

1.	ES6 VS ES5.....	3
2.	VARIABLEN.....	3
2.1.	VERSCHIL TUSSEN VAR EN LET.....	4
3.	FUNCTIONS.....	6
3.1.	FUNCTION SCOPING.....	6
3.2.	NAMED FUNCTIONS.....	7
3.3.	FUNCTIONS EN STRINGS IN ES6.....	9
3.4.	ES6 ARROW FUNCTION (=>).....	10
3.5.	ES6 ARRAYS.....	11
4.	NODES .....	12
4.1.	ROOT NODES .....	12
5.	JAVASCRIPT OUTPUT METHODES .....	14
5.1.	DOCUMENT WRITE EN DOCUMENT.WRITELN .....	14
5.2.	INNERHTML, TEXTCONTENT, INNERTEXT .....	15
5.3.	DOCUMENT.GETELEMENT.....	16
5.3.1.	DOCUMENT.GETELEMENTSBYCLASSNAME.....	16
5.4.	DOCUMENT.QUERYSELECTOR EN DOCUMENT. QUERYSELECTORALL 17	
5.5.	ANDERE DOM ELEMENTEN .....	18
6.	EVENTS EN FORMULIEREN.....	19
6.1.	EVENT (LISTENERS).....	19
6.1.1.	ONLOAD EVENT.....	19
6.1.2.	ONCLICK EVENT .....	20
6.1.3.	GEEN EVENT.....	21
6.1.4.	PREVENTDEFAULT().....	22
6.1.5.	MOUSE EVENT.....	23
6.1.6.	VOORBEELD: TODO LIST .....	26
7.	OBJECT .....	29
7.1.	BUILT-IN OBJECTEN .....	29
7.1.1.	DATE - NEW.....	29
7.1.1.1.	.....GET METHODES 29	
7.1.1.2.	.....SET METHODES 30	
7.2.	EEN INSTANTIE VAN EEN OBJECT .....	31
7.3.	EEN INSTANTIE VAN EEN OBJECT .....	31
7.4.	CONSTRUCTOR.....	32
7.5.	OBJECT CLASS EN INHERITANCE .....	33
7.6.	INHERITANCE .....	33
8.	BOM .....	36
8.1.	SCREEN.....	36
8.2.	COOKIES.....	37
8.3.	COOKIES MAKEN EN LEZEN.....	37
9.	AJAX.....	38
9.1.	WAT IS AJAX? .....	38
9.1.1.	XMLHttpRequest Methodes.....	38

9.1.2.	XMLHttpRequest Properties .....	39
9.1.3.	TXT BESTAND VAN EEN WEBSERVER LADEN. ....	40
9.1.4.	XML BESTAND VAN EEN WEBSERVER LADEN. ....	41
9.1.5.	XML - METHODE GETALLRESPONSEHEADERS. ....	45
9.1.6.	JSON BESTAND VAN EEN WEBSERVER LADEN. ....	46
9.2.	ASYNCHRONOUS VS SYNCHRONOUS? .....	52
9.2.1.	SYNCHRONOUS.....	52
9.2.2.	ASYNCHRONOUS .....	54
9.3.	CALLBACK OF PROMISE .....	54
9.3.1.	CALLBACK.....	54
9.3.2.	PROMISE.....	56
9.3.2.1.	.....ASYNC AWAIT	57
9.3.2.1.1.	.....ASYNC	57
9.4.	AXIOS? .....	59
1.	AJAX.....	61
10.	GEBRUIK VAN API'S.....	61

## 1. ES6 VS ES5

In deze cursus zullen we het hebben over ES6 en de verschillen t.o.v. ES5! Wanneer je deze cursus start heb je de voorgaande ES5 cursus (leren programmeren met javascript) ook doorgenomen. Voor deze cursus maak ik gebruik van de JetBrains editor PHPStorm. Dit is een professionele editor die meerdere talen waaronder ook javascript bevat.

Opmerking: zoals je hieronder kan zien zal deze editor je helpen met de parameters binnen functies. Hieronder zie je bijvoorbeeld **test:** staan. Dit dien je in een andere editor dus NOOIT over te tikken. Het is PHPStorm die je hier begeleid.

```
functionScoped( test: true);
```

Als tweede voorbeeld **message:** dien je dus niet in tikken in een andere editor.

```
let getal1 = parseInt(prompt( message: 'Geef een eerste getal in:'));
```

## 2. VARIABLEN

In ES5 gebruikten we **var** als declaratie van een variabele.

Met de komst van ES6 werden **let** en **const** hier toegevoegd. We hebben dus drie mogelijkheden om te declareren nl.: var, let en const in ES6.

