Inleiding Web Development

Samenvatting (versie 1.0)

https://jessym.com

Inhoudsopgave

2 Ontwikkelomgeving

2.1 Installatie Visual Studio Code

Het is beter als je voor deze cursus de laatste versie van Google Chrome gebruikt (minstens versie 70). Je kunt vervolgens Visual Studio Code downloaden via https://code.visualstudio.com.

Het is verder aan te raden om zowel op Windows als MacOS in te stellen dat je bestandsextensies kunt zien, aangezien we met een aantal verschillende soorten bestanden zullen werken (`.html` , `.css` en `.js`).

2.2 Werken met Visual Studio Code

Het is slim om een aparte map te maken voor deze cursus, bijvoorbeeld *Inleiding Web Development* op je bureaublad.

Als je deze map vervolgens *sleept* naar Visual Studio Code, of opent via het menu, dan kun je aan het begin van elke les een nieuwe map voor die les aanmaken, rechtstreeks vanuit Visual Studio Code.

3 HTML (Structuur)

3.1 Wat is HTML?

HTML (afkorting voor *HyperText Markup Language*) is een zogenaamde *opmaaktaal*, verantwoordelijk voor de algemene structuur en tekst van een website.

Samen met de CSS en JavaScript vormt HTML een trio van technieken, waar elke website op gebaseerd is.

3.2 HTML Tags

Een HTML pagina is niets meer dan een simpel bestand met de extensie `.html` , bijvoorbeeld: `welcome.html` .

Verder is elke HTML pagina opgebouwd uit zogenaamde *elements*, welke (in de meeste gevallen) bestaan uit zogenaamde *opening* en *closing tags*. Een voorbeeld hiervan is het paragraph element: `<p>I am a paragraph!</p>`

Er zijn wel een aantal verplichte elements waar elke HTML pagina mee moet beginnen, waaronder de *doctype*, het *html* en het *body* element. Zie hieronder bijvoorbeeld een minimale HTML pagina.

```
<!DOCTYPE html>
<html>
  <body>
  </body>
</html>
```

Alle content van een HTML pagina wordt vervolgens toegevoegd aan de `<body>`. Zie hieronder een overzicht van een een aantal belangrijke tags.

- `<h1>`: Het grootste header (titel) element
- `<h6>`: Het kleinste header (titel) element
- `<p>`: Het paragraph element
- `<b>`: Het vetgedrukte element
- `<i>`: Het schuingedrukte element
- `<ol>`, `<ul>`: De genummerde en ongenummerde lijst (ordered en unordered list)
- `<li>`: Het list-item element, te gebruiken in combinatie met list elements

3.3 Block en Inline Elements

Er zijn twee belangrijke soorten HTML elements: *inline* en *block* elements. Een inline element probeert altijd achteraan te sluiten op dezelfde regel als het vorige element, maar een block element eist altijd zijn volledige eigen regel.

Om die reden is het wel mogelijk om meerdere inline elements achter elkaar kunt zetten (in horizontale richting), maar is dit niet mogelijk met block elements.

Elk element is van zichzelf ofwel een inline element, ofwel een block element. Voorbeelden van *inline* elements zijn: `<b>`, `<i>` en `<span>`. Voorbeelden van *block* elements zijn: `<h1>` t/m `<h6>`, `<p>`, `<ol>`, `<ul>` en `<div>`.

De elements `<div>` en `<span>` zijn het schoolvoorbeeld van een inline element en een block element, respectievelijk. Verder zijn deze twee elements volledig *neutraal*, wat inhoudt dat ze geen impliciete witruimte om zich heen hebben, zoals het paragraph element dat bijvoorbeeld wel heeft.

3.4 OPDRACHT: Boodschappenlijst

-

3.5 Tag Attributes

Om een bepaald HTML element aan te passen, is het in sommige gevallen mogelijk om een *attribute* toe te voegen aan dat element. Zo'n attribute wordt altijd toegevoegd aan de *opening* tag van een element, en kan gebruikt worden om bijvoorbeeld de tekstkleur van dat element te wijzigen.

```
<h1 style="color: green;">I am now a green header tag!</h1>
```

In het geval van het `<input />` element (een voorbeeld trouwens, van een zogenaamd *self-closing* element, waarbij de opening en closing tags zijn samengesmolten tot 1 enkele tag) kun je de `type` attribute gebruiken, om aan te geven wat voor *soort* input element dit moet worden. En om daarmee aan te geven of de gebruiker in dit input veld bijvoorbeeld tekst kan invoeren, of een getal, of zelfs een bestand kan uploaden. Bijvoorbeeld:

- `<input type="text" />`: Om tekst in te voeren
- `<input type="number" />`: Om een getal in te voeren
- `<input type="password" />`: Om een wachtwoord in te voeren
- `<input type="file" />`: Om een bestand te uploaden

3.6 De `head` Tag

De `<head>` tag bevindt zich in een HTML pagina *boven* de `<body>`, zoals in het voorbeeld hieronder weergegeven.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="icon" href="favicon.png" />
  </head>
  <body>
    <p>Hello</p>
  </body>
</html>
```

En aan dit head element kunnen een aantal belangrijke *child*-elements worden toegevoegd, waaronder de `<title>` element, waarmee je de titel van je pagina kunt instellen (boven in het tabblad).

Verder is het ook mogelijk om via het self-closing `<link />` element, een icoontje aan je pagina te linken (welke ook boven in het tabblad verschijnt).

Dit link element accepteert twee attributes, waarbij de `rel="icon"` attribute aangeeft dat het om een icoontje gaat, en waarbij de `href="favicon.png"` de bestandslocatie van dit icoontje aangeeft (in dit voorbeeld zou er een icoontje genaamd `favicon.png` beschikbaar moeten zijn in dezelfde map als het HTML bestand).

3.7 Anchor Tags

Het anchor element (`<a>`) wordt gebruikt om een link naar een ander HTML bestand, of naar een externe website te maken.

Deze tag heeft een verplichte attribute genaamd `href` , waarmee je de locatie kunt aangeven waar deze link naar moet verwijzen. Voorbeelden zijn:

- `<a href="https://www.google.com">Google</a>` : Om naar de homepage van Google te verwijzen, waarbij je het protocol (*http* of *https*) niet mag vergeten
- `<a href="contact.html">Contact</a>` : Om naar een van je eigen HTML bestanden (zoals `contact.html`) te linken (welke zich, in dit voorbeeld, in dezelfde map moet bevinden als het HTML bestand met deze anchor tag)

3.8 Image Tags

Het self-closing image element (`<img />`) wordt gebruikt om een plaatje toe te voegen aan je website. Via de `src` attribute kun je de locatie van dit plaatje aangeven. En dit mag een plaatje van het internet zijn, maar het kan ook een lokaal plaatje (bijvoorbeeld in dezelfde map als het HTML bestand) zijn. Voorbeelden:

- `<img src="https://example.com/apple.jpg" />`
- `<img src="banana.jpg" />`

3.9 Typische Website Layout

Er bestaat een groot aantal websites dat eenzelfde algemene structuur volgt.

Meestal wordt het bovenste deel van zo'n *standaard* website de *header* genoemd, waar je typisch de naam en het logo van de site ziet staan, plus een aantal links om de site te navigeren, en eventueel een zoekbalk.

Onder de header vind je vervolgens de *main* content, bestaande uit de inhoud waar deze pagina echt om draait.

En tot slot vind je onderaan meestal een soort *footer*, waar je typisch de Algemene Voorwaarden, de Privacy Policy en eventuele merk of copyright informatie kunt vinden.

Uiteraard zijn er ook websites die van deze typische structuur afwijken, maar het is wel een populaire model omdat veel mensen (onbewust) bekend zijn met deze opzet.

Sinds de laatste versie van HTML (versie 5), zijn er dan ook een hoop nieuwe tags bijgekomen, waaronder `<header>`, `<main>` en `<footer>`, welke gebruikt kunnen worden om zo'n typische structuur op te zetten. Bijvoorbeeld:

```
...
<body>
  <header>
    <h1>Marktplaats</h1>
  </header>
  <main>
    <p>Auto te koop!</p>
  </main>
  <footer>
    <p>Copyright</p>
  </footer>
</body>
...
```

En deze nieuwe tags worden ook wel *semantisch* genoemd: als je een tag zoals `<footer>` ziet, dan weet je dat dit het onderste deel van een pagina moet voorstellen, en dat je hier bijvoorbeeld een link naar de Algemene Voorwaarden kunt verwachten.

Dit in tegenstelling tot bijvoorbeeld een `<div>` of een `<span>`, welke niets aangeeft over zijn daadwerkelijke content.

Dus een semantisch element, is een element dat een bepaalde (intrinsieke) betekenis draagt. Andere voorbeelden zijn: `<h1>` en `<p>`.

3.10 OPDRACHT: Fruitwinkel

-

4 CSS (Styling)

4.1 Wat is CSS?

CSS (afkorting voor *Cascading Style Sheets*) is een taal die je gebruikt om HTML pagina's vorm te geven, en is daarmee volledig verantwoordelijk voor de schoonheid en de stijl van een website.

4.2 HTML en CSS Koppelen

Er zijn 3 manieren om CSS te koppelen aan een bestaande HTML pagina.

Slechte Manier 1 (inline)

Door CSS rechtstreeks aan een gekozen HTML element toe te voegen via de `style` attribute.

```
<h1 style="color: red;">I am a red H1 header</h1>
```

Betere Manier 2 (via de `<head>`)

Er bestaat een apart HTML element (`<style>`) dat gebruikt kan worden om een mini-omgeving aan te maken, zodat je de CSS voor deze HTML pagina *rechtstreeks* binnen deze style tag kunt opgeven.

Deze style tag dien je toe te voegen aan de head, zoals hieronder is weergegeven.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      h1 {
        color: red;
      }
    </style>
  </head>
  <body>
    <h1>I am a red H1 header</h1>
  </body>
</html>
```

Beste Manier 3 (apart CSS bestand)

De beste manier is om een apart CSS bestand aan te maken, bijvoorbeeld `style.css`, en deze in dezelfde map als het HTML bestand te plaatsen.

In dat geval kun je een HTML pagina aan deze externe stylesheet *linken*, door een self-closing `<link />` element toe te voegen aan de head.

```
// welcome.html

<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>I am a red H1 header</h1>
  </body>
</html>
```

```
// style.css

h1 {
  color: red;
}
```

4.3 Color, Background Color, Width en Height

De CSS properties `color` en `background-color` worden gebruikt om de tekstkleur en de achtergrondkleur van een HTML element op te geven.

Als je een block element (zoals een div) een achtergrondkleur geeft, dan zul je zien dat block elements, standaard, de volledige breedte van de pagina in beslag nemen.

Je kunt vervolens de properties `width` en `height` gebruiken om de afmetingen van dat block element aan te passen, maar ze verliezen hierdoor niet hun karakteristieke eigenschap (het is nog steeds niet mogelijk om 2 of meer elements op 1 regel te plaatsen, als een van die elements een block element is).

Let goed op dat de properties `width` en `height` *alleen* effect hebben op block elements: als je een span een width probeert te geven, dan gebeurt er niets.

4.4 Text Align

Standaard wordt alle tekst binnen een HTML pagina *links* uitgelijnd. Als je deze richting wilt wijzigen, dan kun je daar de `text-align` property voor gebruiken.

Deze text-align property heeft zowel effect op "pure" tekst, als op inline elements `zelf` (maar niet op block elements). Dus het volgende voorbeeld kan gebruikt worden om een span in het midden uit te lijnen.

```
// welcome.html

...
<body>
  <span>I am now in the center of the page</span>
</body>
...


// style.css

body {
  text-align: center;
}
```

Verder is text-align een voorbeeld van een zogenaamde *cascading* property. Dat wil zeggen dat elk element dat *onder* een element met de stijl `text-align: center;` valt, automatisch ook deze CSS regel overneemt. Dus als je een bepaalde text-align instelt op het body element, dan zal elk HTML element op je pagina (dat onder de body) valt, in deze richting worden uitgelijnd.

Andere voorbeelden van cascading properties zijn `font-size` en `color`.

4.5 `id` en `class`

Het is mogelijk om een HTML element een unieke naam te geven, aan de hand van de `id` attribute. Let hierbij op dat **geen enkel ander element op de pagina** dezelfde id daarna nog mag gebruiken. Bijvoorbeeld:

```
<ul>
  <li>Appel</li>
  <li id="middle-fruit">Banaan</li>
  <li>Citroen</li>
</ul>
```

Als je meerdere elements hetzelfde label wil geven, dan gebruik je daar de `class` attribute voor.

```
<ul>
  <li class="healthy">Appel</li>
  <li class="healthy" id="middle-fruit">Banaan</li>
  <li class="healthy">Citroen</li>
</ul>
```

Vervolgens is het mogelijk om vanuit je stylesheet de CSS id en class selectors (`#` en `.`) te gebruiken, om deze elements een specifieke styling te geven.

```
#middle-fruit {
  color: yellow;
}
.healthy {
  font-size: 20px;
}
```

Het is mogelijk om een element meerdere classes te geven, door deze met spaties van elkaar te scheiden: `<li class="healthy edible tasty">`.

