

Introductie

Wat is JavaScript?

JavaScript, afgekort JS, is een **objectgeoriënteerde** (Eng. *object-oriented*) scripting- of programmeertaal om onder andere interactieve webpagina's te maken. Interactief betekent dat we via **client-side** JavaScript functionaliteit kunnen toevoegen aan een website, bijvoorbeeld: animaties, klikken op knoppen, formuliervalidatie, laden van externe data, 2D/3D graphics, scrolling video ... JavaScript is cross-browser (kan uitgevoerd in alle moderne browsers) en cross-platform (kan uitgevoerd worden op verschillende besturingssystemen). JavaScript behoort tot de drie fundamenteën van het web, namelijk: `HTML`, `CSS` en `JavaScript`.

Via `HTML` is de markuptaal die we gebruiken om webpagina's te structureren. `HTML` geeft een betekenis aan de webinhoud, bijv.: paragrafen, tabellen, afbeeldingen ... `CSS` is de taal van stijlregels die we kunnen gebruiken om `HTML` inhoud te stijlen of op te maken. JavaScript is de scripttaal die we gebruiken om webinhoud dynamisch up te daten en ermee te intrageren.

📁 Mappen & bestanden

```
introduction/  
├─ index.html  
├─ script.js  
└─ style.css :::
```

📄 ./introduction/index.html

html

```
1  <!DOCTYPE html>  
2  <meta charset="UTF-8">  
3  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
4  <title>Voorbeeld</title>  
5  <link rel="stylesheet" href="style.css">  
6  <script defer src="script.js"></script>  
7  <div class="person">  
8      <div class="fullname">Philippe De Pauw - Waterschoot</div>  
9  </div>
```

📄 ./introduction/script.js

js

```

1  const pElement = document.querySelector(".person");
2  pElement.addEventListener("click", function(ev) {
3      const fullName = prompt("Enter full name");
4      pElement.querySelector(".fullname").textContent = fullName;
5  });

```

./introduction/style.css

CSS

```

1  @import url("https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,900;1,400&disp
2
3  .person {
4      background: rgba(92, 0, 253, 1);
5      border-radius: 24px;
6      border: 4px solid rgba(0, 170, 204, 1);
7      color: rgba(0, 170, 204, 1);
8      cursor: pointer;
9      display: inline-block;
10     font-family: "Roboto", sans-serif;
11     font-size: 28px;
12     font-weight: 900;
13     letter-spacing: 0.2em;
14     padding: 6px 12px;
15     text-align: center;
16     text-transform: uppercase;
17 }

```

JavaScript wordt ook vaak gebruikt om functionaliteiten te schrijven voor bestaande software, zoals: Adobe Creative Cloud, Google Docs, Firebase Functions, ... JS kan eveneens gebruikt worden als **server-side** programmeertaal via onder andere Node.js om bijvoorbeeld te communiceren met een databank of om bestanden te manipuleren op de server.

JavaScript draait binnen een **omgeving** (*Eng. environment*) en kan verbonden worden met de objecten binnen deze omgeving waardoor we deze object kunnen manipuleren via code. JavaScript bevat een standaard bibliotheek van objecten (`Array` , `Date` , `Math` ...) en een verzameling van elementen (operatoren, controle structuren ...).

ECMAScript

JavaScript is gestandaardiseerd door [\[Ecma International\]](#)^[1] - European association for standardizing information and communication systems. [Ecma International](#) heeft als doel om een international gestandaardiseerde scripttaal aan te

bieden via de [ECMA-262]-specificatie – beter gekend als **ECMAScript**. ECMAScript is een standaard voor o.a.

JavaScript en voegt bij iedere nieuwe release extra features hieraan toe. [ECMA-262] is ook goedgekeurd door [ISO] als [ISO/IEC.22275].

Programmeertalen ActionScript (Adobe), JScript (Microsoft), JavaScript ... zijn allen een implementatie van de ECMAScript-specificatie. De ECMAScript-specificatie beschrijft niet de DOM API of andere client side Web API's. Deze worden gestandaardiseerd door het [W3C] en [WHATWG].

