

HTML 5, CSS 3 en JavaScript



Introductie

Vertel even over jezelf:

- Waar kom je vandaan?
- Heb je programmeerervaring?
- Wat is je huidige rol of wat wil je het liefst gaan doen?
- Reden voor bijwonen training
- Wat zijn je hobby's / interesses of wat doe je graag?

Materiaal

Cursus materialen:

- PDF met slides
- JavaScript eBook Eloquent JavaScript, met online (browser) variant @ <https://eloquentjavascript.net/>
- Code samples @ <https://codepen.io/teacherStijn>
- Projecten @ <https://github.com/TeacherStijn>
- Bonus m.b.t. Micro front-ends @ Visme.co (invite in mail)

Webpagina's bouwen met HTML

Doelstellingen:

- Kunnen benoemen wat tags en attributen zijn
- Het verschil kunnen beschrijven tussen block en inline elementen
- Kunnen benoemen van de belangrijkste HTML componenten volgens de W3 standard
- Een eenvoudige webpagina kunnen maken
- Een eenvoudig formulier aan een webpagina kunnen toevoegen

Introductie HTML

- HTML staat voor Hyper Text Markup Language
- Taal waarin webpagina's worden geschreven
- Bestaat uit codes die we aan tekst kunnen toevoegen (markup betekent verrijken). Dit worden dan voornamelijk 'tags' (**DEMO**).
- Een (elke!) browser zorgt ervoor dat HTML-tekst wordt geïnterpreteerd naar een webpagina
- Huidige versie is 5.0. Met focus op duidelijk onderscheid tussen structuur en opmaak

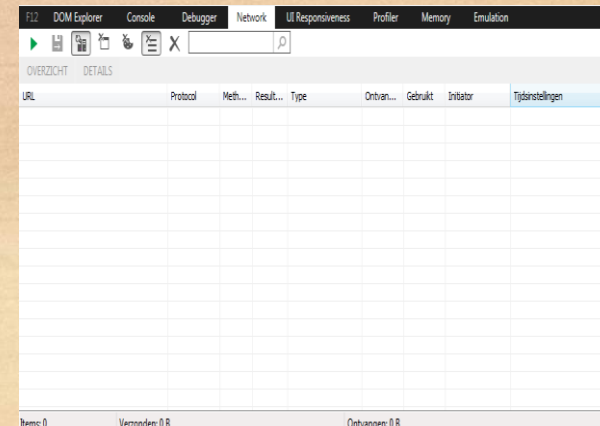
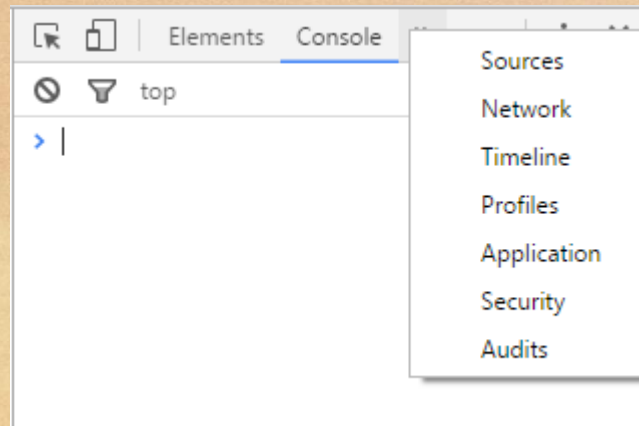
Introductie W3C

Het World Wide Web Consortium, ook wel [W3C](#), is een bedrijf dat zich richt op het opstellen van regels en richtlijnen bij het gebruik van webtalen zoals HTML, CSS en JavaScript.

Als best bruikbare naslag kunnen we navigeren naar [MDN](#).

Debugging

- Foutopsporing voor met name CSS en JavaScript code
- 'Hulpprogramma's voor ontwikkelaars':
 - Google Chrome: <ctrl><shift><i>, óf <f12>
 - Internet Explorwer: <f12>
 - Opera: <ctrl><shift><i>
 - FireFox: gebruik bijvoorbeeld de extensie 'FireBug'
 - OF vanuit je IDE (zoals WebStorm, Vscode of Atom)



Opbouw van een HTML document

- Elementen zijn in HTML beschreven met behulp van tags.
- Een tag wordt geopend met een < en afgesloten met een >. Veel elementen kunnen ook inhoud bevatten.
- Voorbeelden van tags (en daarmee elementen) zijn:
 - `<body>`
 - `<div>`
 - `<p>`
- Iedere tag dient ook afgesloten te worden. Dit is bij de meeste tags middels een gelijknamige tag, met een backslash erin. Respectievelijk: `</body>`, `</p>` en `</div>`.
- Elementen zonder inhoud worden in de openingstag ook afgesloten.
- Zoals: `<input />` en `
`.

Opbouw van een HTML document

De belangrijkste (en volgens W3C essentiële) elementen van een webpagina:

- **<html>** Iedere HTML pagina begint na de doctype met de tag `<html>` en eindigt met de tag `</html>`. Hiermee wordt het begin en einde van een HTML-document gemarkeerd. De browser zal hierdoor herkennen dat het document HTML-code bevat.
- **<head>** Hierin staan de titel van de pagina, informatie over de pagina, eventueel stijlen en eventueel JavaScript code (hierover later meer). De header wordt opgenomen tussen de tags `<head>` en `</head>`. Informatie die in de header getypt is, zal niet in de browser verschijnen.
- **<title>** De titel van de pagina wordt opgenomen tussen de tags `<title>` en `</title>` en komt in de titelbalk van de browser te staan. Deze tag staat altijd *binnen* de `<head>` tag.
- **<body>** Hierin staat de feitelijke inhoud van de pagina. Dit deel begint altijd met de tag `<body>` en eindigt altijd met de tag `</body>`. We zullen later zien dat hierop één uitzondering bestaat.

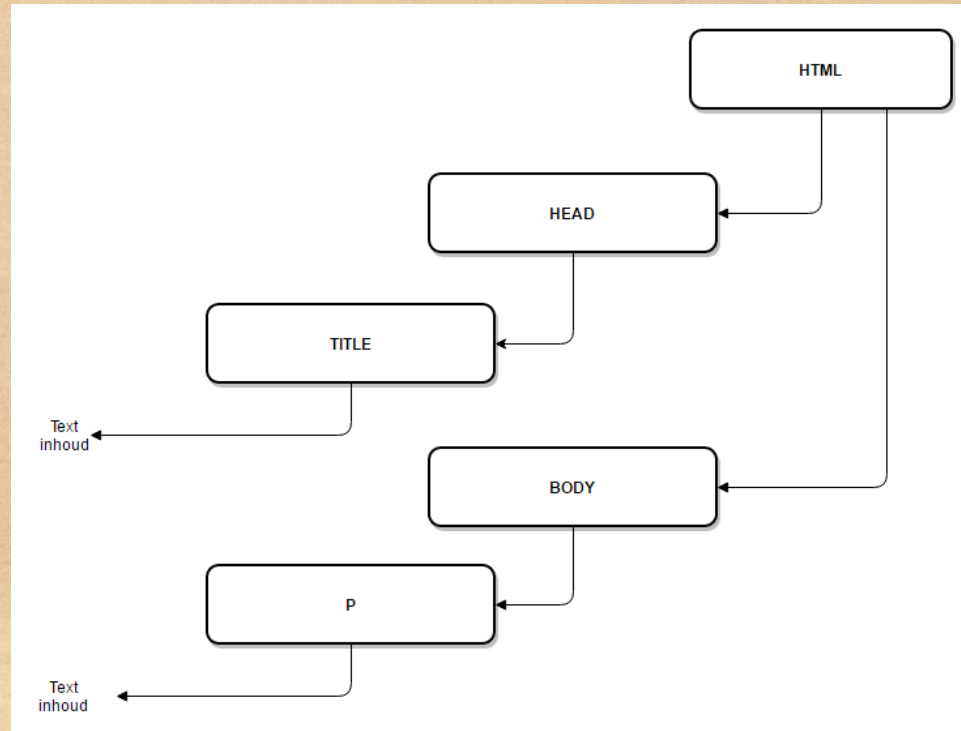
Opbouw van een HTML document

Opbouw basale webpagina volgens W3C HTML 5 standaarden:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Pagina titel</title>
  </head>
  <body>
    </body>
</html>
```


Opbouw van een HTML document

Een webpagina is schematisch goed weer te geven omdat elementen binnen andere elementen geplaatst mogen worden:



Blok- en inline elementen

Elementen die we in de body van de pagina gebruiken, kunnen grofweg in twee structurele groepen worden verdeeld:

Blok elementen: Deze elementen kunnen andere blok elementen en inline elementen bevatten. Blok elementen vormen grotere structuren dan inline elementen.

Belangrijk: Blok elementen beginnen op een nieuwe regel.

Voorbeelden van blok elementen:

```
<div><p><form><table><ol><ul><h1> (t/m) <h6>
```

(DEMO + uitleg elementen)

Blok- en inline elementen

Elementen die we in de body van de pagina gebruiken, kunnen grofweg in twee structurele groepen worden verdeeld:

Inline elementen : Inline elementen kunnen andere inline elementen en data bevatten. Inline elementen maken deel uit van de regel waar ze worden ingevoerd: ze beginnen dus niet op een nieuwe regel.

Voorbeelden van inline elementen: `<a>`

(DEMO + uitleg elementen)

Blok- en inline elementen

Elementen in het algemeen kunnen ook ingedeeld worden naar wat de inhoud ervan mag zijn:

Lege elementen: Dit zijn opmaakelementen zoals regeleinden en lijnen. Ze hebben nooit een eindtag en hebben dus geen inhoud, kortom ze bestaan enkel uit één tag.

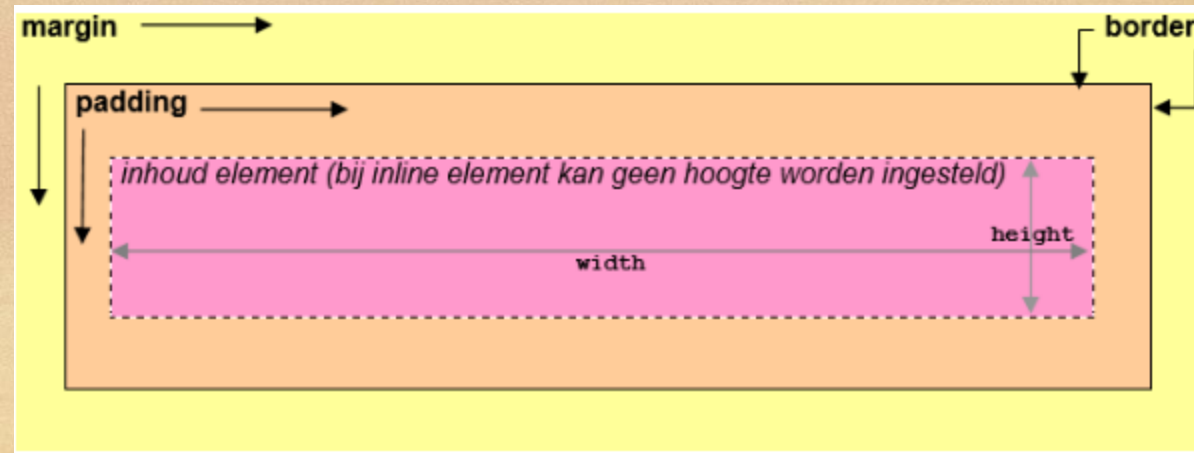
Denk aan: `` en `
`

Container elementen: Deze hebben altijd een eind tag. Tussen de tags bevindt zich de inhoud. De inhoud is altijd het gegeven dat moeten worden gestructureerd en vormgegeven (bv: een tekst).

Denk aan: `<div>`, `<p>` en ``

Ruimtelijke opbouw van elementen

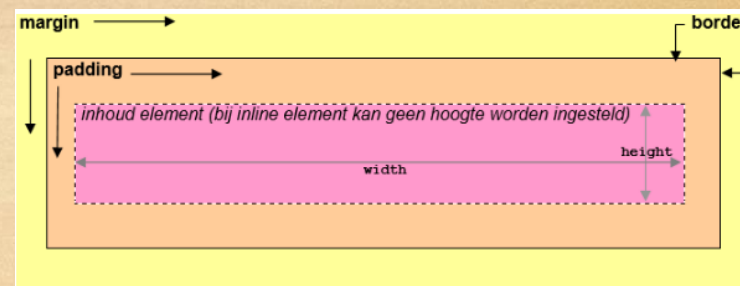
- Elementen zijn de basis van een document. Met behulp van CSS kunnen we verschillende ruimtelijke eigenschappen van een element beïnvloeden.
- Deze ruimtelijk eigenschappen worden in onderstaande figuur weergegeven.



Ruimtelijke opbouw van elementen

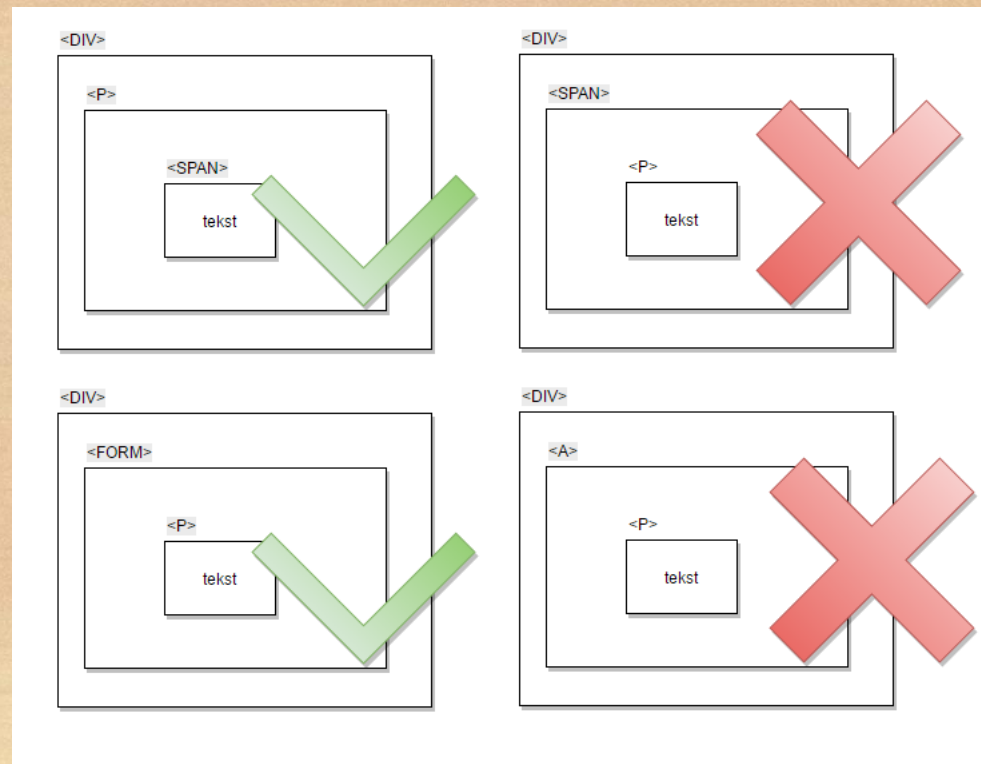
Volgens de CSS definitie horen we te weten dat elk element op het scherm een onzichtbaar kader om zich creëert.

- De *border* is het kader dat om het element heen loopt. Afstand tussen de border en de tekst van het element hangt af van de ingestelde padding.
- *Padding* is de ruimte tussen de inhoud van het element en het kader eromheen. Geen border? Dan is er een onzichtbare van 0px.
- *Margin* is de ruimte tussen de border van het element en de border van aanliggende elementen.
- *Width* en *height* (bij block elementen) betreffen breedte en hoogte van de de inhoud v/h element.



Nesten van elementen

Een schematisch voorbeeld van welke elementen wél, en welke elementen niet binnen elkaar gebruikt mogen worden:



Nesten van elementen

Tenslotte nog enkele punten met betrekking tot de lay-out:

- Lege regels in de HTML-code hebben geen resultaat en worden niet verwerkt in de browser. Ze kunnen enkel de structuur in het document verduidelijken.
- Het afbreken van de regel vindt daarom plaats met behulp van element `
` en niet door in het document een return (enter) in te voeren op de gewenste plaats in de tekst.
- Meerdere opeenvolgende spaties worden in de browser ook gereduceerd tot één spatie.
- Tekst gebruikt standaard de volle breedte van de pagina.

HTML 5 semantische structuur elementen

Deze zeven nieuwe elementen vallen niet onder een categorie block of inline, omdat ze visueel geen effect hebben op de pagina. Ze dienen puur een semantisch doel:

- `<main>` Ten behoeve van vervanging veel gebruikte `<div>` die als 'hoofd' element werden gebruikt.

Let op:

Er mag maar één main element op een pagina zijn. Tevens mag deze géén onderdeel zijn van één van de hierna volgende zes elementen.

HTML 5 semantische structuur elementen

- `<article>` Dit is een element wat gebruikt wordt om aan te geven dat een stuk content bijvoorbeeld een blog (*article*) of forumpost is. Hierdoor zijn daaraan vanuit CSS, maar zeker ook vanuit JavaScript regels aan te koppelen die het werken met dit soort items duidelijker maakt.
- `<section>` Deze tag geeft aan dat elementen die hierin gezet worden bij elkaar horen of minstens gerelateerd zijn aan elkaar. Bijvoorbeeld kopjes, paragrafen en een afbeelding over het thema trainen. Zie het als een div, maar dan met deze beschreven betekenis.

HTML 5 semantische structuur elementen

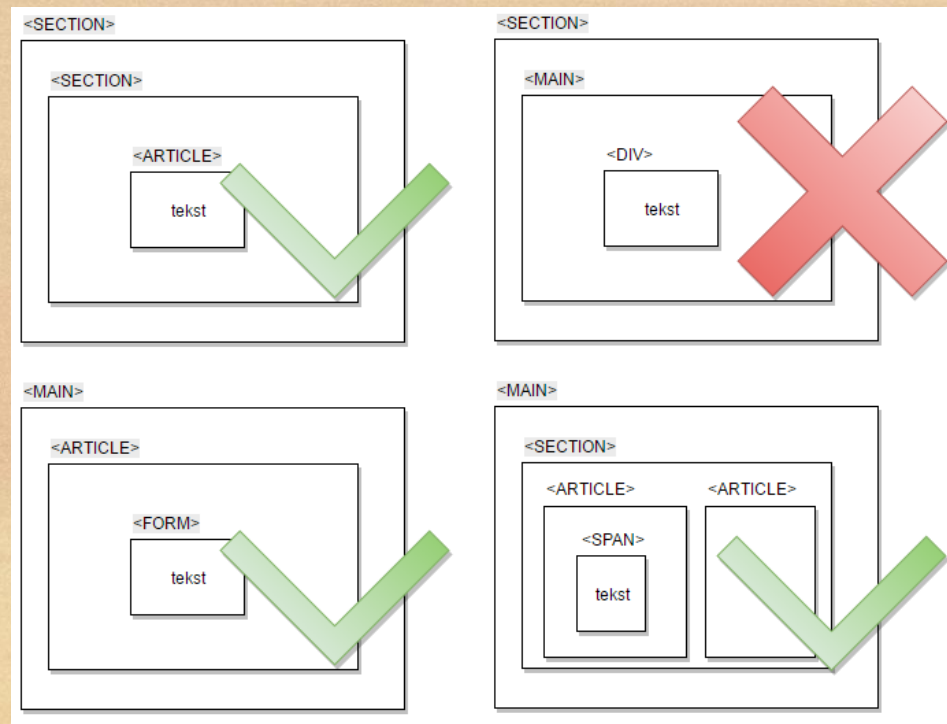
- `<header>` Dient niet verward dient te worden met het `<head>`. Dit element kan meermaals binnen een webpagina worden gebruikt om inleidende informatie te geven over bijvoorbeeld een hele pagina, een sectie of een artikel.
- `<footer>` De tegenhanger van een header element. Ook dit element mag vaker voorkomen binnen één webpagina. Dit element is bedoeld voor informatie over bijvoorbeeld: schrijver van een artikel of webpagina, de algemene voorwaarden, copyright informatie, enzovoorts.

HTML 5 semantische structuur elementen

- `<aside>` Bedoeld om een balk aan de zijkant van de webpagina mee te specificeren (contactinformatie, wordcloud, etc).
- `<nav>` Hierin kunnen wij 'blokken' met navigatie links kwijt. Bedoeld om navigatie-delen van een webpagina te beschrijven. Dus niet iedere link op een pagina hoort in een `<nav>` tag thuis.

HTML 5 semantische structuur elementen

Een schematisch voorbeeld van welke elementen wél, en welke elementen niet binnen elkaar gebruikt mogen worden:



Attributen

- Attributen zijn eigenschappen met eventueel een waarde, die onderdeel zijn van een element.
- De karakteristieken van een element kunnen hiermee nog nauwkeuriger worden bepaald.
- Attributen worden alleen in de begintag van een element opgenomen.
- Bij meerdere attributen per element is de volgorde niet van belang.
- Sommige elementen hebben helemaal geen attributen (zoals ``, `<i>` en `<sub>`).

Voorbeeld attribuut op element: `<input type="text" />`

Attributen

Er zijn een aantal afspraken wat betreft de notatie van elementen (tags) en attributen:

- Tags worden in kleine letters opgenomen (niet verplicht).
- Attributen worden in kleine letters opgenomen (niet verplicht).
- Waarden van attributen worden (in kleine letters) tussen dubbele quotes opgenomen.
- Attribuut en attribuutwaarde worden gescheiden door een is-gelijk-teken (=).

Attributen

We kennen in HTML 5 ook een aantal generieke attributen, waarvan we er enkelen hier zullen toelichten:

- **Accesskey:** toetsencombinatie naar het element
- **Class:** om elementen te categoriseren voor CSS en JavaScript
- **Data-*:** content doorsturen ipv met 'hidden fields'. Veel gebruikt door JavaScript en server talen.
- **Hidden:** is het element (op dit moment) zichtbaar?
- **Id:** variant op het veel gebruikte 'class' element. Zeer toepasbaar indien we een *unieke* naam of referentie naar een element willen maken.
- **Style:** hiermee voegen we CSS stijlen toe aan het element.
- **TabIndex:** een numerieke waarde waarmee wij de zogeheten 'tabvolgorde' kunnen regelen.

Commentaar in HTML

Dit kunnen wij binnen HTML bewerkstelligen met de <!-- en --> tags. Commentaar typen we dan tussen deze twee tags in. Let vooral op het uitroepteken (!) bij de eerste tag:

```
<div>
```

```
    <!-- hier komt straks nog een paragraaf in -->
```

```
</div>
```


Afbeeldingen, hyperlinks en iFrames

Afbeeldingen met het img element

Het opnemen van plaatjes (m.b.v. de tag) kan een webpagina aantrekkelijk maken.

- Is een inline element, dus er kan tekst naast getypt worden.
- Mag in tegenstelling tot andere inline elementen zelf een breedte en hoogte hebben (width en height).
- Heeft veel attributen, waarvan de belangrijkste zijn:
 - **src**: de bron van de afbeelding; een URL of pad.
 - **alt**: tekstweergave indien afbeelding niet toonbaar
 - **title**: tekst bij 'mouse over' en t.b.v. zoekmachines
 - **width** en **height**

Afbeeldingen met het img element

Voorbeeld van een element:

```

```

Is het opgevallen dat het img element met een slash wordt afgesloten? Dat betekent dat er in een image geen andere element of tekst mag komen te staan.

Feitelijk wordt en tag achter de schermen letterlijk vervangen door de *bit representatie* van een fysieke afbeelding.

Afbeeldingen met het img element

Voor de bron (het src attribuut) kunnen wij kiezen voor het gebruik van een absoluut pad of relatief pad:

- **Absoluut pad:** De directory die is opgegeven vanaf de zogeheten root van een schijf. In Windows-kringen wilt dat zeggen: het gehele pad inclusief schijfletter. In Linux wilt dat zeggen: het gehele pad, gezien vanaf de root.

Bijv:

`https://upload.wikimedia.org/wikipedia/en/d/d7/Tingle.png`

Bijv: `c:\games\zelda\tingle.jpg` (deze is onveilig, is niet te gebruiken)

Afbeeldingen met het img element

- **Relatief pad:** De directory die is opgegeven bekeken vanuit ons huidige locatie. Hierbij mag geen drive-letter worden opgegeven en is dus nooit het hele pad naar de file toe, slechts vanuit onze huidige directory het pad naar de file toe.

Bijv: `images/games/zelda/tingle.jpg`

Frames en iFrames

Frames kunnen worden gebruikt om een andere webpagina in te laden binnen de huidige pagina.

- Wordt vaak niet aangeraden wegens:
 - Performance, looks en loss of control

Echter:

- iFrame bieden een optisch beter alternatief én zijn een goede manier om zogeheten Micro front-ends mee op te zetten wegens de mogelijkheden om op onafhankelijke manier je pagina delen op te bouwen

iFrame:

```
<iframe frameborder="0" scrolling="no" src="url_naar_het_boek" width="500" height="500"></iframe>
```


Tabellen, lijsten en speciale tekens

Tabel onderdelen

Bekijk de onderstaande video over de opbouw en het maken van tabellen:

<https://youtu.be/WACK1Alxt3M>

De opbouw van een tabel in een voorbeeld uitgewerkt:

<https://codepen.io/teacherStijn/pen/xxXPbZo>

Lijsten

- Gegevens kunnen we in opsommingen weergeven met behulp van de elementen `` (geordend) en `` (ongeordend).
- Beide soorten lijsten bevatten `` element die de daadwerkelijke content van de lijsten voorstellen.
- Middels het optionele *type* attribuut kunnen wij aangeven welk symbool er voor de opsomming gebruikt wordt. Bijv: *disc* (dicht rondje) en *circle* (open rondje) voor een `` en 1 (1,2,3), a (a,b,c), A (A,B,C), i (i,ii,iii), I (I,II,III) voor een ``.

Lijsten

- Elk ul en ol element hoort één of meer li elementen te bevatten.
- Lijsten kunnen genest worden: in elk li element kunnen één of meer ul of ol elementen opgenomen zijn.
- Indien de tekst langer is dan er op één regel past, wordt deze op de volgende regels ingesprongen weergegeven.

Voorbeeld:

<https://codepen.io/teacherStijn/pen/QWqOwEm>

Speciale tekens

In moderne browsers gaat het weergeven van tekens vaak wel goed, maar in oudere browsers is het gebruik van character entities een must.

< < > > " " ;

& & § § è è

÷ ÷ × × à à

é é ë ë ï ï

â â á á

ö ö ü ü

Formulieren

Het form element

- Definieert het begin en het einde van een formulier
- Het element kan vier attributen bevatten: *method*, *action*, *autocomplete* en *novalidate*.

De attributen '*method*' en '*action*' zijn veruit de twee belangrijkste. Dezen en de algemene werking van een form element zullen we hier bespreken:

<https://codepen.io/teacherStijn/pen/abLVzwo>

Tekstvelden

Een tekstveld maken we middels het `<textarea>` element. Dit heeft een 'rows' en een 'cols' attribuut:

- Rows bepaalt hoe hoog het tekstvak is, uitgedrukt in het aantal rijen (regels) tekst (los v/d input v/d user).
- Cols bepaalt hoe breed het tekstvak is, uitgedrukt in het aantal kolommen (karakters) tekst.

Keuzelijsten

Een keuzelijst maken we middels het `<select>` element.

- Verschillende opties maken we middels `<option>` elementen (binnen het `<select>` element)
- Er kan een default geselecteerde waarde ingesteld worden middels het *selected* attribuut van een `<option>` element

DEMO m.b.t. gebrek aan 'placeholder' attribuut.. Hoe dan?

En voorbeeld:

<https://codepen.io/teacherStijn/pen/PoJOwyE>

Input element

Met dit element wordt een veld gedefinieerd, waarin de gebruiker gegevens kan invoeren. Soort is afhankelijk v/d 'type' attribuut waarde:

- "text": tekstgegevens (al of niet verborgen)
- "checkbox": een selectie m.b.v. aankruisvakjes
- "radio": een selectie m.b.v. keuzerondjes
- "submit": opdracht tot het verzenden van ingevoerde informatie
- "reset": het herstellen van de veldwaarden naar de defaults
- "color": kleuren kiezer
- "range": range kiezer
- "date": datum kiezer
- "file": bestand kiezer

Attributen

- **Pattern**
 - Kan gebruikt worden als extra invoercontrole
 - Maakt gebruik van reguliere expressie volgende POSIX standaard
 - Vooral interessant indien een oude browser nog niet numerieke invoervelden ondersteund.
- **Required**
- **Type** (voor input velden, met als waarden: zie MDN)
- **Min & max**

Validatie

Uitgangspunten:

- Vertrouw de eindgebruiker niet.
- Validatie wil je op zo veel mogelijk plaatsen in de stack doen: in de front-end, back-end, database procedures en met constraints in de tabellen

Validatie in HTML

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation#using_built-in_form_validation

Validatie in JavaScript:

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation#validating_forms_using_javascript

Webpagina's opmaken met CSS

Doelstellingen:

- CSS bestand(en) kunnen toepassen op je HTML pagina('s)
- Elementen kunt opmaken qua kleur, omvang en gebruikte lettertypen
- Elementen kunt positioneren
- Kennis hebben van responsive design overwegingen
- Toepassen van media queries voor toepassen van stijlen op verschillende devices

Webpagina's opmaken met CSS

Manieren om CSS code op je HTML document toe te passen:

- **Embedded:** in de HTML pagina verwerkt, maar wél buiten de beschrijving van de elementen. Vaak wordt bovenaan de pagina een HTML codeblok gemaakt, waarin wij onze CSS code kwijt kunnen.

Voorbeeld:

```
<style type="text/css">
  p { color: green; }
</style>
```


Webpagina's opmaken met CSS

Manieren om CSS code op je HTML document toe te passen:

- **Inline:** zoals de naam al zegt: in de regel. Dus via deze weg kun je CSS code als zogeheten style attribuut toekennen aan een willekeurig HTML element. Kort en snel, maar niet flexibel of beheerbaar.

Voorbeeld:

```
<p style="color: green">
```


Webpagina's opmaken met CSS

Manieren om CSS code op je HTML document toe te passen:

- **Imported:** is een variant op de bovengenoemde linked toepassing. Het importeren van een stylesheet gebeurt hier met het @import commando, binnen <style> HTML element tags.

```
<style type="text/css">  
    @import url(bestandsnaam) mediatype;  
</style>
```


Webpagina's opmaken met CSS

Manieren om CSS code op je HTML document toe te passen:

- **Linked:** is de meest gebruikte manier voor het toekennen van CSS code aan een HTML pagina. Er wordt middels een <link> tag met een href attribuut, verwezen naar een .css bestand dat op de betreffende pagina moet worden toegepast.

```
<link rel="stylesheet" type="text/css"  
href="naam.css" media="all">
```


Meerdere stijl definities

- Wanneer (bijv.) een externe (linked/imported) en interne (embedded) stylesheet zijn gedefinieerd en er vervolgens ook inline styles worden toegepast.
- Er zal één algehele virtuele stylesheet aangemaakt worden.
- De inline style heeft de hoogste prioriteit, gevolgd door de embedded en dan de external style.
- De inline style zal een gelijke stijl die gedefinieerd is in bijvoorbeeld een externe stylesheet, overschrijven.

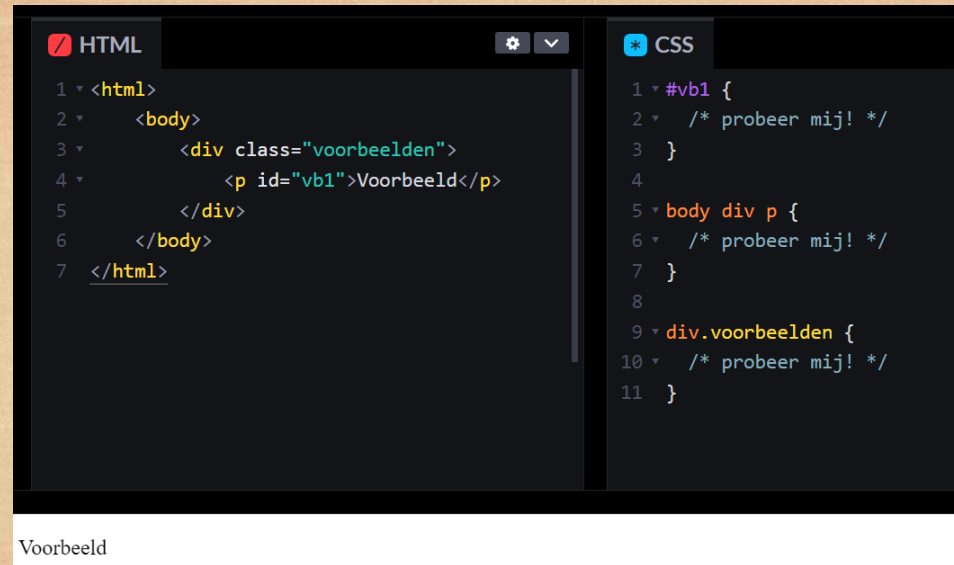
Meerdere stijl definities

Een CSS regel kan met een andere regel overlappen of in conflict zijn

- De meest specifieke regel wint
- Voor elk van de volgende niveaus tellen we het aantal voorkomens van de betreffende selector in de regel. Uiteindelijk plakken we die cijfers aan elkaar:
 - a) het style="..." attribuut binnen een HTML element heeft de hoogste prioriteit
 - b) tel het aantal voorkomens van een id selector
 - c) tel het aantal voorkomens van class selectors en attribuut selectors (element[attribuut] of element[attribuut="..."])
 - d) tel de element en pseudo-element selectors

Meerdere stijl definities

Demo prioriteiten: <https://codepen.io/teacherStijn/pen/KKXypLz>



```
HTML
1 <html>
2   <body>
3     <div class="voorbeelden">
4       <p id="vb1">Voorbeeld</p>
5     </div>
6   </body>
7 </html>

CSS
1 #vb1 {
2   /* probeer mij! */
3 }
4
5 body div p {
6   /* probeer mij! */
7 }
8
9 div.vorbeelden {
10  /* probeer mij! */
11 }
```

Voorbeeld

Welke van de volgende regels zal dan de hoogste prioriteit hebben?

#vb1{color: black;}

body div p{color: green;}

div.vorbeelden {color: red;}

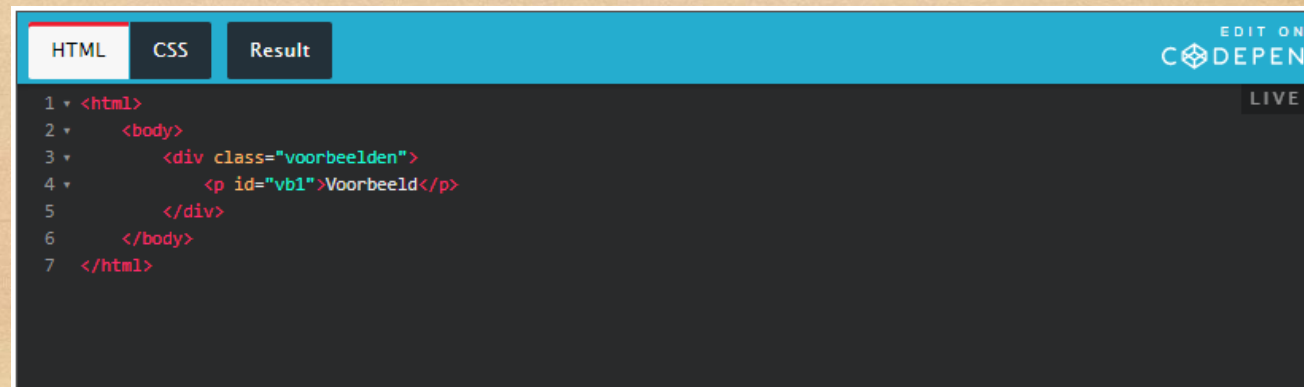
Meerdere stijl definities

In het voorbeeld zal de eerste regel het winnen:

```
#vb1{color: black;}      /* a:0 b:1 c:0 d:0 -> 0,1,0,0 -> score 100 */
```

```
body div p{color: green;} /* a:0 b:0 c:0 d:3 -> 0,0,0,3 -> score 3  */
```

```
div.voorbeelden {color: red;} /* a:0 b:0 c:1 d:1 -> 0,0,1,1 -> score 11 */
```

A screenshot of a CodePen editor interface. The top bar is blue with three tabs: 'HTML' (selected), 'CSS', and 'Result'. On the right of the top bar, it says 'EDIT ON CODEPEN' and 'LIVE'. The main area is dark with red syntax-highlighted HTML code. The code consists of seven lines: 1. <html>, 2. <body>, 3. <div class="voorbeelden">, 4. <p id="vb1">Voorbeeld</p>, 5. </div>, 6. </body>, 7. </html>.

```
1 <html>
2 <body>
3 <div class="voorbeelden">
4 <p id="vb1">Voorbeeld</p>
5 </div>
6 </body>
7 </html>
```


Meerdere stijl definities

Wat nu als er twee conflicterende regels in een stylesheet staan met dezelfde specificiteit?

- De stylesheet wordt van boven naar beneden gelezen en toegepast, dus in dat geval wint de laatste regel.
- Om een regel toch een hogere prioriteit te geven in geval van gelijke specificiteit kan dit worden aangegeven met !important

Voorbeeld:

<https://codepen.io/teacherStijn/pen/JjrOdVp>

Commentaar in CSS

Commentaar notatie

*/ * hier staat code */*

Toelichting

Deze combinatie van een slash en ster (en vise versa), wordt gebruikt om commentaar over meerdere regels in te voegen.

Lettertypen

Lettertype keuze

Fonts worden onderverdeeld in vijf groepen:

- Serif fonts: schreef (kleine decoraties op letters) en verschillende letterbreedte. Voorbeeld fonts: Georgia, Times.
- Sans-serif fonts: verschillende letterbreedte, zonder schreef. Voorbeeld fonts: Arial, Helvetica, Verdana
- Monospace fonts: vaste letterbreedte, met en zonder serifs. Voorbeeld fonts: Andala Mono, Courier, Courier New
- Cursive fonts: extremere ronde vormen en decoraties dan bij serif fonts, vaak gelijkenissen met handschriften. Voorbeeld fonts: Author, Comic Sans
- Fantasy fonts: verzamel categorie, vaak gerelateerd aan verhalen, films, enzovoorts. Voorbeeld fonts: Klingon, Western, Woodblock.

Lettertype keuze

- Om een element een gewenst lettertype te kunnen geven, maken wij gebruik van de **font-family** property.

Voorbeeld:

<https://codepen.io/teacherStijn/pen/yLzPNra>

Meer mogelijkheden - @font-face

We gebruiken de @font-face selector om zelf een nieuw type te declareren:

<https://codepen.io/teacherStijn/pen/WNZXvWv>

(we hoeven hiervan niet alle code te begrijpen)

Verderop in de stylesheet kunnen we naar dit lettertype verwijzen:

```
p {  
  font-family: 'Hobbiton Brush', serif;  
  font-size: 25pt;  
}
```

Zorg er wel voor dat je alternatieve lettertypen opgeeft voor oudere browsers die @font-face niet ondersteunen. In dit geval is de groep 'serif' gebruikt.

Opdracht

Icon font's worden tegenwoordig *enorm* veel gebruikt.
Waarom?

- Voeg zelf een zogeheten 'icon font' toe aan je webpagina
- Gebruik hiervoor de CSS file van 'Material Icons'
- Kijk eventueel hier voor een voorbeeld:

Kleurgebruik, lijsten en animaties

Display eigenschap

- Hoe HTML elementen binnen een tekst worden weergegeven hangt af van de *display* eigenschap.
- Met de display eigenschap kunnen we deze weergave aanpassen, of zelfs helemaal onzichtbaar maken.
- Een veel gebruikte toepassing is het opbouwen van een horizontaal menu.
- Ook kunnen we block elementen inline weergeven of andersom.

Display eigenschap

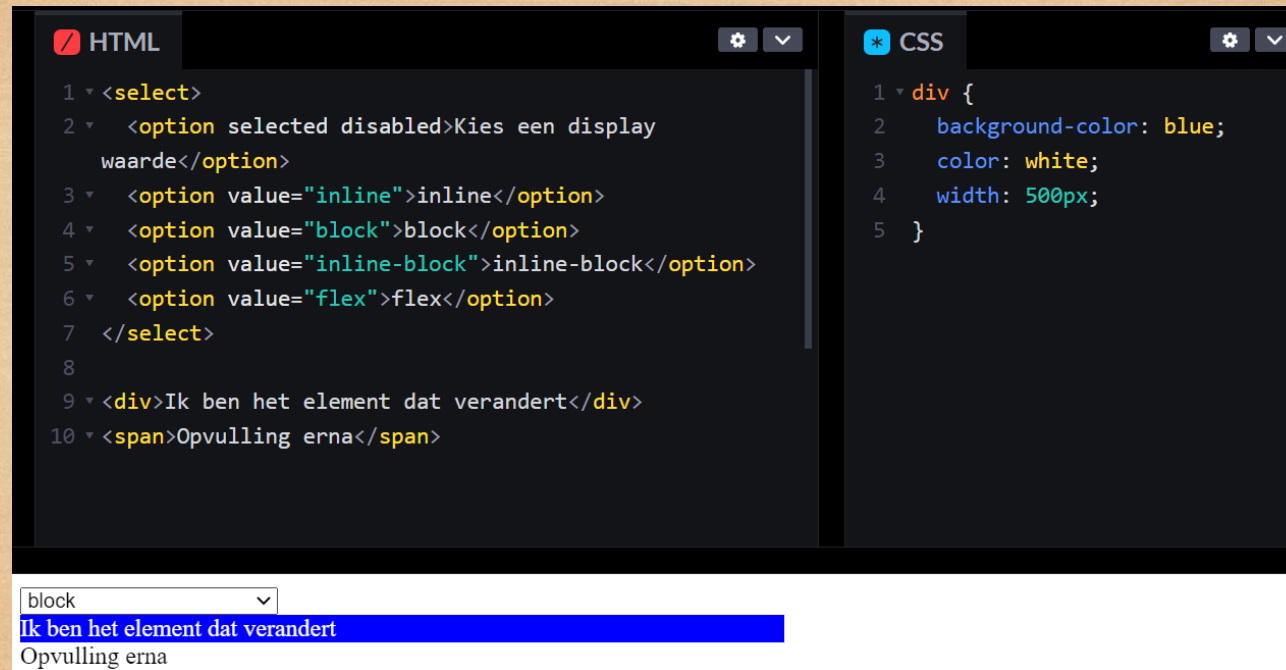
De meest voorkomende waarden zijn:

block, inline, inline-block, flex, grid, table-cell, list-item en none

Er zijn met de komst van CSS 3 een groot aantal andere mogelijke waarden bij gekomen. Omdat dit waarden zijn die vooral op tabellen focussen en in de praktijk niet veel ingezet worden, zullen wij hier niet verder op in gaan.

Een toelichting op de flex en grid sleutelwoorden komt in het hoofdstuk 'positionering' uitgebreid aan bod. De overige mogelijkheden kunt u over lezen op de [MDN website](#).

Display eigenschap



The screenshot shows a code editor with two panels: HTML and CSS. The HTML panel contains a select element with four options: 'inline', 'block', 'inline-block', and 'flex'. The 'block' option is selected. Below the select is a div containing a span with the text 'Opvulling erna'. The CSS panel contains a single rule for the 'div' element, setting 'background-color: blue;', 'color: white;', and 'width: 500px;'. Below the code editor, a preview of the rendered HTML is shown. It features a dropdown menu with 'block' selected, followed by the text 'Ik ben het element dat verandert' in white on a blue background, and 'Opvulling erna' in black on a white background.

```
HTML
1 <select>
2   <option selected disabled>Kies een display
   waarde</option>
3   <option value="inline">inline</option>
4   <option value="block">block</option>
5   <option value="inline-block">inline-block</option>
6   <option value="flex">flex</option>
7 </select>
8
9 <div>Ik ben het element dat verandert</div>
10 <span>Opvulling erna</span>

CSS
1 div {
2   background-color: blue;
3   color: white;
4   width: 500px;
5 }
```

block ▼
Ik ben het element dat verandert
Opvulling erna

Voorbeeld:

<https://codepen.io/teacherStijn/pen/xxXPGMP>

Kleurgebruik

De CSS eigenschappen die ons in staat stellen iets met kleur te doen zullen we hieronder bespreken:

- background-color
- border-color
- color

De waarden (de kleur) voor de genoemde eigenschappen kunnen we invullen met rgb waarden (numeriek of procentueel), met hexadecimale waarden of met kleurnamen.

Kleurgebruik

Zoals u kunt raden, zijn er veel meer mogelijke kleuren te maken dan we in onderstaande tabel tonen. We gaan in de onderstaande tabel echter uit van de beschikbare kleurnamen:

Kleur	Percentage	Numeriek	Hexadecimaal	Kort hex
Red	rgb(100%,0%,0%)	rgb(255,0,0)	#FF0000	#F00
Orange	rgb(100%,40%,0%)	rgb(255,102,0)	#FF6600	#F60
Yellow	rgb(100%,100%,0%)	rgb(255,255,0)	#FFFF00	#FF0
Green	rgb(0%,50%,0%)	rgb(0,128,0)	#008000	
Blue	rgb(0%,0%,100%)	rgb(0,0,255)	#0000FF	#00F
Aqua	rgb(0%,100%,100%)	rgb(0,255,255)	#00FFFF	#0FF
Black	rgb(0%,0%,0%)	rgb(0,0,0)	#000000	F000
Fuchsia	rgb(100%,0%,100%)	rgb(255,0,255)	#FF00FF	#F0F
Gray	rgb(50%,50%,50%)	rgb(128,128,128)	#808080	
Lime	rgb(0%,100%,0%)	rgb(0,255,0)	#00FF00	#0F0
Maroon	rgb(50%,0%,0%)	rgb(128,0,0)	#800000	
Navy	rgb(0%,0%,50%)	rgb(0,0,128)	#000080	
Olive	rgb(50%,50%,0%)	rgb(128,128,0)	#808000	
Purple	rgb(50%,0%,50%)	rgb(128,0,128)	#800080	
Silver	rgb(75%,75%,75%)	rgb(192,192,192)	#C0C0C0	
Teal	rgb(0%,50%,50%)	rgb(0,128,128)	#008080	
White	rgb(100%,100%,100%)	rgb(255,255,255)	#FFFFFF	#FFF

Handige kleur selectie tool! Hier te vinden:

https://www.w3schools.com/colors/colors_picker.asp

List items

Voor het wijzigen van de stijl van lijsten (en), zijn er verschillende CSS properties beschikbaar, waaronder:

- list-style-type
- list-style-image
- list-style-position

Niet alle mogelijke waarden worden hier genoemd. Deze zijn in onze online leeromgeving terug te vinden.

DEMO van alle eigenschappen:

<https://codepen.io/teacherStijn/pen/xxXPGJX>

Animaties

animation

- Bij het animeren met behulp van keyframes maken wij wederom gebruik van een apestaart (@) notatie.
- Dit betekent net als bij het gebruik van @font-face, dat we naar dit deel van de code kunnen verwijzen vanuit een ander stuk CSS code.
- Het idee bij een animatie is dus om bij de animation eigenschap te verwijzen naar de bijbehorende keyframe(s).
- Binnen een @keyframes blok kunnen wij meerdere keyframes (sleutelpunten dus eigenlijk) declareren.
- Ieder keyframe is een zelf bedacht moment binnen de animatie.

Animaties

In het volgende voorbeeld laten we een personage 'bewegen' van klein naar groot met behulp van twee keyframes:

DEMO: <https://codepen.io/teacherStijn/pen/XWezbYp>

Het nu volgende voorbeeld heeft iedere 'animation' eigenschap uitgewerkt:

DEMO: <https://codepen.io/teacherStijn/pen/mdBqJKz>

Borders, padding en margin

Ruimtelijke opbouw van elementen

- Belangrijk is om te weten dat veel browsers bepaalde CSS eigenschappen al voor ons instellen.
- Om dit probleem (deels) tegen te gaan kunnen wij gebruik maken van CSS resets. Dit kan iets heel concreets zijn zoals: `P { margin: 0px; }.`
- Hierdoor wordt dat ons startpunt.
- In de praktijk zijn er al vele CSS reset files ontwikkeld om met de bovengenoemde browser standaard CSS instellingen om te gaan. Eén van de meest gebruikte files hiervoor is *Normalize.css*

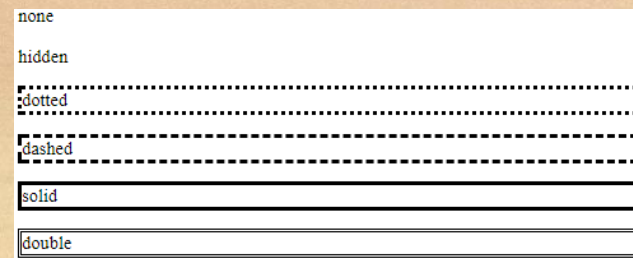
Borders

Voor borders kunnen we drie individuele eigenschappen instellen:

- de vormgeving (style)
- de breedte (width)
- de kleur (color)

border-style

- We kunnen hiermee bijvoorbeeld aangeven of het kader een doorlopend kader moet zijn, of een stippellijn.

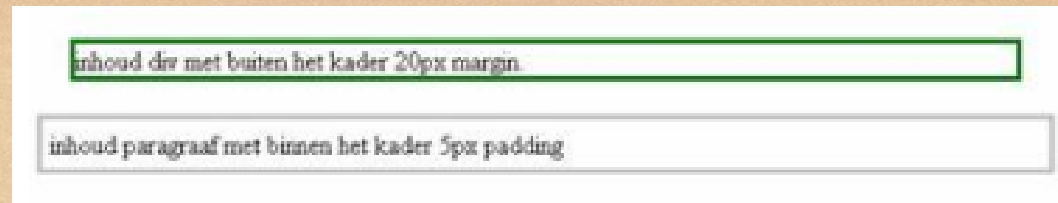


Padding en margin

- De visuele afstand tussen elementen wordt hiermee ingesteld.
- Met margin geven we de afstand aan van een element tot een volgend element.
- Padding is de (evt) ruimte tussen de inhoud en de border van het element.
- Wanneer we bijvoorbeeld tekst of plaatjes die binnen een element staan, verder van het elementkader willen plaatsen, dan moeten we hiervoor de *padding* verhogen. Indien we echter het gehele element (inclusief kader), verder van omringende elementen willen plaatsen, gebruiken we *margin*.

Padding en margin

In dit voorbeeld is gebruik gemaakt van padding op een paragraaf en van margin op een div:



We zien dus dat padding meer 'ademruimte' geeft aan het paragraaf element (binnen de borders). Terwijl we met margin meer ruimte om het gehele element creëren.

margin: default waarde afhankelijk v/d browser

padding: default waarde 1px

Waarden zijn middels de bekende maatvormen op te geven

Box model

- Wanneer we elementen naar eigen wens willen plaatsen op een webpagina, kunnen we o.a. gebruik maken van de *position* en de *float* eigenschap (ander hoofdstuk)
- Binnen CSS kennen we verschillende soorten *boxes*. Dit is een onzichtbaar blok om elementen heen.

Element box

- Elk element kan worden gezien als een box met – van buiten naar binnen – de marge, de border, de padding en de inhoud.
- Als we met CSS de positie van een element willen beïnvloeden, dan heeft dat betrekking op deze hele *element box*, let op: zonder de margin.

Box model

Containing block

- De positie en grootte van een element box wordt meestal bepaald ten opzichte van het omliggende (containing) block element:

```
<div>
```

```
<h1>voorbeeld</h1>
```

```
</div>
```

- Indien een element niet in een block element is geplaatst, dan is het *root element* (het html element) het containing block, ook wel *initial containing block*.

Width en height

- Voor inline replaced elementen kunnen we alleen de breedte instellen, van block elementen ook de hoogte.
- Width en height hebben geen invloed op inline non-replaced elementen, zoals het span element
- De standaard waarde is auto: CSS berekent dan zelf de grootte op basis van de inhoud. Een percentage wordt berekend op basis van het containing block. De lengte kan verder in de eerder genoemde meeteenheden worden uitgedrukt.
- Houd er rekening mee dat de maten width en height betrekking hebben op de inhoud van een element: voor de totale grootte die een element inneemt komen daar nog de padding, border (en margin voor buiten het element) bij.

Width en height

Min-width, min-height, max-width en max-height

- Voor begrenzen van waarden indien geen vaste breedte wordt opgegeven, maar auto of een percentage
- Een kolom met een breedte van 20% is op een telefoon al vrij snel te smal; stel dan bijvoorbeeld een min-width in van 300px.

Extra aandachtspunten

Breedte van een element

- $\text{Width} = \text{breedte} + \text{padding} + \text{breedte border van het element}$
- Wijzigen borderbreedte => de totale element breedte groter

Element horizontaal op pagina uitlijnen

- Breedte instellen op zowel het element als op het parent element.
- Margin instellen op: `0 auto` op het element (waarbij 0 staat voor verticale margin en auto voor de horizontale). Hierdoor wordt deze daadwerkelijk gecentreerd weergegeven.
- Indien je de breedte van in dit geval de div op een percentage instelt (dus ten opzichte van de parent), zal hij mooi meeschalen bij andere beeld formaten.

Extra aandachtspunten

Overflow

- Meer tekst dan element kan bevatten? Tekst zal over de randen van het element 'vloeien'.
- De overflow eigenschap kan de volgende waarden bevatten: *visible, hidden, scroll, auto*
- Vaak ingesteld op 'auto', zodat er een scrollbar verschijnt in het element om zo toch, en netjes, de gehele inhoud te kunnen lezen.

Verticale margins

- Let op het inklappen van verticaal aangrenzende margins.
- Inklappen is alleen van toepassing op margins en niet op borders of padding.
- Gebeurt wanneer 2 elementen boven elkaar een margin hebben (één 'bottom', ander 'top'. De hoogste waarde wordt dan aangenomen!

Positionering

Position property

- Waar een element wordt geplaatst wordt standaard bepaald door de positie en marges van de overige elementen, en van de *document flow* (p verschijnt onder div).
- De document flow geeft aan hoe de volgorde van elementen in een web pagina van invloed is op de positie en grootte van die elementen.
- Met *position* kunnen we deze manier van positioneren van elementen aanpassen.
- Mogelijke waarden: static, relative, absolute en fixed

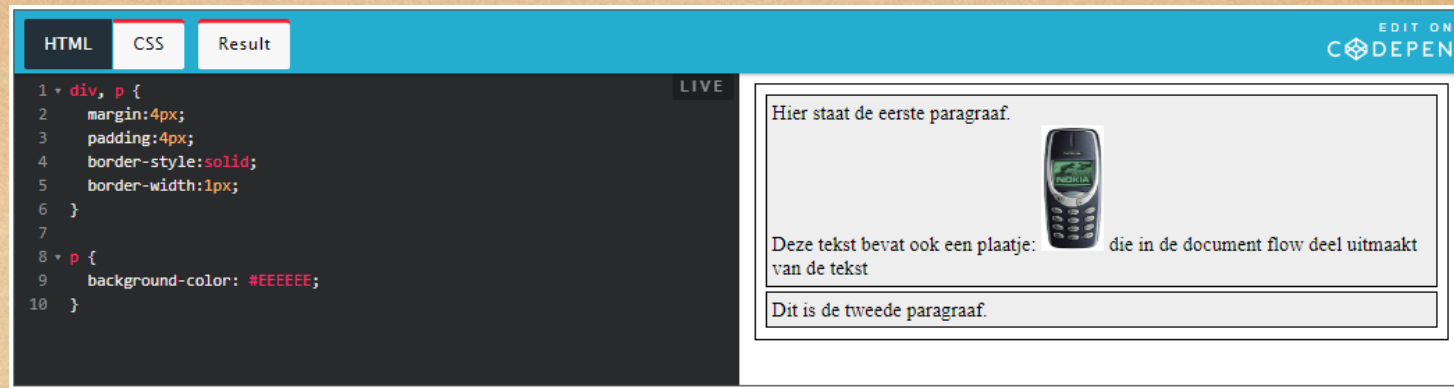
Position property - static

- Dit is de standaard positionering.
- De positie van elementen wordt bepaald door de document flow van de elementen, en van de volgorde van die elementen.
- Bij andere vormen van positionering kunnen we met de afstand van een element ten opzichte van de containing block instellen met waarden voor *top*, *bottom*, *left* en *right* (waarover verderop meer).
- Bij static positionering kunnen dergelijke waarden *niet* worden gebruikt.

Position property - Relative

- Ook hier gaan we uit van de document flow.
- We kunnen de elementen hierbij echter verschuiven ten opzichte van hun normale positie. Dit doen we met behulp van maten voor *top*, *right*, *bottom* of *left*.
- Een positieve waarde voor top betekent bijvoorbeeld dat er aan de bovenkant meer ruimte komt: het element verplaatst dan naar beneden.
- Ook negatieve waarden zijn toegestaan: een negatieve waarde voor *left* betekent dus minder ruimte aan de linker kant: het element verschuift dan naar links.

Position property - Relative



De bovenstaande elementen p en div hebben een border en ze hebben enige ruimte gekregen met behulp van padding en margin. Verder hebben ze een default (static) position.

We zullen nu de positie van het plaatje wijzigen. We moeten daarvoor eerst de position op relative zetten, en vervolgens de ruimte aan de bovenkant vergroten, of de ruimte onder het plaatje verkleinen.

Position property - Absolute

- Het element heeft een vaste positie ten opzichte van het *containing block*.
- Dit containing block moet dan wel een position hebben die niet static is.
- Als zo'n parent element niet wordt gevonden, wordt het initial containing block (het html element) als referentie genomen.

Position property – Fixed

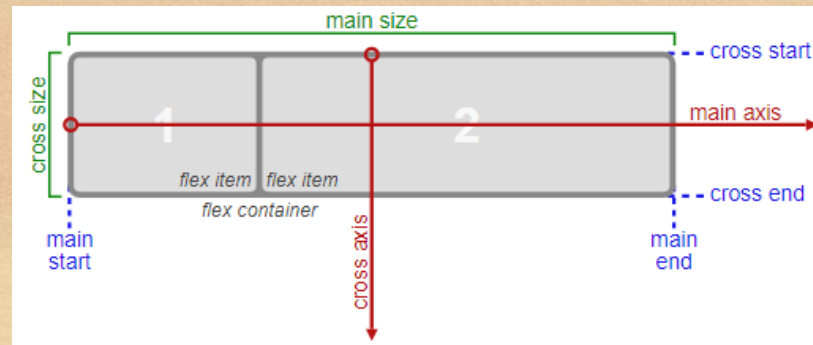
Waarde van position: *fixed*

- Deze vorm van positionering lijkt veel op absolute positionering.
- Het element zal geen deel meer uitmaken van de document flow.
- Er wordt bij fixed positioning niet uitgegaan van het omliggende element voor het bepalen van de positie, maar van het zichtbare deel van het browser venster.
- Een fixed positie wordt onder meer voor kop en voetteksten, reclameblokken en “hulp bij uw aankoop” wizard knoppen.

Het verschil tussen position:*fixed* en position:*absolute* met initial containing block als referentie wordt duidelijk op grote webpagina's, waarbij scrollbars nodig zijn om de rest van de informatie te zien.

Flexbox

- Alternatieve manier van webpagina's opmaken
- Voorheen bedoeld voor netjes uitlijnen van o.a. formulieren
- Bestaat uit flexbox container met daarin flex elementen
- Flexbox element kan weer andere flex elementen bevatten
- Flex element mag in iedere richting worden geplaatst en verkleind qua formaat indien nodig
- Zeer veelzijdig



Flexbox

- Om een flex container te maken, moet de **display** property van het element op 'flex' of 'inline-flex' worden gezet
- Ieder element in een flex container wordt als flex element behandeld, máár:
- Pas zodra een flex element de property **flex** een waarde heeft gegeven, kan deze zich ook pas gedragen als flex element
- De container bepaalt de richting van de elementen in het geheel; horizontale of verticale positionering t.o.v. elkaar

Flex elementen zullen de beschikbare ruimte binnen de container zo optimaal mogelijk benutten

Naslag:

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

Grid

- De vijfde opmaak mogelijkheid naast tabellen, positionering met de 'position' eigenschap, floats en het flexbox systeem
- Goed toepasbaar voor verticale uitlijning en het creëren van tabel-achtige layouts
- Er zijn eigenschappen voor een container element en eigenschappen voor de child elementen (de zogeheten grid items)
- '**display**' property moet worden ingesteld op '**grid**'. Dit vertelt het systeem dat dit element de container wordt voor grid items. De waarden 'subgrid' en 'inline-grid' zijn ook mogelijk, maar vallen buiten de scope van de cursus.

Grid – container eigenschappen

- Een grid is opgebouwd uit kolommen en rijen.
- Wij geven aan hoeveel kolommen en rijen wij willen gebruiken, ieder met de door ons in te vullen breedte (of hoogte, voor rijen).
- Daarnaast kunnen wij iedere kolom en rij een naam geven om later naar terug te kunnen verwijzen.

Voorbeeld:

<https://codepen.io/teacherStijn/pen/poWaKEx>

Responsive web design

Mobile first principe

- Weinig ruimte op een mobiel device (soms zelfs ≥ 200 pixels br)
- Zorgt voor filteren van minder belangrijke content.
- Zorgt weer voor beter semantische organisatie van pagina of form
- Vanuit performance oogpunt: loopt het goed op een mobiel device, dan ook op een desktop omgeving.

DEMO 'Device modus' @ developer tools

Viewport

- Het voor gebruikers zichtbare deel van een webpagina.
- Elementen die bijvoorbeeld breder zijn dan de viewport, zijn alleen volledig te zien wanneer de gebruiker (horizontaal) gaat scrollen. => Niet gebruiksvriendelijk dus!
- Het instellen van de viewport voor op je webpagina kan middels een `<meta>` tag:

`<meta name="viewport" content="width=device-width, initial-scale=1.0">`

- Geeft aan dat de aan te houden viewport, de gehele breedte van het device scherm is.
- Daarnaast staat de initial-scale voor hoe ver de pagina bij openen ingezoomd dient te zijn. Standaard is dus 1.0 (geen zoom).

Relatieve afmetingen

- Probeer waar mogelijk je elementen altijd relatieve maten op te geven.
- Hierdoor kunnen de elementen mee schalen met de viewport.
- Kies voor een indicator op basis van een percentage (%) of view-width (vw).
- Voor teksten kan ook em (element fontsize) gebruikt worden. Waarbij bijvoorbeeld *3em* staat voor 3 maal de grootte van het huidige font.

Touch events

- Al lange tijd wordt niet meer enkel met keyboard en/of muis door webpagina's genavigeerd.
- Touch events, ofwel aanraak momenten, nemen een niet te onderschatten grote rol in bij het navigeren van, naar en binnen webpagina's.
- Een klik met de muis niet hetzelfde is als een druk met de vinger.

Richtlijn: houd elementen die wij met onze vingen moeten kunnen aanraken voor een effect minimaal 40 x 40 pixels breed. Dat is gemiddeld de lengte en breedte van een vinger. Een marge van 5px (dus 45x45) is ook prima.

Break points

- Om een pagina op zo veel mogelijk devices goed weer te geven, kunnen we gebruik maken van break points en media queries.
- Op goede responsive webpagina's wordt de opmaak vaak aangepast aan de hand van het scherm formaat. Ieder omslagpunt van scherm formaat is voor de webpagina als break point in te stellen.
- Bij een breakpoint wordt vaak gekeken naar de pagina breedte.
- Twee of drie breakpoints is het meest gangbaar te gebruiken.
- Media queries zijn veelal het middel om breakpoints mee te configureren.

Break points

Een veel toegepaste verdeling is:

- Max-width van 480px voor mobiele telefoons
- Gemiddeld formaat tablets tot 700px
- Grotere devices tot 1024px
- Alles boven de 1024px

Maar houd in gedachten: meer is niet altijd beter!

Media queries

We kunnen een media query op drie plekken in onze code toepassen:

- In een @import commando, om een aparte CSS file aan te geven voor een specifieke viewport; minst snelle variant; moet de eerste regel in <style> blok zijn.
- In een <link> commando, om een aparte CSS file aan te geven voor een specifieke viewport
- In de CSS code, om één of meerdere CSS stijlen aan te geven voor een specifieke viewport