4.6 OPDRACHT: Regenboogtrap

-

4.7 Display `inline` en Display `block`

Elk HTML element is van zichzelf ofwel een *block* element, ofwel een *inline* element. Maar het is mogelijk om deze eigenschap via CSS te regelen via de `display` property.

Een span met de property `display: block;` zal zich hierdoor als een div gedragen, terwijl een div met de property `display: inline;` zich hierdoor als een span zal gedragen.

4.8 Display `inline-block`

Er bestaat een soort hybride mix tussen inline en block display, genaamd *inline-block*. Elements met de `display: inline-block;` property nemen zowel inline als block properties over:

- (inline) Ze sluiten op dezelfde regel aan met andere inline en inline-block elements.

- (inline) Ze laten zich in een bepaalde richting uitlijnen, als het parent element een bepaalde text-align richting aangeeft.
- (inline) Ze nemen, standaard, niet de volledige paginabreedte in beslag.
- (block) Ze kunnen een specifieke width en height krijgen.

4.9 Padding en Margin

De CSS properties *padding* en *margin* worden gebruikt om extra witruimte aan en rondom een element toe te voegen.

Een element met `padding: 10px;` zal 10 pixels witruimte aan de *binnenkant* van dat element toevoegen (dus rondom de tekst, of rondom een ander element, binnen dat element). De padding maakt dan ook deel uit van het element *zelf*.

Een element met `margin: 10px;`, daarentegen, zal alle content *om zich heen* 10 pixels van zich afduwen. Maar deze margin area maakt officieel geen onderdeel uit van het element zelf.

4.10 Border en Outline

De CSS properties *border* en *outline* worden allebei gebruikt om een rand toe te voegen aan HTML elements.

Een element met `border: 3px solid black;` zal een zwarte doorgetrokken rand van 3 pixels breed hebben. Deze rand is onderdeel van het element *zelf*, dus dat betekent dat de totale afmeting van dit element hierdoor ook in elke richting 3 pixels zal toenemen.

Op dezelfde manier kan een `outline: 3px solid purple;` worden ingesteld, maar deze outline maakt geen deel uit van het element zelf, en verder heeft deze outline ook geen effect op andere elements op de pagina. Dus een een element met `outline: 5000px solid purple;` zal een outline creëren, die *boven* of *onder* alle andere elements op de pagina komt te liggen (maar deze outline zal de andere elements niet *verplaatsen* of wegduwen).

Om die reden is outline vooral handig tijdens het ontwikkelen en tijdens het debuggen, om duidelijk te zien waar een HTML element zich op de pagina bevindt, zonder de andere elements te beinvloeden.

4.11 CSS Box Model

De CSS Box Model beschrijft een soort rechthoek die de layout van elk HTML element vastlegt, en is opgebouwd uit de volgende 4 onderdelen.

- De **content area**, die bijvoorbeeld bestaat uit de tekst `Hello` in het voorbeeld `<p>Hello</p>`
- De **padding area**, die de ruimte *binnen* het element zelf weergeeft, *rondom* de content area (in te stellen met de `padding` property)
- De **border area**, die de rand van het element aangeeft (in te stellen met de `border` property)
- De **margin area**, die de *no-go* zone rondom het element aangeeft en ervoor zorgt dat andere elements worden weggeduwd (in te stellen met de `margin` property)

De outline maakt *geen* onderdeel uit van de CSS Box model, omdat deze property nul effect heeft op de afmetingen van het element zelf, of de positie van de elements eromheen.

4.12 OPDRACHT: Mondriaan

-

4.13 De `hover` Pseudo Class

Er bestaan een aantal *pseudo classes* binnen CSS, welke je kunt gebruiken om een element alleen een bepaalde stijl te geven, als dat element zich in een bepaalde toestand bevindt.

Het beste voorbeeld hiervan is de `:hover` pseudo class, welke gebruikt kan worden om een element een specifieke stijl te geven, wanneer dat element wordt "gehoverd" (dus wanneer je de muis erboven houdt).

Het volgende voorbeeld kan gebruikt worden om een rood element blauw te maken, zodra de gebruiker zijn muis boven dat element houdt.

```
#my-element {
  background-color: red;
}
#my-element:hover {
  background-color: blue;
}
```

4.14 Font Family en Text Shadow

De property *font-family* wordt gebruikt om een HTML element een ander lettertype te geven. Een goede website met gratis letterytypes is https://fonts.google.com, waar je ook instructies zult vinden om die lettertypes aan je CSS pagina toe te voegen, aan de hand van een `font-family: 'Pacifico';` property en een `@import(https://...);` statement bovenin je stylesheet

Verder is het mogelijk om een schaduw in te stellen voor je tekst. Dit kan aan de hand van de *text-shadow* property, welke 4 values accepteert.

```
h3 {
  text-shadow: <offset-x> <offset-y> <blur-radius> <shadow-color>;
}
```

Dus je zou `text-shadow: 2px 2px 4px black;` kunnen gebruiken om een zwarte tekstschaduw toe te voegen, met een "bluriness" van 4 pixels, die 2 pixels naar rechts en 2 pixels naar beneden is verschoven.

4.15 Box Shadow

Naast text-shadow bestaat er ook een CSS property om een schaduw aan een HTML element zelf toe te voegen, namelijk *box-shadow*.

Deze property kan op dezelfde manier gebruikt worden als text-shadow, dus op de volgende manier zou je een simpel grijs vierkant met een subtiele schaduw kunnen maken.

```
#square {
  width: 400px;
  height: 400px;
  background-color: lightgrey;
  box-shadow: 2px 2px 4px grey;
}
```

4.16 OPDRACHT: Blog

-

5 JavaScript (Programmeertaal)

5.1 Wat is JavaScript?

JavaScript is een *programmeertaal* die gebruikt kan worden in een browseromgeving. Dit betekent dat je JavaScript kunt gebruiken om zowel eenvoudige als complexe functionaliteit aan HTML pagina's toe te voegen.

Hieronder zijn een aantal voorbeelden en website features waar JavaScript aan te pas zou komen.

- Google Maps laten zien op je pagina
- Een image slider die je kunt bedienen met de pijltjes van je toetsenbord
- Een simpele browser game, zoals Snake

Bij de term JavaScript kun je meestal aannemen dat het gaat over browseromgeving, maar in principe staat JavaScript *als programmeertaal* los van de browser.