Geschiedenis van JavaScript:

- 1995
Brendan Eich ontwikkelt **Mocha** (naam gekozen door **Marc Andreessen**) in het bedrijf Netscape. Een aantal maanden later werd Mocha hernoemd naar **LiveScript**. Op dat moment had Sun Microsystems het handelsmerk **JavaScript** in handen. Netscape verkreeg een licentie van het handelsmerk JavaScript. LiveScript werd vervolgens hernoemd naar JavaScript in December 1995 vooral omwille dat **Java** op dat moment enorm populair was. JavaScript en Java waren op dat moment niet echt vergelijkbaar.
- 1996
Netscape stelt JavaScript voor aan [Ecma International] om ervoor te zorgen dat webbrowsers deze specificatie kunnen implementeren. Het Ecma International [Technical Committee 39] (TC39) werd aangeduid om JavaScript te laten evolueren naar de standaard scriptingtaal binnen webbrowsers.
- 1997
JavaScript wordt ondersteund door de specificatie van scriptingtalen in de **ECMA-262 Ed.1** standaard. JavaScript wordt gezien als de programmeertaal die de ECMA-262 het meest volgt. ActionScript (Macromedia, nu opgeslorpt door Adobe) en JScript (Microsoft) implementeren ook deze standaard specificatie.
- 1997 **ECMAScript 1 (ES1)**
JavaScript implementeert deze standaard in full glory.
- 1998 **ECMAScript 2 (ES2)**
JavaScript implementeert deze standaard met minimale verbeteringen ten opzichte van de vorige standaard.
- 1999 **ECMAScript 3 (ES3)**
JavaScript implementeert heel wat features om een volwaardige programmeertaal te worden, zoals: reguliere expressies, `try...catch` statements, formatteren van data ...
- 2008 **ECMAScript 4 (ES4)**
Door interne twisten in het technisch comité werd ECMAScript 4 officieel niet goedgekeurd (vooral omdat het te veel nieuwe features bevatte t.o.v. de vorige versie).
- 2009 **ECMAScript 5 (ES5)**
Deze versie wordt volledige ondersteund in alle browsers, behalve voor Internet Explorer 8. Het bevat redelijk wat nieuwe features tegenover ES3, zoals: JSON, Array.prototype.methode, methoden om eigenschappen op te lijsten, strict mode, hangende comma's ...
- 2015 **ECMAScript 6 (ES6)**
ECMAScript 2015. Features: let, const, arrow (`=>`), classes (`class` , `extends` , `constructor` , `super` , `get` , `set` , `static`), enhanced object literals, template strings (```), destructuring, default parameters, rest

parameters, spread operator, iterators, generators, `for...of`, full unicode, modules (`export`, `import`, `from`), module loaders, `Map`, `Set`, `WeakMap`, `WeakSet`, `Proxy`, `Symbol`, `Promise`, `Reflect API`, `Math + Number + String + Array`, tail calls

- 2016 **ECMAScript 7** (ES7)

ECMAScript 2016. Features: `Array.prototype.includes()`, Exponentiation operator (`**`)

- 2017 **ECMAScript 8** (ES8)

ECMAScript 2017. Nieuwe features: `Object.values()`, `Object.entries()`, `String.prototype.padStart()`, `String.prototype.padEnd()`, `Object.getOwnPropertyDescriptors()`, Async Functions, `async function`, `async function expression`, `AsyncFunction`, `await` en Trailing commas in function parameter lists.

- 2018 **ECMAScript 9** (ES9)

ECMAScript 2018. Nieuwe features: Spread in Object Literals and Rest parameters, `for await ... of`, `SharedArrayBuffer`, `Promise.finally()`, `RegExp.dotAll`, `RegExp Lookbehind Assertions`, `RegExp Unicode Property Escapes` en `RegExp Named Capture Groups`.

- 2019 **ECMAScript 10** (ES10)

ECMAScript 2019. Nieuwe features: `Array.Flat()`, `Array.flatMap()`, `Object.fromEntries()`, `String.trimStart()`, `String.trimEnd()`, `Optional Catch Binding`, `Function.toString()`, `.Symbol.description`, `Well Formed JSON.stringify()` en `Array.Sort Stability`.

