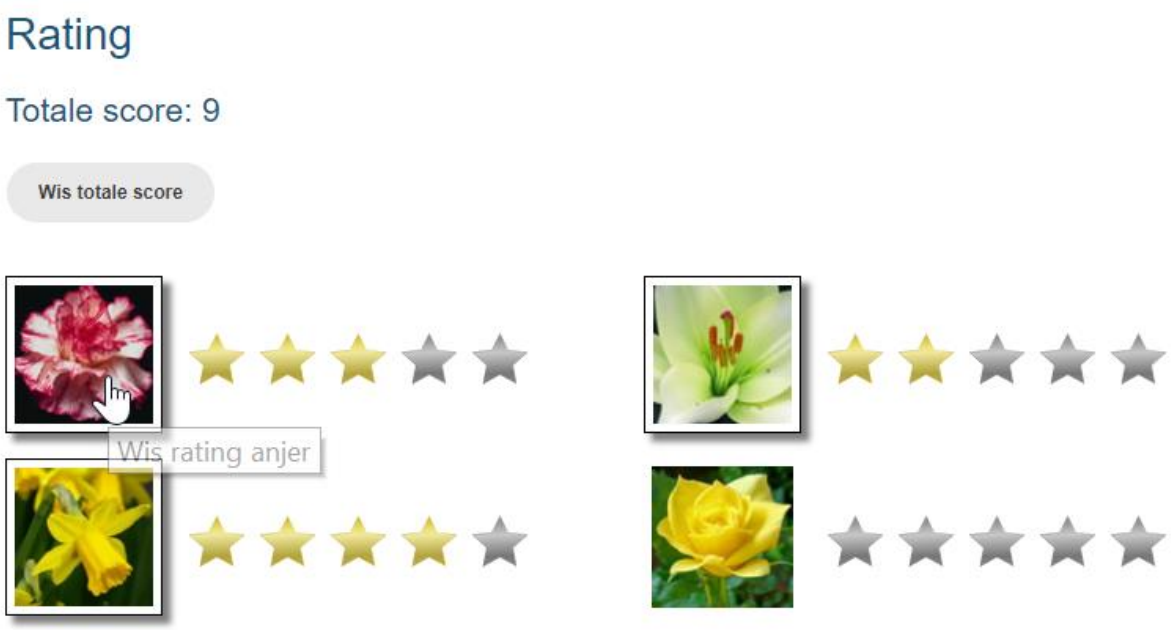
Bij een leeg tekstveld is de variabele filter leeg. Aangezien geen enkele li-tag deze lege filter(-variabele) bevat in zijn attribuut data-g, zullen alle gemeentenamen - verborgen door `$('li').hide()` – onzichtbaar blijven. Dit kunnen we makkelijk oplossen door eerst de inhoud van filter te controleren. Indien filter geen waarde bevat, tonen we altijd de volledige lijst.

```
$('#filter').keyup(function(){
    var filter = $(this).val().toLowerCase();
    if(filter != '') {       // filter bevat een waarde
        $('main li').hide();
        $('main li[data-g*="' + filter + '"]').show();
    } else {                 // filter is leeg
        $('main li').show();
    }
    teller();
});
```

Filter op KA

Toepassing 2: rating

> Open selectors/toep_rating.html.



Rating

De ratingpagina bevat vier afbeeldingen. Klik straks op één van de sterren om een waardeoordeel tussen 1 en 5 uit te brengen. Bovenaan de pagina verschijnt de totale score van alle afbeeldingen. Verder is er een knop voorzien waarmee u de totale score kan wissen. Individuele scores wissen kan door op de afbeelding te klikken.

Verder is het ook belangrijk dat de pagina eenvoudig uit te breiden is. Indien we een vijfde ratingblok toevoegen, blijft de pagina werken zonder het script aan te passen.

Laat ons eerst de belangrijkste delen uit de pagina bespreken. De HTML-code is als volgt opgebouwd:

```
...
<h2>Totale score: <span id="teller">0</span></h2>
...
<section class="columns_2">
    <section>
        <div class="bloem">
            <img src="../assets/fotos/thumb_anjer.jpg"...>
        </div>
        <div class="rating">

<span></span><span></span><span></span><span></span><span></span>
        </div>
    </section>
...
</section>
```

De totale score komt in een span-tag #teller.
Elk ratingblok bestaat uit een section-tag met daarin twee div-tags.

De eerste div-tag met class .bloem bevat enkel een afbeelding. De tweede div-tag met class .rating bevat vijf lege span-tags. Vanuit CSS krijgt elke span-tag een grijze ster als achtergrondafbeelding.

In de CSS-code is vooral de weergave van de sterren belangrijk:

```
<style type="text/css">
.rating span {
    width: 45px;
    height: 37px;
    display: block;
    float: left;
    cursor: pointer;
}
.rate_no {
    background: url(../assets/star_grey.png) no-repeat;
}
.rate_yes {
    background: url(../assets/star_gold.png) no-repeat;
}
</style>
```

De stijlregel .rating span bevat ondermeer de breedte en de hoogte van de span-tags binnen het ratingblok. Daarna volgen twee classes met als achtergrondafbeelding respectievelijk een grijze ster en een gouden ster. Deze classes gaan we dadelijk vanuit ons script aan de span-tags linken.

In het basisscript:

```
$(function() {
    $('.rating span').addClass('rate_no');
    $('.rating span').click(function(){
        // Klik op een ster
    });
    $('#wisRating').click(function(){
        // Wis de totale score
    });
    $('.bloem img').click(function(){
        // Wis de individuele score
    });
    function berekenRating(){
        // Bereken de totale score
    }
});
```

De eerste lijn binnen het script voegt de class rate_no toe aan elke span-tag binnen het ratingblok. De grijze sterren zijn zichtbaar.

Vervolgens komen er drie click-events die we dadelijk stap voor stap gaan uitwerken. Als laatste hebben we nog een functie waarbinnen we de totale score gaan berekenen.

Open de pagina in een browser en bekijk het voorlopige resultaat.

Als eerste gaan we de kleur van de sterren aanpassen. Bij elke klik op een ster krijgen deze ster plus alle voorgaande sterren een andere kleur. Dit doen we door de class rate_yes aan de span tags toe te voegen.

```
$('.rating span').click(function(){
    // Klik op een ster
```

```
    $(this).prevAll().addBack().addClass('rate_yes');
});
```

De rating werkt, maar we hebben nog één probleem. Een volgende ster aanklikken kan, maar een vorige ster aanklikken lukt niet. We kunnen de score dus wel verhogen, maar niet verlagen. Dit is op te lossen door eerst de reeds aanwezige class rate_yes op alle span-tags te wissen en pas daarna de class opnieuw toe te kennen.

> Pas de code als volgt aan en test het resultaat in een browser:

```
$('.rating span').click(function(){
    // Wis de totale score
    $(this).siblings().removeClass('rate_yes');
    $(this).prevAll().addBack().addClass('rate_yes');
});
```

Zodra een afbeelding een score heeft, gaan we de achtergrond van de afbeelding inkleuren. Verder moeten we ook nog de totale score berekenen.

> Pas de code als volgt aan en test het resultaat in een browser:
> (De totale score wordt nog niet berekend.)

```
$('.rating span').click(function(){
    // Wis de totale score
    $(this).siblings().removeClass('rate_yes');
    $(this).prevAll().addBack().addClass('rate_yes');
    $(this).parent().parent().find('.bloem').addClass('drop');
    berekenRating();
});
```

De totale score berekenen is vrij eenvoudig. Tel het aantal span-tags met de class rate_yes. Het aantal kan u ook hier weer opvragen met de eigenschap length.

> Pas de code als volgt aan en test het resultaat in een browser:

```
function berekenRating(){
    // Bereken de totale score
    var teller = $('.rate_yes').length;
    $('#teller').text(teller);
}
```

Om de totale score te wissen, volstaat het om overal de class rate_yes te wissen. De achtergrondkleur van de afbeeldingen verdwijnt en de teller wordt terug op 0 gezet.

> Pas de code als volgt aan en test het resultaat in een browser:

```
$('#wisRating').click(function() {
    // Wis de totale score
```

```
    $('span').removeClass('rate_yes');
    $('.bloem').removeClass('drop');
    berekenRating();
});
```

Door op een foto te klikken, gaan we de individuele score wissen. Kijk eerst even terug naar de broncode van een ratingblok. Vanaf de img-tag navigeren we naar de dichtstbijzijnde section-tag. Hiervoor gebruiken we tweemaal de methode parent() of éénmaal de methode closest('section'). Vervolgens gaan we op zoek naar alle span-tags en verwijderen we de class rate_yes. De achtergrondkleur van de afbeelding wordt gewist en de totale score wordt opnieuw berekend.

Pas de code als volgt aan en test het resultaat in een browser:

```
$('.bloem img').click(function(){
    // Wis de individuele score
    $(this).closest('section').find('span').removeClass('rate_yes');
    $(this).parent().removeClass('drop');
    berekenRating();
});
```

Als laatste controleren we nog even of het script volledig universeel is.

Voeg achter het laatste ratingblok een extra ratingblok toe.
Kopieer hiervoor een bestaand ratingblok en pas het pad naar de afbeelding aan.
(thumb_xxx.jpg wordt thumb_tulp.jpg.)

```
<article">
    ...
    <section>
        <div class="bloem">
            <img src="../assets/fotos/thumb_tulp.jpg"...>
        </div>
        <div class="rating">

<span></span><span></span><span></span><span></span><span></span>
        </div>
    </section>
</article>
```

Test het resultaat in een browser:

Events

Events zijn gebeurtenissen waarop we bepaalde acties kunnen uitvoeren. De browser genereert zelf al een aantal events (voor het laden van de pagina, na het laden van de pagina, ...), maar ook de gebruiker kan events activeren. Bijvoorbeeld door het klikken op een link (clickevent), tijdens het slepen met de muis, door een toets in te drukken, enz ...

Een volledig overzicht vind je hier: http://api.jquery.com/category/events/

Documentevents

Documentevents zijn gebeurtenissen die de browser zelf genereert en waar de gebruiker geen invloed op heeft. jQuery kent vier documentevents, namelijk:

$(document).ready()

Zoals we reeds weten is dit veruit het belangrijkste browserevent. Onze jQurycode kan immers niet werken als ze niet binnen dit event staat. Het ready-event wordt actief zodra de volledige pagina in het DOM is ingelezen, maar de code wordt wel uitgevoerd VOOR alle externe bestanden, zoals afbeeldingen en externe media zijn ingeladen.

Lange notatie: `$(document).ready(function(){ ... });`

Korte notatie: `$(function(){ ... });`

$(window).on('load', function(){...})

Het load-event wordt actief zodra de volledige pagina, inclusief externe bestanden, in het DOM is ingelezen.

```
$(window).on('load', function(){ ... });
```

$(window).on('unload', function(){...})

Het unload-event wordt actief zodra de gebruiker de pagina sluit.

```
$(window).on('unload', function(){ ... });
```

Open **events/documentEvents.html** en bekijk het resultaat in een browser.

Documentevents

```
<body>
    <h1>Documentevents</h1>
    <p><img src="..." alt="..."></p>
</body>
...
<script>
window.onload = function() {
    var imgUrl = 'https://picsum.photos/436/314';
    document.getElementById('foto').src = imgUrl;
    alert('window is loaded (vanilla javascript)\n' + imgUrl);
};
$(function () {
    $('img:eq(0)').attr('src', '../assets/fotos/roos.jpg');
    alert('document is ready (jQuery)');
});
$(window).on('load', function () {
    $('img:eq(0)').attr('src', '../assets/fotos/tulp.jpg');
    alert('window is loaded (jQuery)');
});
$(window).on('unload', function () {
    alert('window is unloaded');
});
</script>
```

Het src-attribuut van de afbeelding heeft weliswaar een waarde, maar vooraleer deze wordt geladen wordt die al overschreven.. De afbeelding wordt vanuit jQuery toegevoegd.

De scripts worden in deze volgorde uitgevoerd:

- Eerst wordt het ready-event (jQuery) uitgevoerd. De roos verschijnt op de pagina.

- Vervolgens wordt het load-event (jQuery) en het onload-event (vanilla javascript) nagenoeg gelijktijdig uitgevoerd. Omdat `$(window).on('load',...)` in de code na `window.onload` staat, wordt de roos vervangen door een tulp.
- Bij het sluiten van de pagina (of bij een reload) wordt het unload-event geactiveerd.

> Plaats in het script de functie window.onload eens achteraan de script-tag en kijk welke afbeelding er op het einde tevoorschijn komt.

> Merk op dat niet elke browser hetzelfde reageert op de verschillende events. (Zie caveats load-event) De meeste browser onderdrukken zelfs het unload-event omdat er in het verleden vaak misbruik van werd gemaakt.

Gebruikersevents

Gebruikersevents zijn steeds gekoppeld aan een selector. De actie staat binnen een zogenoemde callback functie.

Syntax: `$(selector).event(function(){...acties...}).`

Verder is de naam van de methode identiek aan de naam van het event. Bijvoorbeeld: de methode `click()` staat voor het click-event.

Herinner u de term unobtrusive JavaScript. Het JavaScript wordt volledig gescheiden van de HTML-structuur, zodat we de code makkelijk kunnen onderhouden en hergebruiken. Omdat een event vanuit het script aan een selector wordt gekoppeld, noemt men dit event binding. De functie binnen het event noemen we de event handler.

Gebruikersevents kunnen we onderverdelen in verschillende categorieën, namelijk muisevents, formulierevents en toetsenbordevents.

Muisevents

De acties met de muis vormen de grootste groep events. Enkele van deze events hebben we reeds in voorgaande hoofdstukken gebruikt. De belangrijkste muisevents zijn:

- `click()`: klik op de selector.
- `dblclick()`: dubbelklik op de selector.
- *`mouseover()`: plaats de muis boven de selector.
- *`mouseenter()`: plaats de muis boven de selector.
- *`mouseout()`: verwijder de muis van de selector.
- *`mouseleave()`: verwijder de muis van de selector.
- `mousemove()`: verplaats de muis over de selector.
- `mousedown()`: plaats de muis boven de selector en klik op de muistoets.
- `mouseup()`: plaats de muis boven de selector, klik op de muistoets en laat ze weer los.

* Er is een groot verschil tussen `mouseover()`/`mouseenter()` en `mouseout()`/`mouseleave()`.

Het event `mouseover()` wordt getriggerd zodra u met de muis boven de selector komt, maar wordt ook getriggerd indien u over descendant elementen komt. Het event `mouseenter()` houdt daarentegen geen rekening met de descendant elementen. Volgend voorbeeld illustreert het verschil tussen beide events.

> Open **events/mouseOverEnter.html** en bekijk het resultaat in een browser.

```
<article id="overEnter">
    <section>
        <h2>Mouseover</h2>
        <p id="over"> </p>
        <div id="div1"><div id="div2"><div id="div3">
            <p>...</p><p>...</p>
        </div></div></div>
    </section>
    <section>
        <h2>Mouseenter</h2>
        <p id="enter"> </p>
        <div id="div4"><div id="div5"><div id="div6">
            <p>...</p><p>...</p>
        </div></div></div>
    </section>
</article>
.....
<script>
    $(function() {
        var i = 0
        $('#div1').mouseover(function() {    // mouseover
            ++i;
            $('#over').text('mouseover: ' + i);
        });
        var j = 0
        $('#div4').mouseenter(function() {   // mouseenter
            ++j;
            $('#enter').text('mouseover: ' + j);
        });
    });
</script>
```

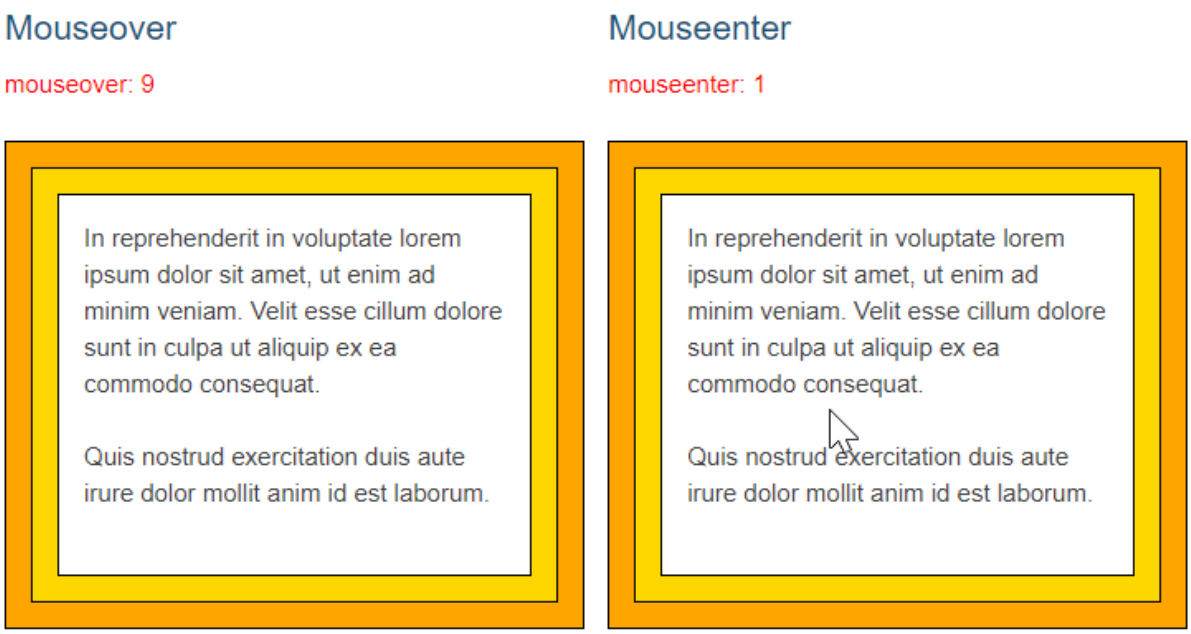
De pagina bevat verschillende div-tags. #div1 heeft als childelement #div2 en verder ook #div3 en nog enkele p-tags als dieper gelegen descendant elementen. #div4 heeft als childelement #div5 en #div6 en enkele p-tags als descendants op een lager niveau.
Op #div1 staat het mouseover-event en op #div4 het mouseenter-event.

> Beweeg de muis verticaal over **#div1**
> Beweeg de muis verticaal over **#div4**

Telkens u het mouseover-event (#div1) activeert, wordt de teller i met één verhoogd en verschijnt de waarde van i in paragraaf #over.
Zodra u de muis één niveau dieper (of hoger) in de boomstructuur verplaatst, wordt de teller verhoogd.

U zal merken dat de teller j enkel verhoogt als u over #div4 komt. De onderliggende descendant elementen zullen de teller niet verhogen.

Het mouseover-event gaat gepaard met event bubbling. Hierover later meer.



Muisevents

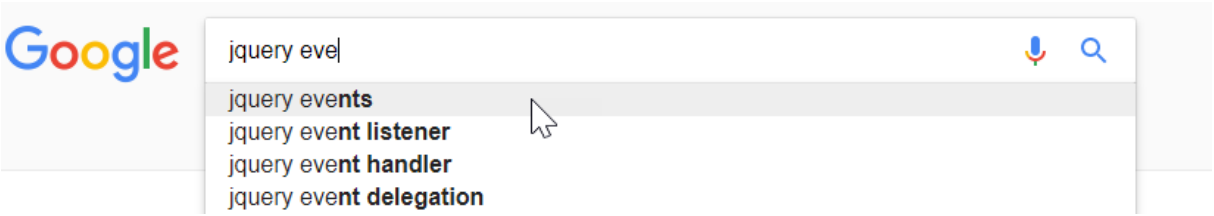
Formulierevents

Volgende events hebben betrekking op formulierelementen zoals tekstvelden, radiobuttons, checkboxen en knoppen.

- `blur()`: de selector verliest de focus.
- `change()`: de waarde van de selector wijzigt.
- `focus()`: de selector krijgt de focus.
- `select()`: selecteer de inhoud van de selector.
- `submit()`: verzend het formulier.

Toetsenbordevents

Toetsenbordevents worden vaak in combinatie met formulieren gebruikt. Denk maar aan een autosuggest tekstveld. De gebruiker typt enkele letters in een zoekveld en automatisch verschijnen er enkele suggesties. Het Google zoekveld maakt bijvoorbeeld gebruik van autosuggest.



Toetsenbordevents

De toetsenbordevents in jQuery zijn:

- `keydown()`: druk een toets in.
- `keypress()`: druk een toets in (werkt niet met speciale toetsen zoals esc, del, ...).
- `keyup()`: laat de toets los.

Indien u een toets lange tijd ingedrukt houdt, wordt `keydown()` maar eenmaal geactiveerd en `keypress()` meermaals geactiveerd.

Gebruikersevents zonder callback functie

Alle voorgaande methodes met een callback functie worden gebruikt als event handler. Dezelfde methodes kunnen ook zonder functie gebruikt worden en zullen dan het event triggeren. Enkele voorbeelden:

- `$('#button1').click()`: activeer het click-event op #button1.
- `$('#naam').focus()`: plaatst de cursor in het (tekst)veld #naam.
- `$('#form1').submit()`: verzend het formulier met id #form1.

Zoals het laatste voorbeeld illustreert, kan u volledig programmatorisch een formulier verzenden, zonder tussenkomst van de gebruiker. Deze methode wordt zeer vaak gebruikt bij contactformulieren. Indien de gebruiker een formulier verzendt, wordt eerst gecontroleerd of alle velden correct zijn ingevuld. (naam, email, enz...). Pas als alle gegevens correct zijn ingevuld, zal het formulier verzonden worden. Bij foutieve gegevens krijgt de gebruiker enkel een foutboodschap.

Event bubbling

Open **events/eventBubbling.html.**

```
<div id="bubble">
    <ul>
        <li>item 0: Ullamco laboris nisi quis nostrud
exercitation</li>
        <li>item 1: <a href="javascript:;">link 0</ex ...</li>
        <li>item 2: Sunt in <a href="javascript:;">link1</a>...</li>
    </ul>
</div>
.....
<script>
$(function() {
    $('#bubble a').click(function() {
        alert('U klikte op een a-tag');
    });
    $('#bubble li').click(function() {
        alert('U klikte op een li-tag');
    });
    $('#bubble ul').click(function() {
        alert('U klikte op een ul-tag');
    });
});
</script>
```

Deze pagina bevat een lijst met daarin enkele links. In de code vindt u een click-event op de a-tag, op de li-tag en op de ul-tag.

> Open de pagina in een browser en klik op een link.
> Welke events worden getriggerd?

Je zal merken dat de drie events getriggerd worden. In JavaScript is het namelijk zo dat een event niet enkel getriggerd wordt op de bedoelde selector, maar dat het event eveneens wordt doorgegeven aan alle parent elementen. Dit effect noemt men event bubbling. Het event borrelt als het ware door het DOM naar boven.

Omdat de parent elementen li en ul eveneens aan een click-event gekoppeld zijn, worden deze events ook uitgevoerd. Indien u in het ontwerp geen rekening houdt met event bubbling, kan dit wel eens tot onverwachte resultaten leiden.

Event methodes en properties

Een event is een object met eigen methodes en properties.

Event methodes

De twee belangrijkste methodes zijn:

- `e.stopPropagation()`: stopt event bubbling.
- `e.preventDefault()`: de default actie wordt niet getriggerd.

> Pas het click-event op de a-tag uit vorige paragraaf als volgt aan:

```
$('a').click(function(e) {
    e.stopPropagation();
    alert('U klikte op een a-tag');
});
```

Het event e wordt als parameter aan de functie toegevoegd. De methode `e.stopPropagation()` zorgt ervoor dat event bubbling wordt uitgeschakeld.

> Open de pagina in een browser en klik opnieuw op de a-tag.
> De events op de li-tag en op de ul-tag worden niet meer uitgevoerd.
> Open **events/eventMethodes.html.**

```
$('a').click(function(e) {
    e.preventDefault();
    alert('Link werkt niet meer');
});
```

De pagina bevat twee externe links. De methode `e.preventDefault()` zorgt ervoor dat de default actie (het openen van de link) niet wordt uitgevoerd.

De positie van `e.preventDefault()` binnen de functie is belangrijk. Stel dat de functie een fout bevat. Alle codes voor de fout worden uitgevoerd, maar alle codes na de fout niet.

> Genereer een fout in de code.
> Maak van alert() bijvoorbeeld alerttt() en test de link.
> e.preventDefault() staat voor de foutieve code, dus de URL wordt niet geopend.

> Plaats e.preventDefault(); achter alerttt() en test de links opnieuw.
> e.preventDefault() staat achter de foutieve code. De URL bij de link werkt nu wel.

Event properties

Enkele event properties:

- `e.currentTarget`: het elementtype waarop het event werd uitgevoerd.
- `e.pageX`: de horizontale positie van de cursor op de pagina.
- `e.pageY`: de verticale positie van de cursor op de pagina.
- `e.timeStamp`: het tijdstip waarop het event werd uitgevoerd. Een timestamp is het aantal ms verstreken sinds het laden van de pagina (Chrome, Firefox) of sinds 1 januari 1970 (Internet Explorer).
- `e.type`: het eventtype.

> Open **events/eventProperties.html** in een browser en klik op de verschillende tags

```
$(function() {
    $('h1, h2, p, li').click(function(e) {
        var props = '';
        props += 'e.currentTarget: ' + e.currentTarget + '<br>';
        props += 'e.pageX: ' + e.pageX + '<br>';
        props += 'e.pageY: ' + e.pageY + '<br>';
        props += 'e.timeStamp: ' + e.timeStamp + '<br>';
        props += 'e.type: ' + e.type + '<br>';
        $('#props').html(props);
    });
});
```

Event properties (h1)

Paragraaf 1 (p)

List1 (h2)

- Ullamco laboris nisi quis nostrud exercitation
- Ex ea commodo consequat
- Sunt in culpa cupidatat oident, eu fugiat nulla pariatur

e.currentTarget: [object HTMLLIElement]
e.pageX: 234
e.pageY: 237
e.timeStamp: 37430.39999995381
e.type: click

Hover-event: methode met een dubbele functie

Sommige events worden regelmatig in combinatie gebruikt. Neem bijvoorbeeld het mouseenter-event en het mouseleave-event om een sectie te tonen en daarna weer te verbergen. Beide combinaties zijn gebundeld in een event met een dubbele uitvoerfunctie.

```
hover(function(){...}, function(){...}).
```

De eerste functie staat voor het mouseenter-event en de tweede functie staat voor het mouseleave-event.

> Open **events/hover.html** en test het resultaat in een browser.

```
$(function() {
    $('#div1').hover(function(e) {
        $(this).addClass('orange');
        $('#res1').text('event.type: '+ e.type);
    }, function(e) {
        $(this).removeClass('orange');
        $('#res1').text('event.type: '+ e.type);
    });
});
```

De eerste functie binnen de methode `hover()` (het mouseenter-event) voegt aan #div1 de class .orange toe. De tweede functie (het mouseleave-event) verwijdert de class .orange. Het eventtype verschijnt net boven de div-tag.

Hover (mouseenter + mouseleave)

event.type: mouseenter

Ut aliquip ex ea commodo consequat. Ut labore et dolore magna aliqua. Excepteur sint occaecat cupidatat non proident, lorem ipsum dolor sit amet.

De methode on()

De methode `on()` is het manusje-van-alles. Alle methodes die we tot hiertoe besproken hebben, kunnen we eveneens via `on()` definiëren.

Syntax: `$(selector).on('event', function(){...});`

Enkele voorbeelden:

- `$(selector).click(function(){...})` is identiek aan `$(selector).on('click', function(){...})`
- `$(selector).mouseenter(function(){...})` is identiek aan `$(selector).on('mouseenter', function(){...})`
- `$(selector).blur(function(){...})` is identiek aan `$(selector).on('blur', function(){...})`

Maar de methode `on()` gaat nog veel verder. Zo kunnen we meerdere events koppelen aan dezelfde functie. We kunnen meerdere events koppelen aan verschillende functies en we kunnen events koppelen aan dynamisch gegenereerde of geïmporteerde HTML-code.

Eén selector en meerdere events met dezelfde functie

Syntax: `$(selector).on('event1 event2 ...', function(){...});`

Open **events/on_1.html** en bekijk het resultaat in een browser.

```
$(function() {
    $('main li').on('click mouseenter', function(e) {
        var ev = e.type;
        var txt = $(this).text();
        $('#resultaat').html('event: '+ ev + ' <br>tekst: ' + txt);
    });
});
```

Eén selector, meerdere events met dezelfde functie

event: click

tekst: Consectetur adipiscing elit.

- Lorem ipsum dolor sit amet.
- Consectetur adipiscing elit.
- Ut ultricies ullamcorper augue et rhoncus.
- Suspendisse ut diam odio. Donec pellentesque bibendum laoreet.

Zowel het click-event als het mouseenter-event voert dezelfde functie uit.

Eén selector en meerdere events met verschillende functies

Syntax: `$(selector).on({event1:function(){...}, event2:function(){...}});`

Open **events/on_2.html** en bekijk het resultaat in een browser.

```
$(function() {
    $('main li').on({
        click: function() {
            $('main li').removeClass('orange');
            $(this).addClass('orange');
        },
        mouseenter: function() {
            $(this).addClass('green');
        },
        mouseleave: function() {
            $(this).removeClass('green');
        }
    });
});
```

Eén selector, meerdere events met verschillende functies

- Lorem ipsum dolor sit amet.
- Consectetur adipiscing elit.
- Ut ultricies ullamcorper augue et rhoncus.
- Suspendisse ut diam odio. Donec pellentesque bibendum laoreet.

De li-tag triggert drie events. Het click-event kleurt de achtergrond van de li-tag oranje. Het mouseenter-event kleurt de achtergrond groen. Het mouseleave-event brengt de achtergrondkleur terug naar zijn vorige toestand.

Merk op dat dit niet werkt voor een mouseenter van een li die oranje is door een clickevent. Het event wordt weliswaar ook dan getriggerd, maar wanneer je meerdere classes toekent aan een

element en daarin krijgt dezelfde property verschillende waarden, dan wordt de laatst toegekende waarde gebruikt, en dat is afhankelijk van de volgorde waarin de classes in de CSS-code gedefinieerd zijn, en niet van de volgorde waarin de classes worden toegekend!

Live event

Alle methodes, met uitzondering van `on()` werken uitsluitend op een selector die bij het openen van de pagina aanwezig is. In de volgende hoofdstukken gaan we op verschillende manieren extra HTML-code in het DOM toevoegen. Een event aan dynamisch gegenereerde code koppelen, kan enkel via on().