Het is bijvoorbeeld ook mogelijk om de programmeertaal JavaScript te gebruiken in een serveromgeving, maar in dat geval wordt JavaScript meestal aangeduid als *Node* of *NodeJS*.

5.2 HTML en JavaScript Koppelen

Er zijn 3 manieren om JavaScript te koppelen aan een bestaande HTML pagina.

Slechte Manier 1 (inline)

Het is mogelijk om kleine stukje JavaScript rechtstreeks aan een HTML element toe te voegen. Het HTML body element heeft bijvoorbeeld een `onload` attribute die gebruikt kan worden om een stukje JavaScript uit te voeren zodra de body is geladen.

```
<!DOCTYPE html>
<html>
  <body onload="alert(10);">
    <h1>Hello</h1>
  </body>
</html>
```

Betere Manier 2 (onder aan de `<body>`)

Er bestaat een apart HTML element (`<script>`) dat gebruikt kan worden om een mini-omgeving te maken, zodat je de JavaScript voor deze HTML pagina *rechtstreeks* binnen deze script tag kunt opgeven.

Deze script tag kun je het beste toevoegen als het *laatste* child van het body element, zoals hieronder is weergegeven.

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Hello</h1>
    <p>Welcome to my website</p>
    <script>
      alert(20);
    </script>
  </body>
</html>
```

Beste Manier 3 (apart JavaScript bestand)

De beste manier is om een apart JavaScript bestand aan te maken, bijvoorbeeld `script.js`, en deze in dezelfde map als het HTML bestand te plaatsen.

In dat geval kun je een HTML pagina aan dit externe bestaand linken, door een `<script>` element toe te voegen aan de head, waarbij je twee belangrijke attributes moet meegeven.

```
// page.html

<!DOCTYPE html>
<html>
  <head>
    <script defer src="script.js"></script>
  </head>
  <body>
    <h1>Hello</h1>
  </body>
</html>
```

```
// script.js

alert(30);
```

De `defer` attribute van het script element is een voorbeeld van een *boolean* attribute (een attribute zonder value). En deze attribute geeft aan dat de browser dit externe JavaScript pas moet uitvoeren, zodra de HTML pagina volledig ingeladen en verwerkt is.

Verder gebruik je de `src` attribute van het script element om de locatie van je externe JavaScript bestand aan te geven.

5.3 Numbers

Een belangrijk concept binnen elke programmeertaal is het concept van *variables*.

In plaats van de ingebouwde `alert` functie rechtstreeks aanroepen met een bepaald getal, is het ook mogelijk om eerst een variable voor dat getal te maken.

```
let myNumber = 55;
alert(myNumber);
```

In dit voorbeeld is `myVariable` een variable van het type *number* (getal). En variables van het type number hebben de eigenschap dat je ermee kunt rekenen.

```
let number1 = 12 + 4;
let number2 = 44 - 7;
let number3 = number1 * (number2 + 100);
let number4 = 6 / 4;
```

Aangezien het niet is toegestaan om spaties of streepjes te gebruiken in de naam van je variables, is het gebruikelijk om de zogenaamde `camelCase` conventie te gebruiken: als een variable uit meerdere woorden bestaat, dan laat je het 2e en het 3e en het 4e (enz.) woord beginnen met een hoofdletter.

```
let ageOfMyNeighbour = 44;
let secondsPerMinute = 60;
```

Tot slot is het meestal fijner om de ingebouwde `console.log` functie te gebruiken, in plaats van `alert`.

```
console.log(22 + 4);
```

5.4 Strings

In de context van programmeren, is een `string` een type variable waarin je een stukje tekst kunt opslaan. Om een string aan te maken, moet je ofwel *double quotes* ofwel *single quotes* (beter) gebruiken.

```
let doubleQuoteString = "This is one way to create strings";
let singleQuoteString = 'This is a better to create strings';
```

Verder is het mogelijk om twee strings achter elkaar te zetten, door ze *bij elkaar op te tellen*.

```
let greeting = 'Nice to meet you ';
let name = 'Jessy';
console.log(greeting + name);
```

5.5 Booleans

JavaScript kent 5 primitieve value types, waarbij **number**, **string** en **boolean** de belangrijkste zijn.

Een *boolean* is een type variable die maar twee values kan hebben: `true` of `false`. Het is niet mogelijk om booleans bij elkaar op te tellen, maar je kunt ze wel op twee andere manieren combineren, aan de hand van de `&&` en `||` ("*en*" en "*of*") operators.

```
let hungry = true;
let thirsty = false;

let hungryAndThirsty = hungry && thirsty;
let hungryOrThirsty = hungry || thirsty;
```

In dit voorbeeld zal de variable `hungryAndThirsty` een boolean zijn met de value `false`, en zal de variable `hungryOrThirsty` een boolean zijn met de value `true`.

5.6 If Statements

Het is mogelijk om een bepaald stukje JavaScript code alleen maar uit te voeren, als er is voldaan is aan een bepaalde voorwaarde. Dit is mogelijk met behulp van *if statements* en *booleans*.

De eenvoudigste versie van een if statement ziet er als volgt uit.

```
let allowMarketingEmails = true;

if (allowMarketingEmails) {
  console.log('Here is a marketing e-mail');
}
```

Maar het is ook mogelijk om meerdere *branches* aan deze if statement toe te voegen.

```
let allowMarketingEmails = false;
let allowPushNotifications = true;

if (allowMarketingEmails) {
  console.log('Here is a marketing e-mail');
} else if (allowPushNotifications) {
  console.log('Here is a push notification');
} else {
  console.log('We have nothing to send you');
}
```

5.7 Truthy en Falsy

Elke value in JavaScript heeft de eigenschap dat hij ofwel *truthy* ofwel *falsy* is. Zie de volgende lijst met alle values die *falsy* zijn binnen JavaScript.

- `false`
- `0`
- `''` (empty string)
- `undefined`
- `null`
- `NaN` (not a number)

Elke value die niet op deze lijst staat, is *truthy* binnen JavaScript.

Aangezien het ook mogelijk is om *niet*-booleans voor een if statement te gebruiken, wordt aan de hand van deze truthy-of-false eigenschap bepaald of de if statement wel of niet wordt uitgevoerd.

```
let age = 0;
let name = 'Jessy';

if (age) {
  console.log('I will NOT be logged to the console');
}

if (name) {
  console.log('I _will_ be logged to the console');
}
```

5.8 Variables Vergelijken

Het is mogelijk om variables van het type *number* met elkaar te vergelijken, aan de hand van de operators `<` (kleinder-dan), `<=` (kleiner-dan-of-gelijk-aan), `>` (groter-dan) en `>=` (groter-dan-of-gelijk-aan).