```
1 @media screen and (min-width: 800px)
2 /* browser breedte van minimaal 800px */
3 @media screen and (min-width: 400px) and (max-width: 600px)
4 /* browser breedte van minimaal 400px en maximaal 600px */
5 @media screen and (max-width: 400px), (min-width: 1000px)
6 /* browser breedte van maximaal 400px OF minimaal 1000px */
```


Webpagina's interactief maken met JavaScript

Doelstellingen:

- Kunnen benoemen waarom JavaScript essentieel is voor modern web applicaties
- Variabelen kunnen maken en uitlezen
- De flow-of-control van een web applicatie kunnen aanpassen met behulp van if en else statements en while- en for-loops.
- Kunnen maken van JavaScript objecten en kunnen uitlezen van de eigenschappen ervan
- Functies en eigenschappen van bestaande JavaScript objecten kunnen gebruiken
- Eigen functies en anonieme callback functies kunnen maken
- Gegevens van een API kunnen ophalen en in verwerken in een CRUD web applicatie

Introductie

Van web pagina tot applicatie

Client side

- **HTML:** structuur van de pagina
 - vroeger ook: opmaak
 - nu: semantiek (header, footer, article, section, nav, etc..)
- **CSS:** opmaak van de pagina
 - kleur, grootte, positie, achtergrond, etc...
- **Javascript:** gedrag van de pagina
 - validatie, interactie, feedback, etc.

Server side

- **PHP / Java / Javascript / ...**
 - Vroeger ook: structuur, opmaak, gedrag
 - Nu: URL routing, database access, rendering data (JSON)

Javascript in de browser

De webbrowser voert Javascript code meteen uit bij het laden van de web-pagina, van boven naar beneden.

Functies worden pas uitgevoerd als deze worden aangeroepen.

Vaak worden functies aangeroepen als reactie op een event (bijvoorbeeld: klikken op een knop).

Javascript in HTML pagina

```
<!DOCTYPE HTML>

<html>

  <head>

    <title>Javascript in HTML pagina</title>

    <script>

      alert('Hello World');

    </script>

  </head>

  <body>

  </body>

</html>
```


Gekoppeld Javascript bestand

```
<!DOCTYPE HTML>

<html>

  <head>

    <title>Javascript in extern bestand</title>

    <script src="scripts\hello.js">

      </script>

  </head>

  <body>

    </body>

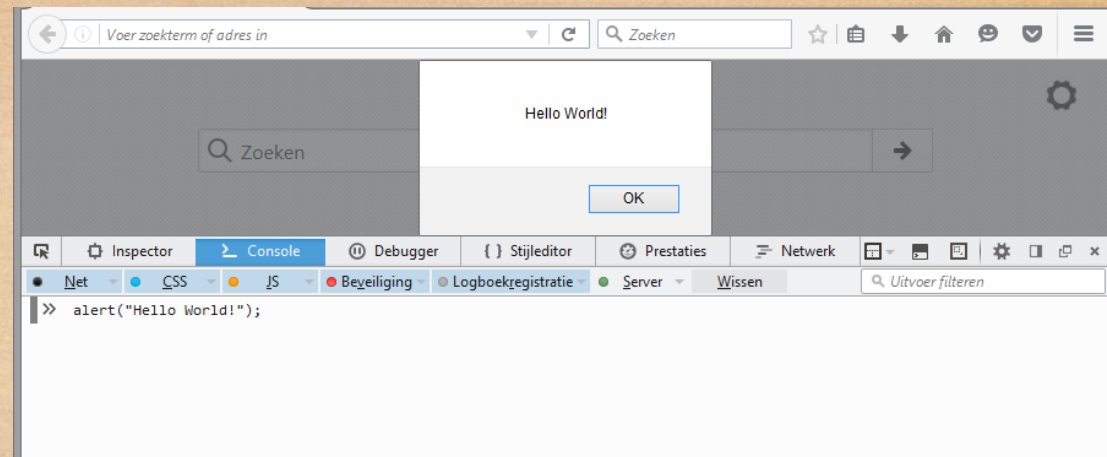
</html>
```

In scripts\hello.js:

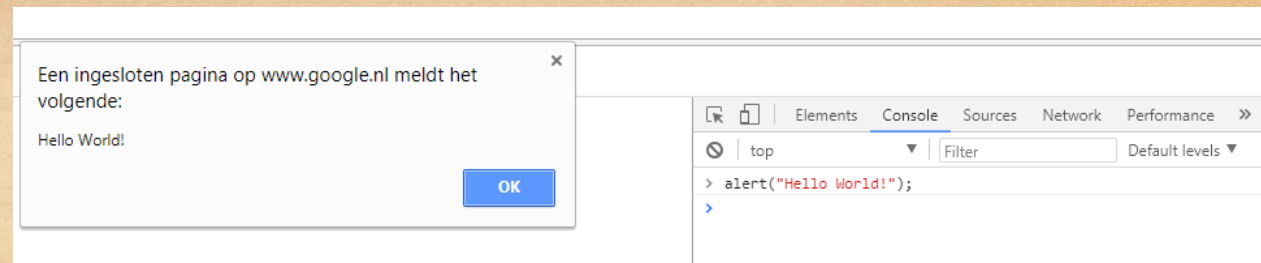
```
alert('Hello World!');
```


In de browser

Firefox Developer Tools



Chrome Developer Tools



Basisbegriffen

Statements

- Een statement is een instructie, zoals: een variabele aanmaken, een variabele vullen, een melding op het scherm tonen, een waarde berekenen.
- Elk statement eindig met ;
- Als ; niet wordt toegevoegd, zal de browser deze zelf toevoegen (automatic semi-colon insertion)
- Statements worden van boven naar beneden uitgevoerd

```
let naam = "Marieke";  
let som = 3 + 4;  
alert("Hello World!");
```


Variabelen

Een variabele is nodig om informatie te bewaren en later te kunnen gebruiken. Een variabele:

- heeft een naam,
- wordt gedeclareerd met `var <naam>` of `let <naam>`
- kan een waarde bevatten (tot die tijd: undefined)
- kan ook leeg worden gemaakt (null)
- kan verschillende soorten data bevatten (loosely typed):
 - string: `let x = "Test"; x = 'voorbeeld';`
 - number: `x = 10.3;`
 - boolean: `let afgerond = false;`
 - array: `let cursussen = ["Javascript", "HTML", "CSS"];`
 - object: `let artikel = { naam: "Pen", prijs: 1.5 };`

Commentaar

Het is goed gebruik om commentaar toe te voegen in Javascript code, zoals:

- doel van de code
- versie
- auteur

Alles tussen `/*` en `*/` wordt als commentaar gezien: de browser zal dat niet proberen uit te voeren

Alles op een regel achter `//` wordt ook als commentaar gezien

Datatypen

Er bestaan twee groepen datatypen:

- primitieve datatypen (immutable)
 - **boolean** (true of false)
 - **null** type (intentionele afwezigheid van een waarde)
 - **undefined** (nog geen waarde toegekend)
 - **number** (64 bits floating point), zoals:
 - +0, -0, NaN (not a number), -Infinity, +Infinity, 2, 20.3
 - **string**: tekstwaarden tussen enkele of tussen dubbele quotes
- complexe datatypes:
 - **arrays**
 - **objecten**
 - **functies**

console.log

Het browser object "console" kan met de functie log(...) debug informatie naar het developer tools scherm schrijven.

Deze informatie komt niet op de webpagina te staan.

Bijvoorbeeld:



Operatoren

Operatoren maken het mogelijk om:

- variabelen te bewerken
- variabelen te vergelijken
- waarden toe te kennen aan variabelen

Bij meerdere operatoren in een statement:

operator precedence (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence)

+ Rekenkundige operatoren

+ (plus)

- (min)

* (keer)

/ (gedeeld door)

% (modulus, rest van een deling)

** (machtsverheffen)

%, * en / hebben voorrang op + en -

** heeft voorrang op de rest

```
console.log(3 + 4 * 5);      // uitvoer 23
```

```
console.log( (3 + 4) * 5 ); // uitvoer 35
```


Strings aan elkaar plakken

Met + kunnen strings aan elkaar worden geplakt.

Eventuele getallen worden daarbij omgezet naar String.

```
console.log(3 + " + " + 4 + " = " + 3 + 4);
```

```
// 3 + 4 = 34
```

```
console.log(3 + " + " + 4 + " = " + (3 + 4));
```

```
// 3 + 4 = 7
```


Assignment operators

Operator	Voorbeeld	Betekent
= (toekenning)	<code>let x = 10; let y = 3;</code>	x is nu 10 y is nu 3
+= (ophoging)	<code>x+=2;</code>	<code>x = x + 2;</code> (x is nu dus 12)
-= (vermindering)	<code>x-=y;</code>	<code>x = x - y;</code> (x is nu dus 9)
= (vermenigvuldiging)	<code>y=x;</code>	<code>y = y * x;</code> (y is nu dus 27)
%= (modulus)	<code>y%=5;</code>	<code>y = y % 5;</code> (y is nu dus 2)
/= (deling)	<code>x/=y;</code>	<code>x = x / y;</code> (x is nu dus 4.5)

Operator	Voorbeeld	Uitleg
<code>==</code> (gelijk)	<code>let x = 3, y = '3'; console.log(x==y); // true</code>	inhoud gelijk, type mag verschillen
<code>===</code> (exact gelijk)	<code>console.log(x===y); // false</code>	inhoud wel, maar type niet gelijk
<code>!=</code> (ongelijk)	<code>console.log(x!=y); // false</code>	inhoud niet ongelijk
<code>!==</code> (exact ongelijk)	<code>console.log(x!==y); // true</code>	inhoud niet ongelijk, maar type wel, dus true
<code><</code> (kleiner dan)	<code>console.log(x<y); // false</code>	x niet kleiner dan y
<code>></code> (groter dan)	<code>console.log(x>y); // false</code>	x niet groter dan y
<code><=</code> (kleiner dan of gelijk aan)	<code>console.log(x<=y); // true</code>	x wel kleiner of gelijk aan y
<code>>=</code> (groter dan of gelijk aan)	<code>console.log(x>=y); // true</code>	x wel groter of gelijk aan y

Condities combineren

- && (logical AND)
- || (logical OR)
- ! (logical NOT)

```
let x = 3;
```

```
let y = 4;
```

```
console.log(x < 4 && y < 4);    // false
```

```
console.log(x < 4 || y < 4);    // true
```

```
console.log(!(x == 4));         // true
```


Programmeerstructuren

Flow of control

Sommige statements moeten conditioneel worden uitgevoerd.

Keuze:

- **als** iets waar is (eenmalig): if (en else), switch
- **zolang** iets waar is: while, do while, for loop
- **voor alle waarden** in een array of object: varianten op de for loop

if

Alleen iets uitvoeren als een conditie waar is:

```
if (conditie)
    statement;
```

of beter:

```
if (conditie) {
    een of meer statements
}
```


if - voorbeeld

```
let x = parseFloat(prompt("geef een getal"));

// vraagt om invoer en zet de tekst om in een getal


if (x < 10) {

    console.log("x is kleiner dan tien!");

}


// uitvoer mogelijk: x is kleiner dan 10
```


if en else

```
let x = parseFloat(prompt("geef een getal"));  
// vraagt om invoer en zet de tekst om in een getal  
  
if (x < 10) {  
    console.log("x is kleiner dan tien!");  
} else {  
    console.log("x is groter of gelijk aan tien!");  
}  
  
// uitvoer mogelijk: x is kleiner dan tien!  
// anders: x is groter of gelijk aan tien!
```


if en else - genest

```
let x = parseFloat(prompt("geef een getal"));
// vraagt om invoer en zet de tekst om in een getal

if (x < 10) {
    console.log("x is kleiner dan tien!");
} else {
    if(x > 10) {
        console.log("x is groter dan tien!");
    } else {
        console.log("x is gelijk aan tien!");
    }
}

// uitvoer mogelijk: x is kleiner dan tien!
// of: x is groter dan tien!
// of: x is gelijk aan tien!
```


if - veel voorkomende fouten

```
let x = 3;  
if (x=2){  
    console.log("x is gelijk aan twee");  
}  
// uitvoer: x is gelijk aan twee
```

```
if(x==4);  
    console.log("x is gelijk aan vier");  
// uitvoer: x is gelijk aan vier
```


switch

Vergelijkt een een variabele met opgegeven waarden.

```
let x = 3;
switch(x) {
  case 1: console.log(1);
  case 2: console.log(2);
  case 3: console.log(3);
  case 4: console.log(4);
}

// uitvoer: 3 EN 4!
```

Switch voert alle code uit vanaf de gevonden waarde, **ook de statements die bij volgende waarden horen.**

switch met break

Vergelijkt een een variabele met opgegeven waarden.

```
let x = 3;
switch(x) {
  case 1: console.log(1); break;
  case 2: console.log(2); break;
  case 3: console.log(3); break;
  case 4: console.log(4); break;
}
// uitvoer: 3
```

Switch voert alle code uit vanaf de gevonden waarde, ook de statements die bij volgende waarden horen!

Na een break; statement wordt het switch blok verlaten

switch met default

Vergelijkt een een variabele met opgegeven waarden.

```
let x = 8;
switch(x) {
  case 1: console.log(1); break;
  case 2: console.log(2); break;
  case 3: console.log(3); break;
  case 4: console.log(4); break;
  default: console.log("anders");
}
// uitvoer: anders
```

Default mag worden toegevoegd, meestal onderaan, maar mag op een andere plek. Als geen van de case waarden voldoet, wordt naar default gesprongen.

while

Voert code uit als, en zolang, de conditie true is.

Binnen blok moet conditie uiteindelijk false worden.

Waarom?