- **const** wordt gebruikt om een variabele mee te geven die niet kan worden overschreven tijdens de uitvoering van het programma.
  - **const** PI = 3.14;
- **let** wordt soms verkeerdelijk gezien als de vervanger van **var**

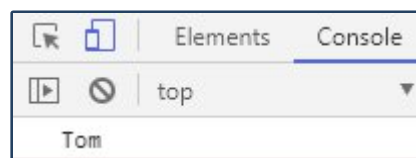
## 2.1. VERSCHIL TUSSEN VAR EN LET

**var** is een function scoped variabele en **let** is een block scoped variabele.

### In ES5:

de variabele naam zit BINNEN de functie en is dus functionScoped, d.w.z. dat hij met het keyword var bereikbaar is. (zie console output)

```
function functionScoped(test){  
  if(test){  
    var naam = 'Tom';  
  }  
  console.log(naam);  
}  
  
functionScoped( test: true);
```



### In ES6:

de variabele BINNEN de functie is blockScoped, d.w.z. enkel toegankelijk binnen het if statement en niet daarbuiten.

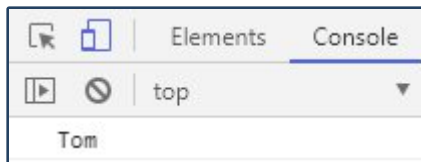
Wanneer we dus de functie laten uitvoeren met de variabele buiten de blockScope, dan krijg je een error.

```
function blockScoped(test){  
  if(test){  
    let naam = 'Tom';  
  }  
  console.log(naam);  
}  
  
blockScoped( test: true);
```

✖ ▶ Uncaught ReferenceError: naam is not defined  
at blockScoped (script.js:14)  
at script.js:16

Wanneer we dit echter BINNEN het IF-statement laten uitvoeren, dus binnen het block, dan wordt dit WEL uitgevoerd.

```
function blockScoped(test){  
  if(test){  
    let naam = 'Tom';  
    console.log(naam);  
  }  
}  
  
blockScoped( test: true);
```



Dit zou het verschil nu duidelijk moeten maken tussen **var** en **let**.

### 3. FUNCTIONS

Functies zijn één van de belangrijkste onderwerpen in gelijk welke programmeer- of scriptingtaal.

We hebben standaard built-in functies van javascript zelf die zeer uitgebreid zijn. Deze worden methods genoemd. Later meer hierover.

Eerst schrijven we onze eigen functie.

EEN FUNCTIE HEEFT STEEDS HET KEYWORD RETURN NODIG OM HET RESULTAAT OVER TE DRAGEN IN DE CODE

Notatie:

```
function naam(parameter, parameter, ....){  
    RETURN resultaat  
}
```

#### 3.1. FUNCTION SCOPING

Alles wat binnen de functie is gedefinieerd, is niet toegankelijk via code buiten de functie. Alleen code binnen dit bereik kan de variabelen/code zien die binnen het bereik zijn gedefinieerd.

#### CODE

Voorbeeld:

```
function foo() {  
    var tekst = 'hallo';  
    console.log(tekst); // => 'hallo'  
}  
console.log(tekst); // reference error
```

Voor geneste functies geldt dezelfde regel

Voorbeeld:

```
function foo() {  
    var tekst = 'hallo';  
    function bar() {  
        var tekst2 = 'world';  
        console.log(tekst); // => 'hallo'  
        console.log(tekst2); // => 'world'  
    }  
    console.log(tekst); // => 'hallo'  
    console.log(tekst2); // reference error  
}  
console.log(tekst); // reference error  
console.log(tekst2); // reference error
```

Wanneer JavaScript een referentie of variabele probeert op te lossen, begint het ernaar te zoeken in het huidige blok (level). Als het dit niet kan vinden in het huidige bereik, klimt het één level op om ernaar te zoeken. Dit proces herhaalt

zich tot de variabele is gevonden. Als de JavaScript-parser de variabele niet kan vinden zal hij een error weergeven.

#### CODE

Voorbeeld:

```
var a = 'hallo';
function foo() {
  var b = 'wereld';
  function bar() {
    var c = 'test';
    console.log(a); // => 'hallo'
    console.log(b); // => 'wereld'
    console.log(c); // => 'test'
    console.log(d); // reference error
  }
}
```

### 3.2. NAMED FUNCTIONS

Functies kunnen een naam krijgen of kunnen ook anonymous (geen naam) worden geschreven. Hieronder een voorbeeld waar beiden aan een variabele worden toegekend.

#### CODE

Voorbeeld:

```
var eenNaam = function sum (a, b) { // named
  return a + b;
}
var geenNaam = function (a, b) { // anonymous
  return a + b;
}
eenNaam (1, 3);
geenNaam (1, 3);
```

Wanneer toegekend aan een variabele blijven de functies PRIVATE in hun eigen scope (blok).