```
let age = 22;
let oldEnough = (age >= 18);
```

In het bovenstaande voorbeeld zal `oldEnough` een variable van het type boolean zijn met als value `true`. De ronde haakjes rondom `age >= 18` zijn niet verplicht, maar ze maken dit stukje code wel iets leesbaarder.

Daarnaast is het ook mogelijk om `===` (is-gelijk-aan) en `!==` (is-niet-gelijk-aan) te gebruiken voor alle type variables.

```
let name = 'Jessy';

if (name === 'Jessy') {
  console.log('Welcome Jessy');
}

if (name !== 'Jessy') {
  console.log('Welcome, whoever you are');
}
```

5.9 Alert, Confirm en Prompt

JavaScript kent drie soorten pop-ups die je kunt gebruiken in een browseromgeving.

- `alert`: Een pop-up waarmee je een simpele mededeling kunt laten zien aan de gebruiker
- `confirm`: Een pop-up waarmee je de gebruiker om een bevestiging kunt vragen, of een soort ja/nee vraag kunt stellen
- `prompt`: Een pop-up waarmee je de gebruiker kunt vragen om een stukje tekst in te typen

Sommige (ingebouwde) functies, waaronder `confirm` en `prompt`, hebben de eigenschap dat ze een zogenaamde *return value* hebben.

Als zo'n functie dan klaar is met zijn uitvoering of berekening, dan is het mogelijk om deze return value op te slaan in een variable.

```
let oldEnough = confirm('Are you old enough?');
let name = prompt('What is your name?');
```

In het geval van `confirm` zal de return value een boolean zijn die `true` is als de gebruiker op "Ok" heeft geklikt, en `false` als de gebruiker op "Cancel" heeft geklikt.

De return value van `prompt` zal een string zijn, gelijk aan de tekst die de gebruiker heeft ingetypt in het popup venster.

5.10 OPDRACHT: Aanmelding

-

5.11 Undefined en Null

Bij de eenvoudige JavaScipt regel `let age = 24;` is er sprake van twee verschillende stappen: *declaration* en *assignment*.

Het is ook mogelijk om deze twee stappen op aparte regels uit te voeren.

```
let age;
age = 24;
```

Op de bovenste regel wordt een variable met de naam `age` aangemaakt (*declaration*), en op de onderste regel wordt die variable een value gegeven (*assignment*).

Je kunt elke variable maar **een keer** aanmaken, maar je kunt een variable **meerdere keren** (re)assignen. Als je een variable gebruikt voordat je deze een value hebt gegeven, dan krijg je `undefined`.

```
let age;

console.log(age);

if (age === undefined) {
  console.log('That variable is undefined');
}
```

In het bovenstaande voorbeeld zal het woordje `undefined` naar de console worden geschreven, gevolgd door de tekst `That variable is undefined`.

Tot slot is het idee achter de value `null` hetzelfde als `undefined`, maar je zult `null` veel minder vaak tegenkomen.

5.12 Let en Const

Naast *let* kun je ook het woordje *const* gebruiken om je variables aan te maken (*declaration*).

Het enige verschil, is dat je `const` variables op 1 regel moet aanmaken en meteen een value moet geven, en dat je een `const` variable nooit meer mag reassignen.

```
const age = 24;;

let name = 'Jessy';
name = 'Alex';
```

Het is beter om altijd `const` te gebruiken, tenzij je niet anders kan.

5.13 Scope

Vrijwel altijd als je twee accolades (`{}`) ziet, is er sprake van een zekere *scope*.

Variables die zijn aangemaakt in een bepaalde scope, kunnen daarbuiten **niet** worden gebruikt. Ze kunnen daarentegen **wel** worden gebruikt in een sub-scope *binnen* die scope.

```
if (true) {
  const age = 12;
  if (true) {
    const name = 'Jessy';
    console.log('I can use both "age" and "name" here');
  }
  console.log('I can only use "age" here, but not "name"');
}

console.log('I cannot use "age" or "name" here');
console.log('The line below will give an error');
console.log(age);
```

5.14 Functies

Er bestaan verschillende manieren om *functies* te maken binnen JavaScript. Een van die manieren werkt als volgt, waarbij de functie als een soort variable wordt aangemaakt.

```
const sayHello = function() {
  console.log('Hello');
};
```

Het is mogelijk om deze functie aan te roepen door een extra regel met `sayHello();` toe te voegen.

Er bestaat verder ook een andere (en modernere) variant van de bovenstaande functie syntax, welke de zogenaamde *fat-arrow* syntax wordt genoemd.

```
const sayGoodbye = () => {
  console.log('Goodbye');
};
```

Dit zijn allebei voorbeelden van functies die geen *arguments* (input) aannemen, en geen *return value* (output) leveren.

5.15 Functies - Verschillende Notaties

Naast de fat-arrow syntax bestaat er ook een mogelijkheid om functies aan te maken zonder het woordje `const`.

```
function sayHello() {
  console.log('Hello');
}
```

De bovenstaande functie lijkt natuurlijk iets minder op een variable door de afwezigheid van het woordje `const`, maar kan op exact dezelfde manier worden aangeroepen met `sayHello();`.

Een belangrijk verschil met de fat-arrow functies, is dat functies die met de bovenstaande syntax zijn aangemaakt, gebruikt kunnen worden **voordat** ze gedefinieerd zijn.

```
sayHello();

function sayHello() {
  console.log('Hello');
}
```

5.16 Arguments

Om een functie te maken die input aanneemt in de vorm van *arguments*, kun je een soort variable toevoegen binnen de ronde haakjes van die functie.

```
function sayHello(name) {
  console.log('Nice to meet you, ' + name);
}
```

Het type en de value van die `name` variable zal nu afhangen van *hoe deze functie wordt aangeroepen*. Want het is vanaf nu mogelijk om een argument mee te geven bij het aanroepen van deze functie: `sayHello('Jessy');`.

In dit specifieke geval zal de `name` variable binnen de `sayHello` functie van het type string zijn, met als value de tekst "Jessy".

5.17 Return Values

Om een functie output te laten leveren aan de hand van een *return value*, gebruik je het woordje `return`.

Vervolgens is het dan mogelijk om de return value van je functie op te slaan in een variable.

```
function guessTheWeather() {
  return 'sunny';
}

const weather = guessTheWeather();
```

Het is belangrijk om te onthouden dat een functie **direct en meteen** eindigt, zodra die functie bij een regel met het woordje `return` is aangekomen.

Het volgende JavaScript programma zal alleen de tekst "Appel" loggen naar de console.