```
x = 1;
while(x<=10){
    console.log(x++);
}
// uitvoer: 1
           2
           3
           ...
           10
```


for loop

De for loop begint met de gehele "loop administratie":

- welke variabele wordt gebruikt?
- hoe lang gaat de loop door?
- wat gebeurt na elke loop?

Veel gebruikt bij het doorlopen van een array.

```
let namen = ["Jan", "Pier", "Joris", "Corneel"];  
for(let i=0;i<namen.length;i++){  
    console.log(namen[i] + " heeft een baard.");  
}
```


for ... of

Nieuwe manier om een array te doorlopen.

```
let namen = ["Jan", "Pier", "Joris", "Corneel"];  
for(let naam of namen){  
    console.log(`${naam} heeft een baard.`);  
}
```


for ... in

Om de eigenschappen van een object te doorlopen.

```
let artikel = { naam: "pen", prijs: 1.5 };  
for(let eigenschap in artikel){  
    console.log(eigenschap, artikel[eigenschap]);  
}
```

Uitvoer:

```
naam pen  
prijs 1.5
```


Opdracht

Gegeven is de volgende array:

```
let prijzen = [10, 3.5, 2.7, 8];
```

Bereken de totale prijs en toon deze op de console.

Gebruik eerst de for loop.

Doe het daarna nog eens met de for ... of loop.

Functions

Wat is een functie?

Een functie is:

- een blok van 0 of meer statements
- met een naam
- en 0 of meer parameters (invoer)
- en optionele uitvoer (return waarde)

Zo kan de groep statements vaker worden uitgevoerd.

Voorbeeld

```
function telop(a, b){  
    // a en b zijn parameters, vergelijkbaar met variabelen  
    // deze zijn alleen binnen de functie bekend  
    return a + b;  
}
```

```
console.log(telop(3,4)); // 7  
console.log(telop(6,7)); // 13
```


Een functie is ook een object

Dit mag ook:

```
let telop = function(a, b) {  
    return a + b;  
}
```

```
console.log(telop(3, 6)); // 9
```


parameters zijn optioneel

```
let telop = function(a, b) {  
    return a + b;  
}
```

```
console.log(telop(2)); // NaN
```

```
telop = function(a, b) {  
    a = a || 0; // als a niet gevuld is: 0  
    b = b || 0; // als b niet gevuld is: 0  
    return a + b;  
}
```

```
console.log(telop(2)); // 2
```


typeof en instanceof

Vaak is controle van datatypen van argumenten gewenst.

- Controle primitieve typen: `typeof`
- Controle complexe typen: `instanceof`

```
console.log(typeof 'test' == 'string');    //true
console.log(typeof true == 'boolean');      //true
console.log(typeof 123 == 'number');        //true
console.log(function() {} instanceof Function); //true
console.log({ x:1 } instanceof Object);    //true
console.log([1,2,3,4] instanceof Array);   //true
```


foutsituatie: throw

```
function telop(a, b){  
    if(typeof a !== 'number' || typeof b !== 'number'){  
        throw 'foute invoer, gebruik getallen';  
    }  
    return a + b;  
}  
  
console.log(telop('test', 4));  
  
// Uncaught foute invoer, gebruik getallen
```


foutsituatie afvangen: try en catch

```
function telop(a, b){
    if(typeof a !== 'number' || typeof b !== 'number'){
        throw 'foute invoer, gebruik getallen';
    }
    return a + b;
}

try { // iets wat mis kan gaan:
    console.log(telop(3,4));      // gaat goed
    console.log(telop('test', 4)); // gaat mis, naar catch
    console.log(telop(4,5));      // wordt niet uitgevoerd
}

catch (fout){ // als er iets mis gaat:
    console.log("Er ging iets mis: " + fout);
}

// 7

// Er ging iets mis: foute invoer, gebruik getallen
```


Opdracht

Maak een functie `maal(a, b)` die `a` en `b` met elkaar vermenigvuldigt.

Zorg ervoor dat het ook werkt als `b` niet is opgegeven (welke waarde krijgt `b` als `b` niet wordt opgegeven?)

Optioneel: maak het ook mogelijk om meer waarden op te geven, bijvoorbeeld:

```
let result = maal(3,4,5);
```

```
console.log(result); // 60
```


anonieme functies

Als een functie maar op één plek wordt gebruikt, hoeft deze geen naam te hebben.

Voorbeeld: doe iets met elke waarde in een array.

```
let namen = [ "Jan", "Pier", "Joris", "Corneel" ];  
  
function toonNaam(naam) {  
    console.log(naam.toUpperCase());  
}  
  
namen.forEach(toonNaam);  
  
// toonNaam wordt uitgevoerd op elke naam  
// en wordt alleen daar gebruikt  
  
  
  
// uitvoer:  
  
JAN  
  
PIER  
  
JORIS  
  
CORNEEL
```


anonieme functies - vervolg

Als een functie maar op één plek wordt gebruikt, hoeft deze geen naam te hebben.

Voorbeeld: doe iets met elke waarde in een array.

```
let namen = [ "Jan", "Pier", "Joris", "Corneel" ];
namen.forEach(function(naam) {
    console.log(naam.toUpperCase());
});
// een anonieme functie wordt uitgevoerd op elke naam

// uitvoer:

JAN
PIER
JORIS
CORNEEL
```


arrow functies

Als een functie maar op één plek wordt gebruikt, hoeft deze geen naam te hebben.

Voorbeeld: doe iets met elke waarde in een array.

```
let namen = [ "Jan", "Pier", "Joris", "Corneel" ];  
namen.forEach(naam => console.log(naam.toUpperCase()));  
// een arrow functie wordt uitgevoerd op elke naam
```

```
// uitvoer:
```

```
JAN
```

```
PIER
```

```
JORIS
```

```
CORNEEL
```


Opdracht

Gegeven is deze array:

```
let getallen = [1,3,5,7,9];
```

Doorloop deze array met `getallen.forEach(...)` en toon elk getal verdubbeld. Doe dit met (naar keuze):

- een zelf gedefinieerde functie
- een anonieme functie
- een arrow functie

closures

Een functie kan ook een functie of een object retourneren.

Deze houdt toegang tot de context waarin de geretourneerde functie is aangemaakt.

```
function counter(startWith, step){
  let result;
  return function() {
    result= result===undefined? // bestaat result nog niet?
      startWith:                // begin dan met startwaarde
      result+step;              // hoog anders op met step
    return result;
  };
}

let teller = counter(0,2); // teller is een functie
                        // die bij elke aanroep de
                        // variabele result kan lezen

console.log(teller()); // 0
console.log(teller()); // 2
console.log(teller()); // 4
```


IIFE

IIFE: Immediately Invoked Function Expression

Anonieme functie die meteen eenmalig wordt aangeroepen.

Onder meer om global scope schoon te houden.

Vaak ook een closure.

```
let teller = (function (startWith, step){  
    let started = false;  
    return function() {  
        if(started) return startWith+=step;  
        started=true;  
        return startWith;  
    };  
})(1,2);  
  
// nu bestaat alleen de functie teller in global scope  
console.log(teller()); // 1  
console.log(teller()); // 3  
console.log(teller()); // 5
```


Live Demo

Toepassing van IIFE's zien wij vaak in de vorm van een afgesloten module:

<https://codepen.io/teacherStijn/pen/PoJOZyE>

```
const huisBeheer = (function(){  
  const data = [];  
  return {  
    voegToe: function(){},  
    bekijk: function(){  
    };  
  })();
```

Bekijk ook dit voorbeeld: <https://codepen.io/teacherStijn/pen/EGVBgp>

Bonus: ECMAScript modules

- Sleutelwoorden 'import' en 'export' maken functies en objecten respectievelijk als verwijzing bruikbaar of juist naar buiten kenbaar.
- `type="module"` bij de `<script>` tag om aan te geven dat we met een module te maken hebben.
- Named exports en default exports (wel of geen klassenaam bij het import statement plaatsen)
- De code dient te draaien op een (al dan niet lokale) webserver(!)

(e.v.t. v.b. code **import_export.zip @ Github/teacherStijn**)

Objecten

Objecten

Objecten zijn verzamelingen eigenschappen met een naam, en waarden.

De waarden kunnen primitieve waarden, arrays, functies en objecten zijn.

Oude manier:

```
let artikel = new Object();  
  
artikel.naam = "pen";  
  
artikel.prijs=1.5;
```

Nieuwe manier, de JavaScript notatie:

```
let artikel = {  
  naam:"pen",  
  prijs: 1.5  
}
```

```
console.log(artikel.naam); // pen
```


Objecten met functies

```
let artikel = {  
  naam: "pen",  
  prijs: 1.5,  
  totaal: function(aantal){  
    return this.prijs * aantal;  
  }  
}  
  
// this.prijs: de prijs van het object  
// waar de functie totaal toe behoort  
  
console.log(artikel.totaal(3)); // 4.5
```


eigenschappen benaderen

Een eigenschap wordt direct benaderd via punt notatie:

```
let artikel = {  
  naam: "pen",  
  prijs: 1.5,  
  totaal: function(aantal){  
    return this.prijs * aantal;  
  },  
  type: { punt: "fijn",  
         kleur: "zwart" }  
}  
  
console.log(artikel.type.kleur); // zwart
```


eigenschappen indirect benaderen

```
let artikel = {  
  naam: "pen",  
  prijs: 1.5,  
}  
  
console.log(artikel["naam"]);           // pen
```

```
function getWaarde(ding, eigenschap){  
  return ding[eigenschap];  
}  
  
console.log(getWaarde(artikel, "prijs")); // 1.5
```

```
for(let prop in artikel){  
  console.log(artikel[prop]);  
}  
  
// pen  
  
// 1.5
```


Constructor function

Een functie kan met **new** <functie> worden gebruikt om een object aan te maken.

Deze functie hoort geen return waarde te hebben.

Wat in de functie aan this wordt toegekend, wordt een eigenschap van het nieuwe object.

Conventie: de naam begint met een hoofdletter.

```
function Artikel(naam, prijs){  
    this.naam=naam;  
    this.prijs=prijs;  
}
```

```
let a = new Artikel("pen", 1.5);  
console.log(a.naam, a.prijs); // pen 1.5  
let b = new Artikel("potlood", 0.5);  
console.log(b.naam, b.prijs); // potlood 0.5
```


prototype

Objecten gemaakt met een functie krijgen een eigen prototype object.

Eigenschappen (met name functies) die voor elk object van dat type hetzelfde zijn kunnen aan het prototype worden toegekend.

```
Artikel.prototype.totaal = function(aantal) {  
    return this.prijs * aantal;  
};  
  
let c = new Artikel("gum", 0.8);  
console.log(c.totaal(2)); // 1.6
```

Meer over prototypes en inheritance: <https://codepen.io/teacherStijn/pen/dqxReL>

Nieuw: class

Met class kan automatisch een constructor function worden aangemaakt, met functies die in het prototype terecht komen:

```
class Artikel {  
    constructor(naam, prijs){ // geen keyword function  
        this.naam = naam;  
        this.prijs = prijs;  
    }  
    totaal(aantal){           // geen keyword function  
        return this.prijs * aantal;  
    }  
}
```


Date

Met de constructor functie Date kan met datums en tijden worden gewerkt.

```
let nu = new Date();  
console.log(nu.getFullYear()); // 2017  
console.log(nu.getMonth());    // 11 (0-based)  
console.log(nu.getDate());     // 28 (0-based)  
console.log(nu.getHours());    // 12  
console.log(nu.getMinutes());  // 53
```

```
let nieuwjaar = new Date(2018,0,0);
```

Date heeft ook set-functies om een Date object te wijzigen.

String properties en functies

String heeft behalve de property `length` ook veel functies.

Deze zijn terug te vinden om MDN (zoek op "MDN javascript String")

```
let tekst = "Dit is een voorbeeld.";

console.log(tekst.length);           // 21

console.log(tekst.toUpperCase()); // DIT IS EEN VOORBEELD.

console.log(tekst.charAt(2));        // t

console.log(tekst.split(" "));

    // ["Dit", "is", "een", "voorbeeld."]

console.log(tekst.substr(4,2));      // is

console.log(tekst.substring(4,6)); // is

console.log(tekst.slice(-6,-1));    // beeld
```


arrays

Een array is een verzameling waarden. Dit kunnen verschillende typen waarden zijn, bijvoorbeeld:

```
let test = [1, "test", {min: 1, max: 4}, [2,3,4]];
```

```
console.log(test[0]);    // 1
```

```
console.log(test[3][0]); // 2
```

```
console.log(test[2].max); // 4
```

Een array is ook aanpasbaar, bijvoorbeeld:

```
test[9]=10; // test.length is nu 10,  
// waarden tussen test[3] en test[9] zijn undefined  
test.length=2; // test is nu [1, "test"];
```


array methods

```
let fruit = ["appels", "peren", "mandarijnen", "bessen"];  
fruit.push("bananen"); // voegt toe aan het eind  
fruit.unshift("kersen"); // voegt toe aan het begin  
let laatste = fruit.pop(); // verwijdert laatste  
let eerste = fruit.shift(); // verwijdert eerste  
fruit.splice(2,1); // verwijdert element op 3e positie  
fruit.splice(1,0,"aardbeien"); //voegt op 2e positie toe  
fruit.splice(1,1,"perziken"); //vervangt op 2e positie  
console.log(fruit.slice(1,-1)); // 2e tot voorlaatste  
let copy = fruit.slice(); // geen argumenten:  
                        // copy van hele array  
console.log(fruit.indexOf("perziken")); // 1
```


array arrow functions

```
let fruit = ["appels", "peren"];

fruit = fruit.map(naam => naam.toUpperCase());

fruit.forEach((naam, i) => console.log(`${i}: ${naam}`));

// uitvoer

0: APPELS

1: PEREN

let getallen = [1,3,5,2,7,9,10,6];

let somEven = getallen.filter(n => n%2==0)

                        .reduce((a,b) => a+b);

console.log(somEven); // 18

getallen.sort(); // [1,10,2,3,5,6,7,9]

getallen.sort((a,b) => a-b); // [1,2,3,5,6,7,9,10]
```


JSON

Formaat voor het versturen/ontvangen van gegevens.

Vergelijkbaar met Javascript object formaat:

- key-value paren
- met beperkingen:
 - key altijd tussen dubbele quotes
 - value:
 - number
 - string tussen dubbele quotes
 - array
 - object (volgens zelfde beperkingen)
 - niet: functie

In bestand: extensie .json

Uitwisseling met server: MIME-type is application/json

JSON voorbeeld

```
let artikel = {
  naam: "pen",
  prijs: 1.5,
  totaal: function(aantal){
    return this.prijs * aantal;
  },
  type: { punt: "fijn",
         kleur: "zwart" }
}

let json = JSON.stringify(artikel);
console.log(json); // {"naam":"pen","prijs":1.5,
                    // "type":{"punt":"fijn","kleur":"zwart"}}

artikel = JSON.parse(json);
console.log(artikel.type.kleur); // zwart
```


AJAX - Asynchronous Javascript And XML

Nu vooral gebruikt om objecten in JSON formaat met server uit te wisselen.