- 2020 **ECMAScript 11** (ES11)

ECMAScript 2020. Nieuwe features: `?.`, `??`, `globalThis`, `Promise.allSettled()`, `import()`, `BigInt`.

- 2021 **ECMAScript 12** (ES12)

ECMAScript 2021. Nieuwe features: <https://dev.to/naimlatifi5/ecmascript-2021-es12-new-features-2l67>

JavaScript is geen Java

JavaScript en Java zijn gelijkwaardig op bepaalde elementen, maar verschillend op andere. JavaScript was eerst gekend als LiveScript, maar werd hernoemt naar JavaScript omwille van de gelijkenissen met Java. JavaScript volgt grotendeels de **syntaxis** van Java, de **naamgeving** en de **control flow**.

JavaScript bevat **geen static typing** zoals in Java, maar wel **dynamic typing**. Dynamic typing betekent dat variabelen, klassen en methoden niet gedeclareerd hoeven te worden voordat het gebruikt kan worden, bij static typing moet dit wel gebeuren.

JavaScript is **loosely typed** (of **weak typed**), Java **strongly typed**. Loosely typed betekent dat variabelen, klassen en methoden niet van een specifiek datatype hoeven te zijn. Ze zijn niet gebonden aan een bepaald datatype, maar bevatten wel een type tijdens het uitvoeren. In Java moet dit wel gebeuren!

JavaScript kan niet standaard aan het bestandensysteem, Java kan dit wel.

Wat kan je ermee doen?

Met de JavaScript programmeertaal kunnen we heel veel doen:

- waarden bewaren in variabelen, zoals in het voorgaande voorbeeld waarin het element met de klasse `.person` bewaren in de variabele `pElement` .
- code uitvoeren als **antwoord** (*Eng. response*) op bepaalde gebeurtenissen. Wanneer een gebruiker klikt op het element met de klasse `.person` , dan tonen we een popup waarin de gebruiker een andere naam kan ingeven.
- ...

Naast de standaard JavaScript functionaliteiten worden extra functionaliteiten aangeboden via gestandaardiseerde **Browser API's**. API's zijn een verzameling van bouwstenen die ons toelaten om uitgebreide applicaties te schrijven op een eenvoudige manier met beperkte code. Deze bouwstenen zijn direct bruikbaar en zijn tevens uitvoerig getest door de community.

Browser API's zijn binnen een webbrowser ingebouwd en kunnen daarom toegevoegd worden aan iedere webgebaseerde applicatie waardoor extra functionaliteiten kunnen toegevoegd worden, bijvoorbeeld:

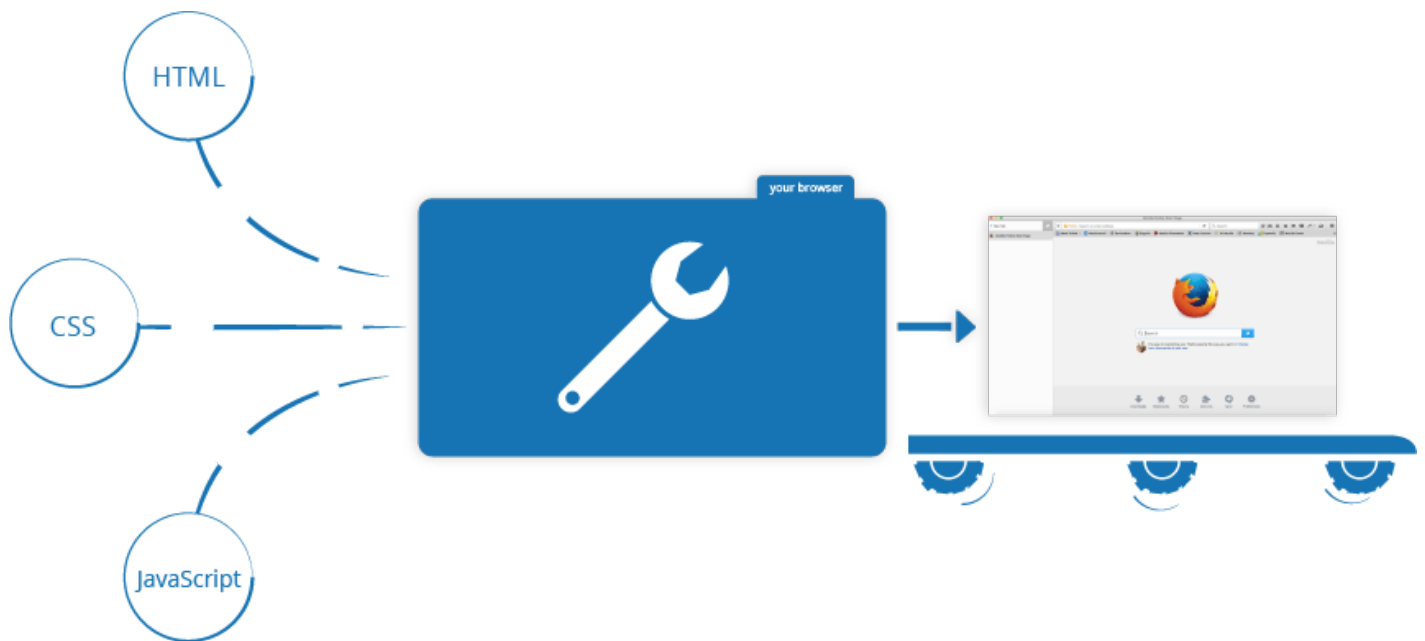
- De **Document Object Model (DOM) API** laat toe om HTML and CSS te manipuleren.
- De **Geolocation API** vraagt de geographische informatie van de gebruiker op, waaronder de latitude en de longitude. Met deze informatie kan o.a. **Google Maps** , **mapbox** ... jouw locatie weergeven op een map.
- De **Canvas** en **WebGL** APIs laten toe om 2D- en 3D-computergrafiek te genereren en te animeren.
- **Audio and Video APIs** zoals **HTMLMediaElement** en **WebRTC** laten toe om audio en video af te spelen en te manipuleren of video te streamen van een web camera naar één of meerdere computers.
- Met de **Notification** API kunnen we ervoor zorgen dat webpagina's systeem notificaties kunnen aanspreken van de eindgebruiker ook wanneer een webpagina niet in het actieve tabblad wordt weergegeven en ook wanneer andere applicaties op de voorgrond actief zijn.
- **Fetch API** wordt gebruikt om externe data in te laden.
- **Gamepad** API wordt gebruikt om gamepads en andere game controllers in connecteren met een webpagina.
- Met **Vibration** API kunnen we toegang hebben tot de vibratie hardware van een device, indien er ondersteuning daarvoor is voorzien.
- **WebXR Device** API brengt de wereld van XR (AR en VR) dichterbij om te draaien binnen een webbrowser.
- ...

Naast de Browser API's zijn ook externe API's / bibliotheken beschikbaar waarnaar we verwijzingen moeten doen via het `<script>` element of via `require` / `import` statements in JavaScript. We kunnen bijvoorbeeld de **Google Maps API** , **OpenStreetMap API** , **Google Firebase API** , ... integreren in onze toekomstige applicatie.

Wat doet JavaScript op een webpagina?

Wanneer we een webpagina laden in een webbrowser, zal code (HTML, CSS en JavaScript) uitgevoerd worden in een uitvoerbare omgeving en dit door de ingebouwde browserengine. Dit is vergelijkbaar met een meubelbedrijf die start met ruwe materialen (cf.: code) en als output een meubel oplevert (cf. webpagina of een verzameling van webpagina's, kortom een website).

JavaScript wordt uitgevoerd door de ingebouwde JavaScript engine, meestal nadat de HTML en CSS samengesteld zijn. Dit zorgt ervoor dat de structuur (HTML) en de opmaak (CSS) al aanwezig zijn voordat JavaScript de webpagina kan manipuleren.



What is JavaScript doing on your page?. Bron: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

HTML en CSS kunnen gemanipuleerd worden via JavaScript via de **Document Object Model API**. Het is belangrijk in dit geval dat we eerst HTML en CSS laden en dan pas JavaScript. JavaScript kan ook uitgevoerd worden voordat alle HTML inhoud geladen is.