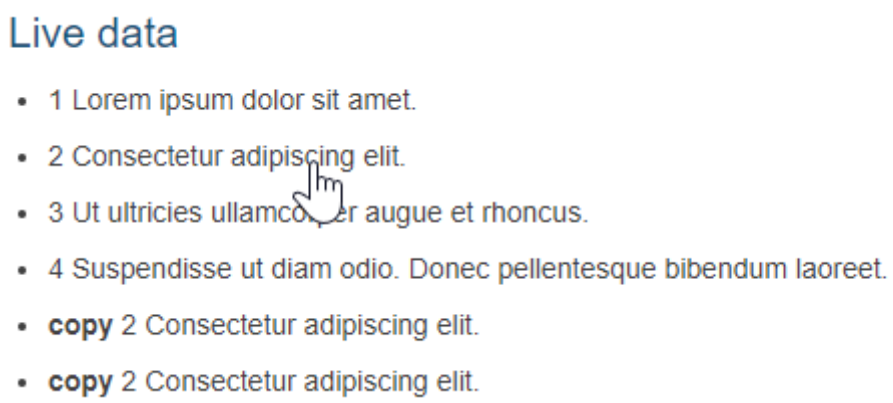
> Open **events/on_3.html** en bekijk het resultaat in een browser.

```
$(function() {
    $('main li').click(function(){
        var txt = '<li><b>copy</b> ' + $(this).html() + '</li>';
        $('main ul').append(txt);
    });
});
```

De variabele txt maakt een nieuwe li-tag met daarin het woord copy, gevolgd door de tekst van het geklikte item. Op de volgende lijn wordt de waarde van txt met `append()` achteraan de ul-tag toegevoegd.

> Klik enkele malen op één van de vier listitems.
> Klik nu één van de nieuw gegenereerde items.

Je zal merken dat het click-event niet wordt uitgevoerd.



Om de gegenereerde code ook aan het click-event te koppelen, zijn we genoodzaakt om het event via de methode `on()` te koppelen.

Syntax: `$(ancestor).on('event', 'selector', function(){...});`

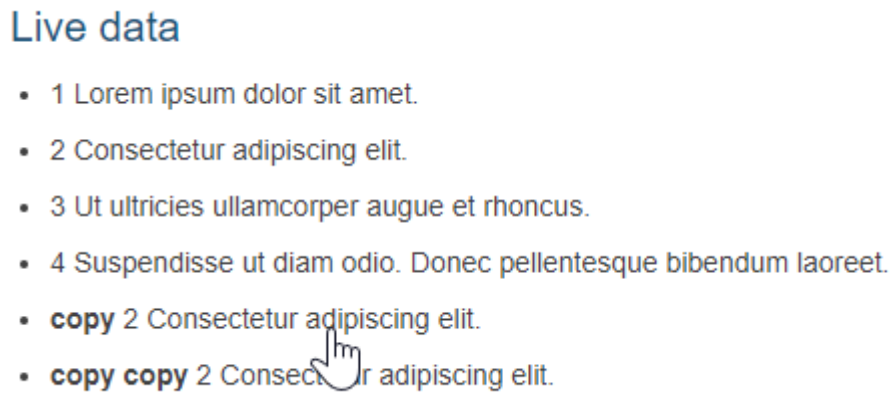
Waar de selector normaal voor de methode staat, verhuist deze nu binnen de methode. Voor de selector komt een ancestor tag vanwaar het event actief wordt. De ancestor tags voor onze li-tag zijn ul, article, div#outline en body.

Dus: `$('li').click(function(){});` wordt: `$('ul').on('click', 'li', function(){});` of: `$('body').on('click', 'li', function(){});`

> Pas de code als volgt aan:

```
$(function() {
    $('main ul').on('click', 'li', function(){
        var txt = '<li><b>copy</b> ' + $(this).html() + '</li>';
        $('main ul').append(txt);
    });
});
```

De dynamisch gegenereerde li-tags zijn nu ook aanklikbaar en zullen opnieuw een copy achteraan de lijst toevoegen.



Wat je uiteindelijk voor `on()` plaatst, is afhankelijk van de toepassing.

In dit voorbeeld kan je `$('main ul')` eventueel vervangen door `$('body ul')`. Dit geeft immers hetzelfde resultaat. De algemene regel is: plaats voor `on()` het element waarbinnen het event invloed mag hebben.

Event handlers verwijderen

De methode `off()` verwijdert de events die gekoppeld zijn vanuit `on()`.

Toepassing 1: reactietest

> Open **events/toep_reactieTest.html** en bekijk het resultaat in een browser.

```
<section id="speelveld">
    <div id="blok"></div>
</section>
...
<script>
$(function() {
    var startTijd, eindTijd;
    var wachttijd = 0;
    var speelH = $('#speelveld').height() - $('#blok').height();
    var speelW = $('#speelveld').width() - $('#blok').width();
    $('#start').click(function(e) {
        // klik op startknop
    });
    ('#blok').click(function(e) {
        // klik op blok
    });
});
</script>
```

De section-tag #speelveld krijgt vanuit CSS een gele achtergrondkleur en staat relatief gepositioneerd. Binnen #speelveld staat een div-tag #blok. Deze staat absoluut gepositioneerd binnen #speelveld. Zodra u op de startknop klikt, gaan we na enkele seconden #blok op een willekeurige plaats binnen #speelveld positioneren. De reactietijd is de tijd tussen het verschijnen van #blok en het moment dat u op #blok klikt.

De variabelen speelH en speelW bepalen de hoogte en de breedte waarbinnen we #blok mogen positioneren. #speelveld is 500px breed. #blok is 40px breed. Dus de x-coördinaat van #blok kan minimum 0px en maximum 460px bedragen. Voor de y-positie maken we een gelijkaardige berekening.

Pas het click-event op de knop als volgt aan:

```
$('#start').click(function(e) {
    $('#blok').hide();
    var blokX = Math.floor(Math.random() * speelW) + 'px';
```

```
    var blokY = Math.floor(Math.random() * speelH) +'px';
    $('#blok').css({top: blokY, left: blokX});
    startTijd = e.timeStamp;
    setTimeout(function(){
        $('#blok').show();
    },wachttijd);
});
```

Telkens we op de klop klikken, gaan we #blok verbergen. De variabelen blokX en blokY geven ons een willekeurig getal tussen 0px en 460px. Op de vijfde lijn wordt #blok opnieuw gepositioneerd.

Het tijdstip waarop we op de knop klikken, wordt bewaard in de variabele startTijd.

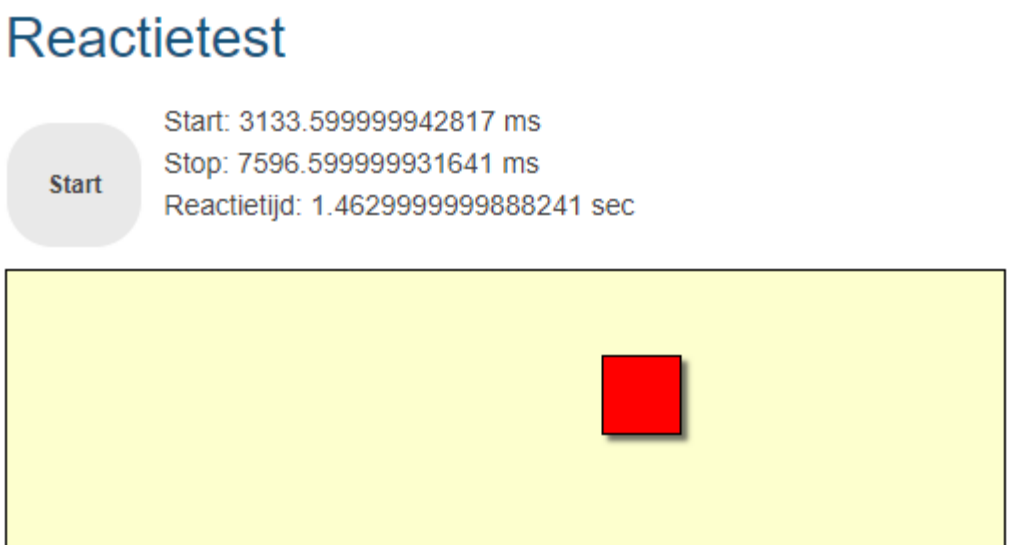
Als laatste maken we #blok opnieuw zichtbaar. Dit gebeurt via de (JavaScript) methode `setTimeout()` met een tijdsvertraging van 0 sec. (de variabele wachttijd staat voorlopig nog op 0.)

> Bewaar de pagina en test het resultaat in een browser.
> Klik enkele malen op de knop.
> Het blok verschijnt telkens op een willekeurige plaats.
> Pas vervolgens de code achter het #blok aan:

```
$('#blok').click(function(e) {
    eindTijd = e.timeStamp;
    var tijd = (eindTijd - startTijd - wachttijd)/1000;
    $('#uwScore').html('Start: ' + startTijd + ' ms<br>' +
        'Stop: ' + eindTijd + ' ms<br>' +
        'Reactietijd: ' + tijd + ' sec');
});
```

De variabele eindTijd bevat het tijdstip waarop we op het blok klikken. Zowel startTijd als eindtijd bevatten een timestamp in ms. De variabele tijd geeft het verschil tussen het verschijnen van het blok en het moment waarop we op het blok klikken. (Vergeet niet de wachttijd in rekening te brengen.) Door het tijdsverschil te delen door 1000, bekomen we een getal in sec. De reactietijd wordt in de p-tag #uwScore getoond.

> Bewaar de pagina en test het resultaat in een browser.
> Klik op de startknop en vervolgens op het rode blok.



Zet bovenaan het script de variabele wachttijd op 3000 en test het resultaat.

Animaties en effecten

Met `animate()` kan u bijna elke CSS-eigenschap dynamisch wijzigen. `animate()` wijzigt binnen een bepaalde tijdspanne één of meerdere CSS-eigenschappen van waarde a naar waarde b. Bijvoorbeeld: wijzig de breedte en de dekking van een afbeelding van 100% naar 20% over een tijdspanne van 3 seconden, wordt:

```
$('img').animate({width: '20%', opacity: .2},3000);
```

Om het u nog makkelijker te maken, zijn een drietal veel gebruikte animaties gebundeld in een eigen methode. Deze methodes noemt men in jQuery effecten.

Verder kan u verschillende animaties/effecten met elkaar combineren. Ofwel lopen deze animaties simultaan ofwel sequentieel (in queue). In een sequentiële animatie kan u eventueel nog pauzes inlassen.

Verder bespreken we nog enkele methodes om de animatie te stoppen en om terug te keren naar de initiële toestand.

Een volledig overzicht van alle animaties/effecten vindt u op:
http://api.jquery.com/category/effects/

Basiseffecten

Alle effecten die we hier bespreken, gebruiken tot drie optionele parameters. Neem bijvoorbeeld hide(). Dit effect verbergt het geselecteerde element. Syntax: `hide([duration] [, easing] [, callback])`

- **Duration of tijdsduur.**
  Dit is de tijdspanne waarover de animatie loopt. U kan de tijd opgeven in milliseconden of met de strings slow , normal of fast (respectievelijk 600 ms, 400 ms en 200 ms).
- **Easing of versnelling.**
  jQuery kent twee optionele versnellingsparameters: linear en swing (de default waarde). Via plugin's zijn extra versnellingsparameters beschikbaar. Hierover later meer.
- **Callback**
  Deze functie wordt uitgevoerd zodra de animatie volledig is afgerond.

> Open **animatie/basisEffecten.html** en bekijk de broncode.
> Test de pagina in een browser.

Deze pagina bevat enkele div-tags. Het click-event op de verschillende knoppen activeert een animatie.

Show(), hide() en toggle()

Deze effecten passen tijdens de animatie de CSS-eigenschappen height, width en opacity aan. Aan het einde van de animatie wordt voor `hide()` de eigenschap `display:none` toegevoegd. Op het einde van `show()` komt de display-eigenschap terug naar de initiële toestand. Voor blocklevel elementen zoals een div-tag of een h1-tag is de initiële toestand `display:block`. Voor inline elementen zoals een span-tag of een strong-tag is de initiële toestand `display:inline`.

> Bekijk de effecten die bij de links van div1 horen.
> Test de pagina in een browser.

```
$(function() {
    var snelheid = 1000;
    // animatie op div1
    $('#verbergDiv1').click(function(){
        $('#div1').hide(snelheid);
    });
    $('#toonDiv1').click(function(){
```

```
        $('#div1').show(snelheid);
    });
    ...
});
```

De animatiesnelheid kan u aanpassen met de variabele snelheid. Initieel staat snelheid ingesteld op 1000 (1000 ms of 1 sec). Als snelheidswaarde kan u een getal ingeven of de strings slow, normal en fast. De twee eerste knoppen maken #div1 respectievelijk onzichtbaar en zichtbaar.

```
$(function() {
    var snelheid = 1000;
    ...
    $('#toggleDiv1').click(function(){
        $('#div1').toggle(snelheid);
    });
    ...
});
```

Met de methode `toggle()` kan u de vorige toestand omkeren. Zichtbaar wordt onzichtbaar en omgekeerd. Telkens u op de knop #toggleDiv1 klinkt, wijzigt de toestand. (Zie ook paragraaf 6.5.)

```
$(function() {
    var snelheid = 1000;
    ...
    $('#toggleDiv1Callback').click(function(){
        $('#div1').toggle(snelheid, function(){
            if ($('#div1').is(':visible')) {
                alert('Div1 is zichtbaar');
            } else {
                alert('Div1 is onzichtbaar');
            }
        });
    });
});
```

De vierde knop activeert het toggle-effect, voorzien van een callback functie. De if-voorwaarde geeft als waarde true of false. Afhankelijk van deze waarde wordt de bijbehorende tekst getoond.

`$('#div1).is(':visible')` is niet hetzelfde als `$('#div1:visible').`

`$('#div1:visible')` is een selector en bevat dan enkel het object, en niet de waarde true of false. Het resultaat van het if-else-statement is bijgevolg altijd false en de boodschap is dus alert('Div1 is onzichtbaar').

SlideUp(), slideDown() en slideToggle()

Ook deze effecten tonen/verbergen het element, maar nu met een schuifeffect.

Deze methodes passen enkel de CSS-eigenschap height aan om uiteindelijk te eindigen met de eigenschap `display:none` voor `slideUp()` en `display:block` of `display:inline` voor `slideDown()`.

Bekijk de effecten die bij de links van div2 horen.
Test de pagina in een browser.

slideUp div2 | slideDown div2 | slideToggle div2 | slideToggle div2 met callback

Lorem ipsum dolor sit amet, velit esse cillum dolore consectetur adipisicing elit. Duis aute irure dolor ut enim ad minim veniam, mollit anim id est laborum. Ut aliquip ex ea commodo consequat. Velit esse cillum dolore sunt in culpa quis nostrud exercitation.

```
$(function() {
    var snelheid = 1000;
    $('#verbergDiv2').click(function(){
        $('#div2').slideUp(snelheid);
    });
    $('#toonDiv2').click(function(){
        $('#div2').slideDown(snelheid);
    });
    $('#toggleDiv2').click(function(){
        $('#div2').slideToggle(snelheid);
    });
    $('#toggleDiv2Callback').click(function(){
        $('#div2').slideToggle(snelheid, function(){
            if ($('#div2').is(':visible')) {
                alert('Div2 is zichtbaar');
            } else {
                alert('Div2 is onzichtbaar');
            }
        });
    });
});
```

FadeIn(), fadeOut() en fadeToggle()

Ook deze methodes tonen/verbergen het element, maar nu met een vervagingseffect.

Deze methodes passen enkel de CSS-eigenschap `opacity` aan om uiteindelijk te eindigen met de eigenschap display:none voor `fadeOut()` en `display:block` of `display:inline` voor `fadeIn()`.

Bekijk de effecten die bij de links van div3 horen.
Test de pagina in een browser.

Div3: fadeIn, fadeOut en fadeToggle

fadeOut div3 | fadeIn div3 | fadeToggle div3 | fadeToggle div3 met callback

Ut aliquip ex ea commodo consequat. Ut labore et dolore magna aliqua. Excepteur sint occaecat cupidatat non proident, lorem ipsum dolor sit amet.

```
$(function() {
    var snelheid = 1000;
    $('#verbergDiv3').click(function(){
        $('#div3').fadeIn(snelheid);
    });
    $('#toonDiv3').click(function(){
        $('#div3').fadeOut(snelheid);
    });
    $('#toggleDiv3').click(function(){
        $('#div3').fadeToggle(snelheid);
    });
    $('#toggleDiv3Callback').click(function(){
        $('#div3').fadeToggle(snelheid, function(){
            if ($('#div3').is(':visible')) {
                alert('Div3 is zichtbaar');
            } else {
                alert('Div3 is onzichtbaar');
            }
        });
    });
});
```

FadeTo()

Bij alle voorgaande methodes verdwijnt de div-tag volledig van het scherm. De vrijgekomen plaats wordt ingenomen door de onderliggende elementen of, met andere woorden, aan de div-tag wordt de CSS-eigenschap `display:none` toegevoegd.

De methode `fadeTo()` gebruikt de CSS-eigenschap visibility (`hidden` of `visible`). Ook hiermee kan u de div-tag laten verdwijnen, maar de plaats wordt niet vrijgegeven. De onderliggende elementen schuiven dus niet naar boven.

De syntax van `fadeTo()` is ook iets anders dan de voorgaande:

```
.fadeTo( duration, opacity [, easing] [, callback] )
```

De parameters `duration` en `opacity` zijn verplicht. De opacity of dekking is een getal tussen 0 (onzichtbaar) en 1 (zichtbaar).

> Bekijk de effecten die bij de links van div4 horen.
> Test de pagina in een browser.

Div4: fadeTo

fadeTo 50% div4 | fadeTo 0% div4 | fadeTo 100% div4

Lorem ipsum dolor sit amet, velit esse cillum dolore consectetur adipisicing elit. Duis aute irure dolor ut enim ad minim aliquip ex ea commodo consequat. Velit esse cillum dolore sunt in culpa quis nostrud exercitation.

```
$(function() {
    var snelheid = 1000;
    $('#fadeDiv4_50').click(function(){
        $('#div4').fadeTo(snelheid, .5);
    });
    $('#fadeDiv4_0').click(function(){
        $('#div4').fadeTo(snelheid, 0);
    });
    $('#fadeDiv4_100').click(function(){
        $('#div4').fadeTo(snelheid, 1);
    });
});
```

Aangepaste animaties

Alle voorgaande effecten hebben betrekking op het tonen/verbergen van een element. Maar wat als we andere CSS-eigenschappen willen animeren? Hiervoor gebruiken we de methode `animate()`.

Met `animate()` kan u enkel CSS-eigenschappen aanpassen die een getal bevatten. Eigenschappen die uit een string bestaan kan u niet wijzigen.

Bijvoorbeeld: de achtergrondkleur en de achtergrondafbeelding zijn niet aanpasbaar. Er zijn echter nog twee niet-CSS-eigenschappen, namelijk `scrollTop` en `scrollLeft`.

Syntax: `.animate({properties} [, duration] [, easing] [, callback])`

- Properties bevat een map (of object literals) van één of meerdere eigenschappen die u wilt wijzigen.
  Zoals beschreven in hoofdstuk 2 gebruikt een object literal accolades om alle eigenschappen te groeperen en een komma als scheidingsteken.
  Bijvoorbeeld: `animate({width: '50%',left: 200},...)`
- Duration of tijdsduur.
- Easing of versnelling.
- Callback functie die wordt uitgevoerd na het beëindigen van de animatie.

Voor de syntax van de properties gelden speciale regels.

CSS-eigenschappen die uit één woord bestaan, kan u al dan niet tussen aanhalingstekens zetten.

- Goed: `animate({'width': '50%'})`
- Goed: `animate({width: '50%'})`

CSS-eigenschappen die uit twee woorden bestaan, MOET je tussen aanhalingstekens zetten of indien je het woord in camelCase zet (beide woorden aan elkaar schrijven, maar elk volgend woord beginnen met een hoofdletter), zijn aanhalingstekens niet verplicht.

- Goed: `animate({'border-width': '20px'}).`
- **Fout**: `animate({border-width: '20px'}).`
- Goed: `animate({'borderWidth': '20px'}).`
- Goed: `animate({borderWidth: '20px'}).`

Indien de waarde een eenheid bevat, zet deze dan steeds tussen aanhalingstekens. Indien de waarde geen eenheid bevat, mag u de aanhalingstekens weglaten. (De default eenheid is px.)

- Goed: `animate({'width': '50%'}).`
- Fout: `animate({'width': 50%}).`
- Goed: `animate({width: '300px'}).`
- Goed: `animate({width: '300'}).` (300 = '300px')
- Goed: `animate({width: 300}).`

> Open **animatie/aangepasteAnimaties.html**.
> Bekijk de broncode en test de pagina in een browser.

De pagina bevat zowel simultane als sequentiële animaties.

Simultane animaties

De eerste div-tag bevat een animatie waarbij de vier CSS-eigenschappen width, left, opacity en border gelijktijdig wijzigen.

> Bekijk de animatie die bij de knop animatie div1 hoort.
> Test het resultaat in een browser.

```
$(function() {
    var snelheid = 2000;
    $('#anim1').click(function(){
        $('#div1').animate({
            width: '50%',
            left: 200,  // vervang waarde door '+=200'
            opacity: 0.25,
            borderWidth: '20px'
        }, snelheid);
    });
});
```

imultane animaties

> Vervang de waarde van left : 200 door left : '+=200'.
> Test het resultaat in een browser.

De waarde +=200 staat voor waarde = waarde + 200. Telkens u op de knop drukt, start de animatie opnieuw en wordt het blok 200 pixels naar rechts geschoven. Na een aantal kliks schuift het blok zelfs volledig buiten het beeld.

Sequentiële animaties

Een sequentiële animatie wordt opgebouwd door een reeks enkelvoudige animaties achter elkaar te plaatsen. Hiervoor gebruiken we zoals steeds de punt-notatie. U kan de methode animate() gebruiken, maar ook alle reeds besproken effecten kan je aan de ketting toevoegen.

Bijvoorbeeld: `$('#div2').animate(...).slideUp(...).animate(...).`

> Bekijk de animatie die bij de knop animatie div2 hoort.
> Test het resultaat in een browser.

```
$(function() {
    var snelheid = 2000;
    $('#anim2').click(function(){
        $('#div2').animate({
            width: '50%'
        }, snelheid).animate({
            left: 200
        }, snelheid).animate({
            opacity: .25
        }, snelheid).slideUp(snelheid);
    });
});
```

Simultane + sequentiële animaties

In voorgaand voorbeeld bevatte animate() telkens één eigenschap. Zo ontstond een volledig sequentiële animatie. Door meerdere eigenschappen te combineren, kan u makkelijk een combinatie van sequentiële en simultane animaties bouwen.

Bekijk de animatie die bij de knop animatie div3 hoort.
Test het resultaat in een browser.

```
$(function() {
    var snelheid = 2000;
    $('#anim3').click(function(){
        $('#div3').animate({
            width: '50%',
            left: 200,
            opacity: 0.25
        }, snelheid).slideUp(snelheid);
    });
});
```

Eerst worden de eigenschappen width, left en opacity simultaan gewijzigd. Op het einde van deze cyclus komt het slide-effect in actie.

Animatie pauzeren: delay()

In een sequentiële animatie kan u tussen de opeenvolgende stappen een pauze inlassen met de methode delay(). De vertraging wordt zoals steeds opgegeven in milliseconden.

Bekijk de animatie die bij de knop animatie div4 hoort.

```
$(function() {
    var snelheid = 2000;
    $('#anim4').click(function(){
        $('#div4').animate({
            width: '50%',
            left: 200,
            opacity: 0.25
        }, snelheid).delay(3000).slideUp(snelheid);
        });
});
```

In bovenstaand voorbeeld zit er tussen beide animaties een wachttijd van 3 seconden.

Animatie beëindigen: finish()

De methode finish() beëindigt de volledige, al dan niet sequentiële, animatie die op dat moment wordt uitgevoerd en toont het object dadelijk met de CSS-eigenschappen in zijn eindtoestand.

Bekijk de animatie die bij de knop animatie div5 hoort.

```
$(function() {
    var snelheid = 2000;
    $('#anim5').click(function(){
        $('#div5').animate({
            width: '50%'
        }, snelheid).animate({
            left: 200
```

```
        }, snelheid).animate({
            opacity: .25
        }, snelheid);
    });
    $('#finish5').click(function(){
        $('#div5').finish();
    });
});
```

Animatie afbreken: stop()

De methode `stop()` kent twee optionele parameters.

Syntax: `.stop([clearQueue] [, jumpToEnd])`

- **clearQueue** (`true` of `false`)
  Indien true worden alle volgende animaties uit de sequentie gestopt.
  (Default waarde is false.)

- **jumpToEnd** (`true` of `false`)
  Indien true springt de momenteel lopende animatie dadelijk naar zijn eindpunt en wordt dan gestopt. Indien false loopt de momenteel lopende animatie gewoon verder en wordt op het einde pas gestopt. (Default waarde is false.)

Bekijk de code die bij de resetknoppen hoort.

```
<input class="button" type="button" id="reset1" class="resetAnim"
data-id="div1" value="reset div1">
...
<script>
    $(function() {
        $('.resetAnim').click(function(){
            var deDiv = $(this).attr('data-id');     // of
$(this).data('id');
            $('#' + deDiv).stop(true,true).attr('style', '');
        });
    });
</script>
```

De vier resetknoppen op deze pagina hebben allemaal dezelfde class .resetAnim. Elke knop heeft een attribuut data-id met als waarde de naam van de te stoppen div-tag.

Als eerste plaatsen we de waarde van data-id in de variabele deDiv. Deze waarde halen we op via de methode `attr('data-id')` of via `data('id')`. De selector op de volgende lijn is `$('#' + deDiv)`. Voor deDiv = div1 wordt dit $('#div1').

Vervolgens plaatsen we de methode stop() op de geselecteerde div-tag. Omdat alle effecten en animaties in het DOM stijleigenschappen aan het element toevoegen, volstaat het om deze stijleigenschappen terug leeg te maken. Dit gebeurt via de methode `attr('style', '')`.

De methodes `attr()` en `data()` wordt later nog in detail besproken, maar voorlopig is het wel nuttig om te weten wat er juist gebeurt. Heel wat methodes die later volgen, hebben twee

betekenissen. Een methode `attr()` kan zowel een getter als een setter zijn. Voor de methode `data()` gebruiken we enkel de getter.

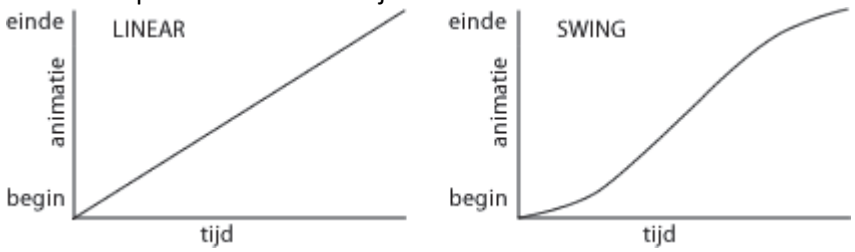
- **Getter** (haal de waarde van een eigenschap op).
  `attr('naam')`.
  `attr('data-naam')` is identiek aan `data('naam')`.
- **Setter** (geef de eigenschap een nieuwe waarde).
  `attr('naam', 'waarde')`.

Bijvoorbeeld: lees en wijzig de tekst van de knop reset1.

- `$('#reset1').attr('value')` geeft als resultaat reset div1.
- `$('#reset1').attr('value', 'wis animatie')` wijzigt de tekst in de knop van reset div1 naar wis animatie.

Versnellingsparameter

De versnelling of Easing-parameter bepaalt volgens welke curve de animatie verloopt. jQuery kent maar twee versnellingsparameters: linear en swing (de default waarde). Onderstaande grafieken tonen het snelheidsverloop in functie van de tijd.



Indien u de easing-plugin installeert, kan u nog tientallen extra versnellingsparameters gebruiken. Info en download: http://gsgd.co.uk/sandbox/jquery/easing/

> Open **animaties/versnellingsfunctie.html**.
> Bekijk de animaties en test het resultaat in een browser.

```
<div class="select">
    <select id="easing">
        <optgroup label="jQuery Core">
            <option value="linear">linear</option>
            <option value="swing">swing</option>
        </optgroup>
        <optgroup label="jQuery easing plugin">
            <option value="easeInOutBounce">easeInOutBounce</option>
            <option value="easeInOutQuart">easeInOutQuart</option>
            <option value="easeInOutCubic">easeInOutCubic</option>
            <option value="easeInOutQuint">easeInOutQuint</option>
            <option value="easeInOutSine">easeInOutSine</option>
            <option value="easeInOutExpo">easeInOutExpo</option>
            <option value="easeInOutCirc">easeInOutCirc</option>
            <option
value="easeInOutElastic">easeInOutElastic</option>
            <option value="easeInOutBack">easeInOutBack</option>
        </optgroup>
```

```
    </select>
    <i class="fa fa-angle-down fa-2"></i>
</div>
...
<h3>Easing effect = <span id="effect">...</span></h3>
...
<script src="../js/jquery.easing.1.3.js"></script>
<script>
    $(function() {
        var hoogte = $('#div1').height();
        var snelheid = 1000;
        $('#easing').change(function(){
            var easing = $(this).val();
            $('#effect').text(easing);
            $('#div1').animate({height:120}, snelheid, easing)
                .delay(300)
                .animate({height:hoogte}, snelheid, easing);
        });
    });
</script>
```

De eerste lijn linkt naar de easing-plugin zodat we de nieuwe versnellingsparameters kunnen gebruiken in onze animaties.

Het script bevat drie variabelen. In de variabele hoogte plaatsen we de werkelijke hoogte van de #div1. Dit gebeurt met de getter-functie height(). Als we eventueel later de inhoud van #div1 aanpassen, wordt ook de werkelijke hoogte van #div1 automatisch herrekend.

In de variabele easing bewaren we de versnellingsparameter. Telkens we een andere waarde uit de lijst met ID easing selecteren, wordt deze waarde uitgelezen en in de variabele easing bewaard. De gekozen waarde wordt ook zichtbaar op de pagina. `$('#effect').text(easing)` plaatst de waarde als tekst binnen de span-tag effect.