```
let request = new XMLHttpRequest(); // leeg request object

request.onreadystatechange = function(response) { // bij response:

    if (request.readyState === XMLHttpRequest.DONE) { //klaar (4)

        if (request.status === 200) { // gelukt (200)

            let list = JSON.parse(request.responseText); // lees response

            list.filter( item => item.postId<5) // update DOM

                .forEach(item => console.log(item.email));

        } else {

            console.log("Couldn't load comments :(");

        }

    }

};

request.open('GET', // verzend request

'https://jsonplaceholder.typicode.com/comments', true);

request.send();
```


Fetch API

Sinds ES6 kun je ook gebruik maken van het gemak van Promises om gegevens van of naar een server te sturen:

- Promises moeten worden ondersteund in de browser
- Er kan ook een options object worden meegestuurd met o.a. headers voor bijv. Authenticatie
- Ontvangen gegeven moet eerst naar json worden omgezet middels de voor dat object ingebakken json() functie. Dat levert weer een Promise op, waarmee wij met wederom een then() functie abonneren

Voorbeeld:

<https://codepen.io/teacherStijn/pen/GRMOpzo>

Opdracht

Maak een class Boek waarmee boek-objecten kunnen worden aangemaakt. Elk boek-object heeft een auteur en een titel. Geef ook een functie toString(): deze geeft een boek als tekst terug.

Maak vervolgens een array van boeken. Probeer daarin diverse functies van een array uit, zoals:

- alle boeken tonen met forEach
- sorteren op titel
- zoeken op auteur

Zie documentatie bij Array

Opdracht

Haal gegevens op van een API naar keuze (bij voorkeur zonder OAuth authenticatie i.v.m. tijd), maar je mag gerust een JSON api kiezen waarbij een API-key nodig is (en vaak een account voor registratie)

Kies bijvoorbeeld een API uit de volgende lijst:

<https://github.com/TeacherStijn/public-apis>

- Kies een API zonder OAuth en liefst 'No' CORS
- Plak de URL die je vindt in de adresbalk en dan wil je *enkel* JSON code zien
- Gebruik eventueel een JSON formatter om overzicht te houden over welke properties je wilt gaan gebruiken

DOM manipulatie

Het Document Object Model

Het Document Object Model (DOM) is:

- een programmeer interface voor HTML en XML documenten,
- een geheugenpresentatie van de webpagina,
- vormt een boomstructuur in het geheugen,
- kan worden gelezen en aangepast,
- kan weer naar het scherm worden weggeschreven.

DOM uitlezen

Om elementen in document te vinden: document.

functie	geeft
<code>getElementsByTagName(...)</code>	HTMLCollection (elementen)
<code>getElementsByClassName(...)</code>	HTMLCollection
<code>getElementsByName(...)</code>	HTMLCollection
<code>getElementById(...)</code>	element
<code>querySelector(...)</code>	element (gebruikt CSS selector)
<code>querySelectorAll(...)</code>	HTMLCollection (gebruikt CSS selector)

HTMLCollection omzetten naar array: `Array.from(collection);`

Ook vanaf element te gebruiken.

DOM elementen vinden

```
<section id="voorbeeld">
  <a class="item" href="http://www.stijnjanssen.nl">Cursussen</a>
  <a href="http://www.tweakers.net">Tweakers website</a>
</section>
<span class="item">Voorbeeld</span>
```

```
let voorbeeld = document.getElementById("voorbeeld");
console.log(voorbeeld.childElementCount); // 2
let links = document.getElementsByTagName("a");
console.log(links[1].textContent); // Tweakers website
let test = document.getElementsByClassName("item");
console.log(test.item(1).textContent); // Voorbeeld
let test2 = document.querySelector("#voorbeeld .test");
console.log(test2.textContent); // Cursussen
```


DOM elementen uitlezen

Veel gebruikte eigenschappen van een element:

- **textContent**: tekst inhoud van het element
- **innerHTML**: de HTML inhoud van het element
- **children**: de elementen binnen het element
- **parentNode**: het bovenliggende element

DOM attributen uitlezen

In de HTML file:

```
<section id="voorbeeld">  
  <a class="item" href="http://www.stijnjanssen.nl">Cursussen</a>  
  <a href="http://www.tweakers.net">Tweakers website</a>  
</section>  
  
<span class="item">Voorbeeld</span>
```

In de JavaScript file:

```
console.log(links[0].getAttribute('href'));  
console.log(links[0].href);
```


DOM wijzigen

In de HTML file:

```
<div id="result"></div>
```

In de JavaScript file:

```
let link = document.createElement('a');    // maak element
let tekst = document.createTextNode('Cursussen'); // maak tekst
let pad = 'https://www.stijnjanssen.nl'

link.appendChild(tekst);                    // voeg tekst toe aan element
link.setAttribute('href', pad);            // geef attribuut aan element

let result = document.getElementById("result");
result.appendChild(link);
```

HTML resultaat:

```
<section id="result">
    <a href="http://www.stijnjanssen.nl">Cursussen</a>
</section>
```


Alternatief: innerHTML

In de HTML file:

```
<section id="result"></section>
```

In de JavaScript file:

```
const content = 'Klik hier maar even';
```

```
const mijnLink = `\${content}</a>`
```

```
let result = document.getElementById("result");
```

```
result.innerHTML = mijnLink;
```

Zie ook:

<https://codepen.io/teacherStijn/pen/EGVBgp>

class aanpassen

```
<style>
  .rood { color:red;}
  .sterk { font-weight: bold; }
  .onderstreept { text-decoration: underline;}
</style>
<div id="test" class="rood">test</div>
```

```
let test = document.getElementById("test");
```

```
test.classList.add("sterk");
```

```
test.classList.remove("rood");
```

```
if(test.classList.contains("sterk")){
```

```
  test.classList.replace("sterk", "onderstreept");
```

```
}
```

```
test.classList.toggle("onderstreept"); // zet aan/uit
```


Opdracht

- Maak een array met artikelen (namen).
- Genereer hieruit een lijst op het scherm (li-elementen binnen een leeg ul-element).
- Probeer dit eerst met een String template, zoals eerder voorbeeld. Probeer het ook met createElement en appendChild.

Optioneel: geef elk element ook een data attribuut.

Events

Events afhandelen

- Events zijn gebeurtenissen op de pagina, zoals:
 - pagina is geladen
 - er is op een knop gedrukt
 - een veld is ingevuld
- Het reageren op zo'n event wordt **event handling** genoemd
- Daarvoor zijn twee ingrediënten nodig:
 - een **event listener**: wordt wakker als een opgegeven event optreedt
 - een **callback functie**: wordt aangeroepen door de event listener als het event optreedt

Veel gebruikte events

- **window**
 - **load**: pagina is geladen
 - **unload**: pagina is verlaten
 - **beforeunload**: voordat de pagina wordt verlaten
- **element**
 - **focus**: cursor komt in element
 - **blur**: cursor verlaat element
 - **change**: element is gewijzigd
- **form**
 - **submit**: formulier wordt verzonden
 - **reset**: verzenden formulier wordt gecancelld

Keyboard en mouse events

- **keyboard**
 - **keypress**: op toets gedrukt
 - **keydown**: toets is ingedrukt
 - **keyup**: toets is losgelaten
- **mouse**
 - **click**: in element op muis geklikt
 - **mouseover**: muis pointer op element
 - **mousemove**: muis wordt verplaatst

Reageren op een event

Een veel gebruikte event is **het load event** van window: pas dan is het hele document in de DOM geladen.

Dit gaat bijvoorbeeld fout:

```
<html>
  <head><title>lukt niet</title>
  <script>
    var mijnDiv = document.getElementById("test");
    // div met id="test" is hier nog niet bekend
    console.log(mijnDiv.textContent);
  </script>
</head>
  <body><div id="test">Voorbeeld</div></body>
</html>
```


Wacht tot document geladen is

```
<html>

  <head><title>lukt wel</title>

  <script>

    window.onload=function () {

      var mijnDiv = document.getElementById("test");

      // div met id="test" is nu wel bekend

      console.log(mijnDiv.textContent);

    }

  </script>

</head>

  <body><div id="test">Voorbeeld</div></body>

</html>
```


on<event> of addEventListener

- Met on<event> kunnen we een functie koppelen aan een event.
- Een volgend **on<event>** van het zelfde type overschrijft de eerste.
- Niet overschrijven? Dan **addEventListener**(<event>, <functie>, <propagatie>).

```
window.addEventListener("load", mijnFunctie, false);
```

Propagatie: wordt het event vervolgens naar parents doorgegeven (false), of naar children (true).

callback functies

De functie die wordt aangeroepen bij een event wordt "callback functie" genoemd.

Deze krijgt het event zelf als argument mee.

```
<input id="test" type="text">
```

```
var test = document.getElementById("test");  
test.addEventListener("keydown",  
    function(e) { console.log(e.key); }, false);
```


Properties van events

- **target**: het element dat het event heeft afgevuurd
- keyboard events:
 - **key** (de toets zelf, als string)
 - **ctrlKey, metaKey, shiftKey, altKey** (boolean)
- mouse events:
 - **clientX, clientY, screenX, screenY, movementX, movementY**
 - **ctrlKey, metaKey, shiftKey, altKey**

Voorbeeld aanpassen DOM

```
window.addEventListener("click",function(e){  
    var newDiv = document.createElement("div");  
    var newText = document.createTextNode(  
        `positie: [${e.clientX}, ${e.clientY}]`);  
    newDiv.appendChild(newText);  
    document.body.appendChild(newDiv);  
});  
  
// nu worden steeds nieuwe DIV elementen  
// toegevoegd aan de body  
// bijvoorbeeld  
// <div>positie: [18, 27]</div>
```


Element vervangen in DOM

```
window.addEventListener("click",function(e){  
    var newDiv = document.createElement("div");  
    var newText = document.createTextNode(  
        `positie: [${e.clientX}, ${e.clientY}]`);  
    newDiv.appendChild(newText);  
    newDiv.id="test";  
    var test=document.getElementById('test');  
    if(test) // als <div id="test"> al bestaat:  
        test.parentNode.replaceChild(newDiv,test);  
    else // anders nieuw aanmaken  
        document.body.appendChild(newDiv);  
});
```

```
// nu wordt steeds één div vervangen
```


Opdracht – maak een CRUD app van je API data

1) Zorg dat jouw array in losse elementen op de HTML pagina verschijnt (bijv. in LI's)

Bonus: toon je array items in de Bootstrap Cards die we eerder hebben toegevoegd

Tip: gebruik `.innerHTML` property

2) Maak de LI (of div of..) elementen klikbaar door een `.addEventListener()` eraan te koppelen

3) Zorg dat bij klikken erop, er in een andere DIV detail info van dit item wordt getoond

4) En/of maak een klikbare (rode?) button het huidige element verwijderd

Tip: werk de array én de GUI (view) bij

Bonus: Frameworks en Micro front-ends

Doelstellingen:

- Kunnen benoemen waarom Frameworks worden gebruikt
- Kunnen benoemen wat Micro front-ends zijn en wat de overwegingen zijn voor gebruik
- Kunnen uitleggen wat custom componenten zijn
- Zelf een custom component kunnen maken

Materiaal @ Visme op uitnodiging

Workshop

Doelstelling:

Bouw een eenvoudige webshop met een JavaScript DIY module

Gebruik de volgende opdrachtomschrijving:

<https://github.com/TeacherStijn/Casus-JavaScript-Webshop>