Elke browser tab is een aparte **container** (*Eng. execution environment*) om code uit te voeren. Dit betekent dat de code in iedere tab onafhankelijk van elkaar kan uitgevoerd worden, zonder met elkaar te intrageren. Het is wel mogelijk om over tabs heen code uit te voeren, maar omwille van beveiligingsredenen doen we dit niet.

Client-side JavaScript is code dat uitgevoerd wordt op de computer van de gebruiker. Wanneer de webpagina wordt bekeken door de gebruiker, dan wordt alle content (dus ook de code) gedownload van de server. Vervolgens worden deze bestanden uitgevoerd in de browser. Content kan hardgecodeerd (*Eng. hard coded*) worden in de HTML.

Websites die hard-coded content bevatten noemen we **statische websites**. Statische websites kunnen we dynamisch maken door de inhoud synchroon of asynchroon te laden van externe bronnen via `XMLHttpRequest` of `fetch`.

Server-side JavaScript is code dat uitgevoerd wordt op de server. Het resultaat van deze code wordt gedownload door de gebruiker tijdens het bezoeken van een specifieke url via een webbrowser. De gedownloade assets worden vervolgens gevisualiseerd in de browser. Andere populaire server programmeertalen zijn: PHP, ASP.NET, Ruby, Python, Java ... Server-side code genereert nieuwe dynamische content op de server, bijv.: het ophalen van data uit een databank.

Starten met JavaScript

Starten met JavaScript is vrij eenvoudig. Wat je nodig hebt is een basisontwikkelomgeving en een engine die JavaScript-bestanden kan uitvoeren. Met Node.js Command-Line Interface kunnen we JavaScript-bestanden uitvoeren via de Command Prompt (Windows) of Terminal (macOS). Met de ingebouwde engine in een webbrowser kunnen we ook JavaScript uitvoeren.

Node.js

Met de Node.js command Line Interface kunnen we JavaScript-bestanden uitvoeren zonder dat HTML en dus ook een browser nodig is. Dit is handig om ons te focussen op één programmeertaal. We hebben reeds de noodzakelijke tools geïnstalleerd, waaronder de Node.js versie **16.17.0 LTS**.

Versie van actieve Node.js opvragen:

```
$ node -v
v16.17.0
```

We maken een `index.js` bestand met de volgende code:

`./index.js`

```
1  const firstName = "Philippe";
2  const surName = "De Pauw - Waterschoot";
3
4  const message = `
5  My personal information
6  =====
```

```

7   Firstname:\t${firstName}
8   Surname:\t${surName}
9   `;
10
11  console.log(message);

```

Vervolgens voeren we het `index.js` bestand uit via Node.js CLI:

```
$ node index.js
```

Resultaat:

```

My personal information
=====
Firstname:      Philippe
Surname:        De Pauw - Waterschoot

```

Node.js bevat standaard een verzameling van [ingebouwde modules](#), waaronder de [os module](#). Met deze module kunnen informatie van het **Operating System** ophalen.

```

1  const os = require("os");
2  const info = `
3  Platform:\t${os.platform()}
4  Release:\t${os.release()}
5  `;
6  console.log(info);

```

Resultaat na `node index.js` :

```

Platform:      darwin
Release:       18.7.0

```

Via de ingebouwde module [HTTP](#) kunnen we zelfs een server implementeren:

```

1  const http = require("http");
2  const port = 8081;
3  http
4  .createServer(function (req, res) {
5    res.writeHead(200, { "Content-Type": "text/html" });


```




```
6     res.end("Hello World!");
7   })
8   .listen(port);
```

Bezoeken van de url `http://localhost:8081/` in de browser resulteert in een webpagina (`'Content-Type': 'text/html'`) met de tekst `Hello World!` . De statuscode `200` dat de **response** `OK` is, kortom succesvol.

Browser

In Firefox kunnen we op JavaScript-code schrijven met de [Web Console](#) 

De Web Console toont ons informatie over de huidige geladen webpagina en bevat tevens een commandline waarmee we JavaScript expressies kunnen uitvoeren op de huidige webpagina. Deze console kunnen we openen via `⌘ Shift+Ctrl+I` (Windows), `⌘ Cmd+⌘ Alt+K` (macOS) of selecteer vervolgens **Console** uit het **Developer menu** onder het **Tools menu**. Om dan meerdere regels te kunnen typen gebruiken we `Ctrl+B` (`Cmd+B` op een Mac). Vroeger werd hiervoor Scratchpad gebruikt maar dit is verouderd.