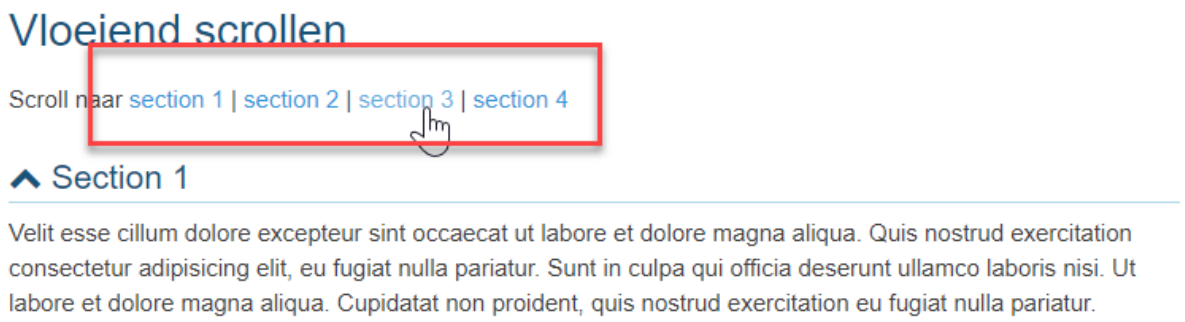
Nu de animatie zelf.
`$('#div1').animate({height:100}, snelheid, easing)` maakt #div1 120 pixels hoog. De verplaatsing gebeurt volgens de gekozen versnelling. Vervolgens wachten we 0,3 seconden `delay(300)` om daarna terug naar de originele hoogte te animeren via `animate({height:hoogte}, snelheid, easing);`

De easing-plugin bevat meer versnellingsparameters dan deze in de keuzelijst. Indien je de jQuery-UI-plugin gebruikt, hoef je de easing-plugin niet meer apart te linken. De volledige easing-plugin zit namelijk geïntegreerd in de jQuery-UI-plugin.

Toepassing 1: vloeiend scrollen

Open **animaties/toep_scroll.html.**
Test de links in een browser

De navigatie bovenaan verwijst met een ankerlink naar een bepaalde plaats in de pagina. In deze toepassing gaan we de overgangen via een animatie vloeiend laten verlopen. Daarna gaan we onderaan elke section-tag dynamisch een extra link toevoegen die naar het anker verwijst dat net voor het navigatieblok staat.



Pas het script als volgt aan.

```
$(function() {
    $('#scrollNav a').click(function(e){
        e.preventDefault();
    });
});
```

De callback functie van het click-event geeft het event zelf als parameter terug (aangeduid met de parameter e).
De methode `e.preventDefault()` zorgt ervoor dat we de normale werking van de links uitschakelen. In de volgende stap wordt de link immers vervangen door een scroll-effect.

Bewaar de pagina en test de links in een browser. (Geen enkele link werkt nog.)
Voeg nu de animatie toe.

```
$(function() {
    $('#scrollNav a').click(function(e){
        e.preventDefault();
        var anker = $(this).attr('href');
        var ankerOffset = $(anker).offset().top;
        $('html, body').animate({scrollTop:ankerOffset}, 500);
    });
});
```

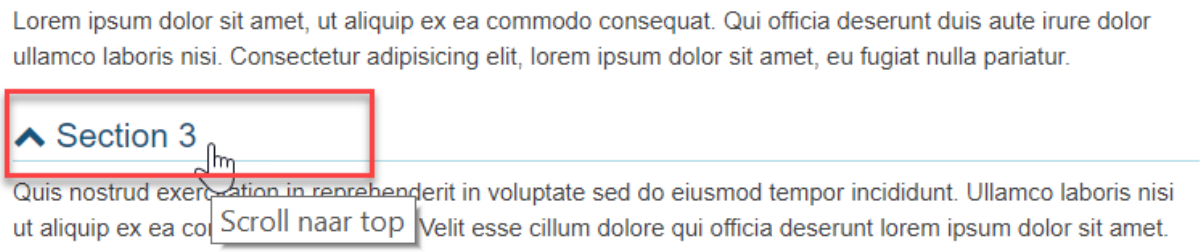
Eerst lezen we de waarde van het link-attribuut href uit en plaatsen deze in de variabele anker. De variabele ankerOffset bevat de afstand van het gekozen anker ten opzichte van de browserrand.
De methode `offset()` geeft ons de afstand van de selector ten opzichte van het document. `offset()` is op zich een object met twee properties, namelijk `left` en `top`. Om de juiste afstand tot de bovenrand te kennen, gebruiken we dus `offset().top`.

Bewaar de pagina en test de animaties in een browser.
Plaats op de titels binnen de section-tag een link naar anker #top.

```
$(function() {
    ...
    $('section > h2').css('cursor', 'pointer')
        .attr('title', 'Scroll naar top')
        .click(function(){
            var topOffset = $('#top').offset().top;
            $('html, body').animate({scrollTop:topOffset}, 200);
    });
});
```

De selector `$('section > h2')` gaat op zoek naar elke h2-tag die zich op het eerste niveau binnen de section-tag bevindt. Om de gebruiker te tonen dat de h2-tag een link wordt, gaan we de cursor aanpassen met `.css('cursor', 'pointer')` en krijgt de link een tooltip via `.attr('title', 'Scroll naar top')`. Daarna volgt het click-event. De code achter het click-event is bijna identiek aan de code van de links uit voorgaande functie.

> Bewaar de pagina en test de links op de h2-tags in een browser.



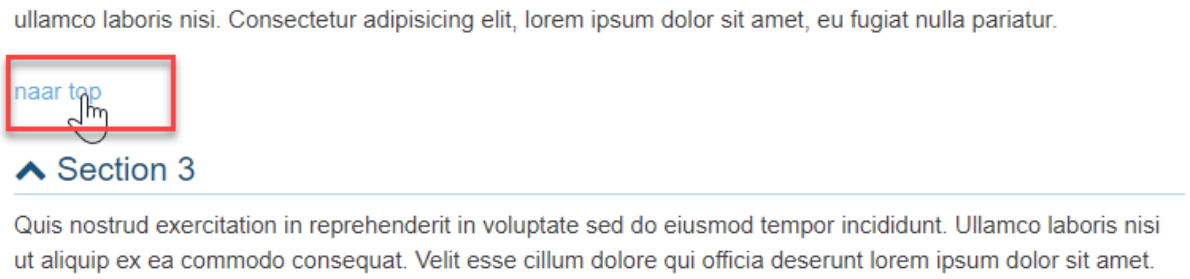
Als laatste gaan we nog een extra tekstlink toevoegen. Hiervoor gaan we extra HTML-code genereren die we dynamisch op het einde van elke sectie injecteren.

> Plaats op de titels aan het einde van elke section-tag een link naar anker #top.

```
$(function() {
    ...
    $('section').append('<p><a href="#top">naar top</a></p>');
});
```

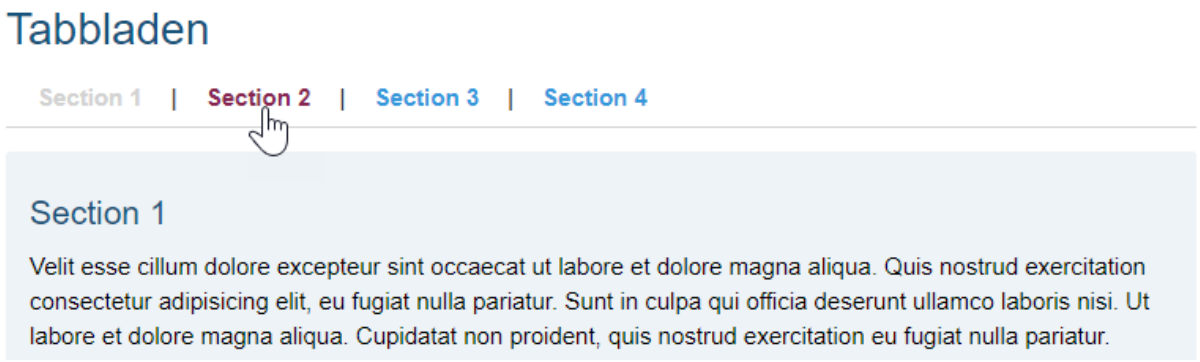
De methode `append('...')` voegt net voor de eindtag van section een nieuwe paragraaf met een link toe. Ook deze methode wordt later nog in detail besproken.

> Bewaar de pagina en test de geïnjecteerde links in een browser.



Toepassing 2: tabbladen

Tabbladen worden vaak gebruikt op pagina's die zeer veel informatie bevatten. U kan de informatie mooi in blokken onderverdelen en via verschillende tabs de juiste data tonen of verbergen.



Open **animaties/toep_tabs.html** en bekijk de HTML-code.

```
<ul class="tabNav">
    <li><a href="#!" class="active">Section 1</a></li>
    <li><a href="#!">Section 2</a></li>
    <li><a href="#!">Section 3</a></li>
    <li><a href="#!">Section 4</a></li>
</ul>
....
<section class="message-bar background-gray-light">
    <h2>Section 1</h2>
    <div>...</div>
</section>
<section class="message-bar background-gray-light">
    <h2>Section 2</h2>
    <div>...</div>
</section>
<section class="message-bar background-gray-light">
    <h2>Section 3</h2>
    <div>...</div>
</section>
<section class="message-bar background-gray-light">
    <h2>Section 4</h2>
    <div>...</div>
</section>
```

De navigatie is opgebouwd via een lijst met (dummy) links. Vanuit CSS worden de links horizontaal geplaatst en, afhankelijk van de toestand, krijgt de tab een andere achtergrondkleur.

In normale toestand is de achtergrondkleur van een tab lichtblauw en in de hover toestand is de achtergrond donkerrood.

Telkens men op een link klikt, komt de tab in zijn actieve toestand. Nu wordt de kleur grijs. De actieve toestand vindt u in de class .active.

Bij elke link hoort natuurlijk ook een datablok. Hiervoor gebruiken we een section-tag. De volgorde van de li-tags bepaalt welke section-tags zichtbaar/onzichtbaar worden.
De eerste link-tag toont de eerste section-tag, de tweede link toont de tweede section-tag, enz...

We gaan het script zo universeel mogelijk opbouwen. Als u later een extra link en een extra section-tag toevoegt, moet dit automatisch als tabblad functioneren.

Pas het script als volgt aan:

```
<style>
    main > section:not(:first-of-type) {
        display: none;
    }
    ...
</style>
```

Eerst gaan we, vanuit css, alle section-tags, behalve de eerste, verbergen.

Test het resultaat in een browser.
Voeg nu het click-event toe.

```
$(function() {
    $('.tabNav a').click(function(){
        var index = $('.tabNav a').index(this);
        $('.tabNav a').removeClass('active');
        $(this).addClass('active');
        $('section:visible').hide();
        $('section:eq(' + index + ')').show();
    });
});
```

De navigatie bevat voorlopig vier links. Om te achterhalen welke link er werd aangeklikt, maken we gebruik van de methode index(). De selector `$('.tabNav a')` komt viermaal voor op de pagina. Voor de eerste link is `$('.tabNav a').index(this)` gelijk aan `0`, voor de tweede link is dit 1, enz... De index van de geklikte link, bewaren we in de variabele index.

Vervolgens verwijderen we de reeds aanwezige class .active en voegen de class via `$(this).addClass('active')` opnieuw toe aan de geklikte link.

De variabele index hebben we nodig voor de section-tags. Eerst verbergen we alle section-tags om vervolgens de juiste section-tag te tonen. Voor de derde link wordt dit:
`$('section:eq(2)').show();`

Test het resultaat in een browser.

De tabnavigatie is in principe klaar, maar we gaan nog even een animatie toevoegen.

Vervang de twee laatste lijnen uit het click-event door deze code:

```
$(function() {
    $('.tabNav a').click(function(){
        var index = $('.tabNav a').index(this);
        $('.tabNav a').removeClass('active');
        $(this).addClass('active');
        $('section:visible').slideUp(function(){
            $('section:eq(' + index + ')').slideDown();
        });
    });
});
```

Met een slide-effect verdwijnt de zichtbare section-tag. Zodra het effect is afgerond, wordt de nieuwe data zichtbaar.

Test het resultaat in een browser.
Klik tweemaal na elkaar op hetzelfde tabblad.

De data verdwijnt en wordt vervolgens terug zichtbaar. We willen het slide-effect enkel als we op een ander tabblad klikken. Als we op het reeds actieve tabblad klikken, mag er helemaal niets gebeuren.

Plaats een if-voorwaarde rond het slide-effect.

```
$(function() {
    $('.tabNav a').click(function(){
        var index = $('.tabNav a').index(this);
        $('.tabNav a').removeClass('active');
        $(this).addClass('active');
        if ($('section:eq(' + index + ')').is(':hidden')) {
            $('section:visible').slideUp(function(){
                $('section:eq(' + index + ')').slideDown();
            });
        }
    });
});
```

Bewaar de pagina en test het resultaat in een browser.
Voeg ook eens een vijfde link en een vijfde section-tag toe.

DOM manipulatie

Het Document Object Model of kortweg DOM is een W3C-standaard die de structuur van een HTML-document beschrijft. Hierdoor wordt het mogelijk om via een scripttaal zowel de inhoud als de structuur en de stijl van een object te wijzigen.

In voorgaande hoofdstukken hebben we al enkele eigenschappen van een object aangepast. Denk maar aan de methodes `css()` en `addClass()`. In dit hoofdstuk gaan we nog een stap verder en gaan we eveneens objecten toevoegen, wijzigen, wissen en klonen.

Een volledig overzicht van alle DOM methodes vindt je op:
http://api.jquery.com/category/manipulation/

HTML-attributen

Elke HTML-tag heeft tal van attributen die we zeer eenvoudig kunnen wijzigen, wissen en toevoegen. Hiervoor gebruiken we de universele methodes attr() en removeAttr(). De meest gebruikte attributen (style, class en data-xxx) krijgen hun eigen methodes.

Open **dom/htmlAttributen.html** en bekijk de broncode.
Test de pagina in een browser.

attr(), removeAttr() en data()

Syntax:

- Getter (haal de waarde van een attribuut op).
  ```
  .attr(attributeName).
  ```
- Getter voor data-xxx (haal de waarde van een data-xxx attribuut op).
  ```
  .data(dataSuffix).
  ```
- Setter (geef het attribuut een nieuwe waarde).
  ```
  .attr(attributeName,value)
  .attr(attributes)
  .attr(attributeName,function(index,value))
  ```
- Wis een attribuut.
  ```
  .removeAttr(attributeName)
  ```

Het data-attribuut kan u op twee manieren opvragen, namelijk via attr('data-xxx') of via data('xxx'). Een data-attribuut toevoegen kan ENKEL via attr('data-xxx', value)!

In onderstaand voorbeeld gaan we met de eerste knop (#attr1) het href-attribuut van elke link binnen #sect1 kopiëren naar het title-attribuut van de link. De tweede knop (#attr2) verwijdert het title-attribuut van elke link.

```
$(function() {
    $('#attr1').click(function() {
        $('#sect1 a').each(function(index, element) {
            $(this).attr('title', $(this).attr('href'));
        );
    });
    $('#attr2').click(function() {
```

```
        $('#sect1 a').removeAttr('title');
    });
});
```

★ Vervang in het script lijn.

```
$(function() {
    $('#attr1').click(function() {
        $('#sect1 a').each(function(index, element) {
            $(this).filter('a[href^="http://"]').attr('title',
'Externe link naar: ' + $(this).attr('href') );
        );
    });
});
```

Nu krijgen enkel de uitgaande links het title-attribuut.
Een lokale link, zoals de link homepage, heeft geen title-attribuut.

Kijk ook eens naar het gemeenschappelijk script op alle pagina's binnen deze website (map _jq/les.js). Hier vindt u onder andere een script dat elke externe link op de pagina's automatisch in een nieuw browservenster opent.
$('a[href^="http://"]').filter('a:not([target])').attr('target','_blank');

css()

Deze methode wijzigt het style-attribuut. In hoofdstuk 2 wordt de methode css() al uitgebreid besproken.

- Getter (haal de waarde van een CSS-eigenschap op).
  `.css(propertyName).`
- Setter (geef de CSS-eigenschap een nieuwe waarde).
  `.css(propertyName,value)`
  `.css(properties)`
  `.css(propertyName,function(index,value))`
- Wis het style-attribuut (met removeAttr() !!!)
  `.removeAttr('style')`

Het voorbeeld op de pagina onderlijnt alle links binnen #sect2 en kleurt ze groen.

```
$('#sect2 a').css('text-decoration', 'underline').css('color',
'green');
// of via een map
$('#sect2 a').css({'text-decoration': 'underline', 'color':
'green'});
```

CSS attribuut: css() methode

- http://www.jquery.com/
- http://www.jquerymobile.com/
- http://api.jquery.com/
- homepage

addClass(), removeClass(), toggleClass() en hasClass()

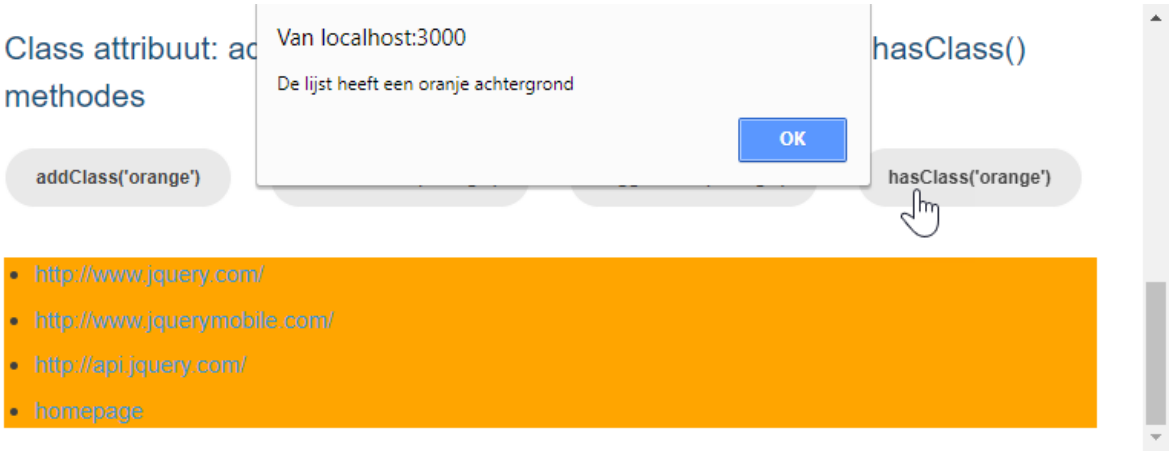
```
$('#orange1').click(function() {
    $('#sect3 ul').addClass('orange');
});
$('#orange2').click(function() {
    $('#sect3 ul').removeClass('orange');
});
$('#orange3').click(function() {
    $('#sect3 ul').toggleClass('orange');
});
$('#orange4').click(function() {
    if ($('#sect3 ul').hasClass('orange')) {
        alert('De lijst heeft een oranje achtergrond');
    } else {
        alert('De lijst heeft een witte achtergrond');
    }
});
```

De methode `addClass()` voegt één of meerdere CSS-classes toe aan een object. Meerdere classes worden gescheiden door een spatie. Bijvoorbeeld `addClass('class1 class2')`. Met `removeClass()` kan u één of meerdere classes verwijderen.

Met `toggleClass()` kan u een reeds toegekende class verwijderen of een nog niet toegekende class toevoegen.

De laatste methode, hasClass(), controleert of een bepaalde class reeds is toegekend aan een element of niet. Deze methode geeft `true` of `false` terug en kan u bijgevolg makkelijk verwerken in een voorwaardelijke instructie.

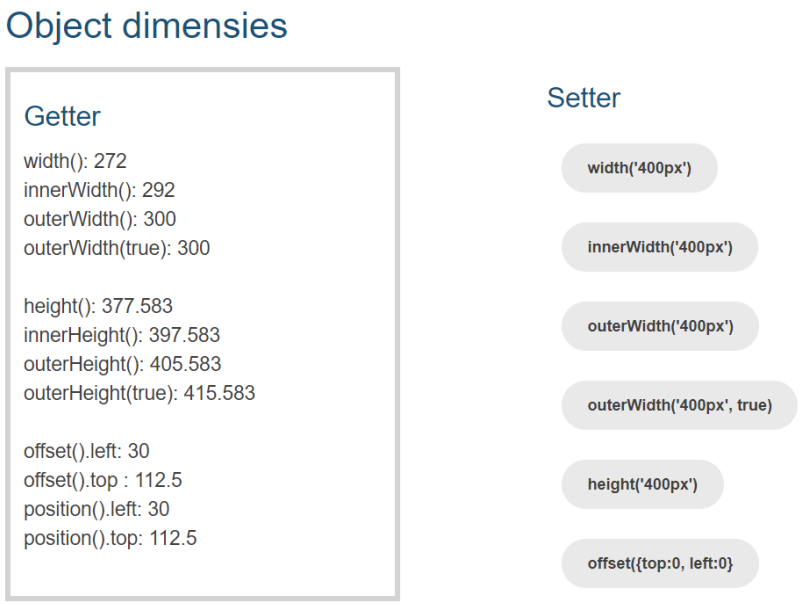
> Test de vier knoppen in een browser.



Object dimensies

jQuery kent enkele handige methodes om de afmetingen van een object op te vragen en zijn positie binnen de pagina te bepalen.

> Open **dom/dimensies.html** en bekijk de broncode.
> Test de pagina in een browser.



De acht laatste getallen zijn verschillend op Mac en op Windows. Beide sytemen gebruiken immers een verschillend lettertype! Bovenstaande afbeelding toont de Windows-dimensies.

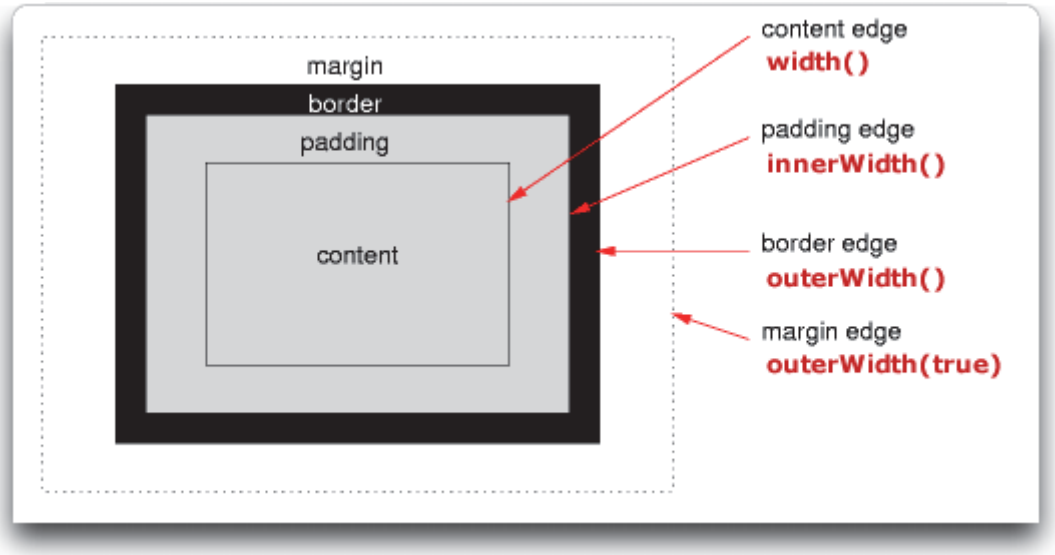
```
<style type="text/css">
#sect1 {
    width: 300px;
    float:left;
    border: solid 4px lightGrey;
    padding: 10px;
    margin: 5px 5px 10px 5px;
    cursor: text;
}
...
</style>
```

```
$('#sect1').on('mouseleave keyup', function(){
     toonEigenschappen();
});
$('#sect1').mouseleave();
$('#width').click(function() {
    $('#sect1').width('400px');
    toonEigenschappen();
});
$('#innerWidth').click(function() {
    $('#sect1').innerWidth('400px');
    toonEigenschappen();
});
$('#outerWidth').click(function() {
    $('#sect1').outerWidth('400px');
    toonEigenschappen();
});
$('#outerWidthTrue').click(function() {
    $('#sect1').outerWidth('400px', true);
    toonEigenschappen();
});
$('#height').click(function() {
    $('#sect1').height('400px').css('overflow', 'auto');
    toonEigenschappen();
});
$('#offset').click(function() {
    $('#sect1').offset({top:150, left:30});
    toonEigenschappen();
});
function toonEigenschappen(){
    sec = $('#sect1');
    eigenschappen = 'width(): '+sec.width()+'<br>';
    eigenschappen += 'innerWidth(): '+ sec.innerWidth() + '<br>';
    eigenschappen += 'outerWidth(): '+ sec.outerWidth() +
'<br><br>';
    eigenschappen += 'outerWidth(true): '+ sec.outerWidth(true) +
'<br>';
```

```
    eigenschappen += 'height(): '+ sec.height() + '<br>';
    eigenschappen += 'innerHeight(): '+ sec.innerHeight() + '<br>';
    eigenschappen += 'outerHeight(): '+ sec.outerHeight() + '<br>';
    eigenschappen += 'outerHeight(true):'+ sec.outerHeight(true) +
'<br>';
    eigenschappen += 'offset().left: '+ sec.offset().left + '<br>';
    eigenschappen += 'offset().top : '+ sec.offset().top + '<br>';
    eigenschappen += 'position().left: '+ sec.position().left +
'<br>';
    eigenschappen += 'position().top: '+ sec.position().top +'<br>';
    $('#sect1 p:first').html(eigenschappen);
}
```

Zowel de breedte als de hoogte van een element kunnen we bepalen aan de hand van vier methodes. Voor de breedte zijn dit:

- `width()`: de breedte van de box (500px).
- `innerWidth()`: breedte + padding (500px + 2x10px = 520px).
- `outerWidth()`: breedte + padding + rand (520px + 2x4px = 528px).
- `outerWidth(true)`: breedte + padding + rand + margin (528px + 2x5px = 538px).



De positie van de linkerbovenhoek van een object kunnen we als volgt bepalen:

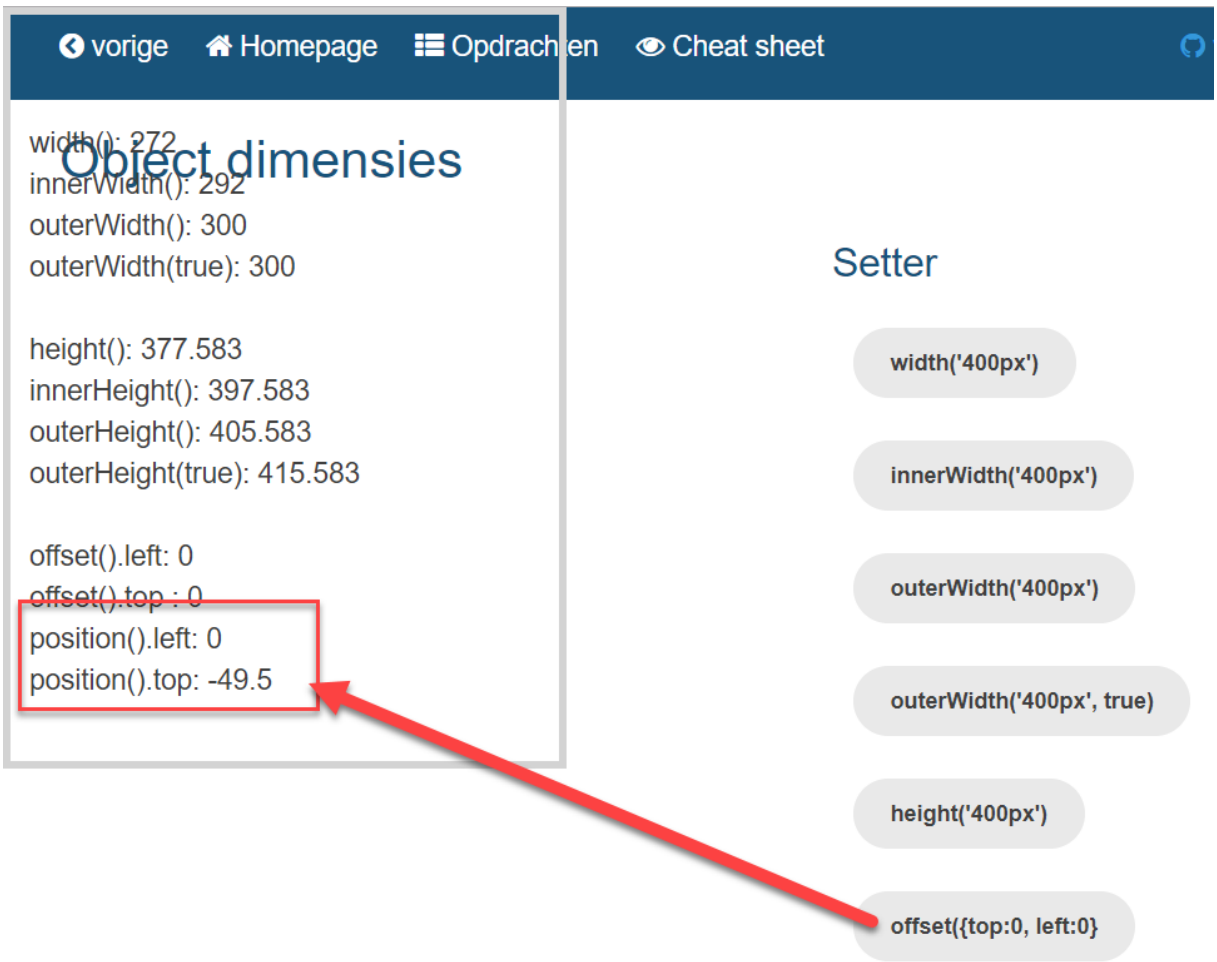
`offset().left` en `offset().top` geven respectievelijk de X- en Y-coördinaat van het object ten opzichte van de body-tag.

`position().left` en `position().top` geven de X- en Y-coördinaat ten opzichte van het dichtstbijzijnde parent element dat ZELF gepositioneerd staat.

In ons voorbeeld geeft `position()` de afstand ten opzichte van de body-tag.

> Voeg aan het document volgende stijlregel toe:
> `main {position:relative;}`
> Klik op de offset knop.

Je zal merken dat de waardes voor `offset()` niet wijzigen. De waardes voor `position()` worden nu herrekend naar de main-tag.



Enkel `width()`, `height()` en `offset()` kennen een setter-functie.

Elementen toevoegen

Binnen jQuery kunnen we op verschillende manieren nieuwe inhoud aan de pagina toevoegen. We kunnen zowel volledig nieuwe inhoud genereren en aan de pagina toevoegen, als reeds bestaande inhoud verplaatsen of dupliceren (klonen).