```
function showFruits() {
  console.log('Appel');
  return;
  console.log('Banaan');
}

showFruits();
```

5.18 Timeouts

In veel gevallen roep je je functies zelf aan, maar er bestaan ook manieren om je functies *door JavaScript* aan te laten roepen.

De ingebouwde `setTimeout` kan worden gebruikt om een functie pas over een paar seconden uit te voeren.

```
function showTime() {
  const time = new Date().toTimeString();
  console.log(time);
}

setTimeout(showTime, 2000);
```

In dit voorbeeld wordt de `showTime` functie over 2000 milliseconden (= 2 seconden) *door JavaScript* uitgevoerd.

Om die reden is het **essentieel** dat je geen ronde haakjes achter het `showTime` argument van de `setTimeout` functie plaatst. Het is de bedoeling om de `showTime` functie *mee te geven* als argument, niet om deze functie *zelf uit te voeren*.

Verder zal het aanroepen van de `setTimeout` functie *zelf* altijd minder dan 1 milliseconden tijd in beslag nemen, aangezien `setTimeout` alleen maar hoeft

af te stemmen met de browser dat de opgegeven functie over een aantal seconden wordt uitgevoerd.

In het volgende voorbeeld zul je vrijwel direct de tekst "Banaan" zien worden gelogd naar de console, gevolgd door de tekst "Appel" twee seconden later.

```
function showApple() {
  console.log('Appel');
}

setTimeout(showApple, 2000);
console.log('Banaan');
```

5.19 Intervals

In JavaScript kun je ook de ingebouwde `setInterval` functie gebruiken om een van je functies *herhaaldelijk* uit te voeren.

Net als `setTimeout` accepteert deze functie als eerste argument de functie die je wil uitvoeren, en als tweede argument hoeveel milliseconden je wil wachten tussen elke uitvoering van die functie.

```
function showTime() {
  console.log(new Date().toTimeString());
}

setInterval(showTime, 3000);
```

Hiermee zal elke 3 seconden de huidige tijd naar de console worden geschreven. Let wel op dat deze functie na 3 seconden pas voor de eerste keer wordt uitgevoerd, dus op tijdstip 3, op tijdstip 6, op tijdstip 9, op tijdstip 12, enzovoort.

Als je wil dat deze functie ook op tijdstip 0 wordt aangeroepen, dan zul je dat een keer zelf moeten doen met `showTime();`.

5.20 OPDRACHT: Kaartverkoop

-

5.21 Arrays

Een *array* is een voorbeeld van een zogenaamde *data structure*, waarbij je een data structure kunt zien als een bepaalde constructie die je in staat stelt om *meerdere* values op te slaan in *een* enkele variable.

En het eenvoudigste voorbeeld van een data structure is dus de array, welke je simpelweg kunt zien als een soort rijtje met values.

```
const myFirstArray = ['a', 'b', 'c', 44, true];
```

De array uit het bovenstaande voorbeeld bestaat uit 5 verschillende values. Het is ook mogelijk om deze *lengte* van een array op te vragen via de `myFirstArray.length` property.

Verder heeft elk element in een array een unieke *index*, waarbij het binnen de programmeertalen gebruikelijk is om vanaf **0** te beginnen met tellen. Dus het eerste (meest linkse) element krijgt index 0, het volgende element krijgt index 1, enzovoort.

Via deze index is het ook mogelijk om een bepaald element in de array te selecteren. In het onderstaande voorbeeld zal de variable `myElement` van het type number zijn, en zal het getal 44 naar de console worden geschreven.

```
const myFirstArray = ['a', 'b', 'c', 44, true];
const myElement = myFirstArray[3];
console.log(myElement);
```

5.22 Arrays - `includes`

Elke array heeft toegang tot een ingebouwde functie, waarmee je kunt controleren of die array een bepaalde value wel of niet heeft.

```
const randomValues = ['test', 44, false];

if (randomValues.includes(44)) {
  console.log('I _will_ be logged to the console');
}

if (randomValues.includes('TEST')) {
  console.log('I will NOT be logged to the console');
}
```

De return value van deze `includes` functie is een simpele boolean, welke `true` of `false` is, afhankelijk of het element zich wel of niet in deze array bevindt.

Let erop (net zoals bij de `randomValues.length` property voor de lengte van een array), dat je deze `includes` functie aanroept **op** de array zelf, waarbij je altijd een punt (`.`) moet gebruiken.

Dus `randomValues.includes(44);` in plaats van gewoon `includes(44);`.

5.23 Arrays - `push`, `pop`, `unshift` en `shift`

Het is eenvoudig om een element met een bepaalde index te wijzigen binnen een array. Om het middelste element van `const myArray = ['a', 'b', 'c'];` te wijzigen naar de string "x", kun je simpelweg de syntax `myArray[1] = 'x';` gebruiken.

Arrays hebben verder nog een aantal ingebouwde functies (of *methods*), waarmee je elements aan een bestaande array kunt toevoegen, of juist kunt verwijderen.

- `push`: voeg een nieuw element toe aan het *eind* van een array
- `pop`: verwijder het *laatste* element van een array
- `unshift`: voeg een nieuw element toe aan het *begin* van een array
- `shift`: verwijder het *eerste* element van een array

In het voorbeeld hieronder heb ik telkens met een comment (herkenbaar aan de dubbele slash `//`) aangegeven wat de value van de `myArray` variable op dat moment is.

```
const alphabet = ['aa', 'bb', 'cc'];

alphabet.push('dd');
console.log(alphabet); // ['aa', 'bb', 'cc', 'dd']

alphabet.pop();
console.log(alphabet); // ['aa', 'bb', 'cc']

alphabet.unshift(44);
console.log(alphabet); // [44, 'aa', 'bb', 'cc']

alphabet.shift();
console.log(alphabet); // ['aa', 'bb', 'cc']

alphabet.shift();
console.log(alphabet); // ['bb', 'cc']

alphabet.pop();
console.log(alphabet); // ['bb']
```

5.24 Loops

Loops zijn een manier om een bepaald stukje JavaScript code herhaaldelijk uit te voeren. Om bijvoorbeeld een bericht 10x te loggen naar de console, kun je de volgende *for-loop* gebruiken.

```
for (let index = 0; index < 10; index += 1) {
  console.log('Hello ' + index);
}
```

De belangrijkste use case voor een for-loop is de *array*. Met behulp van een for-loop kun je namelijk over alle elements van een array *itereren*.