In Chrome kunnen we JavaScript schrijven via de ingebouwde [Console](#)  binnen de DevTools. Je kan deze DevTools openen via `⌘ Shift+Ctrl+I` (Windows), `⌘ Cmd+⌘ Alt+K` (macOS) of selecteer **Hulpprogramma's voor ontwikkelaars** menu onder het menu **Meer hulpprogramma's**.

Uiteraard kunnen we ook JavaScript toevoegen aan een webpagina. In het `HTML`-bestand van deze webpagina kunnen we interne JavaScript schrijven via een of meerdere `<script>` tags of kunnen we een of meerdere JavaScript-bestanden linken via dezelfde `<script>` tags maar voorzien van minstens een attribuut genaamd `src` met als waarde een verwijzing naar het JavaScript-bestand.

JavaScript wordt op een gelijkaardige manier als **CSS** toegevoegd aan een `HTML` pagina.

`<script>` blok

JavaScript kan toegevoegd worden binnenin een `<script>` element in de `<head>` .

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <script>
5        // JavaScript goes here
6      </script>
```

html

```
7     </head>
8     <body></body>
9 </html>
```

Als we op deze manier scripts toevoegen dan moet de code wel luisteren naar de gebeurtenis `DOMContentLoaded`. Deze gebeurtenis treedt op wanneer de inhoud van de webpagina geladen is.

```
1 <script>
2     document.addEventListener("DOMContentLoaded", function () {
3         console.log("Document Loaded and ready to be used!");
4     });
5 </script>
```

JavaScript kan ook toegevoegd worden binnenin een `<script>` element net voor het sluiten van het `<body>` element. We hoeven niet meer te luisteren naar de gebeurtenis `DOMContentLoaded` omdat alle DOM elementen reeds geladen zijn.

```
1 <!DOCTYPE html>
2 <html lang="en">
3     <head> </head>
4     <body>
5         <script>
6             (void function () {
7                 console.log("Document Loaded and ready to be used!");
8             })();
9         </script>
10    </body>
11 </html>
```

In het bovenstaande voorbeeld gebruiken we een **IIFE** (Immediately Invoked Function Expression). Dit is een JavaScript functie expressie die direct wordt uitgevoerd na de definitie. Variabelen, functies ... binnen deze IIFE kunnen niet buiten deze IIFE aangesproken worden.

Extern JavaScript-bestand

Wanneer de JavaScript-code te lang wordt en wanneer we de code willen structureren per onderwerp of feature, dan kunnen we best werken met externe JavaScript-bestanden.

Werkwijze:

1. Maak een folder aan voor jouw nieuwe webapplicatie, bijv.: `tinder`
2. Open deze folder in Visual Studio Code of een andere IDE (Integrated Development Environment)
3. Maak een `HTML`-bestand aan binnen deze folder
4. Maak vervolgens een subfolder `js` aan
5. Maak een bestand `main.js` aan binnen deze subfolder
6. Link het extern JavaScript-bestand in de `HTML` via het `<script>` element door het `src` attribuut in te stellen met als waarde het **relatief pad** naar dit JavaScript-bestand. In dit geval bevat de waarde de volgende string: `/js/main.js`. Het `<script>` element kunnen we toevoegen in de `<head>` of net voor het sluiten van het `<body>` element.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <script src="/js/main.js"></script>
5   </head>
6   <body></body>
7 </html>
```

De JavaScript-code moet wel luisteren naar de gebeurtenis `DOMContentLoaded`, wanneer we deze code vermelden in de `<head>`. Deze gebeurtenis treedt op wanneer de inhoud van de webpagina geladen is. We kunnen dit wel vervangen door speciale attributen te gebruiken van het `<script>` waaronder `defer`.

```
1 document.addEventListener("DOMContentLoaded", function () {
2   console.log("The Tinder App is loaded!");
3 });
```