Binnen de selector

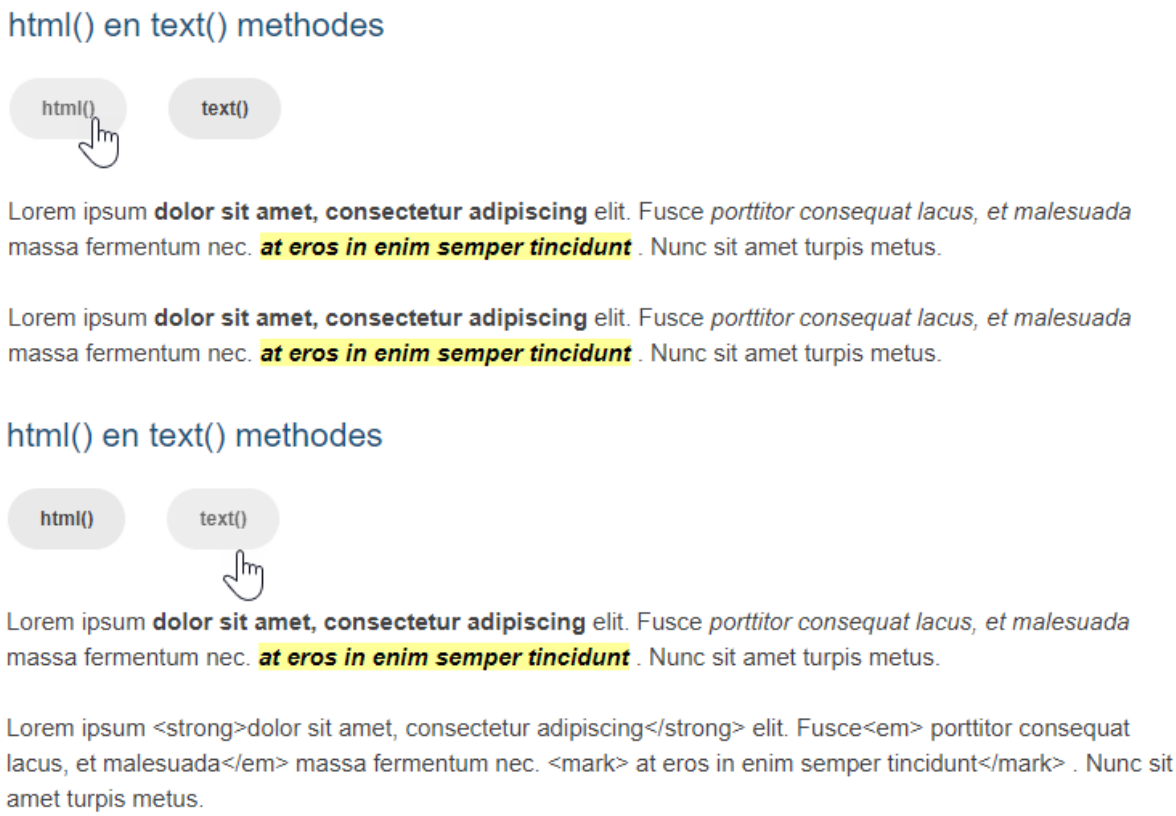
Eerst gaan we na, hoe we bestaande of nieuwe inhoud binnen een element toevoegen. Twee methodes werden in voorgaande hoofdstukken reeds herhaaldelijk gebruikt.

> Open **dom/inside.html** en bekijk de broncode.
> Test de pagina in een browser.

```
data = $('#p1').html();
$('#inside1').click(function() {
    $('#p2').html(data);
});
```

```
$('#inside2').click(function() {
    $('#p2').text(data);
});
```

Vooraan het script leest de methode `html()` de inhoud van #p1 uit en plaatst deze in de variabele data. Een klik op #inside1 kopieert de volledige inhoud, inclusief HTML-tags, van data naar #p2.
De methode `text()` daarentegen kopieert enkel de tekst, zonder HTML-opmaak, naar #p2.



Om nieuwe inhoud binnen een element toe te voegen, kunnen we vier methodes gebruiken: twee methodes om gegevens vooraan toe te voegen en twee methodes om de inhoud achteraan toe te voegen.

`append()` en `prepend()` worden altijd voorafgegaan door een selector. Binnen de methode komt de inhoud die we respectievelijk achteraan of vooraan de selector toevoegen.

```
$('#inside3').click(function() {
    $('#sect2 p.groen').append('<b> *** append *** </b>');
});
$('#inside4').click(function() {
    $('#sect2 p.groen').prepend('<b> *** prepend *** </b>');
});
```

append() en prepend() methodes

Met `appendTo()` en `prependTo()` draaien we de logica om. Eerst komt de inhoud die we wensen toe te voegen en binnen de methode staat de selector.

```
$('#inside5').click(function() {
    $('<b> *** appendTo *** </b>').appendTo('#sect3 p.groen');
});
$('#inside6').click(function() {
    $('<b> *** prependTo *** </b>').prependTo('#sect3 p.groen');
});
```

appendTo() en prependTo() methodes

`appendTo()` en `prependTo()` volgen op het eerste zicht een vreemde logica die we binnen jQuery niet gewend zijn. Het resultaat van append() en appendTo() is in dit voorbeeld net hetzelfde. Meestal kan u dus één van beide methodes kiezen, maar in uitzonderlijke gevallen geniet appendTo() toch de voorkeur. Hierover dadelijk meer.

Buiten de selector

In deze voorbeelden gaan we de nieuwe inhoud niet binnen, maar net voor of net achter de geselecteerde selector plaatsen.

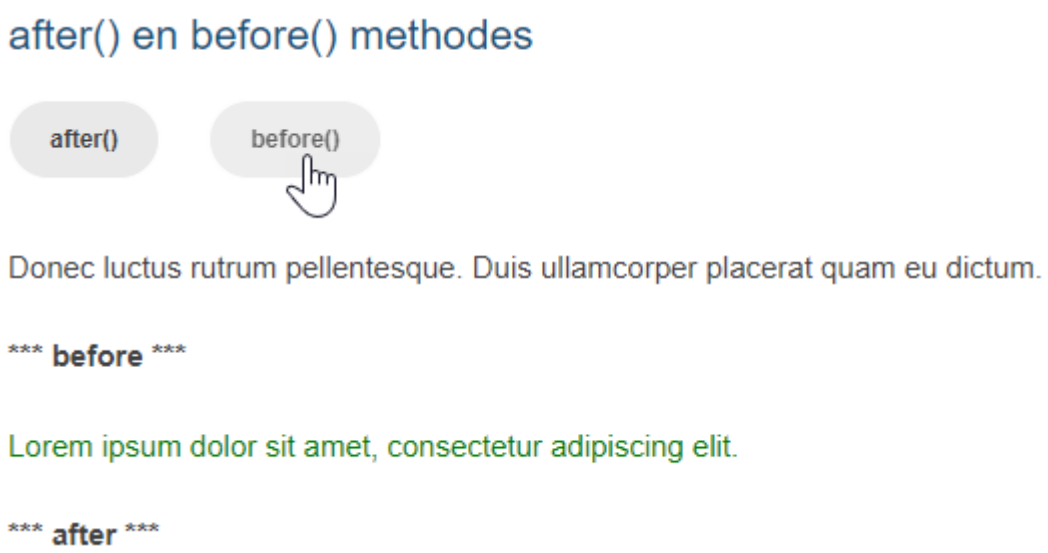
> Open **dom/outside.html** en bekijk de broncode.
> Test de pagina in een browser.

Ook hier beschikken we weer over vier methodes: twee methodes met de selector vooraan en twee methodes met de selector binnen de methode.

`after()/insertAfter()` en `before()/insertBefore()` plaatsen de nieuwe inhoud respectievelijk achter of voor de selectie.

```
$('#outside1').click(function() {
    $('#sect1 p.groen').after('<p><b>*** after ***</b></p>');
});
$('#outside2').click(function() {
```

```
    $('#sect1 p.groen').before('<p><b>*** before ***</b></p>');
});
```




```
$('#outside3').click(function() {
    $('<p><b>*** insertAfter ***</b></p>')
        .insertAfter('#sect2 p.groen');
});
$('#outside4').click(function() {
    $('<p><b>*** insertBefore ***</b></p>')
        .insertBefore('#sect2 p.groen');
});
```

Rond de selector

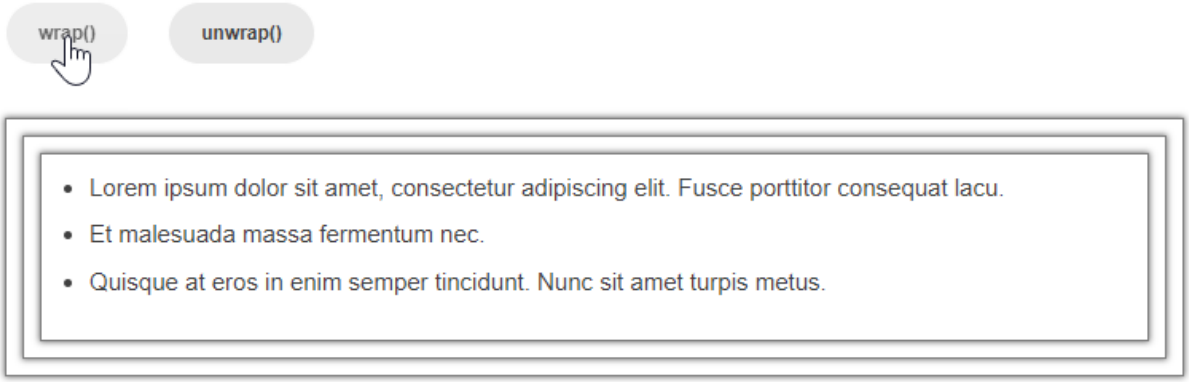
Hier volgen enkele methodes waarbij we nieuwe inhoud (meestal extra tags) rond een selectie plaatsen.

> Open **dom/around.html** en bekijk de broncode.
> Test de pagina in een browser.

In onderstaand voorbeeld plaatst de methode wrap() een div-tag rond #lijst1. Telkens u op de knop klikt, wordt een nieuwe div-tag toegevoegd. Met unwrap() kan u de PARENT-tag terug verwijderen. Klik maar eens enkele keren op unwrap() en u zal merken dat de volledige layout verdwijnt.

```
$('#around1').click(function() {
    $('#lijst1').wrap('<div class="drop-shadow"></div>');
});
$('#around2').click(function() {
    $('#lijst1').unwrap();
});
```

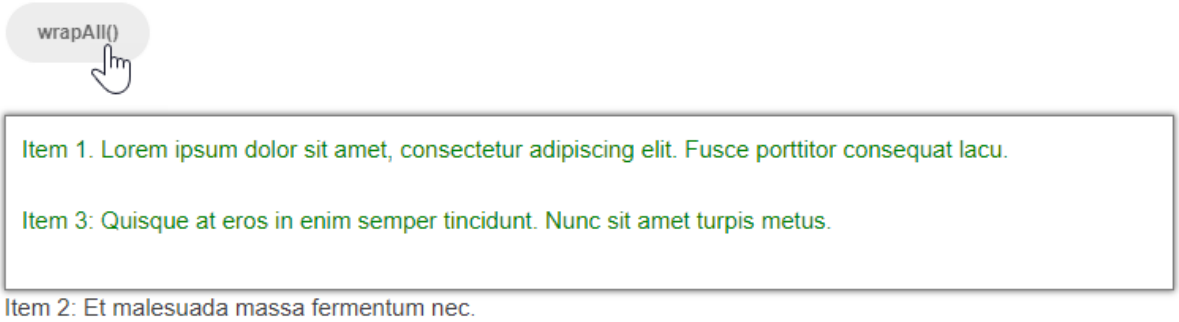
wrap() en unwrap() methodes



De tweede section-tag bevat verschillende p-tags waarvan twee p-tags met de class .groen. De methode `wrapAll()` brengt eerst alle p-tags met class .groen samen en plaatst hier vervolgens in zijn geheel een div-tag rond. De methode `wrap()` zal rond elke p-tag apart een div-tag plaatsen.

```
$('#around3').click(function() {
    $('#sect2 p.groen').wrapAll('<div class="drop-shadow"></div>');
});
```

wrapAll() methode



De HTML-code wijzigt van:

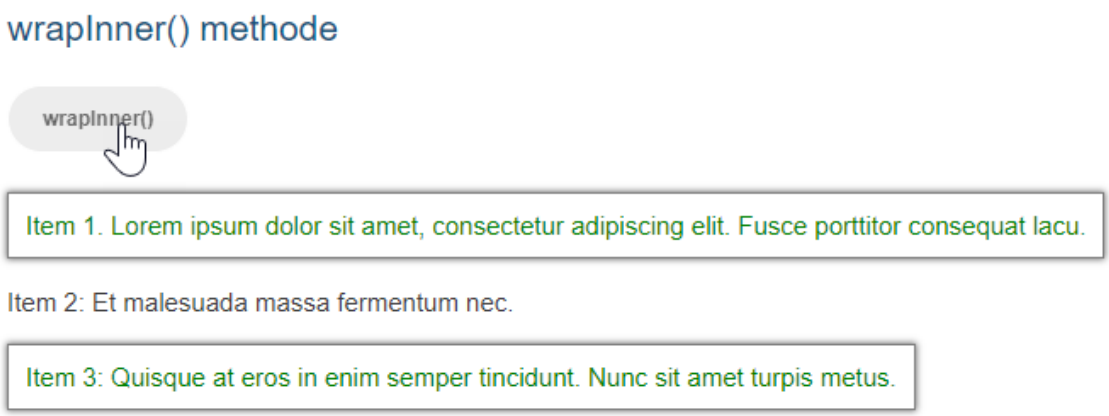
```
...
<p class="groen">Item 1...</p>
<p>Item 2...</p>
<p class="groen">Item 3...</p>
...
```

Naar:

```
...
<div class="drop-shadow">
    <p class="groen">Item 1...</p>
    <p class="groen">Item 3...</p>
</div>
<p>Item 2...</p>
...
```

Waar `wrap()` een nieuwe tag rond de selectie plaatst, plaatst `wrapInner()` een nieuwe tag binnen de selectie.

```
$('#around4').click(function() {
    $('#sect3 p.groen').wrapInner('<span class="drop-
shadow"></span>');
});
```



In bovenstaand voorbeeld wijzigt de code.

- van: `<p class="groen">Item 1...</p>`
- naar: `<p class="groen"><span class="drop-shadow">Item 1...</span></p>`

before() vs insertBefore()

We komen nog even terug op het verschil tussen `before()/insertBefore()`.

> Open **dom/before_insertBefore.html** en bekijk de broncode.
> Test de pagina in een browser.

```
$('#sect1 p:first').before('<img
src="../assets/fotos/thumb_tulp.jpg">');
$('<img src="../assets/fotos/thumb_tulp.jpg">').insertBefore('#sect2
p:first');
```

Aan de eerste paragraaf van elke section-tag wordt een afbeelding toegevoegd. Voorlopig nog niets aan de hand. Beide methodes geven immers hetzelfde resultaat.

Stel dat we via de class .rood een kader rond de afbeelding plaatsen. In principe kunnen we de klasse natuurlijk rechtstreeks op de img-tag plaatsen en dan zien we ook geen verschil tussen beide methodes. Indien we de klasse echter vanuit jQuery toevoegen, dan is er wel degelijk een verschil.

> Pas de code als volgt aan:

```
$('#sect1 p:first').addClass('rood').before('<img
src="../assets/fotos/thumb_tulp.jpg">');
```

```
$('<img
src="../assets/fotos/thumb_tulp.jpg">').addClass('rood').insertBefor
e('#sect2 p:first');
```

Binnen de tweede section-tag krijgt de afbeelding een rode kader, maar binnen de eerste section-tag komt het kader rond de p-tag te staan, binnen een ketting vanaf de selector. Aan de selector wordt een klasse toegevoegd en vervolgens wordt de nieuwe code toegevoegd.

De structuur van onze ketting is namelijk:

- kader op p-tag: $(selector).addClass(...).before(nieuweCode);
- kader op foto: $(nieuweCode).addClass(...).insertBefore(selector);

Ook `addClass()` en `before()` van plaats wisselen heeft totaal geen zin. Beide geven immers net hetzelfde resultaat.

- kader op p-tag: `$(selector).addClass(...).before(nieuweCode);`
- kader op p-tag: `$(selector).before(nieuweCode).addClass(...);`

Indien u toch met before() werkt, zal u de code moeten opsplitsen over twee lijnen. Onderstaande code geeft wel het gewenste resultaat:

```
$('#sect1 p:first').before('<img
src="../assets/fotos/thumb_tulp.jpg">');
$('#sect1 img:first').addClass('rood');
```

before()

Ut enim ad minim veniam, duis aute irure dolor velit esse cillum dolore. In reprehenderit in voluptate sed do eiusmod tempor incididunt excepteur sint occaecat. Cupidatat non proident, consectetur adipisicing elit, lorem ipsum dolor sit amet. Ut enim ad minim veniam, eu fugiat nulla pariatur. Mollit anim id est laborum.

Sed do eiusmod tempor incididunt qui officia deserunt quis nostrud exercitation. In reprehenderit in voluptate ut labore et dolore magna aliqua. Eu fugiat nulla pariatur.

insertBefore()

Lorem ipsum dolor sit amet, velit esse cillum dolore consectetur adipisicing elit. Duis aute irure dolor ut enim ad minim veniam, mollit anim id est laborum. Ut aliquip ex ea commodo consequat. Velit esse cillum dolore sunt in culpa quis nostrud exercitation.

In reprehenderit in voluptate excepteur sint occaecat duis aute irure dolor. Ut enim ad minim veniam, ut labore et dolore magna aliqua. Cupidatat non proident, sed do eiusmod tempor incididunt. Consectetur adipisicing elit, lorem ipsum dolor sit amet, ut labore et dolore magna aliqua. In reprehenderit in voluptate cupidatat non proident, excepteur sint occaecat.

Besluit:

- Wilt je enkel nieuwe inhoud toevoegen, dan kanjeu zowel before() als insertBefore() gebruiken.
- Indien je, VOOR het toevoegen, nog extra jQuery methodes toepast op de nieuwe inhoud, gebruik dan STEEDS insertBefore().

> Dezelfde logica geldt natuurlijk ook voor append()/appendTo(), prepend()/prependTo(), after()/insertAfter(), before()/insertBefore() en replaceWith()/replaceAll().
> De laatst genoemde methodes komen dadelijk aan bod.

Elementen verwijderen

> Open **dom/remove.html** en bekijk de broncode.
> Test de pagina in een browser.

```
$('#remove1').click(function() {
    $('#lijst1 b').empty();
});
$('#remove2').click(function() {
    $('#lijst2 b').remove();
});
$('#remove3').click(function() {
    var lijst3 = $('#lijst3').detach();
    lijst3.insertAfter('h1');
});
```

De eerste twee methodes geven op het eerste zicht net hetzelfde resultaat, maar schijn bedriegt.

empty() remove() detach()

lijst1

- Lorem ipsum adipiscing elit. Fusce porttitor consequat lacu.
- Et malesuada massa fermentum nec.
- Quisque at eros in . Nunc sit amet turpis metus.

lijst2

- Lorem ipsum adipiscing elit. Fusce porttitor consequat lacu.
- Et malesuada massa fermentum nec.
- Quisque at eros in . Nunc sit amet turpis metus.

De methode `empty()` maakt enkel de b-tags binnen #lijst1 leeg. De b-tags zelf, blijven in de code aanwezig. De methode `remove()` verwijdert de inhoud samen met de b-tags.

De methode `detach()` verwijdert de selectie uit het beeld, maar niet uit het geheugen. Omdat de inhoud niet echt verdwenen is, kunnen we deze makkelijk verplaatsen. U kan dit best vergelijken met een knip-en-plakoperatie. In ons voorbeeld knippen we #lijst3 uit het document en plakken deze vervolgens net achter de h1-tag.

Elementen verwijderen

- Lorem ipsum **dolor sit amet, consectetur** adipiscing elit. Fusce porttitor consequat lacu.
- Et malesuada massa fermentum nec.
- Quisque at eros in **enim semper tincidunt**. Nunc sit amet turpis metus.

empty(), remove() en detach() methode

empty() remove() detach()

lijst1

- Lorem ipsum adipiscing elit. Fusce porttitor consequat lacu.

Elementen vervangen

Open **dom/replace.html** en bekijk de broncode.
Test de pagina in een browser.

```
$('#replace1').click(function() {
    $('#lijst1 li.groen').replaceWith('<li><b>Cras augue</b></li>');
});
$('#replace2').click(function() {
    $('<li><b>Cras augue</b></li>').replaceAll('#lijst2 li.groen');
});
```

Met beide methodes kan u een element volledig vervangen. Bij `replaceWith()` staat de selector vooraan. Bij `replaceAll()` staat de vervangende code vooraan. In beide voorbeelden wordt elke li-tag met class .groen volledig door een nieuwe li-tag vervangen.

Elementen klonen

Stel dat we vanuit AJAX een aantal tweets inlezen en deze op onze pagina integreren. Elke tweet krijgt een gelijkaardige lay-out. Dit kunnen we perfect met de methodes die we in dit hoofdstuk hebben besproken. Echter de code die we moeten toevoegen is nogal uitgebreid. Waarschijnlijk gebruiken we een section-tag met daarin een hx-tag, een p-tag en nog een link. Stel dat we dit voor x tweets moeten herhalen. U merkt het al. Dit is een zeer omslachtige en tijdrovende bezigheid.

Voor dit soort toepassingen gebruiken we dus beter een sjabloon dat reeds de volledige structuur en lay-out bevat. Dit sjabloon gaan we dan met de methode clone() x maal kopiëren en plakken (klonen). Daarna hoeven we enkel nog de inhoud aan te passen en de toepassing is klaar.

> Open **dom/clone.html** en bekijk de broncode.
> Test de pagina in een browser.

Zodra u de pagina in een browser opent, verschijnt de tijd in de span-tag binnen de lijst. Klik op het listitem en de tijd verschijnt in een alertbox.

```
<ul>
    <li class="temp">Het is nu: <span class="tijd"></span></li>
</ul>
...
<script>
var uur = tijd();
$('.tijd').text(uur);
...
$('main li').click(function() {
    alert($(this).text());
});
...
function tijd(){
    d = new Date();
    uur = d.getHours()+'h '+d.getMinutes()+'min
'+d.getSeconds()+'sec';
    return uur;
}
</script>
```

> Klik enkele keren op de knop nieuwe tijd.

```
$('#clone').click(function() {
    uur = tijd();
    $('main li.temp').clone().removeClass('temp').insertBefore('main
li:first');
});
```

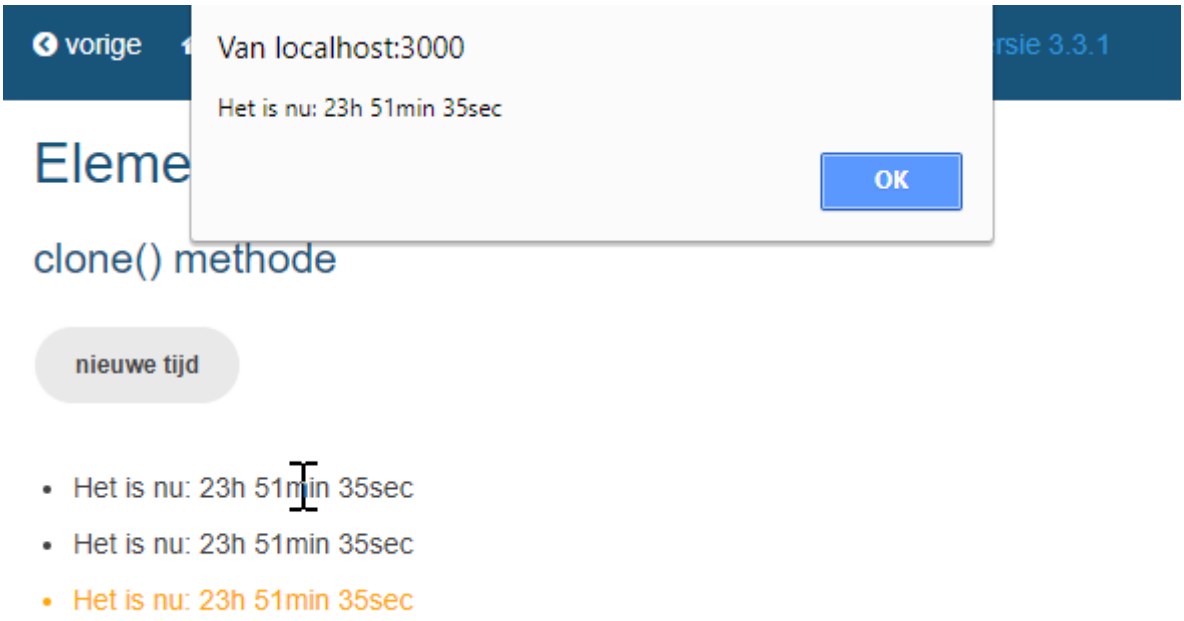
Het listitem met class .temp wordt gekloond. Vervolgens wordt in het gekloonde item de class .temp verwijderd. De nieuwe code wordt vooraan de lijst toegevoegd.

Telkens u op de knop klikt, verschijnt er een perfecte kopie van het laatste item uit de lijst. We klonen voorlopig enkel de li-tag. Klik maar eens op een gegenereerd item. De alertbox verschijnt enkel bij het laatste item, niet op de gekloonde items.

> Pas de code als volgt aan en test het resultaat in een browser:

```
$('#clone').click(function() {
    uur = tijd();
    $('main
li.temp').clone(true).removeClass('temp').insertBefore('main
li:first');
});
```

Met de parameter true gaat u zowel de li-tag als het gekoppelde event dupliceren. Vanaf nu verschijnt bij elk item een alertbox.

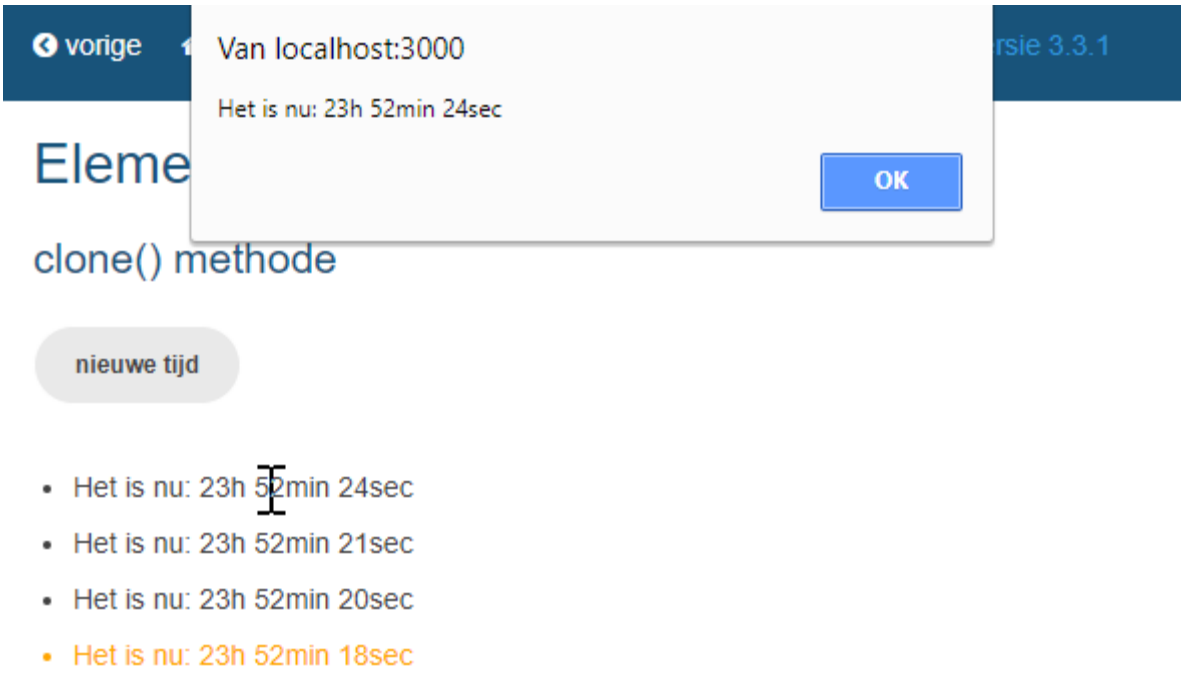


Als laatste gaan we niet enkel het item met zijn event handlers klonen, maar we gaan eveneens de datum opnieuw berekenen.

Pas de code als volgt aan en test het resultaat in een browser:

```
$('#clone').click(function() {
    uur = tijd();
    $('main
li.temp').clone(true).removeClass('temp').find('.tijd').text(uur).en
d().insertBefore('main li:first');
});
```

Voor we de gekloonde gegevens toevoegen, gaan we binnen de selectie op zoek naar span.tijd en passen de inhoud aan. De variabele uur verwijst naar de functie `tijd()`. Daarna keren we via `end()` terug naar de eerste selectie (li.temp) en voegen nu pas de nieuwe code vooraan de lijst toe. Ziehier het resultaat:



Toepassing 1: Lightbox

Tabbladen worden vaak gebruikt op pagina's die zeer veel informatie bevatten. U kan de informatie mooi in blokken onderverdelen en via verschillende tabs de juiste data tonen of verbergen.

Open **dom/toep_lightbox.html**

Elke thumbnail opent een grote afbeelding in hetzelfde browservenster. Ziehier de HTML-code voor één van de afbeeldingen:

```
<p>
<a class="lightbox" href="fotos/anjer.jpg">
<img src="fotos/thumb_anjer.jpg" width="100" height="100"
alt="anjer">
</a>
...
</p>
```

Pas het script als volgt aan:

```
$('a.lightbox').click(function(e){
e.preventDefault();
$('<div id="overlay"></div>').fadeIn('slow').appendTo('body');
});
```

`e.preventDefault()` blokkeert de links op de thumbnails. Vervolgens gaan we een extra div-tag met id *#overlay* aan de body-tag toevoegen.

De overlay eigenschappen staan volledig in CSS beschreven.

```
<style>
#overlay {
width: 100%;
height: 100%;
position: absolute;
top: 0;
left: 0;
z-index: 100;
background: rgba(255,255,255,.7) url(fotos/pixel.png) no-repeat
center center;
}
</style>
```

div#overlay heeft een breedte en hoogte van 100%. De achtergrondkleur is volledig wit, maar 70% transparant. De div-tag staat absoluut gepositioneerd en heeft een z-index van 100. De z-index zorgt ervoor dat de div-tag boven alle andere elementen uitstijgt en het volledig scherm met een witte, deels transparante, waas bedekt.

> Klik op één van de afbeeldingen.

De overlay komt tevoorschijn, maar we kunnen hem nog niet laten verdwijnen.

> Voeg onderstaande code toe en bekijk het resultaat in een browser.

```
$('a.lightbox').click(function(e){
    e.preventDefault();
    $('<div id="overlay"></div>').fadeIn('slow').appendTo('body');
});
$('body').on('click', '#overlay', function(){
    $('#overlay').fadeOut('slow', function(){
        $(this).remove();
        $('body').css('overflow', 'auto');
    });
});
```

> Vervang de twee laatste lijnen uit het click-event door deze code:

```
$(function() {
    $('.tabNav a').click(function(){
        var index = $('.tabNav a').index(this);
        $('.tabNav a').removeClass('active');
        $(this).addClass('active');
        $('section:visible').slideUp(function(){
            $('section:eq(' + index + ')').slideDown();
        });
    });
});
```

Door op div#overlay te klikken verdwijnt de div-tag langzaam uit beeld. Na de animatie wordt de div-tag volledig verwijderd via `remove()`.