```
const array = ['a', 'b', 'c'];

for (let index = 0; index < myArray.length; index += 1) {
  const item = array[index];
  console.log('Element at index ' + index + ' is equal to: ' + item);
}
```

Maar aangezien er een betere en modernere manier bestaat om over een array te itereren, raad ik je aan om for-loops niet te gebruiken.

Een nog minder gebruikte loop is trouwens de *while-loop*, die er als volgt uitziet.

```
let number = 10;

while (number >= 0) {
  console.log(number);
  number -= 1;
}
```

Dus, zoals ik al zei, is het beter om loops te vermijden. Je moet er namelijk ook ontzettend goed op letten dat je geen fout maakt in de logica van een loop, waardoor hij nooit tot een einde komt.

Zulke loops worden *infinite loops* genoemd, en kunnen ervoor zorgen dat je browser crasht.

5.25 Arrays - Iteration

Elke array heeft een ingebouwde functie genaamd `forEach`, welke gebruikt kan worden om over de elements van die array te *itereren*.

Deze `forEach` functie lijkt in enige zin op de `setTimeout` en `setInterval` functies, aangezien je voor het eerste argument van `forEach` een functie moet

meegeven, welke zal worden uitgevoerd voor elk element in de array.

```
function showElement(element, index) {
  console.log('Item at index ' + index + ' is equal to ' + element);
}

const array = ['a', 'b', 'c'];
array.forEach(showItem);
```

Het is essentieel dat je de `showItem` *meegeeft* aan de `forEach` functie, en dat je je functie niet zelf aanroept. Dus `array.forEach(showItem);` in plaats van `array.forEach(showItem());`.

Verder is het gebruikelijk om een fat-arrow functie rechtstreeks mee te geven aan de *forEach* functie.

```
['a', 'b', 'c'].forEach((element, index) => {
  console.log('Item at index ' + index + ' is equal to ' + element);
});
```

5.26 OPDRACHT: Gemiddelde

-

5.27 Objects en Properties

Behalve de array bestaat er in JavaScript nog een belangrijke data structuur genaamd *object*.

Net als een array, bestaat een object ook uit meerdere values. Het grote verschil is dat je in het geval van een array een *index* gebruikt, om de locatie van de verschillende values aan te geven, terwijl je in het geval van een object een (zelfgekozen) *key* gebruikt.

```
const catalogue = {
  sweater: 30,
  shirt: 20,
  sale: true
};
```

Dit is een voorbeeld van een object met drie verschillende *properties*. Het is vervolgens mogelijk om de value van een property te selecteren, aan de hand van de property key.

```
const catalogue = {
  sweater: 30,
  shirt: 20,
  sale: true
};

const property = catalogue.sweater;
console.log(property); // 30
```

In het bovenstaande voorbeeld is `property` een variable van het type number (met als value het getal 30).

Tot slot is het ook mogelijk om de value van property te wijzigen, of om een property volledig te verwijderen uit een object.

```
const catalogue = {
  sweater: 30,
  shirt: 20,
  sale: true
};

catalogue.sweater = 50;
delete catalogue.shirt;
```

5.28 Objects en Methods

Behalve primitieve values als number, string en boolean, is het ook mogelijk om een object een *functie* als property te geven. Zulke functies die deel uitmaken van een object worden meestal *methods* genoemd, in plaats van *functies*.

Er bestaan twee verschillende soorten syntax om een method aan een functie toe te voegen.

```
const person = {
  firstName: 'Jessy',
  sayHello: function() {
    console.log('Hello, this is the traditional syntax');
  },
  sayGoodbye: () => {
    console.log('Goodbye, this is the fat-arrow syntax');
  }
};
```

Maar het is beter om de moderne variant van de traditionele syntax te gebruiken.

```
const person = {
  firstName: 'Jessy',
  sayHello() {
    console.log('Hello, this is the (modern) traditional syntax');
  },
  sayGoodbye: () => {
    console.log('Goodbye, this is the fat-arrow syntax');
  }
};
```

In het geval van de traditionele (of de modern traditionele) syntax, is het in dit voorbeeld mogelijk om gebruik te maken van het woordje `this`, dat naar het object *zelf* verwijst (zolang je deze method **op** het object zelf aanroept).

```
const person = {
  firstName: 'Jessy',
  introduce() {
    return 'Hi, my name is ' + this.firstName;
  },
};

const introduction = person.introduce();
console.log(introduction); // 'Hi, my name is Jessy'
```

5.29 Modelleren

De primitieve values (number, string, boolean en null/undefined) vormen samen met de standaard data structures (array en object) een ontzettend krachtig systeem om allerlei concepten in de echte wereld te modelleren.

Zie hieronder hoe je bijvoorbeeld een "verkoper" zou kunnen modelleren op een website als Marktplaats of Ebay.

```
const seller = {
  username: 'tunghori',
  location: 'Philippines',
  feedback: {
    positive: 13609,
    neutral: 19,
    negative: 2
  },
  items: [
    { name: 'Captain Rex', price: 7.99 },
    { name: 'Imperial Death Trooper', price: 10.99 },
    { name: 'Interrogation Droid', price: 2.99 }
  ]
};
```

5.30 OPDRACHT: Random People

-

6 JavaScript (De DOM)

6.1 Wat is de DOM?

In principe staat JavaScript *als programmeertaal* los van de browser. Maar er bestaat een ingebouwd object, genaamd `document`, waar je gebruik van kunt maken om toegang te krijgen tot de HTML pagina waar het huidige JavaScript bestand deel van uitmaakt.

Aangezien dit (soms gigantische) `document` object de volledige HTML pagina bevat, wordt dit ook wel het *Documet Object Model*, of simpelweg *de DOM* genoemd.

6.2 HTMLElements

Om het HTML body element te selecteren, kun je de `body` property van het ingebouwde `document` object gebruiken.

```
const bodyElement = document.body;
```

Bovenstaande `bodyElement` is een variable van het type object, uit de categorie `HTMLElement`. Dat betekent dat we voor zulke objects toegang hebben tot

bepaalde properties en methods, specifiek voor `HTMLElement`s.

Als het body element van onze HTML pagina bijvoorbeeld een class van `main-content` had, dus `<body class="main-content">`, dan zou het mogelijk zijn om deze class via JavaScript als string op te vragen, aan de hand van de `className` property voor `HTMLElements`.