Linken we het JavaScript-bestand net voor het sluiten van de `<body>`, dan hoeven we deze luisteraar niet te registreren. Wanneer in de `<head>`, wanneer boven `</body>`? Alles hangt af van de functionaliteit die de code zal bieden.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head> </head>
4   <body>
5     <script src="/js/main.js"></script>
6   </body>
7 </html>
```

Het gekoppeld `main.js` JavaScript-bestand:

```
1 (void function () {
2   console.log("Document Loaded and ready to be used!");
3 })();
```

Inline JavaScript afhandelaar

In **HTML** kunnen we in een element via **specifieke attributen**, zoals: `onclick`, `onkeypress`, `onfocus` ..., **afhandelaars** (*Eng. handler*) registreren die uitgevoerd zullen worden wanneer een specifieke gebeurtenis optreedt. De gebeurtenis “klikken op een element” komt overeen met het attribuut `onclick`. De waarde van dit attribuut is de afhandelaar. In de afhandelaar definiëren we de instructies die uitgevoerd moeten worden.

```
1 <button onclick="createNewPerson();">New Person</button>
```

In het bovenstaande voorbeeld koppelen we een afhandelaar aan de gebeurtenis `onclick`. Deze “Event Handler” voert de instructie `createNewPerson()` uit. Dit is een functie die we geïmplementeerd hebben in een gekoppeld extern JavaScript-bestand.

```
1 function createNewPerson() {
2   alert("This will create a new form in order to create a new person.");
3 }
```

De functie `createNewPerson()` toont een popup venster met daarin de string `'This will create a new form in order to create a new person.'`.

Stel dat we een 10-tal elementen hebben met dezelfde event handlers, dan moeten we een 10-tal keer dezelfde luisteraar koppelen. Wijzigen we bijvoorbeeld de functienaam naar `createPerson`, dan moeten we deze wijziging doorvoeren op een 10-tal plaatsen.

Opgelet

Vermijd het gebruik van inline JavaScript afhandelaars, omdat:

- de **HTML** code moet leesbaar zijn en dient hoofdzakelijk om de webpagina semantisch te structureren
- de JavaScript-code moet beheersbaar blijven, dus werk steeds met externe JavaScript-bestanden en verdeel deze per feature (*Eng. separate by concern*).

```

1 <div class="articles">
2   <article class="article">
3     <span class="btnReadMore" data-id="1">Read More...</span>
4   </article>
5   ...
6 </div>

```

In het voorbeeld definiëren we een reeks artikelen via het `<article>` element met `"article"` als waarde van het `class` attribuut. Elk artikel bevat een `` element die fungeert als knop met identificatie `class="btnReadMore"` en een speciaal **zelfgeschreven** (*Eng. custom*) attribuut `data-id` waarin we de unieke identificatie (primaire sleutel in een databank) bijhouden van ieder artikel. We registreren geen inline JavaScript afhandelaar, dit doen we via een extern gekoppeld JavaScript-bestand.

```

1 const articleElements = document.querySelectorAll(".articles .article");
2 articleElements.forEach(function (element) {
3   element
4     .querySelector(".btnReadMore")
5     .addEventListener("click", function (ev) {
6       gotoArticleDetail(this.dataset.id);
7     });
8 });
9 function gotoArticleDetail(id) {
10   alert(
11     "The browser will navigate to the details page of the article with id " + id
12   );
13 }

```

Via het **Document Object Model** kunnen we de **HTML** en **CSS** manipuleren. Dit zullen we leren in een apart hoofdstuk **DOM**. De JavaScript-code gaat als volgt:

1. Selecteer alle artikelen via de query `.articles .article`
2. Loop doorheen deze artikel elementen via de `forEach` methode van het array object
3. Voor ieder element, selecteer het sub-element met de selector `.btnReadMore`
4. Voor dit element registreer een event handler die luistert naar het `click` event
5. Binnen deze luisteraar roepen we de function `gotoArticleDetail` aan met als parameter de waarde van het attribuut `data-id`
6. De functie `gotoArticleDetail` toont een popup met de tekst: `'The browser will navigate to the details page of the article with id x'`. `x` bevat de waarde van het attribuut `data-id`

1. De voorloper van **Ecma International** was **ECMA** (*European Computer Manufacturers Association*). ↩

[Grammatica en datatypen](#) →