Omdat `div#overlay` een dynamisch gegenereerd element is, kunnen we het click-event niet rechtstreeks aanspreken via `$('#overlay').click(...)`.

Als laatste gaan we de vergrote foto binnen het document openen. Hiervoor gaat u het click-event op de thumbnails vervolledigen.

Voeg volgende code toe:

```
$('a.lightbox').click(function(e){ e.preventDefault();
    e.preventDefault();
    $('<div id="overlay"></div>').fadeIn('slow').appendTo('body');
    var fotoURL = $(this).attr('href');
    $('#overlay').css('background-image', 'url(' + fotoURL + ')');
    $('body').css('overflow', 'hidden'); // Verberg scrollbars
});
```

Het pad naar de grote afbeelding vinden we in het href-attribuut van de link. Dit pad bewaren we in de variabele *fotoURL*. Vervolgens vervangen we de achtergrondafbeelding (de preloader) van de overlay door de foto uit *fotoURL*.

De laatste codelijn zorgt ervoor dat eventuele scrollbalken op het browservenster verdwijnen.

Inleiding tot AJAX

Alle moderne websites maken tegenwoordig gebruik van AJAX. Denk maar aan Gmail, Google Drive, Facebook en Twitter.

In tegenstelling tot een klassieke webpagina worden bij AJAX-gestuurde pagina's delen van de pagina ververst zonder de volledige webpagina opnieuw in te laden. De pagina's worden hierdoor veel interactiever en intuïtiever. De gebruiker krijgt meer het gevoel dat hij in een desktopapplicatie werkt dan op een website. Websites die hoofdzakelijk gebruik maken van AJAX noemt men ook wel Rich Internet Applicaties of RIA's.

AJAX is geen technologie op zich, maar een algemene term voor het ontwerpen van interactieve pagina's waarbij gegevens uit een extern bestand worden opgehaald en vervolgens dynamisch worden getoond. Het grote voordeel van AJAX t.o.v. uitsluitend klassieke webpagina's is dat alle gegevens dynamisch in de browser worden geladen. Alle interactiviteit (sorteren, filteren, ...) gebeurt dus volledig binnen de browser zonder dat er een nieuwe connectie met de server nodig is.

De communicatie binnen een klassieke website gebeurt als volgt:

- De browser laadt een pagina.
- De gebruiker klikt op een link.
- De browser vraagt een volledig nieuwe pagina op.
- De webserver stuurt de nieuwe pagina naar de browser.
- De browser toont de nieuwe pagina.

De communicatie met een AJAX-pagina verloopt als volgt:

- De browser laadt een pagina.
- De gebruiker klikt op een link.
- De JavaScript engine verwerkt de aanvraag en stuurt de aanvraag in de achtergrond door naar een externe pagina.
- De JavaScript engine ontvangt de gegevens en past het DOM aan.
- De browser toont de nieuwe gegevens en hoeft daarvoor niet de volledige pagina te herladen.

Omdat de browser minder met de webserver communiceert en omdat de verwerking volledig lokaal gebeurt, krijgen we een zeer snelle respons en een snelle pagina update.

In dit hoofdstuk geven we een korte inleiding tot AJAX en verduidelijken we enkele begrippen. De oefeningen komen in de twee volgende hoofdstukken aan bod.

Een volledig overzicht van alle AJAX-methodes vind je hier: http://api.jquery.com/category/ajax/

Historie

De techniek om asynchroon gegevens op te halen, bestaat al meer dan 20 jaar. In 1998 ontwikkelde Microsoft een systeem om via een ActiveX control gegevens in de achtergrond op te halen. Enkele jaren later implementeerden alle andere browsers een gelijkaardig principe, maar nu gebaseerd op het gestandaardiseerde XMLHttpRequest protocol (kortweg XHR).

Google was het eerste bedrijf dat XHR op grote schaal in zijn toepassingen ging verwerken. In de beginjaren kwamen de externe gegevens uitsluitend uit een XML-document. Zo is de term AJAX ontstaan. AJAX was het acroniem voor Asynchronous JavaScript And XML.

Ondertussen is de omschrijving van AJAX al achterhaald. Via AJAX kunnen we niet enkel XML importeren, maar eveneens tekst, HTML, JSON en JavaScript.

Het is niet zo evident om op een universele manier AJAX te verwerken. De ene browser gebruikt XHR, de andere ActiveX. Gelukkig hoeven wij ons hier geen zorgen over te maken. jQuery zorgt immers voor een correcte verwerking in de verschillende browsers.

Wat is XML?

XML is het acroniem voor eXtensible Markup Language. Net zoals HTML, beschrijft XML de datastructuur van gegevens, niet de opmaak. Een XML-document is, net zoals HTML, opgebouwd met tags (nodes in het XML jargon) en attributen.

Neem als voorbeeld een adresboek in de vorm van een XML-document.

XML begint steeds met de header of proloog. De proloog bevat informatie over de document encoding en de XML versie. Dit is ondermeer belangrijk voor het programma dat de XML-code gaat verwerken (de XML-parser). Na de proloog volgt de rootnode adresboek.

Ons adresboek bevat twee personen. Elke persoon staat beschreven in een eigen adresnode met als attribuut een uniek id. Binnen de adresnode komen de childnodes: voornaam, naam en email.

```
<?xml version="1.0" encoding="UTF-8"?>
<adresboek>
    <adres id="1">
        <voornaam>Lorem</voornaam>
        <naam>Ipsum</naam>
        <email>lorem.ipsum@example.com</email>
    </adres>
    <adres id="2">
        <voornaam>Morbi</voornaam>
        <naam>Dui</naam>
        <email>morbi.dui@example.com</email>
    </adres>
    <adres id="3"> ... </adres>
</adresboek>
```

Een goed gestructureerd XML-document is zelfbeschrijvend. Dit wil zeggen dat nodenamen iets vertellen over de inhoud van de node.

Als het document voldoet aan alle syntaxregels van XML, noemt men dit well-formed (of goed gevormd). Een well-formed document kan door de meeste parsers, zoals een webbrowser, correct verwerkt worden.

Een well-formed document kan u makkelijk in Chrome of in Firefox testen. Open het XML-document in de browsers. Als de boomstructuur verschijnt, is het document well-formed en zijn de gegevens te verwerken via jQuery.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<adresboek>
   <script id="tinyhippos-injected"/>
  ▼<adres id="1">
     <voornaam>Lorem</voornaam>
     <naam>Ipsum</naam>
     <email>lorem.ipsum@example.com</email>
   </adres>
  ▼<adres id="2">
     <voornaam>Morbi</voornaam>
     <naam>Dui</naam>
     <email>morbi.dui@example.com</email>
   </adres>
   <adres id="3">...</adres>
 </adresboek>
```

well-formed XML

Indien het document fouten bevat, is het niet well-formed en krijgt u dit te zien:

This page contains the following errors:

```
error on line 7 at column 13: Opening and ending tag mismatch: email line 0 and adres
```

Below is a rendering of the page up to the first error.

Lorem Ipsum

Eindnode email ontbreekt

In een XML-document dat uitsluitend voor eigen gebruik is ontworpen, kan u de nodenamen vrij kiezen. In een universeel, gestandaardiseerd XML-document zoals RSS, ATOM en XHTML ligt de naamgeving vast.

Om de correctheid van het document te controleren, maakt men gebruik van een DTD of van een XML Schema.

Een DTD of XML schema documenteert als het ware het XML-bestand. Hierin wordt ondermeer beschreven welke nodes in het XML-document moeten/mogen voorkomen en welke inhoud de nodes bevatten (tekst, enkel getallen, …). Aan de hand van dit controlebestand kan de parser de XML-nodes zowel op syntax als op inhoud valideren.

Een well-formed XML-bestand dat ook nog voldoet aan de bijbehorende DTD of Schema, noemt men een valid XML-bestand.

Wat is JSON?

JSON is het acroniem voor JavaScript Object Notation. JSON beschrijft, net zoals XML, de datastructuur van gegevens. JSON is een onderdeel van JavaScript, en is bijgevolg relatief eenvoudig in het DOM te verwerken.

De eenvoud van JSON heeft geleid tot een grote populariteit ervan, met name als een alternatief voor XML. Binnen JSON staan gegevens gestructureerd in de vorm van een JavaScriptobject of als een JavaScript array.

Ziehier een JSON equivalent van ons XML adresboek.

```
{
    "adresboek":[
        {
            "id":1,
            "voornaam":"Lorem",
            "naam":"Ipsum",
            "email":"lorem.ipsum@example.com"
        },
        {
            "id":2,
            "voornaam":"Morbi",
            "naam":"Dui",
            "email":"morbi.dui@example.com"
        },
        {
            ...
        }
    ]
}
```

Vergelijk de syntax van bovenstaand JSON object met de syntax van een object literal in paragraaf 2.11.4.

In JSON moeten alle namen en stringwaardes tussen dubbele aanhalingstekens staan, zoniet kan jQuery het bestand niet verwerken!

- Goed: "voornaam":"Lorem"
- Fout: voornaam : "Lorem"
- Fout: 'voornaam' : "Lorem"
- Fout: 'voornaam' : 'Lorem'

Requests filteren met GET en POST

Het is perfect mogelijk om gegevens (HTML, JSON, XML, ...) te verwerken uit een volledig statisch document. Het wordt natuurlijk nog interessanter indien we de gegevens dynamisch vanuit een database kunnen genereren. Vanuit een statische pagina is de response altijd hetzelfde. Op een dynamische pagina is de response afhankelijk van een filter of parameter. Het filteren gebeurt meestal vanuit een formulier.

Neem bijvoorbeeld de zoekfunctie van Google. De response is afhankelijk van de zoekterm die u in het formulier invult.

Zoals u weet, kan u een formulier verzenden via GET of via POST.

Formulier verzenden met de GET-methode

GET plakt alle formuliervelden achter de URL. Alle zoekmachines gebruiken GET.

Zoek via Google op jquery tutorials en bekijk de URL.

```
http://www.google.be/search?...&q=jquery+tutorials&.....
```

De structuur van de URL met een GET-request is als volgt:

```
action?naam1=waarde1&naam2=waarde2&naam3=waarde3
```

Action is de URL van de pagina die het formulier zal verwerken. Van elk formulierveld wordt zowel de naam van het veld als de waarde in het veld aan de URL toegevoegd. Alle velden zijn gescheiden door een &-teken.

Omdat GET de waarde of de inhoud van elk object zichtbaar maakt in de URL, kan u deze methode bijvoorbeeld niet gebruiken op een loginpagina. Het paswoord mag immers niet zichtbaar zijn in de URL. De lengte van de URL is ook beperkt tot enkele honderden karakters (browserafhankelijk).

Formulier verzenden met de POST-methode

De tweede methode, POST, geeft de gegevens via de header door. De gegevens zijn nu niet zichtbaar in de URL. Het aantal karakters dat kan worden doorgestuurd, is vrijwel onbeperkt.

Voor een AJAX request naar een statische pagina hebt u geen webserver nodig.

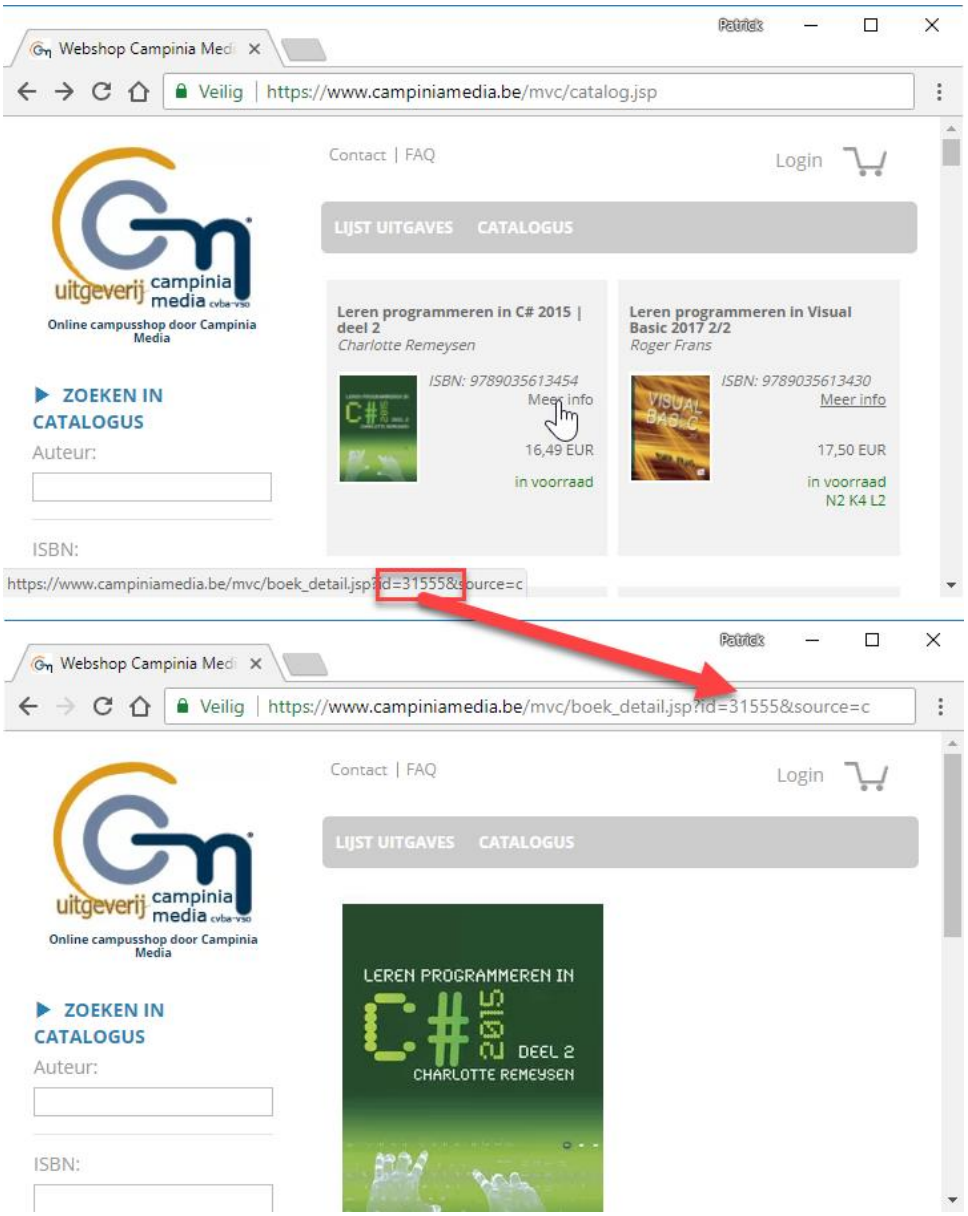
Voor een AJAX request met GET- en POST-variabelen in combinatie met een database hebt u altijd een webserver nodig. Formuliergegevens kan u dan enkel verwerken via een server-side scripttaal zoals PHP of ASP.NET. Hierdoor bent u wel verplicht om via een webserver te werken.

GET-methode zonder formulier

Omdat de GET-methode alle parameters achter de URL plaatst, is het perfect mogelijk om de parameters dadelijk in een link te verwerken. Dit wordt vaak gebruikt in een master/detail relatie.

De masterpagina toont een beknopt overzicht van alle items. Elk item heeft een link naar de detailpagina en stuurt in de URL een unieke identificatiecode mee.

Neem als voorbeeld de startpagina van Campinia Media.



Elke link "Lees verder" verwijst naar dezelfde detailpagina:

```
https://www.campiniamedia.be/mvc/boek_detail.jsp?id=xxxxx&source=c
```

De parameter id filtert de juiste gegevens uit de database en toont de gedetailleerde informatie over het gevraagde boek.

Zes soorten AJAX requests

De methode $.ajax() is een jQuery's low-level AJAX implementatie. Dit is tevens de meest uitgebreide, maar ook moeilijkste methode.

Gelukkig zijn er ook nog vijf afgeleide methodes met minder toeters en bellen, maar wel veel eenvoudiger te begrijpen en te gebruiken. Onderstaande tabel geeft een overzicht van de verschillende methodes met hun mogelijkheden en beperkingen.

	Moeilijkheidsgraad	GET request	POST request	Response				
				Tekst	HTML	Script	JSON(P)	XML
$(selector).**load**(url [, data][, callback])	*	x	x	x	x			
$.getScript(url [, callback])	**					x		
$.getJSON(url [, data][, callback])	**	x					x	
$.get(url [, data][, callback][, dataType])	**	x		x	x	x	x	x
$.post(url [, data][, callback][, dataType])	**		x	x	x	x	x	x
$.ajax(settings)	***	x	x	x	x	x	x	x

Zonder webserver kan u enkel koppelen met statische bestanden. Om gegevens uit een database te verwerken, zal u altijd een server-side script moeten gebruiken en bent u natuurlijk wel verplicht om een webserver te gebruiken.

Omdat niet iedereen vertrouwd is met server-side scripts (PHP, ASP.NET, ...) gaan we de oefeningen dadelijk opsplitsen over twee verschillende hoofdstukken.

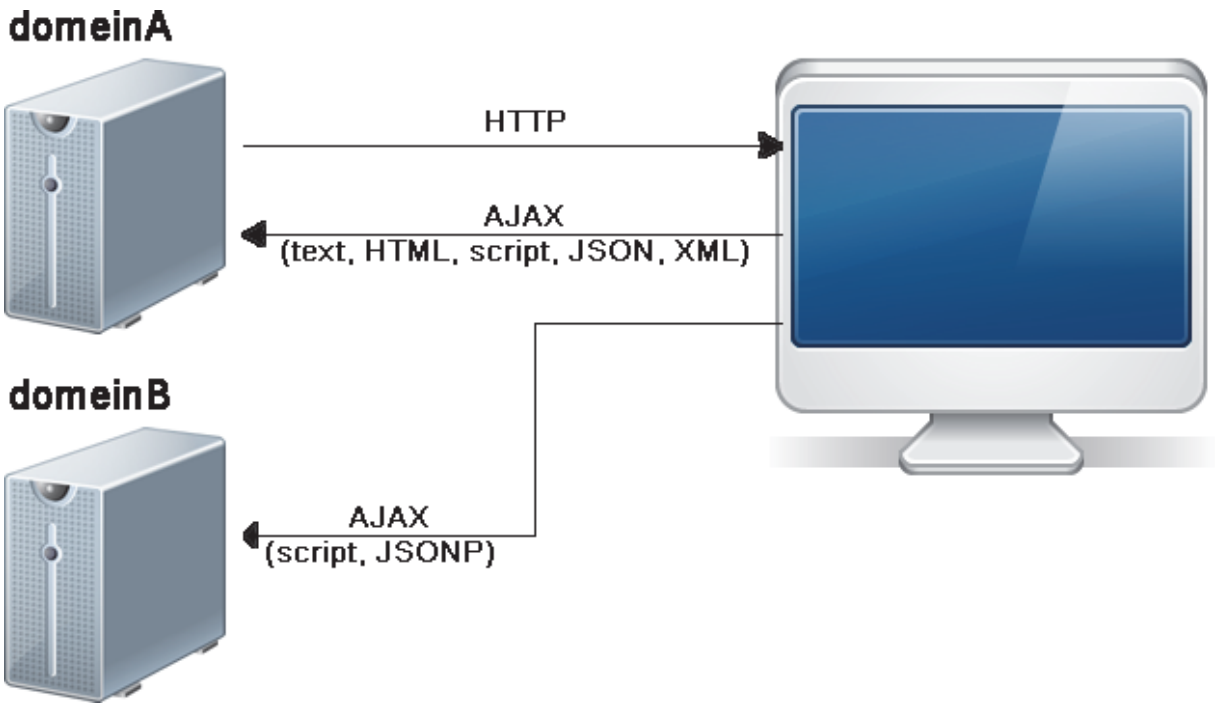
Same origin policy

Uit veiligheidsredenen bevatten alle browsers enkele beperkingen. Eén van deze beperkingen is de same origin policy. Of, met andere woorden, alle gegevens die u via AJAX ophaalt, moeten afkomstig zijn van hetzelfde domein. Hetzelfde domein betekent meer specifiek: dezelfde server, zelfde protocol, zelfde domeinnaam en dezelfde poort. Het is bijvoorbeeld niet mogelijk dat een webpagina op domeinA gegevens ophaalt uit domeinB.

Er zijn echter twee uitzonderingen. Het ophalen van externe JavaScripts en gegevens in JSONP formaat (let op de P achteraan) zijn wel toegestaan.

JSONP staat voor "JSON with Padding". Wanneer de webserver van domeinB zodanig staat geconfigureerd dat deze toelaat dat andere gebruikers zijn gegevens in JSON-formaat mogen ophalen, spreekt men van JSONP. De structuur van een JSON-bestand en van een JSONP-bestand is identiek.

Onderstaande afbeelding toont de AJAX mogelijkheden/beperkingen, rekening houdend met de same origin policy.

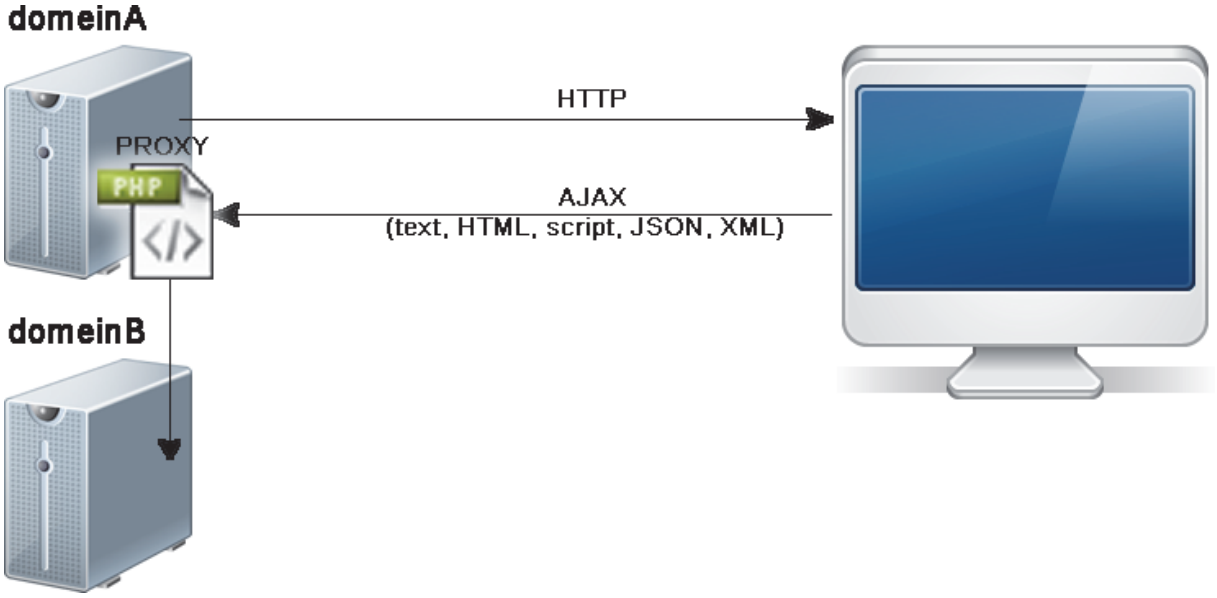


Same origin policy is een beperking van de browser. Ook indien u zonder webserver werkt (domeinA is dan de lokale computer) geldt bovenstaand schema.

Cross-site scripting

Indien u over een webserver beschikt, kan u de same origin policy makkelijk omzeilen. Op de webserver plaatst u een proxyscript. Dit proxyscript haalt de inhoud van een externe pagina (html, XML, JSON, ...) op en toont dit in zijn eigen pagina.

In plaats van in een AJAX request te verwijzen naar de externe pagina, verwijst u nu naar de proxypagina. Voor de browser lijkt het alsof de gegevens afkomstig zijn van het eigen domein. Op deze manier kunnen we dus alle externe gegevens perfect via AJAX verwerken. Hierover later meer.



Het proxyscript is een server-side script en is niet gebonden aan de same origin policy.

AJAX zonder server-side scripting

Omdat niet iedereen over voldoende kennis beschikt om zelf server-side scripts te schrijven, beperken we ons in dit hoofdstuk tot AJAX voorbeelden waarvoor u geen webserver hoeft te gebruiken. Uiteindelijk zal uw website wel ergens op een server terecht komen, maar voor deze oefeningen maakt de servertechnologie niet uit. Gratis webhosting ondersteunt vaak geen server-side scripts, maar deze oefeningen zullen perfect werken. Onderstaande tabel toont een overzicht van welk type data u met welke methode kan verwerken. De meest eenvoudige methode wordt aangeduid met twee sterretjes.

In dit hoofdstuk bespreken we enkel de combinaties met een grijze achtergrond.

	load()	$.getScript()	$.getJSON()	$.get()	$.post()	$.ajax()
Tekst	**			*	*	*
HTML	**			*	*	*
JavaScript		**		*	*	*
JSON			**	*	*	*
JSONP			**	*	*	*
XML				**	**	*

Google Chrome heeft naast de same origin policy nog een beperking. Zonder webserver kan Chrome standaard geen AJAX verwerken.

Load()

De methode `load()` is de meest eenvoudige en tevens de meest gebruikte methode om HTML in de achtergrond op te halen.

In tegenstelling tot alle andere AJAX-methodes wordt `load()` rechtstreeks aan een selector gekoppeld.

Response: met deze methode kunnen we enkel tekst en HTML ophalen.

Syntax: `load(url [, data] [, callback])`

- URL:
  De URL van de op te halen pagina.
- Data:
  Een string (GET-request) of een map (POST-request) waarmee we de gegevens op de URL kunnen filteren. Aangezien we in dit hoofdstuk geen server-side script gebruiken, kunnen we deze parameter in dit hoofdstuk nog niet verwerken.

- Callback:
  Deze functie wordt uitgevoerd zodra alle gegevens zijn opgehaald.

Load(): HTML zonder callback

Open **ajaxLokaal/bloemen.html** en **ajaxLokaal/load1.html**.
Bekijk de broncode en test het resultaat in een browser.

De pagina bloemen.html is een klassieke HTML-pagina. Elke article-tag heeft een uniek id en bevat informatie over één bloem.

```
<section>
    <h1>Overzicht bloemen</h1>
    <h1>Overzicht bloemen</h1>
    <article id="Anjer">...</article>
    <article id="Lelie">...</article>
    <article id="Narcis">...</article>
    <article id="Roos">...</article>
    <article id="Tulp">...</article>
</section>
```

De inhoud van de section-tag gaan we volledig tonen in de pagina load1.html.

```
<ul class="tabNav">
    <li><a href="#!" class="active">Anjer</a></li>
    <li><a href="#!">Lelie</a></li>
    <li><a href="#!">Narcis</a></li>
    <li><a href="#!">Roos</a></li>
    <li><a href="#!">Tulp</a></li>
</ul>
<div id="data"></div>
...
<script>
$(function() {
    // De URL en de selectie binnen de pagina zijn gescheiden door
een spatie!
    $('#data').load('bloemen.html section');
});
</script>
```

Open **load1.html** in een browser.

Load (zonder callback functie)

Anjer | Lelie | Narcis | Roos | Tulp

Overzicht bloemen

Anjer

Lorem ipsum dolor sit amet, ullamco laboris nisi consectetur adipisicing elit. Sed do eiusmod tempor incididunt quis nostrud exercitation ut labore et dolore magna aliqua. Velit esse cillum dolore cupidatat non proident, qui officia deserunt.

Duis aute irure dolor mollit anim id est laborum. Eu fugiat nulla pariatur. Ut aliquip ex ea commodo consequat.

Info: http://nl.wikipedia.org/wiki/Anjer

Lelie

Ut aliquip ex ea commodo consequat. Quis nostrud exercitation velit esse cillum dolore duis aute irure dolor. Excepteur sint occaecat eu fugiat nulla pariatur. Ut enim ad minim veniam, in reprehenderit in voluptate sunt in culpa.

Ullamco laboris nisi quis nostrud exercitation ut labore et dolore magna aliqua. Velit esse cillum dolore sed do eiusmod tempor incididunt in reprehenderit in voluptate. Ullamco laboris nisi mollit anim id est laborum.

Toon volledige section-tag

Bij het laden van de pagina wordt de volledige inhoud van bloemen.html ingelezen en verschijnt alle data binnen de section-tag van bloemen.html in #data van load1.html. De navigatie bovenaan de pagina heeft voorlopig nog geen invloed op het resultaat.

> Merk op dat altijd de volledige HTML-code wordt ingelezen. De lay-out van bloemen.html (de CSS-eigenschappen) wordt nooit mee overgenomen. Load1.html bevat eigen stijlregels om de ingelezen data opnieuw te formatteren.

Meestal hebben we niet de volledige inhoud van een externe pagina nodig, maar enkel een bepaald deel. Het deel dat we willen ophalen, gaan we dan afbakenen met een id. Op bloemen.html heeft elke bloem reeds een uniek id. Indien we enkel de gegevens van #Roos willen tonen, moeten we het juiste id aan de URL toevoegen.

> Pas de code aan en test het resultaat in een browser.