```
const bodyElement = document.body;
console.log(bodyElement.className); // 'main-content'
```

6.3 Get Element(s) By XXX

De body is het makkelijkste HTML element om te selecteren vanuit JavaScript, omdat we daarvoor gewoon de `document.body` property kunnen gebruiken.

Het is echter ook mogelijk om elements op basis van hun *id*, hun *class* of hun *tagname* te selecteren, aan de hand van de volgende methods van het `document` object.

- `document.getElementById('favourite-image')`: deze method geeft als return value het unieke `HTMLElement` met een id van "favourite-image" (of `null` als dat element niet bestaat)
- `document.getElementsByClassName('red-fruit')`: deze method geeft als return value een `HTMLCollection`, bestaande uit alle `HTMLElement`s met de class "red-fruit"
- `document.getElementsByTagName('span')`: deze method geeft als return value een `HTMLCollection`, bestaande uit alle `HTMLElement`s van het type "span"

Hierbij is het belangrijk om te onthouden dat `HTMLCollection`s *array-like* worden genoemd, maar geen echte arrays zijn. Dit betekent dat het bijvoorbeeld *wel* mogelijk om het eerste element van zo'n HTMLCollection `myCollection` te selecteren via de bekende array syntax `myCollection[0]`, maar je hebt geen toegang tot de meeste array methods zoals `push`, `pop` en `forEach`.

Om een `HTMLCollection` om te zetten in een echte array kun je de `Array.from` functie gebruiken.

```
const htmlCollection = document.getElementsByTagName('red-fruit');
const realElementsArray = Array.from(htmlCollection);
realElementsArray.forEach((element) => {
  console.log(element.tagName);
});
```

6.4 Parents, Siblings en Children

Vrijwel elk HTML element heeft een *parent*. Daarnaast kan een element ook een of meerdere *siblings* (zuster elementen), en een of meerdere *children* elements hebben.

Om een `HTMLCollection` bestaande uit alle child elements te krijgen, kun je de property `myElement.children` gebruiken.

Om het vorige of het volgende sibling element te selecteren, kun je de properties `myElement.previousElementSibling` of `myElement.nextElementSibling` properties gebruiken.

En, tot slot, om het parent element van een element te selecteren, kun je de property `myElement.parentElement` gebruiken.

6.5 Elements Aanpassen

Om de tekstkleur en de achtergrondkleur van een `HTMLElement` aan te passen vanuit JavaScript, kun je gebruik maken van de `style` property.

```
const div = document.getElementsByTagName('div')[0];

div.style.color = 'red';
div.style.backgroundColor = 'pink';
```

Hierbij is het belangrijk om CSS properties met een of meerdere streepjes, zoals `background-color` en `box-shadow`, om te zetten in hun camelCase variant `backgroundColor` en `boxShadow`.

Verder is het bijvoorbeeld ook mogelijk om een class in te stellen voor een HTML element, via `myElement.className = 'red-fruit';` of om de *tekst* van een element in te stellen via `myElement.innerText = 'Hello';`.

6.6 OPDRACHT: Fruitgalerie

-

6.7 Elements Toevoegen - `appendChild`

Het toevoegen van een nieuw element bestaat altijd uit twee stappen.

1. Maak een nieuw element aan, met behulp van de `document.createElement`
2. Geef je nieuwe element een plaatsje in de DOM

Het onderstaande voorbeeld geeft aan hoe je een een blauw vierkant kunt toevoegen aan de DOM, met behulp van de `appendChild` method, welke je aan moet roepen **op** het parent element waar je je nieuwe element aan toe wilt voegen.

```
// Nieuw element aanmaken
const newDiv = document.createElement('div');
newDiv.style.width = '200px';
newDiv.style.height = '200px';
newDiv.style.backgroundColor = 'blue';

// Nieuw element toevoegen aan het body element
const parent = document.body;
parent.appendChild(newDiv);
```

De `appendChild` method kan alleen gebruikt worden om je nieuwe element als **laatste** child toe te voegen aan de parent waarop je deze method aanroept.

6.8 Elements Toevoegen - `insertBefore`

Naast `appendChild` kan ook `insertBefore` gebruikt worden om een nieuw HTML element aan een bepaalde parent toe te voegen via JavaScript.

Alleen geeft `insertBefore` iets meer keuzevrijheid over de locatie van het nieuwe element binnen de parent, aangezien deze method als tweede argument een `HTMLElement` accepteert waar**vóór** het nieuwe element moet worden toegevoegd.

In het onderstaande voorbeeld wordt een nieuwe list item aangemaakt, en vervolgens *tussen* de twee bestaande list items geplaatst.

```
// page.html

<!DOCTYPE html>
<html>
  <head>
    <script defer src="script.js"></script>
  </head>
  <body>
    <ol>
      <li>Appel</li>
      <li id="lemon">Citroen</li>
    </ol>
  </body>
</html>
```

```
// script.js

const banana = document.createElement('li');
banana.innerText = 'Banaan';

const lemon = document.getElementById('lemon');
const list = lemon.parentElement;

list.insertBefore(banana, lemon);
```

6.9 OPDRACHT: Vlag

-

6.10 Elements Verwijderen - `removeChild`

De `removeChild` method wordt aangeroepen **op** de *parent* van het HTML element dat je wil verwijderen, waarbij het te-verwijderen element als argument wordt weergegeven. Bijvoorbeeld:

```
const spanToRemove = document.getElementsByTagName('span')[0];

const parent = spanToRemove.parentElement;
parent.removeChild(spanToRemove);
```

6.11 Buttons en Clicks

Een veelgebruikt HTML element is de `<button>`, en er bestaat een manier om een JavaScript functie uit te voeren elke keer dat de gebruiker klikt op deze knop.

```
...
<body>
  <button>Click Me</button>
</body>
...
```

Eerst moet je deze button als `HTMLElement` uit de DOM selecteren. Daarna kun je een zogenaamde *event listener* registreren **op** dit element, welke zich *abonneert* op een bepaalde actie.

Vervolgens zal JavaScript de functie aanroepen die je deze event listener meegeeft, elke keer als de gekozen actie zich voordoet.

Om bijvoorbeeld de tekst "Hello" te loggen naar de console, elke keer als er op de button wordt geklikt, kun je het volgende voorbeeld gebruiken.

```
const button = document.getElementsByTagName('button')[0];

function sayHello() {
  console.log('Hello');
}

button.addEventListener('click', sayHello);
```

Het eerste (string) argument van deze `addEventListener` method geeft het *type* actie aan waar je je op wilt abonneren, in dit geval "click". Het tweede (functie) argument verwijst naar de functie die moet worden uitgevoerd, elke keer als deze actie zich voordoet.

En uiteraard is het ook mogelijk om een fat-arrow functie rechtstreeks mee te geven als tweede argument van `addEventListener`.

```
const button = document.getElementsByTagName('button')[0];
button.addEventListener('click', () => {
  console.log('Hello');
});
```

6.12 OPDRACHT: Lichtknop

-