```
$(function() {
    $('#data').load('bloemen.html #Roos');
});
```

Load (zonder callback functie)

Anjer | Lelie | Narcis | Roos | Tulp

Roos

Sed do eiusmod tempor incididunt ut enim ad minim veniam, ut aliquip ex ea commodo consequat. Sunt in culpa consectetur adipisicing elit, excepteur sint occaecat. Eu fugiat nulla pariatur.

Duis aute irure dolor in reprehenderit in voluptate quis nostrud exercitation. Qui officia deserunt ut enim ad minim veniam, velit esse cillum dolore. Sunt in culpa ut labore et dolore magna aliqua. Cupidatat non proident, mollit anim id est laborum. Quis nostrud exercitation eu fugiat nulla pariatur.

Ut labore et dolore magna aliqua. Duis aute irure dolor ut enim ad minim veniam, velit esse cillum dolore. In reprehenderit in voluptate eu fugiat nulla pariatur. Mollit anim id est laborum.

Info: http://nl.wikipedia.org/wiki/Rozen

Toon enkel #Roos

Het is niet zo dat we met deze functie maar een deel van de pagina inlezen. We lezen nog steeds ALLE data in, maar we tonen maar een deel van de gegevens.

> Binnen `load()` kunnen we eender welke selector gebruiken. Meestal gebruikt men hiervoor een id, maar het is perfect mogelijk om bijvoorbeeld enkel de titels te tonen.

Voor `$('#data').load('bloemen.html h2');` is dit het resultaat:

Load (zonder callback functie)

Anjer | Lelie | Narcis | Roos | Tulp

Anjer

Lelie

Narcis

Roos

Tulp

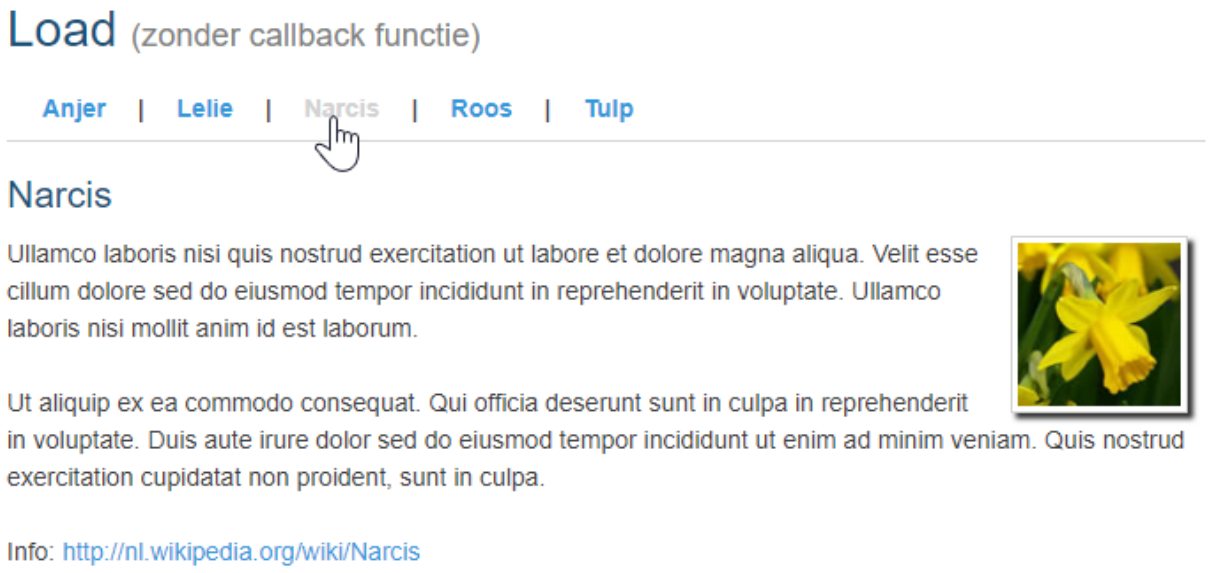
Toon alle h2-tags

Als laatste gaan we de navigatie integreren.

Pas de code aan en test het resultaat in een browser.

```
$(function() {
   $('.tabNav a').click(function (e) {
        var id = $(this).text();
        $('.tabNav a').removeClass('active');
        $(this).addClass('active');
        $('#data').load('bloemen.html #' + id);
    });
    $('.tabNav a:first').click();
});
```

De tekst binnen elke link komt overeen met het id van de te tonen sectie (let op: hoofdlettergevoelig). De pagina werkt feilloos, maar de code is zeker niet optimaal. Bij elke klik wordt immers de volledige pagina opnieuw ingelezen om uiteindelijk maar een deel te tonen. Het is natuurlijk veel interessanter dat we de pagina maar éénmaal inlezen en dat de navigatie vanuit het DOM de juiste id's toont of verbergt. Dit kunnen we perfect oplossen door de callback functie te gebruiken.



Intaractieve versie

Load(): HTML met callback

★ Open ajaxLokaal/load2.html en bekijk het script.

Open **ajaxLokaal/load2.html** en bekijk het script.

```
$(function() {
    $('#data').load('bloemen.html article', function () {
        $('#data article:not(:first)').hide();
    });
    $('.tabNav a').click(function () {
        $('.tabNav a').removeClass('active');
```

```
        $(this).addClass('active');
        var id = '#' + $(this).text();
        $('#data article').hide().filter(id).show();
    });
});
```

Bij het openen van de pagina worden alle article-tags van bloemen.html in #data geplaatst. Vanuit de callback functie worden alle article-tags, behalve de eerste, onzichtbaar gezet. Alle verdere interacties bevinden zich binnen het click-event op de links.

Klikt u bijvoorbeeld op de link Roos, dan wordt de actie:

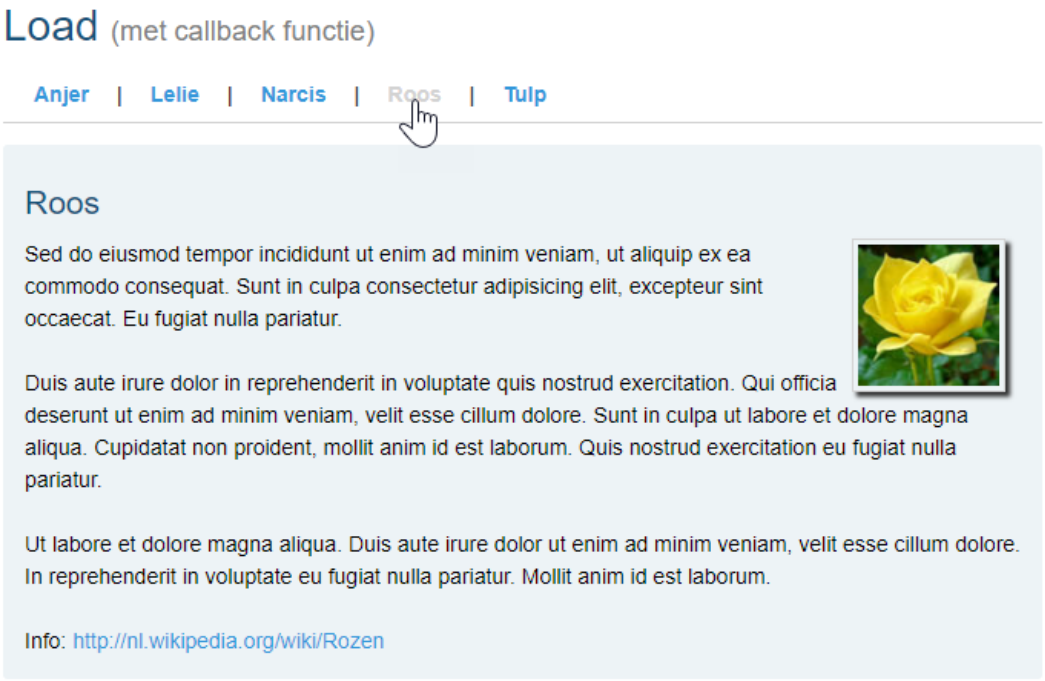
```
$('#data article').hide().filter('#Roos').show();
```

Als laatste gaan we nog een kleine animatie toevoegen.

> Vervang de laatste lijn binnen het click-event door onderstaande code.
> Test het resultaat in een browser.

```
$(function() {
    ...
    $('.tabNav a').click(function () {
        $('.tabNav a').removeClass('active');
        $(this).addClass('active');
        var id = '#' + $(this).text();
        if ($(id).is(':hidden')) {
            $('#data article:visible').slideUp('slow', function () {
                $(id).slideDown('slow');
            });
        }
    });
});
```

Bij elke klik controleren we eerst of het id reeds zichtbaar is. Enkel indien het id onzichtbaar staat, wordt de animatie uitgevoerd. De animatie zelf is een sequentieel slideUp en slideDown effect.



Interactieve versie met animatie

$.getScript()

Met `$.getScript()` kan u een extern JavaScript inlezen en uitvoeren. Zoals we uit voorgaand hoofdstuk weten, is deze methode niet gebonden aan de same origin policy van de browser. We kunnen dus perfect scripts van zowel de lokale site als van een externe server inlezen.

Merk op dat, met uitzondering van `load()`, alle AJAX methodes geen rechtstreekse koppeling hebben met een selector. De opgehaalde gegevens worden vanuit de callback functie gekoppeld.

Response: met deze methode kunnen we enkel JavaScript inlezen.

Syntax: `$.getScript(url [, callback])`

- URL:
  De URL van het op te halen JavaScript.
- Callback:
  Deze functie wordt uitgevoerd zodra het script volledig is ingelezen.

> Open **ajaxLokaal/data/alert.js** en **ajaxLokaal/script.html**.
> Bekijk de broncode en test het resultaat in een browser.

```
// code alert.js
alert('Deze boodschap is afkomstig van een extern script.');
function wijzigTitel(titel) {
    $('h1').text('*** ' + titel + ' ***');
}
```

En de code van script.html.

```
<input class="button" type="button" id="alert" value="$.getScript:
alert.js">
...
<script>
$('#alert').click(function(){
    $.getScript('data/alert.js', function() {
        var titel = $('#alert').val();
        wijzigTitel(titel);
    });
});
</script>
```

Pas op het moment dat u op de knop klikt, wordt het script alert.js ingelezen. Het script toont een alertbox en past de tekst van de h1-tag op script.html aan.

$.getJSON()

Met `$.getJSON()` kunnen we enkel gegevens in JSON-formaat ophalen. Filteren kan enkel via de GET-methode.

Response: JSON en JSONP. Syntax:

```
$.getJSON(url [, data] [,
callback]).done(callback).fail(callback).always(callback)
```

- **URL:**
  De URL van het op te halen bestand.
- **Data:**
  Een string of een map waarmee we de gegevens op de URL kunnen filteren. (Een map wordt binnen de functie automatisch omgezet naar een string.) Aangezien we in dit hoofdstuk geen server-side script gebruiken, kunnen we deze parameter in deze oefeningen nog niet verwerken.
- **Callback:**
  Deze functie wordt uitgevoerd zodra alle gegevens zijn opgehaald.
- **.done(callback)** (optioneel):
  Ook deze functie wordt uitgevoerd zodra alle gegevens zijn opgehaald, maar deze bevat GEEN data.
- **.fail(callback)** (optioneel):
  Deze functie wordt enkel uitgevoerd indien er iets nies loopt (URL niet bereikbaar, timeout, ...).
- **.fail(callback)** (optioneel):
  Deze functie wordt altijd uitgevoerd.

JSON is een relatief nieuw formaat, maar wint aan populariteit ten opzichte van XML. Daarom gaan we dit formaat iets gedetailleerder bespreken.

Een JSON-object kan enkel waardes bevatten (een array) of kan complexe naam/waardeparen (een object) bevatten. Elk van deze gegevensstructuren wordt aan de hand van een oefening verduidelijkt.

$.getJSON(): JSON met een eenvoudige literal array

Open **ajaxLokaal/data/bloemen1.json** en **ajaxLokaal/json1.html**.
Bekijk de broncode en test het resultaat in een browser.

Het JSON-object bevat vijf waardes.

```
["Anjer","Lelie","Narcis","Roos","Tulp"]
```

```
<h2>Aantal items: <span id="aantal"></span> </h2>
<section id="data">
    <ul id="bloemenlijst"></ul>
</section>
...
<script>
$(function () {
    $.getJSON('data/bloemen1.json', function (data) {
        console.log('first success');
        console.log('data', data);
        console.log('data[0]', data[0]);
        $('#aantal').text(data.length);
        $.each(data, function (index, value) {
            $("#bloemenlijst").append('<li>' + this + '</li>\n\t');
        });
    }).done(function () {
        console.log('second success');
    }).fail(function (err) {
        console.log('error', err);
    }).always(function () {
        console.log('complete');
    });
});
</script>
```

Json1.html leest de array uit bloemen1.json. De parameter data in de callback functie bevat de array.

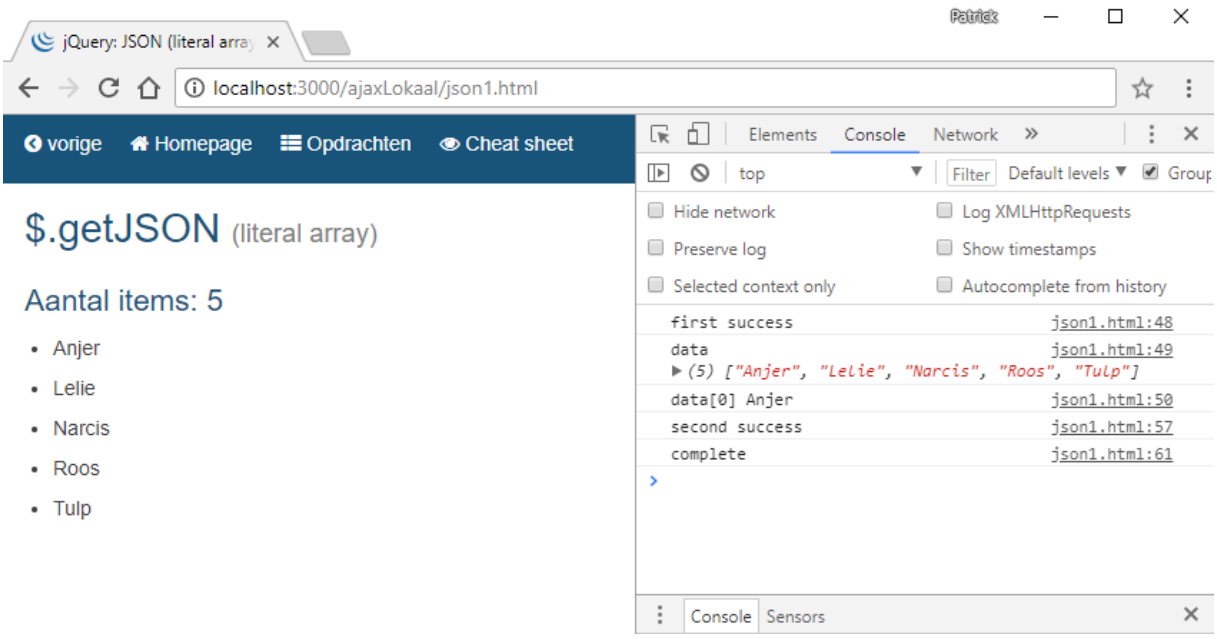
U kan deze waarde van data makkelijk uitlezen.

Bijvoorbeeld: de code `console.log('data', data);` toont de volledige array in het consolevenster van de browser.

`data.length` berekent het aantal items in de array. Deze waarde wordt achteraan de h2-tag toegevoegd.

De methode `$.each()` doorloopt alle items in de array en plaatst de waarde telkens in een nieuwe li-tag.

Ziehier het resultaat:



JSON met een eenvoudige literal array

$.getJSON(): JSON met één object

> Open **ajaxLokaal/data/bloemen2.json** en **ajaxLokaal/json2.html**.
> Bekijk de broncode en test het resultaat in een browser.

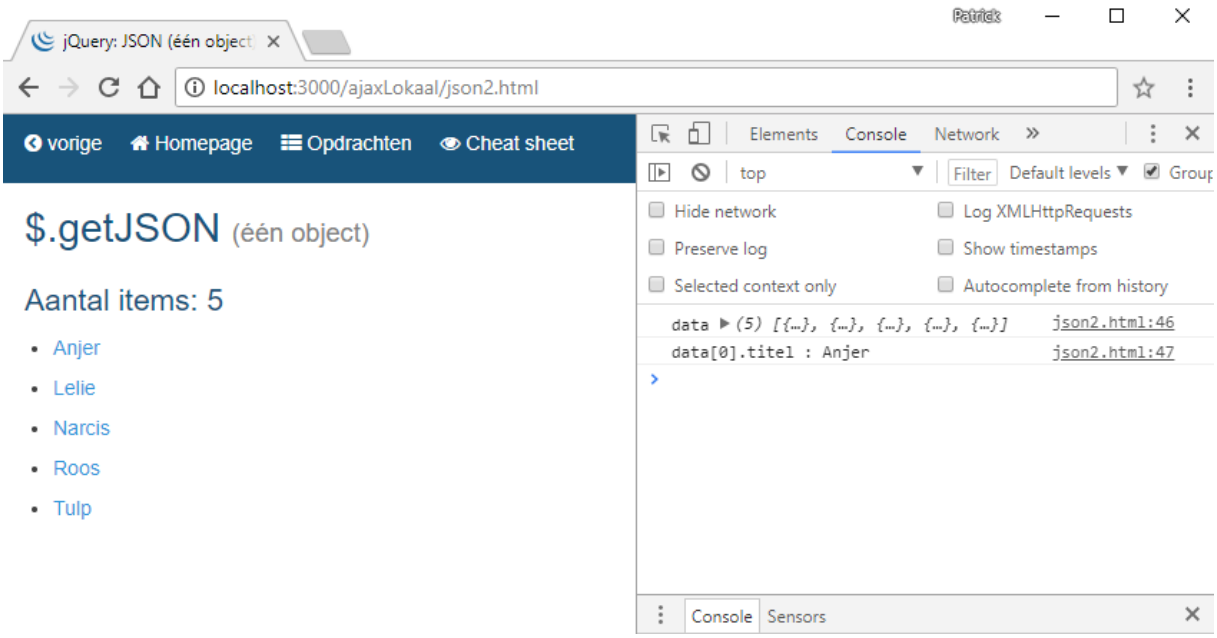
Het JSON-object bevat opnieuw informatie over vijf bloemen, maar elke bloem heeft nu een titel en een url.

```
[
    {"titel": "Anjer","url": "http://nl.wikipedia.org/wiki/Anjer"},
    {"titel": "Lelie","url": "http://nl.wikipedia.org/wiki/Lelie"},
    {"titel": "Narcis","url":
"http://nl.wikipedia.org/wiki/Narcis"},
    {"titel": "Roos","url": "http://nl.wikipedia.org/wiki/Rozen"},
    {"titel": "Tulp","url": "http://nl.wikipedia.org/wiki/Tulp"}
]
```

```
<h2>Aantal items: <span id="aantal"></span> </h2>
<section id="data">
    <ul id="bloemenlijst"></ul>
</section>
...
<script>
$(function () {
    $.getJSON('data/bloemen2.json', function (data) {
        console.log('data', data);
        console.log('data[0].titel :', data[0].titel);
        $('#aantal').text(data.length);
        $.each(data, function (index, value) {
            $("#bloemenlijst").append('<li><a href="' + this.url +
'">' + this.titel + '</a></li>\n\t');
```

```
        })
    }).fail(function (err) {
        console.log('error', err);
    });
});
</script>
```

De parameter data bevat nu een array van vijf objecten (zie console).



JSON met één objecten

Kijk even terug naar hoofdstuk 2. Binnen een constructor functie verwijst het kernwoord this naar het object. Binnen de each-lus lezen we met this.titel en this.url de waardes van elk object uit en voegen we deze toe aan de lijst bloemenlijst.

$.getJSON(): JSON met meerdere objecten

Open **ajaxLokaal/data/bloemen3.json** en **ajaxLokaal/json3.html**.
Bekijk de broncode en test het resultaat in een browser.
Bekijk de JSON-structuur.

```
{
    "bloemzaden": [
        {"titel": "......","omschrijving": "....."},
        {"titel": "......","omschrijving": "....."},
        ....
    ],
    "bloem": [
        {"titel": "...","foto": "...","url": "...","omschrijving":
"..."},
```

```
        {"titel": "...","foto": "...","url": "...","omschrijving":
"..."},
        ...
    ]
}
```

```
<section>
    <ul id="bloemzadenlijst"></ul>
</section>
<section class="sjabloon drop-shadow">
    <h2>titel</h2>
    <img src="../assets/fotos/thumb_roos.jpg">
    <div>Omschrijving</div>
    <p><a href="#">Meer info</a></p>
    <div class="reset"></div>
</section>
...
<script>
    $(function () {
        $.getJSON('data/bloemen3.json', function (data) {
            console.log('data', data);
            console.log('data.bloemzaden', data.bloemzaden);
            console.log('data.bloem', data.bloem);
            console.log('data.bloem[0].titel :',
data.bloem[0].titel);
            // bloemzaden
            $.each(data.bloemzaden, function (index, value) {
                $("#bloemzadenlijst").append('<li><strong>' +
this.titel + '</strong><br>' + this.omschrijving + '</li>\n\t');
            });
            // bloemen
            $.each(data.bloem, function (index, value) {
                $('.sjabloon').clone()
                    .find('h2').text(this.titel).end()
                    .find('div:first').html(this.omschrijving).end()
                    .find('a').attr('href', this.url).end()
                    .find('img').attr('src', '../assets/fotos/' +
this.foto).end()

.removeClass('sjabloon').insertBefore('.sjabloon');
            });
            $('.sjabloon').remove();
        }).fail(function (err) {
            console.log('error', err);
        });
    });
</script>
```

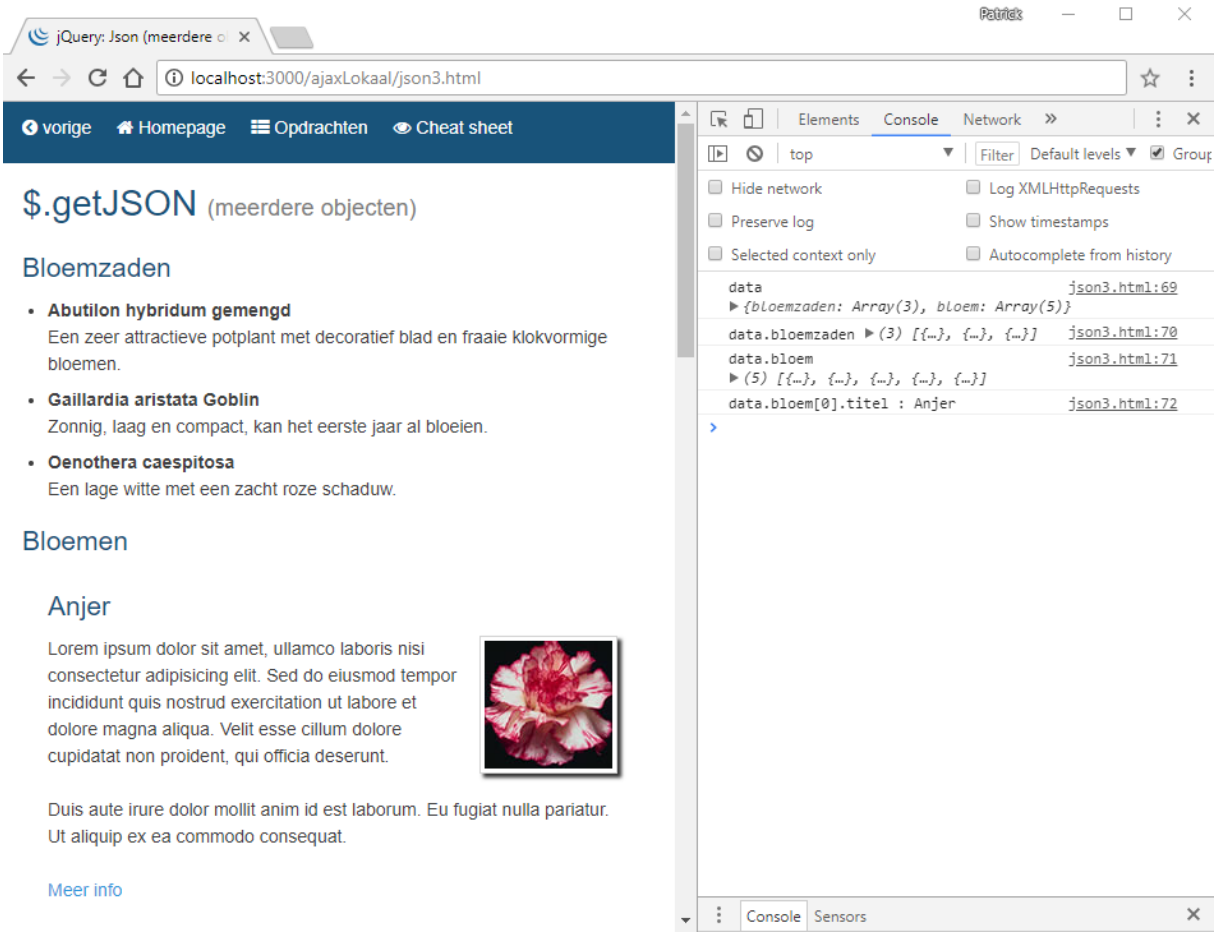
De parameter data bevat één samengesteld object. De verschillende deelobjecten kan u met de puntnotatie benaderen. `data.bloemzaden` is een array van drie objecten, `data.bloem` een array van vijf objecten.

Het eerste deelobject `data.bloemzaden` wordt, naar analogie met voorgaande oefening, in een lijst getoond.

Om het object `data.bloem` makkelijk te formatteren, maken we gebruik van het sjabloon (de section-tag met class .sjabloon). In de each-lus staat maar één statement! Om de code leesbaar te houden, wordt het statement verdeeld over meerdere lijnen.

Binnen de lus maken we een kopie van het sjabloon. De methode `find()` selecteert telkens één element binnen de section-tag en vult deze met de waardes uit het object. Op het einde van de ketting wordt de class .sjabloon verwijderd. Na het beëindigen van de lus wordt het originele sjabloon gewist.

Ziehier het resultaat:



JSON met meerdere objecten

$.get() en $.post()

Beide methodes worden normaal gebruikt om via een formulier gegevens te filteren. Het formulier wordt respectievelijk verzonden via de GET- of via de POST-methode.

Response: Tekst, HTML, JavaScript, JSON, JSONP en XML.

Syntax: `$.get(url [, data] [, callback] [, dataType]).done(callback).fail(callback).always(callback)`

- **URL:**
  De URL van de op te halen pagina.
- **Data:**
  Een string (voor de GET-methode) of een map (voor de POST-methode) waarmee we de gegevens op de URL kunnen filteren.
- **Callback:**
  Deze functie wordt uitgevoerd zodra alle gegevens zijn opgehaald.
- **DataType:**
  Het datatype (text, html, script, json, jsonp of xml) dat we verwachten terug te krijgen van de server.
- **.done(callback)** (optioneel):
  Ook deze functie wordt uitgevoerd zodra alle gegevens zijn opgehaald, maar deze bevat GEEN data.
- **.fail(callback)** (optioneel):
  Deze functie wordt enkel uitgevoerd indien er iets nies loopt (URL niet bereikbaar, timeout, ...).
- **.fail(callback)** (optioneel):
  Deze functie wordt altijd uitgevoerd.

Maak er een goede gewoonte van om de vierde parameter (dataType) altijd toe te voegen. Indien deze parameter ontbreekt, zal jQuery het datatype zelf bepalen, maar dit loopt vaak mis.

In feite is het er niet echt voor bedoeld, maar we kunnen beide methodes perfect zonder filter gebruiken. Merk ook op dat dit de eerste methodes zijn waarmee we XML kunnen verwerken.

$.get(): JSON met meerdere objecten

Open **ajaxLokaal/getJSON**
Bekijk de broncode en test het resultaat in een browser.

Deze toepassing is een bijna exacte kopie van json3.html.

Vervang $.getJSON door $.get en voeg achteraan het datatype json toe.

```
$.getJSON('data/bloemen3.json', function(data) {     // Code
json3.html
    ...
});

$.get('data/bloemen3.json', function(data) {         // Code
getJSON.html
    ...
```

```
}, 'json');
```

$.get(): XML

Open **ajaxLokaal/data/bloemen.xml** en **ajaxLokaal/getXML.html**.
Bekijk de broncode en test het resultaat in een browser.

Ziehier de XML-structuur:

```
<?xml version="1.0" encoding="utf-8"?>
<bloemen>
    <bloem id="1">
        <titel>Anjer</titel>
        <foto>thumb_anjer.jpg</foto>
        <url>http://nl.wikipedia.org/wiki/Anjer</url>
        <omschrijving><![CDATA[<p>Lorem
ipsum...</p>]]></omschrijving>
    </bloem>
    <bloem id="2">
        <titel>Lelie</titel>
        <foto>thumb_lelie.jpg</foto>
        <url>http://nl.wikipedia.org/wiki/Lelie</url>
        <omschrijving><![CDATA[<p>Ut aliquip
ex...</p>]]></omschrijving>
    </bloem>
    ....
<bloemen>
```

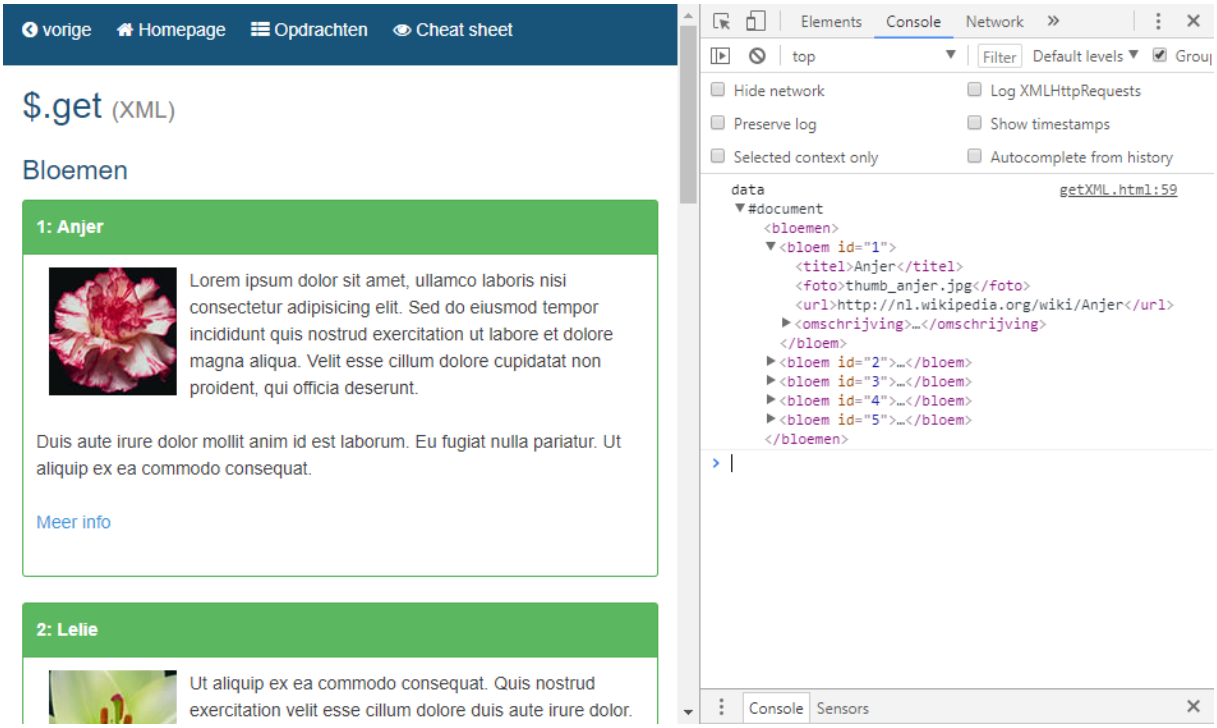
De node omschrijving bevat HTML-tags. Deze tags zijn niet voor de XML-parser bedoeld en worden daarom met een CDATA blok omsloten.

En de code:

```
<div class="sjabloon">
    <div class="card-success">
        <h3 class="card-header">titel</h3>
        <div class="card-body">
            <img src="../assets/fotos/thumb_roos.jpg">
            <div>Omschrijving</div>
            <p><a href="#!">Meer info</a></p>
        </div>
    </div>
</div>
...
<script>
    $(function () {
        $.get('data/bloemen.xml', function (data) {
            console.log('data', data);
            var bloem = $(data).find('bloem');
            $.each(bloem, function (index, value) {
                var id = $(this).attr('id');
                var titel = $(this).find('titel').text();
                var url = $(this).find('url').text();
```

```
                var foto = $(this).find('foto').text();
                var omschrijving =
$(this).find('omschrijving').text();

                $('.sjabloon').clone()
                    .find('.card-header').text(id + ': ' +
titel).end()
                    .find('a').attr('href', url).end()
                    .find('img').attr('src', '../assets/fotos/' +
foto).end()
                    .find('.card-body div').html(omschrijving).end()

.removeClass('sjabloon').insertBefore('.sjabloon');
            });
            $('.sjabloon').remove();
        }, 'xml').fail(function (err) {
            console.log('error', err);
        });
    });
</script>
```

De parameter data in de callback functie is nu een object van het type document (Chrome) of XMLdocument (Firefox).



Net zoals in een HTML-document kunnen we nodes en attributen het makkelijkst benaderen via de methode `find('node')`. De methode `text()` leest de waarde van een node uit en de methode `attr('id')` de waarde van het attribuut id.

Net zoals in het JSON voorbeeld worden de gegevens getoond op basis van een sjabloon dat we enkele malen klonen. De variabele bloem bevat een array van alle bloem nodes. Binnen de each-lus worden per bloem alle nodes en attributen opgehaald en in een copy van het sjabloon geplaatst.

Vergeet ook hier niet het datatype xml te vermelden!

```
$.get('data/bloemen.xml', function(data){....}, 'xml' );
```

$.get(): JSON vs. XML

Zeker in het begin is het even wennen aan de manier waarop we gegevens uit een JSON- en XML-document moeten benaderen. Vandaar nog even het verschil tussen beide vermelden.

- **JSON** is een object van het type object.
  Binnen een object gebruiken we de puntnotatie om dieper in de boomstructuur af te dalen.
  Waarde ophalen via: data.naam1.naam2. ... .naamx
- **XML** is een object van het type document.
  Hier gebruiken we net dezelfde syntax als in het DOM. Om snel verschillende niveaus af te dalen gebruikt u best find().
  Waarde van een node ophalen: $(data).find('node').text()
  Waarde van een attribuut ophalen: $(data).find('node').attr('attr')

Als voorbeeld gaan we alle gegevens van één persoon weergeven vanuit een JSON-bestand en vanuit een XML-bestand. De structuur van beide bestanden is niet optimaal, want we gebruiken overdreven veel geneste nodes, maar het is in dit voorbeeld enkel de bedoeling om aan te tonen hoe we best de boomstructuur benaderen.

Open **ajaxLokaal/data/adres.xml** en **ajaxLokaal/data/adres.json**.

```
<?xml version="1.0" encoding="utf-8"?>
<adresboek>
    <adres id="1">
        <naam>Naam1</naam>
        <email>
            <email1>naam1@example.com</email1>
            <email2>naam2@example.com</email2>
        </email>
    </adres>
    ...
</adresboek>
```

```
{
    "adres": [
        {"id": "1","naam": "Naam1","email": {
            "email1": "naam1@example.com",
            "email2": "naam2@example.com"}
        },{"id": "2","naam": "Naam2","email": {
            "email1": "naam3@example.com",
            "email2": "naam4@example.com"}
        }
```

```
    ]
}
```

Beide bestanden volgen ongeveer eenzelfde logica.

In XML bevat de rootnode adresboek twee adresnodes. Elk adres heeft een attribuut id en twee childnodes, naam en email. De emailnode heeft op zich ook twee childnodes, email1 en email2.

In JSON zien we het adres object. Elk adres heeft drie properties waarvan twee properties (id en naam) een waarde bevatten en de derde propertie (email) bevat een object.

> Open **ajaxLokaal/get_JSON_XML.html**.
> Test de pagina in een browser.

```
// XML
$.get('data/adres.xml', function(data) {
    $('#xmlData h2').text($(data).find('naam:first').text());
    $('#xmlData span:first')
        .text($(data).find('adres:first').attr('id'));
    $('#xmlData
span:eq(1)').text($(data).find('email1:first').text());
    $('#xmlData
span:last').text($(data).find('email2:first').text());
}, 'xml');
```

```
// JSON
$.get('data/adres.json', function(data) {
    $('#jsonData h2').text(data.adres[0].naam);
    $('#jsonData span:first').text(data.adres[0].id);
    $('#jsonData span:eq(1)').text(data.adres[0].email.email1);
    $('#jsonData span:last').text(data.adres[0].email.email2);
}, 'json');
```

Ziehier het resultaat:

$.get (JSON vs XML)

Naam1

id= 1
Email-1 = naam1@example.com
Email-2 = naam2@example.com

Naam1

id= 1
Email-1 = naam1@example.com
Email-2 = naam2@example.com

$.post()

`$.get()` en `$.post()` volgen net dezelfde logica. Aangezien we in dit hoofdstuk geen formulierfilters gebruiken, mogen we in alle voorgaande voorbeelden `$.get()` vervangen door `$.post()` en de toepassing zal blijven werken.

Vervang in één van voorgaande voorbeelden `$.get()` door `$.post()` en test het resultaat in een browser.

$.ajax()

De methode $.ajax() is de meest uitgebreide maar tevens ook de moeilijkste AJAX methode. Alle voorgaande methodes zijn een vereenvoudigde afleiding van $.ajax(). Een aantal eigenschappen die we in voorgaande methodes gebruiken (zoals bijvoorbeeld foutafhandeling) zijn in deze methode rechtstreeks geïntegreerd.

Response: Tekst, HTML, JavaScript, JSON, JSONP en XML.

Syntax: `$.ajax(settings)`

- **Settings:**
  Een set van naam/waarde eigenschappen om de methode te configureren. In totaal zijn er meer dan dertig eigenschappen. Voor een volledig overzicht, zie:
  http://api.jquery.com/jQuery.ajax/

De belangrijkste settings zijn:

- **url:**
  De URL van de op te halen pagina.
- **dataType:**
  Het datatype (text, html, script, json, jsonp of xml) dat we verwachten terug te krijgen van de server.
- **data:**
  De data die we als filter mee sturen.
- **type:**
  Verzend de extra parameters via GET of POST (default is GET).
- **beforeSend():**
  deze callback functie wordt uitgevoerd net voor de gegevens worden opgehaald. Dit is de ideale plaats om een preloader te tonen.
- **success():**
  na het inlezen van de externe gegevens wordt deze callback functie uitgevoerd. U kan deze functie vergelijken met de callback uit voorgaande AJAX-methodes.
- **error():**
  deze callback functie wordt uitgevoerd indien er zich een fout voordoet.

Voor de meeste toepassingen komt u al een heel eind met de vijf voorheen besproken methodes. Bij wijze van voorbeeld gaan we één van de voorgaande toepassingen volledig herschrijven met de methode `$.ajax()`.

Open **ajaxLokaal/ajax.html**.
Test de pagina in een browser.

```
$.ajax({
    url: 'data/bloemen2.json',
    dataType: 'json',
    success: function(data) {
        $('#aantal').text(data.length);
        $.each(data, function(index, value) {
            $("#bloemenlijst").append('<li><a href="' + this.url +
'">'
                + this.titel + '</a></li>\n\t');
        });
    },
    error: function(jqXhr, textStatus, error) {
        alert( "error\n" + error );
    }
});
```

Deze toepassing reageert net hetzelfde als json2.html. De code binnen de callback functie van `$.getJSON()` verhuist naar de callback van `success`.

Externe gegevens ophalen via JSONP

We hebben het al enkele malen aangehaald: het is perfect mogelijk een JSON-bestand van een ander domein in te lezen. De externe server moet deze cross-domain-data echter wel toestaan. U kan dus niet zomaar eender welk JSON-bestand vanaf een externe server inlezen.

Even zoeken op Google en u vindt voldoende JSONP aanbieders.

Om een JSONP request uit te voeren, hoeft u meestal aan het einde van de URL enkel de parameter callback=? toe te voegen. Meer is het niet.

In deze voorbeelden gebruiken we enkel de methode $.getJSON(), maar u kan perfect de methodes $.get(), $.post() of $.ajax() gebruiken.

Voor deze oefeningen zijn we volledig afhankelijk van services van derden. Deze services kunnen in verloop van tijd wijzigen of zelfs helemaal verdwijnen. Het is dus best mogelijk dat tijdens het lezen van dit boek bepaalde oefeningen niet meer het gewenste resultaat opleveren.

Filminfo ophalen via IMDB

De Internet Movie Database, of kortweg IMDB is een online databank met informatie over films, series en games. IMDB biedt voorlopig zelf nog geen eigen API aan, maar er zijn reeds verschillende externe hosts die voor u de gegevens in JSONP formaat kunnen aanleveren. Een van die hosts is http://www.omdbapi.com/.

> Om seze service te kunnen gebruiken, moet u eerst een gratis apikey genereren!

By ID or Title

Parameter	Required	Valid Options	Default Value	Description
i	Optional*		<empty>	A valid IMDb ID (e.g. tt1285016)
t	Optional*		<empty>	Movie title to search for.
type	No	movie, series, episode	<empty>	Type of result to return.
y	No		<empty>	Year of release.
plot	No	short, full	short	Return short or full plot.
r	No	json, xml	json	The data type to return.
callback	No		<empty>	JSONP callback name.
v	No		1	API version (reserved for future use).

omdbapi API

De API is zeer eenvoudig. In onze toepassing verwerken we dadelijk drie parameters, namelijk t voor de titel, y voor het jaartal en plot. Al deze parameters zijn optioneel.

Voor we de code bekijken kunnen we best eerst de structuur van de JSON response bestuderen.

Open **ajaxLokaal/JSONP_imdb.html** in een browser.
Open op deze pagina de voorbeeldlink Zoek Pulp Fiction.


```
{"Title":"Pulp Fiction","Year":"1994","Rated":"R","Released":"14 Oct 1994","Runtime":"154 min","Genre":"Crime,
Drama","Director":"Quentin Tarantino","Writer":"Quentin Tarantino (stories), Roger Avary (stories), Quentin
Tarantino","Actors":"Tim Roth, Amanda Plummer, Laura Lovelace, John Travolta","Plot":"The lives of two mob hitmen, a
boxer, a gangster's wife, and a pair of diner bandits intertwine in four tales of violence and
redemption.","Language":"English, Spanish, French","Country":"USA","Awards":"Won 1 Oscar. Another 62 wins & 69
nominations.","Poster":"https://ia.media-
imdb.com/images/M/MV5BMTkxMTA5OTAzMl5BMl5BanBnXkFtZTgwNjA5MDc3NjE@._V1_SX300.jpg","Ratings":[{"Source":"Internet Movie
Database","Value":"8.9/10"},{"Source":"Rotten Tomatoes","Value":"94%"},
{"Source":"Metacritic","Value":"94/100"}],"Metascore":"94","imdbRating":"8.9","imdbVotes":"1,514,292","imdbID":"tt01109
12","Type":"movie","DVD":"19 May 1998","BoxOffice":"N/A","Production":"Miramax
Films","Website":"N/A","Response":"True"}
```


JSON zonder formatering

Om de response zo klein mogelijk te houden, zijn alle overbodige spaties verwijderd. Dit maakt het ons wel extra moeilijk om de structuur te interpreteren. Gelukkig bestaat er een Chrome extensie JSON Formatter waarmee we JSON data kunnen herformatteren.

(De link staat eveneens op de voorbeeldpagina.)

```
{
    "Title": "Pulp Fiction",
    "Year": "1994",
    "Rated": "R",
    "Released": "14 Oct 1994",
    "Runtime": "154 min",
    "Genre": "Crime, Drama",
    "Director": "Quentin Tarantino",
    "Writer": "Quentin Tarantino (stories), Roger Avary (stories), Quentin Tarantino",
    "Actors": "Tim Roth, Amanda Plummer, Laura Lovelace, John Travolta",
    "Plot": "The lives of two mob hitmen, a boxer, a gangster's wife, and a pair of diner bandits intertwine in four tales of violence and
    redemption.",
    "Language": "English, Spanish, French",
    "Country": "USA",
    "Awards": "Won 1 Oscar. Another 62 wins & 69 nominations.",
    "Poster": "https://ia.media-imdb.com/images/M/MV5BMTkxMTA5OTAzMl5BMl5BanBnXkFtZTgwNjA5MDc3NjE@._V1_SX300.jpg",
    "Ratings": [
        {
            "Source": "Internet Movie Database",
            "Value": "8.9/10"
        },
        {
            "Source": "Rotten Tomatoes",
            "Value": "94%"
        },
        {
            "Source": "Metacritic",
            "Value": "94/100"
        }
    ],
    "Metascore": "94",
    "imdbRating": "8.9",
    "imdbVotes": "1,514,292",
    "imdbID": "tt0110912",
    "Type": "movie",
    "DVD": "19 May 1998",
    "BoxOffice": "N/A",
    "Production": "Miramax Films",
    "Website": "N/A",
    "Response": "True"
}
```


JSON met formatering

De structuur is nu makkelijk leesbaar. Een aantal van deze velden wordt in onze toepassing verwerkt. Let vooral op Poster. Dit veld bevat de URL naar de coverafbeelding.

> Vul in het hidden-field apikey uw eigen apikey in!
> Bekijk de broncode en test het resultaat in een browser.
> Zoek op pulp fiction.

Let in de code vooral op de structuur van het formulier!

```
<form name="imdbForm" id="imdbForm" method="get"
action="http://www.omdbapi.com/">
    <div>
        <label for="t">Zoek titel:</label>
        <div class="input">
            <input type="text" name="t" id="t" required value="pulp
fiction">
        </div>
    </div>
    <div>
        <label for="y">Jaar:</label>
        <div class="input">
            <input type="number" name="y" id="y">
        </div>
    </div>
    <div>
        <label for="plot">Plot:</label>
        <div class="select">
            <select name="plot" id="plot">
                <option value="short">korte samenvatting</option>
                <option value="full">volledig</option>
            </select>
            <i class="fa fa-angle-down fa-2"></i>
        </div>
        <input type="hidden" name="apikey" id="apikey"
value="xxxxxx">
        <input class="button" type="submit" name="submit"
id="submit" value="zoek">
    </div>
</form>
```

Kies voor alle formuliervelden net dezelfde naam als de parameters in de API. Als we dadelijk het formulier verzenden, komen de juiste parameters automatisch correct in de URL terecht.

En nu het script.

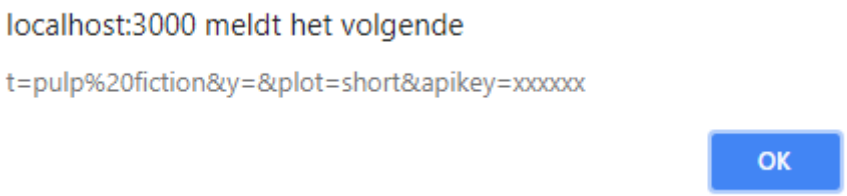
Omwille van de soms grote vertragingstijden, werken we bij het ophalen van externe gegevens best altijd met een preloader.

Zodra het formulier verzonden wordt (submit-event), gaan we met e.preventDefault() voorkomen dat de pagina wordt omgeleid naar de URL op de form-tag (http://www.omdbapi.com/). Vervolgens wordt de preloader getoond.

```
$('#imdbForm').submit(function (e) {
    e.preventDefault();
    $('#imdbInfo').hide();
    $('.preloader-success').show();
    var pars = $(this).serialize();
    // alert(pars);
    $.getJSON('http://www.omdbapi.com/?callback=?&' + pars, function
(data) {
        console.log('data', data)
        $('#titel').text(data.Title);
        $('#genre').text(data.Genre);
        $('#release').text(data.Released);
        $('#acteurs').text(data.Actors);
```

```
        $('#director').text(data.Director);
        $('#runtime').text(data.Runtime);
        $('#plotInfo').text(data.Plot);
        $('#cover').attr('src', data.Poster);
        $('#link').attr('href', 'http://imdb.com/title/' +
data.imdbID);
        if (data.Poster) {
            data.Poster.length > 10 ? $('#cover').show() :
$('#cover').hide();
        }
        $('#imdbInfo').show();
    }).fail(function (err) {
        console.error('error', err);
        alert(JSON.stringify(err));
    }).always(function () {
        $('.preloader-success').hide()
    });
});
```

De methode serialize() leest alle formuliervelden uit, zet de naam + waardes om naar de juiste URL decoding en plaatst deze in een string.



Dit is tevens het correcte formaat waarin we de gegevens moeten doorsturen naar de API. Vervolgens maken we een JSONP request met als URL:
http://www.omdbapi.com/?callback=?&t=pulp+fiction&y=&plot=short&apikey=xxxxxx

Let vooral op de extra parameter callback=?.

Zonder deze parameter geen JSONP en dus geen resultaat!

De logica binnen de callback functie kennen we ondertussen. De gegevens worden uitgelezen en op de juiste plaats op de pagina getoond. Zodra alles verwerkt is, wordt de preloader verwijderd.

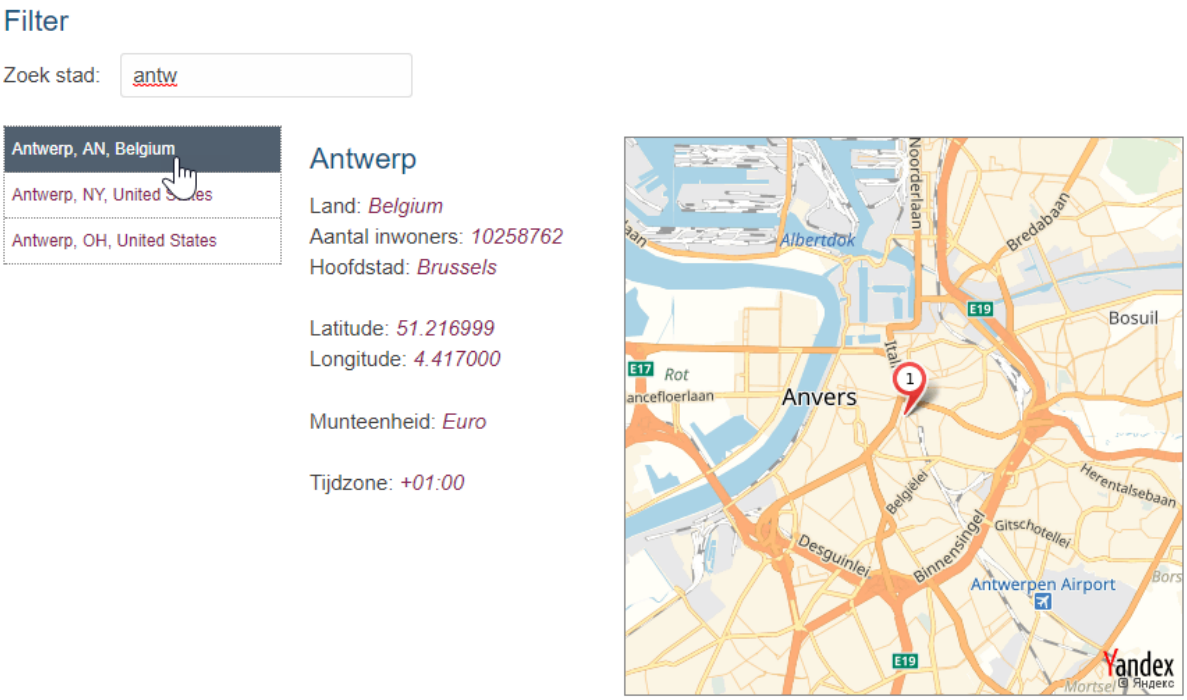
Nog een kleine opmerking over de afbeelding. Voor sommige films bestaat (nog) geen cover. Indien Poster de waarde N/A heeft, maken we #cover onzichtbaar.

```
if (data.Poster) {
    data.Poster.length > 10 ? $('#cover').show() :
$('#cover').hide();
}
```

Stadsinfo opvragen via GeoBytes

GeoBytes geeft ons algemene informatie (coördinaten, land, munteenheid, ...) van steden over de ganse wereld.

Open **ajaxLokaal/JSONP_stadsInfo.html**
Bekijk de broncode en test het resultaat in een browser.



In deze toepassing maken we gebruik van drie API's.

- Links: GeoBytes API voor het tonen van de stadslijst.
- Midden: GeoBytes API voor het tonen van de stadsinfo.
- Rechts: Yandex Maps API voor het stadsplan.

Meer info over de gebruikte API's vindt u hier:

- http://www.geobytes.com/free-ajax-cities-jsonp-api.htm
- https://tech.yandex.com/maps/doc/staticapi/1.x/dg/concepts/input_params-docpage/

Als voorbeeld gaan we zoeken naar alle steden die de tekenreeks antw bevatten.

Open **http://gd.geobytes.com/AutoCompleteCity?q=antw** in een browser.
Herformatteer het resultaat met JSONlint.

```
[
    "Antwerp, AN, Belgium",
    "Antwerp, NY, United States",
    "Antwerp, OH, United States"
]
```


Raw Parsed

De details kunnen we opvragen via een tweede URL op GeoBytes.

> Open **http://gd.geobytes.com/GetCityDetails?fqcn=Antwerp,+AN,+Belgium** in een browser.
> Herformatteer het resultaat met JSONlint.

(De waarde voor de parameter fqcn kan u eventueel kopiëren uit voorgaande array.)


```
{
    "geobytesforwarderfor": "",
    "geobytesremoteip": "193.190.124.5",
    "geobytesipaddress": "Antwerp, AN, Belgium",
    "geobytescertainty": "0",
    "geobytesinternet": "BE",
    "geobytescountry": "Belgium",
    "geobytesregionlocationcode": "BEAN",
    "geobytesregion": "Antwerpen",
    "geobytescode": "AN",
    "geobyteslocationcode": "BEANANTW",
    "geobytesdma": "0",
    "geobytescity": "Antwerp",
    "geobytescityid": "6037",
    "geobytesfqcn": "Antwerp, AN, Belgium",
    "geobyteslatitude": "51.216999",
    "geobyteslongitude": "4.417000",
    "geobytescapital": "Brussels",
    "geobytestimezone": "+01:00",
    "geobytesnationalitysingular": "Belgian",
    "geobytespopulation": "10258762",
    "geobytesnationalityplural": "Belgians",
    "geobytesmapreference": "Europe ",
    "geobytescurrency": "Euro",
    "geobytescurrencycode": "EUR",
    "geobytestitle": "Belgium"
}
```


Raw Parsed

De coördinaten geobyteslatitude en geobyteslongitude uit dit bestand gebruiken we om een statische afbeelding op Yandex te berekenen.

> Open de afbeelding in een browser:
> https://static-maps.yandex.ru/1.x/?lang=en-US&z=12&l=map&size=400,400&ll=4.417000,51.216999&pt=4.417000,51.216999,pm2rdl1

Als we dit nu allemaal combineren, krijgen we volgende code:

```
$('#geoForm').submit(function (e) {
    e.preventDefault();
});

// stadslijst
$('#q').keyup(function (e) {
    if ($('#q').val().length > 2) {
        var pars = $('#geoForm').serialize();

$.getJSON('http://gd.geobytes.com/AutoCompleteCity?callback=?&' +
pars, function (data) {
            $('#stadslijst li').remove();
            $.each(data, function (index, value) {
                $('#stadslijst ul').append('<li>' + this + '</li>');
            });
            $('#stadslijst li:first').text() == '' ? $('#stadslijst,
#stadInfo').hide() : $('#stadslijst').show();
        });
    }
}).keyup();

// detail van de stad
$('#stadslijst ul').on('mouseenter', 'li', function () {
    var stad = $(this).text();

$.getJSON('http://gd.geobytes.com/GetCityDetails?callback=?&fqcn=' +
stad, function (data) {
        $('#stadInfo h2').text(data.geobytescity);
        $('#land').text(data.geobytescountry);
        $('#inwoners').text(data.geobytespopulation);
        $('#hoofdstad').text(data.geobytescapital);
        $('#tijdzone').text(data.geobytestimezone);
        $('#munt').text(data.geobytescurrency);
        var lat = data.geobyteslatitude;
        $('#lat').text(lat);
        var lng = data.geobyteslongitude;
        $('#lng').text(lng);
        var lngLat = lng + ',' +lat;
        var mapURL = 'https://static-maps.yandex.ru/1.x/?lang=en-
US&z=12&l=map&size=400,400&ll=' + lngLat + '&pt=' + lngLat +
',pm2rdl1';
        $('#map').attr('src', mapURL);
    });
    $('#stadInfo').show();
});
```

Het keyup-event wordt getriggerd vanuit het zoekveld q. Indien dit veld meer dan twee karakters bevat, komt de eerste JSON request in actie. De gegevens die we als een array ontvangen, worden in een ul-lijst verwerkt. Eerst worden de oude items uit de lijst gewist en daarna wordt de lijst opnieuw opgebouwd.

Telkens we de muis op één van de steden plaatsen (mouseenter) worden de stadsdetails opgevraagd en wordt het middenstuk geüpdatet. Met de coördinaten kunnen we via de Yandex API een nieuwe afbeelding ophalen.

Merk op dat de stadslijst volledig dynamisch wordt opgebouwd.

Op een dynamisch gegenereerde lijst kunnen we enkel het on-event gebruiken.

- Goed: `$('#stadslijst ul').on('mouseenter', 'li', function(){...}).`
- Fout: `$('#stadslijst ul li').mouseenter(function(){...}).`

JSONP met $.get(), $.post() of $.ajax()

Werkt u liever met deze methodes, vergeet dan niet het juiste formaat te vermelden.

De URL-parameter callback=? komt dan te vervallen.

- $.get('url', function(data) {...}, 'jsonp');
- $.post('url', function(data) {...}, 'jsonp');
- $.ajax({..., dataType: 'jsonp' , ...});

AJAX met server-side scripting

Heb je al ervaring met dynamische websites, dan ben jezeker vertrouwd met een master/detail-relatie. Op de overzichtspagina, de master, komt een beknopt overzicht van alle items uit de database. Vanuit een link waarbij we het id meesturen, komen we vervolgens op de detailpagina terecht. Elke actie resulteert in een nieuwe connectie met de database en met de webserver.

Misschien hebt u ook al een contactformulier ontworpen? De gebruiker vult zijn gegevens in en komt na het verzenden op een bedankpagina terecht.

Deze twee "klassiekers" gaan we in dit hoofdstuk omvormen tot een AJAX gestuurde pagina.

In het tweede deel van dit hoofdstuk gaan we externe data (HTML, JSON en XML) via een proxyscript ophalen en in een eigen webpagina verwerken.

In dit hoofdstuk bespreken we enkel de combinaties met een grijze achtergrond.

	load()	$.getScript()	$.getJSON()	$.get()	$.post()	$.ajax()
Tekst	**			*	*	*
HTML	**			*	*	*
JavaScript		**		*	*	*
JSON			**	*	*	*
JSONP			**	*	*	*
XML				**	**	*

Alle toepassingen in dit hoofdstuk maken gebruik van server-side PHP-scripts. Beschikt u niet over PHP-hosting, dan kan u best een lokale testserver installeren. Bijvoorbeeld:

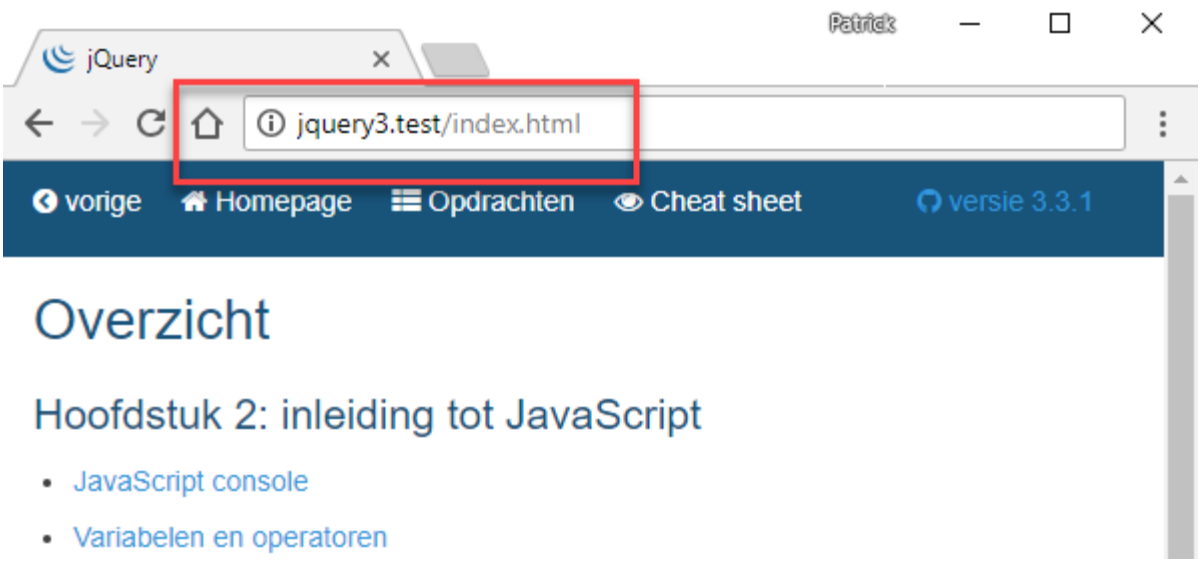
- AMPPS (Windows - Mac): https://www.ampps.com/
- WAMPserver (Windows): http://www.wampserver.com/
- MAMP (Mac): http://www.mamp.info/

Het installeren en configureren van een webserver en PHP valt buiten het bestek van deze cursus.

Maak in de root van de webserver een nieuwe map jQuery aan of maak een lokaal domein aan. Kopieer ALLE oefenbestanden naar de map jQuery.

Ziehier het resultaat:

- Op een externe webserver is de URL: http://www.mijnDomein.com/jQuery/
- Op een lokale webserver met lokaal domein is de URL: http://jquery3.test/

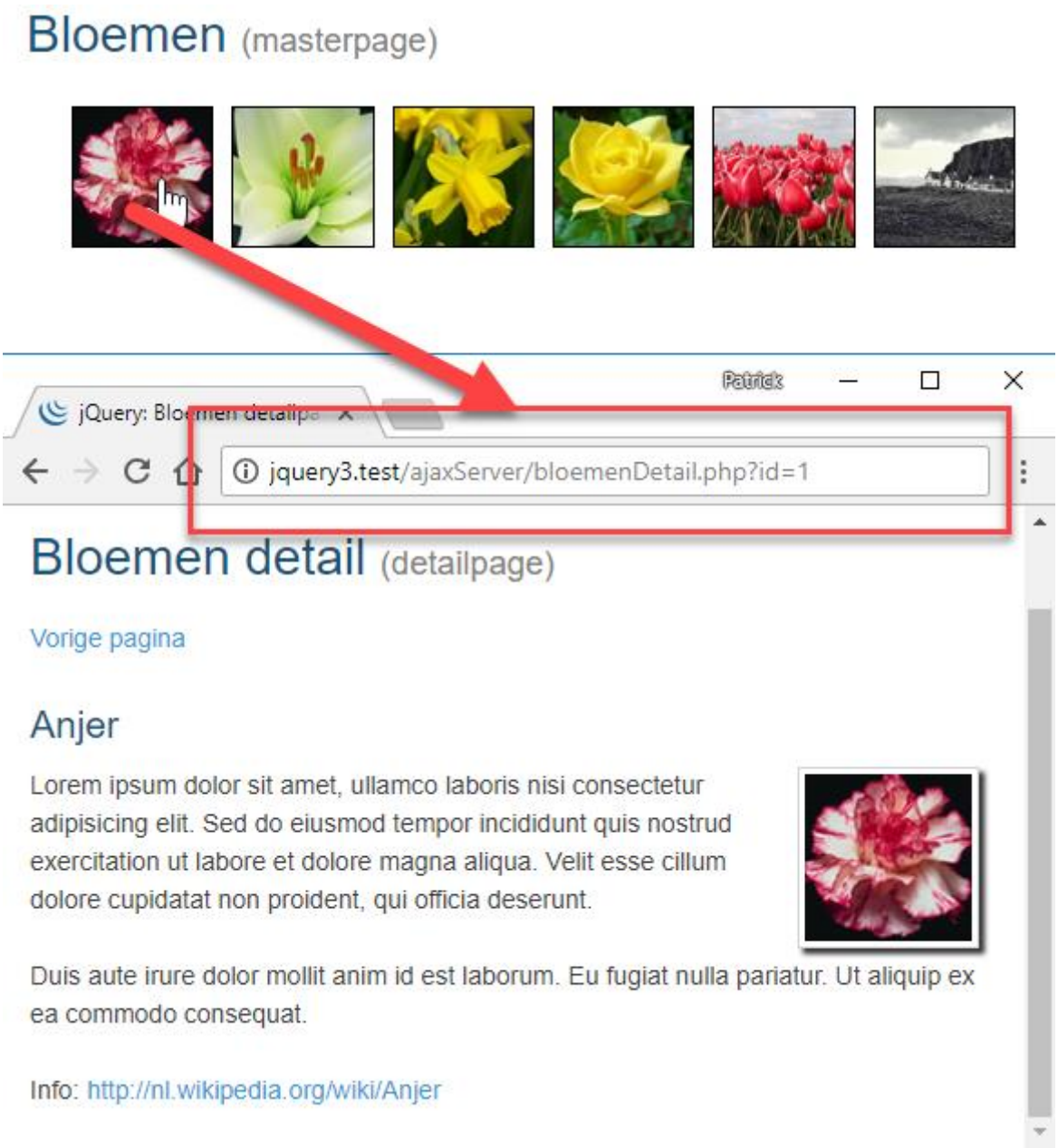


Lokale webserver met lokaal domein

> Alle voorbeelden uit vorig hoofdstuk kan u via de webserver ook testen in Chrome.

Master/detailrelatie met load()

> Open **http://jquery3.test/ajaxServer/bloemenMaster1.html** in een browser.
> Klik op één van de afbeeldingen.



Elke afbeelding op de hoofdpagina verwijst naar dezelfde detailpagina, enkel de parameter id=x is verschillend.

Het bestand ajaxServer/bloemenDetail.php bevat een PHP-script dat in functie van het id een bepaald gedeelte van de pagina toont. Voor alle id's groter dan 5 (of kleiner dan 1) verschijnt een foutboodschap. Bijvoorbeeld http://jquery3.test/ajaxServer/bloemenDetail.php?id=100 geeft:

We gaan de hoofdpagina zodanig aanpassen dat een deel van de detailpagina rechtstreeks op de hoofdpagina verschijnt. Een browser die geen JavaScript ondersteunt zal nog steeds de site op de klassieke manier kunnen bekijken. Ook zoekrobots, zoals Google, ondersteunen geen JavaScript, maar kunnen de site op deze manier nog volledig indexeren.

> Bekijk vanuit de browser de broncode van de detailpagina.
> Bijvoorbeeld: **http://jquery3.test/ajaxServer/bloemenDetail.php?id=1**

Het deel van de pagina dat we dadelijk op de hoofdpagina integreren, staat binnen `<section id="bloemInfo">`

> Open **ajaxServer/bloemenMaster1.html** en bekijk de broncode.

```
$('#_ a').click(function(e) {
    e.preventDefault();
    var url = $(this).attr('href');
    $('#detail').load(url + ' #bloemInfo', function(){
        ...
    });
});
```

Het script werkt voorlopig niet omdat de selector op het click-event nog niet in orde is. Dit passen we dadelijk aan.

Binnen het click-event zorgt `e.preventDefault()` ervoor dat de echte links op de afbeeldingen niet meer werken. Op de derde lijn wordt de link van de detailpagina uitgelezen en vervolgens in de methode `load()` verwerkt. Vergeet niet het id #bloeminfo in de uiteindelijke link te verwerken, anders gaan we de volledige pagina inbedden!

> Wijzig de selector `$('#_ a').click(...)` in `$('#master a').click(...)` en test het resultaat in een browser. Let op: de pagina openen vanaf de webserver: **http://jquery3.test/ajaxServer/bloemenMaster1.html**

Enkel de link "Vorige pagina" is nog storend. Deze kunnen we makkelijk binnen de callback van load() verwijderen.

> Voeg de code `$('#bloemInfo p:first').remove()` toe en test de pagina opnieuw.

```
$('#detail').load(url + ' #bloemInfo', function(){
    $('#bloemInfo p:first').remove();
});
```

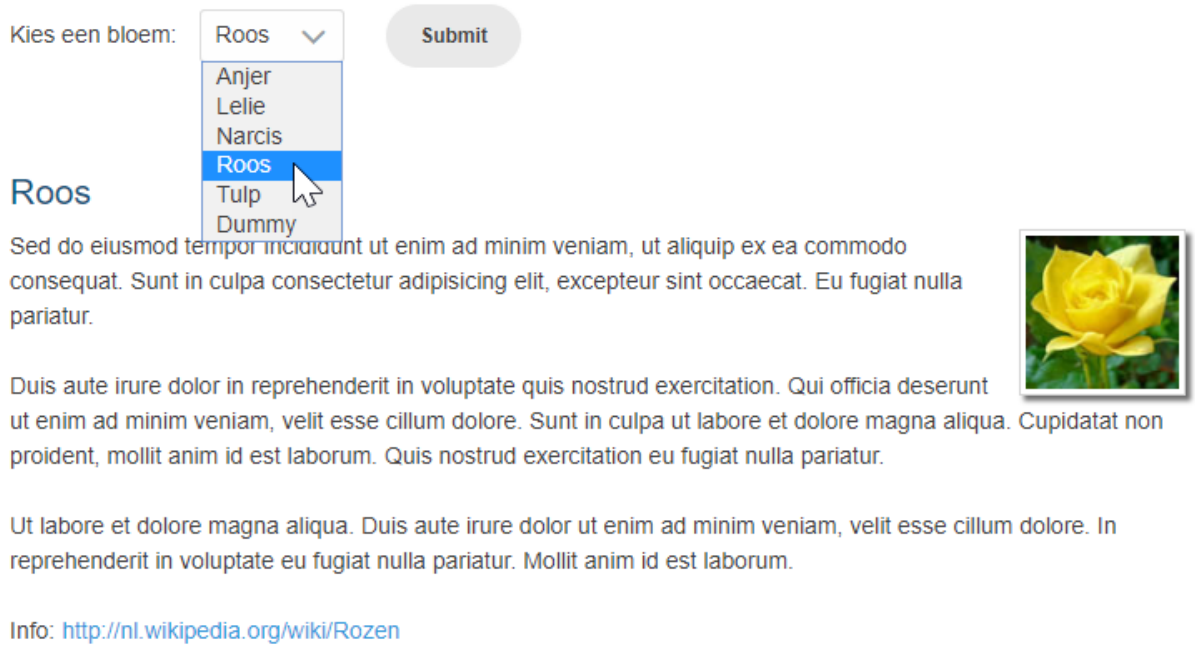
Master/detailrelatie met $.get()

Open **ajaxServer/bloemenMaster2.html** en bekijk de broncode.

Deze pagina reageert net hetzelfde als voorgaande toepassing maar nu wordt de detailpagina opgevraagd vanuit een keuzelijst.

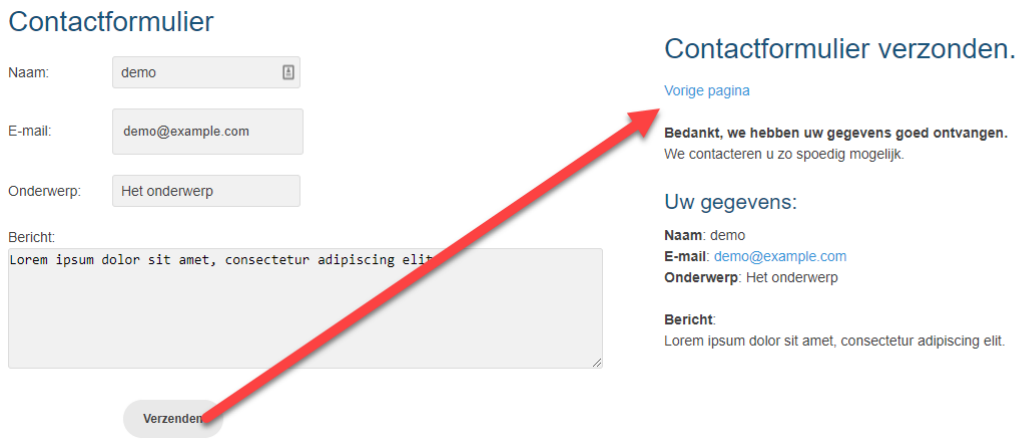
```
$('#form').submit(function(e) {
    e.preventDefault();
    var str = $(this).serialize();
    $.get('bloemenDetail.php', str, function(data){
        $('#detail').html($(data).find('#bloemInfo'));
        $('#bloemInfo p:first').remove();
    }, 'html');
});
$('#id').change( function(){
    $('#form').submit();
});
('#form').submit();
```

De selectie binnen de keuzelijst wordt in de variabele str verwerkt. Voor de keuze Roos wordt str gelijk aan id=4.
Binnen de callback van `$.get()` selecteren we enkel de section-tag #bloeminfo en plaatsen deze inhoud in #detail. Vervolgens de link verwijderen en de toepassing is klaar.



Contactformulier verzenden met $.post()

Open **http://jquery3.test/ajaxServer/contactForm.html** in een browser en verzend het formulier.



De pagina contactSend.php leest alle waardes uit het formulier en verwerkt deze via PHP in contactSend.php. (In deze demo wordt natuurlijk geen echte mail verstuurd.)

> Bekijk vanuit de browser de broncode van
> **http://jquery3.test/ajaxServer/contactSend.php**

De boodschap die binnen `<section id="message">` verschijnt, gaan we nu in contactForm.html opnemen.

> Open **ajaxServer/contactForm.html** en bekijk de broncode.
> Wijzig de selector van `$('#_').submit(...)` in `$('#form').submit(...)`.

```
$('#form').submit(function(e) {
    e.preventDefault();
    var str = $(this).serialize();
    $(this).hide();
    $.post('contactSend.php', str, function(data){
        $ ('#detail').html($(data).find('#message'));
    }, 'html');
});
```

De gegevens uit het formulier worden eerst met `$(this).serialize()` omgevormd en vervolgens via `$.post()` verzonden. Net zoals in voorgaande toepassing gaan we binnen de callback de juiste gegevens filteren en in #detail tonen.

> Open **http://jquery3.test/ajaxServer/contactForm.html** opnieuw in een browser en test het resultaat.

Cross-site scripting (XSS)

Met Cross-site scripting, afgekort als XSS, plaatsen we op onze webserver een proxyscript. Dit script haalt een extern bestand op en toont als resultaat een exacte kopie van de externe pagina. Door een

AJAX call naar de proxypagina te maken, kunnen we op deze manier de same-origine policy beveiliging van de browser omzeilen.

Voor we de werking van het script bekijken gaan we nog even de PHP-configuratie controleren. De proxy maakt namelijk gebruik van de cURL-functie van PHP en deze is in de default configuratie van WAMPserver nog niet actief.

Open **http://jquery3.test/ajaxServer/XSS/curl.php** in een browser.

PHP cURL

De PHP-functie cURL is actief.

PHP cURL

De PHP-functie cURL is niet actief !!!

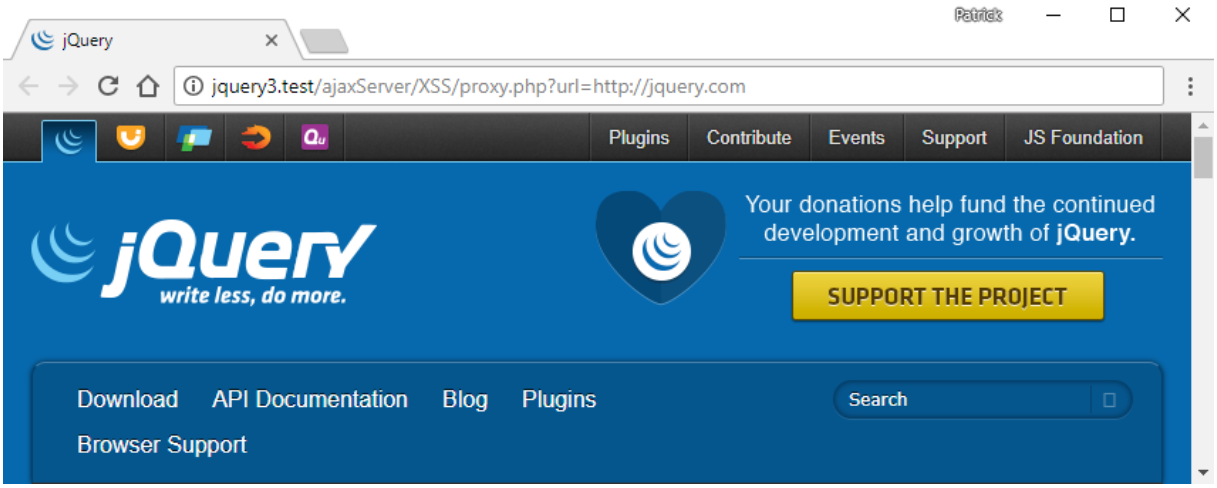
Gelieve deze extensie eerst te activeren.

Indien de achtergrond groen kleurt, is de configuratie in orde en kan u verder met de oefeningen. Verschijnt de foutboodschap op een rode achtergrond, dan moet u de PHP-configuratie nog aanpassen!

Het proxyscript

Het proxyscript is een universeel script waarmee u zowel HTML, JSON als XML kan ophalen. Geef de gewenste URL via de stringparameter url mee aan het script.

Open volgende URL in een browser:
http://jquery3.test/ajaxServer/XSS/proxy.php?url=http://jquery.com



Stel dat de URL zelf ook nog stringparameters heeft, dan moeten we de string eerst coderen, anders loopt het grondig mis. Neem bijvoorbeeld:

http://jquery3.test/...../proxy.php?url=https://www.google.be/search?hl=nl&q=jquery

De URL bevat een &-teken en dit wordt door het proxyscript geïnterpreteerd als een eigen variabele, waardoor de pagina een foutief resultaat toont. Om dit te vermijden gaan we de URL via de JavaScriptmethode `encodeURIComponent()` coderen.

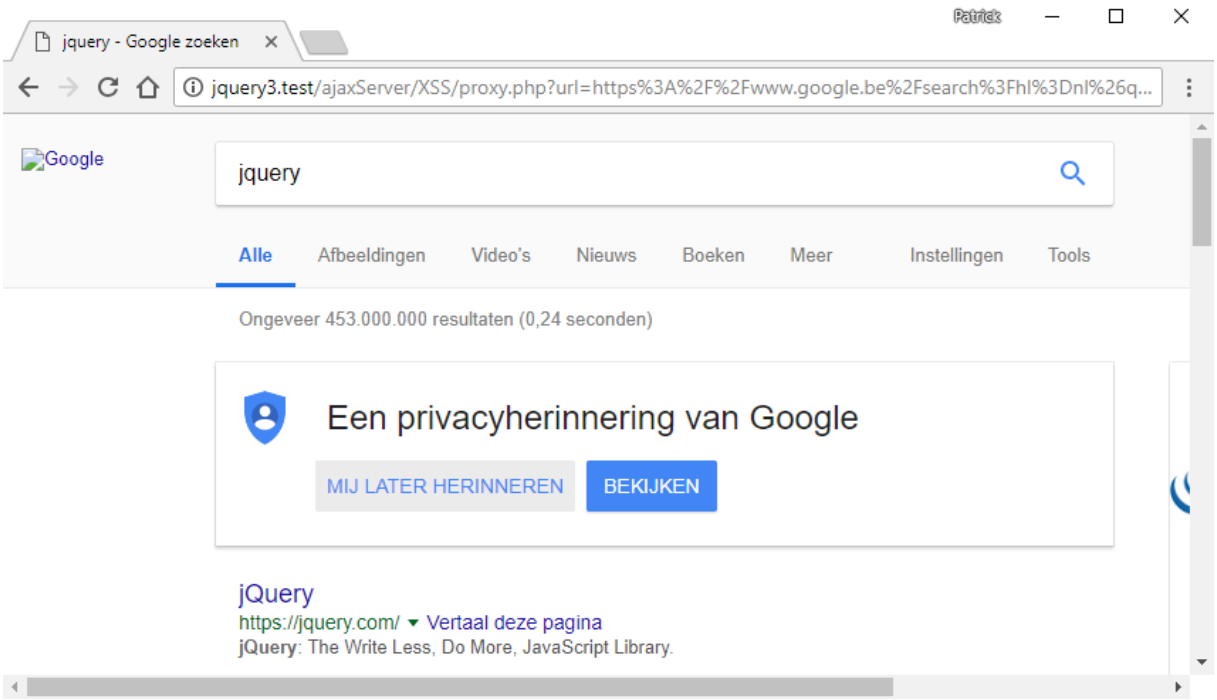
De gecodeerde Google URL wordt dan:

```
https%3A%2F%2Fwww.google.be%2Fsearch%3Fhl%3Dnl%26q%3Djquery
```

Als we deze string als URL aan het proxyscript geven, bekomen we wel het juiste resultaat.

> Merk op dat bepaalde afbeeldingen op de pagina ontbreken. Alle afbeeldingen met relatieve paden worden niet getoond. Afbeeldingen met absolute paden zijn zichtbaar.

Werkt u op Windows hosting, gebruik dan het proxyscript proxy.asp. De werking is identiek aan proxy.php.



In dit hoofdstuk gaan we van elk datatype (HTML, JSON en XML) een voorbeeld uitwerken.

Net zoals in voorgaand hoofdstuk zijn we ook hier volledig afhankelijk van services van derden. Het is dus best mogelijk dat bij het uittesten van de voorbeelden in dit boek bepaalde oefeningen niet meer het gewenste resultaat opleveren.

JSON: Google Geocoding

Met de Google Geocoding API kan u de coördinaten van een straatnaam of van een volledig adres opvragen. Het resultaat wordt default in JSON-formaat aangeboden.
https://developers.google.com/maps/documentation/geocoding/

Open **http://jquery3.test/ajaxServer/XSS/googleGeo.html** in een browser.

Google Geocoding

- Voorbeeld JSON: Vogelzang - Mijr

Zoek plaats: Vogelzang Submit

3 locaties gevonden

2114 Vogelenzang, Nederland
52.3196324,4.5780826

> Open de twee voorbeeldlinks (Vogelzang en mijr) bovenaan de pagina.
> Bestudeer de JSON-structuur.

Indien de zoekopdracht geen resultaat oplevert, is results leeg en geeft `status zero_results`.

```
{
    "results" : [],
    "status" : "ZERO_RESULTS"
}
```

Bij een zoekopdracht met resultaat, staat status op OK. Results bevat nu deze structuur:

```
{
    "results" : [
        {
        "address_components" : [...],
        "formatted_address" : "Vogelzang, 9870 Zulte, België",
        "geometry" : {
            "bounds" : { ...},
            "southwest" : {...}, "location" : {
                "lat" : 50.93021560,
                "lng" : 3.46115240
            },
            "location_type" : "GEOMETRIC_CENTER",
            "viewport" : {...}
        },
        "types" : [ "route" ]
        }, ...
        {...}
    ],
    "status" : "OK"
}
```

De waardes die ons interesseren zijn:

- `results.formatted_address`

- `results.geometry.location.lat`
- `results.geometry.location.lng`

Net zoals in één van voorgaande toepassingen, kunnen we de coördinaten lat en lng opnieuw gebruiken om een statische map te genereren.

> Open **ajaxServer/XSS/googleGeo.html** en bekijk de broncode.

```
$(function () {
    $('#mapsForm').submit(function (e) {
        e.preventDefault();
        $('.preloader-success').show();
        $('#aantal').text('');
        $('#map').empty().hide();
        var str = $(this).serialize();
        var url =
'http://maps.googleapis.com/maps/api/geocode/json?sensor=true&';
        var url = encodeURIComponent(url + str);
        $.getJSON('proxy.php?url=' + url, function (data) {
            if (data.status != 'OK') {
                $('#aantal').text('Geen locaties gevonden');
            } else {
                var aantal = data.results.length;
                $('#aantal').text(aantal + ' locaties gevonden');
                $.each(data.results, function (index, value) {
                    var adres = this.formatted_address;
                    var lat = this.geometry.location.lat;
                    var lng = this.geometry.location.lng;
                    var map = 'https://static-
maps.yandex.ru/1.x/?lang=en-US&z=14&l=map&size=600,200&ll=' + lng +
',' + lat + '&pt=' + lng + ',' + lat + ',pm2rdl1';
                    $('#map').append('<p><b>' + adres + '</b><br>' +
lat + ',' + lng + '</p>');
                    $('#map').append('<p><img src="' + map +
'"></p>');
                });
                $('#map').show();
            }
        }).done(function () {
            $('.preloader-success').hide();
        }).fail(function (err) {
            console.log('error', err);
        });
    });
});
```

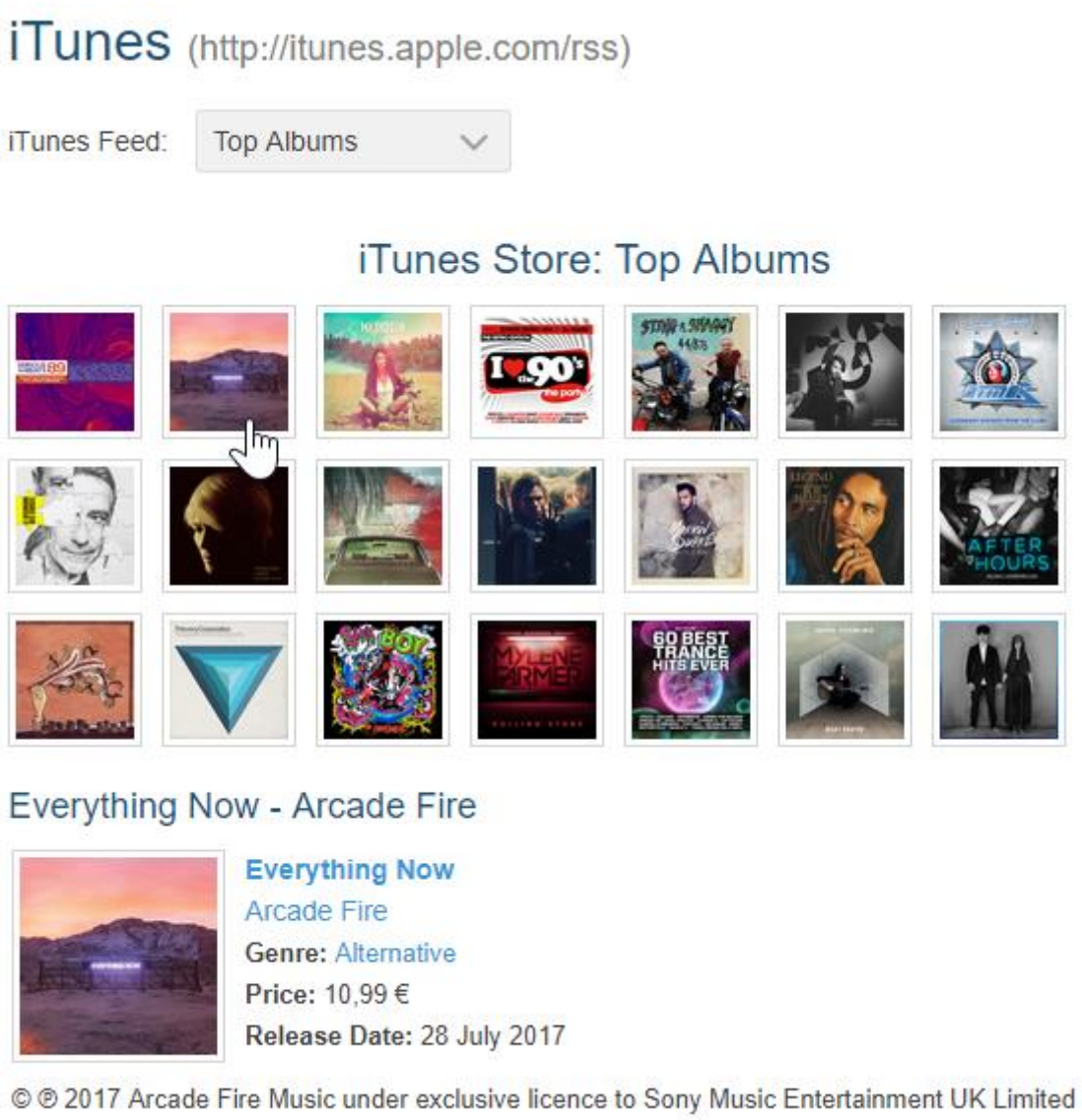
Telkens we nieuwe coördinaten opvragen, gaan we eerst de oude gegevens uit de section-tag #map wissen. Vervolgens wordt de URL van de op te vragen pagina samengesteld. Merk op dat de URL een &-teken bevat waardoor we de URL moeten coderen met `encodeURIComponent()`.

Binnen de methode `$.getJSON()` wordt eerst de status gecontroleerd. Enkel indien de response resultaten bevat, gaan we de gewenste waardes binnen een each-lus uitlezen. De afbeelding wordt opgehaald en #map wordt opnieuw gevuld.

XML: iTunes feed

In het laatste voorbeeld gaan we een iTunes RSS-feed in onze pagina verwerken.

> Open **http://jquery3.test//ajaxServer/XSS/iTunes.html** in een browser.



Via de link http://itunes.apple.com/rss kan u desgewenst een eigen feed maken en deze aan de keuzelijst toevoegen.

> Open een willekeurig gegenereerde feed in Firefox of Internet Explorer
> Bestudeer de XML-structuur.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns:im="http://itunes.apple.com/rss" ...>
    ...
    <title>iTunes Store: Top Albums</title>
    <entry>
        <im:image height="55">...</im:image>
        <im:image height="60">...</im:image>
        <im:image height="170">...</im:image>
        <content type="html">&lt;table ... /table&gt; </content>
    </entry>
    ...
</feed>
```

Het document is gestructureerd volgens een ATOM-feed. Apple heeft echter een eigen namespace aan de feed toegevoegd. Alle iTunes gerelateerde nodes beginnen met im:.

In onze toepassing gaan we drie nodes verwerken. De titel boven de thumbnails halen we uit de node titel. Het pad voor de thumbnails is de eerste im:img node. De volledige omschrijving staat in html binnen de node content.

Klik op een thumbnail om de omschrijving te tonen.

Open **ajaxServer/XSS/itunes.html** en bekijk de broncode.

```
$('#thumbs, #detail').hide();

$('#url').change(function (e) {
    $('.preloader-success').show();
    $('#thumbs, #detail').empty();
    var url = $(this).val();
    $.get('proxy.php?url=' + url, function (data) {
        $('#thumbs, #detail').show();
        $('h2').text($(data).find('title:first').text());
        var entry = $(data).find('entry');
        $.each(entry, function (index, value) {
            //Eerste selector (im\\:image:eq(1))  voor FF en IE,
tweede (image:eq(1)) voor Chrome en Safari
            var img = $(this).find('im\\:image:eq(1),
image:eq(1)').text();
            var title = $(this).find('title').text();
            var content = $(this).find('content').text();
            $('#thumbs').append('<img src="' + img + '" data-id="' +
index + '">');
            $('#detail').append('<div id="entry' + index + '"><h3>'
+ title + '</h3>' + content + '</div>');
        });
        $('#detail div').not(':first').hide();
    }, 'xml').done(function () {
        $('.preloader-success').hide();
    }).fail(function (err) {
        console.error('error', err);
    });
}).change();
```

```
$('#thumbs').on('click', 'img', function () {
    var id = $(this).data('id');
    $('#detail div:visible').hide();
    $('#entry' + id).show();
});
```

Het uitlezen van de afbeelding `im:image` vraagt wat extra uitleg. Voor Chrome en Safari hoeft u geen rekening te houden met de prefix. Omdat de pagina drie dezelfde nodes bevat en we enkel de eerste node met de kleinste afbeelding gebruiken, wordt de selector `image:eq(1)`. Voor Firefox en Internet Explorer is de prefix wel belangrijk. Een dubbelpunt moet u wel altijd laten voorafgaan door twee backslashes. De selector wordt dus `im\\:image:eq(1)`. Door beide te combineren, werkt de selector `find('im\\:image:eq(1), image:eq(1)')` in alle browsers.

Binnen de each-lus worden alle afbeeldingen aan #thumbs toegevoegd. De index wordt via het attribuut data-id aan de afbeelding toegevoegd.

Voor de eerste thumbnail wordt dit: `<img src="https://...xxx.jpg" data-id="0" >`.

Dit nummer gebruiken we straks om de juiste details te tonen.

Alle content nodes komen in een div-tag binnen #detail te staan. Ook hier verwerken we de index in het id van de div-tag. Voor de eerste div-tag wordt dit: `<div id="entry0">...</div>`.

Op het einde van de each-lus worden alle div-tags, behalve de eerste, onzichtbaar gemaakt.

Telkens men op een thumbnail klikt, wordt de oude div-tag onzichtbaar en de geselecteerde div-tag wordt zichtbaar. Klik bijvoorbeeld op de tweede thumbnail.

De variabele id is dan 1 en div-tag #entry1 wordt zichtbaar.