#### CODE

Voorbeeld: onderstaande geeft een reference error.

```
var eenNaam = function sum (a, b) { // named
  return a + b;
}
sum(1,3) //Reference error
```

## CODE

Voorbeeld: zelf geschreven functie.

```
function berekenLeeftijd(geboorteJaar){
    return huidigJaar - geboorteJaar;
}

var huidigJaar = 2020;
var geboorteJaar = prompt('Geef uw geboortejahr in:, YYYY:');

var resultaat = berekenLeeftijd(geboorteJaar);

if(resultaat >=0) {
    console.log('Het aantal jaren tussen ' + huidigJaar + 'en '
    + geboorteJaar + ' is ' + resultaat);
}else{
    console.log('Het resultaat kan niet negatief zijn.');
```

## RESULTAAT

```
Het aantal jaren tussen 2020en 1973 is 47
```

- Enkele regel commentaar: 2 enkele slashen (//)
- /\* meerdere regels commentaar \*/



### 3.3. FUNCTIONS EN STRINGS IN ES6

Wanneer we variabelen binnen in een string wilden weergeven in ES5 dan dienden we gebruik te maken van concatenatie (plus, komma of functie). In ES6 kunnen we de variabelen binnen een string embedden (incapselen). In het voorbeeld hieronder ziet u het verschil in schrijfwijze. Let op: in ES6 gebruiken we de backticks om dit te bewerkstelligen (`).

```
let naam = 'Tom';

//ES5
document.write( text: 'Mijn naam is:' + naam + '<br>');
//ES6
document.write( text: `Mijn naam is:${naam}`);
```

Ook functies kunnen binnen een string automatisch worden aangesproken.

```
/**functies binnen strings**/
function som(a,b) {
    return a+b;
}

let getal1 = parseInt(prompt( message: 'Geef een eerste getal in:'));
let getal2 = parseInt(prompt( message: 'Geef een tweede getal in:'));
document.write( text: `De som van:${getal1} + ${getal2} is ${som(getal1,getal2)}`);
```

### 3.4. ES6 ARROW FUNCTION (=>)

In ES6 kunnen we gebruik maken van de arrow function om bijvoorbeeld een array eenvoudiger te doorlopen.

In het onderstaande voorbeeld maken we gebruik van de map functionaliteit. Een map is een collectie van sleutels met telkens hun value. Dit lijkt op een object, maar het verschil hier is dat een map een collectie kan bevatten van verschillende datatypes.

In onderstaand voorbeeld zie je twee parameters staan: **el** en **index**. **el** staat voor de value, dus een waarde en **index** staat voor de locatie van deze waarde.

In ES5 dienden we de function(el) nog aan te spreken met daarnaast de return om de functie een resultaat te laten bepalen.

In ES6 gebruiken we de **arrow function** en retourneren we de uitkomst onmiddellijk.

Zie de voorbeelden hieronder:

```
const arrayGetallen = [1000,2000,3000,4000];

//ES5
var verschil = arrayGetallen.map( callbackfn: function(el) {
    return 5000 - el;
});
document.write( text: verschil + '<br>');
//resultaat: 4000,3000,2000,1000

//ES6
let verschil2 = arrayGetallen.map( callbackfn: el => 5000 - el);
document.write( text: `${verschil2}<br>`);

verschil2 = arrayGetallen.map( callbackfn: (el, index)=> `Element ${index + 1} = ${5000-el}`);
document.write( text: `${verschil2}`);
```

### 3.5. ES6 ARRAYS

Wanneer we arrays doorlopen dan spreken we eigenlijk over associatieve arrays in javascript, nl. sleutel en telkens hun value.

Ook hier kunnen we in ES6 de verkorte arrow function notatie gebruiken om ieder element van een array te doorlopen.

Met de `querySelectorAll` halen we iedere tag op die de naam resultaat draagt en voegen we die toe aan de `const mijnClass`.

Vervolgens doorlopen we deze nieuwe constante indien er meerdere zouden zijn op een pagina en wordt deze aan de variabele `mijnArray` toegekend.

Voor iedere element die de `className` resultaat draagt zal er een rood kleur worden toegekend. (ES5 voorbeeld).

In ES6 kunnen we dit in één enkele lijn schrijven. Hier zullen we ieder element wijzigen in groen.

Kopieer ook eens de onderstaande p tag meerdere keren in de pagina en je zal zien dat de `querySelectorAll` alle tags met dezelfde `className` resultaat zal aanpassen.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
  <p class="resultaat">
    Dit is een test.
  </p>

  <script src="script.js"></script>
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-...>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js">
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js">
</body>
</html>
```

```
//ES5
const mijnClass = document.querySelectorAll( selectors: '.resultaat' );

var mijnArray = Array.prototype.slice.call(mijnClass);
mijnArray.forEach( callbackfn: function(e1) {
  e1.style.color = 'red';
});

//ES6
Array.from(mijnClass).forEach( callbackfn: e1 => e1.style.color = 'green');
```

## 4. NODES

Binnen in de DOM hebben we de mogelijkheid om te navigeren tussen nodes in javascript. Iedere tag wordt als een element of ook wel node genoemd gezien. Er bestaan enkele methodes om iedere node of meerdere noden tegelijk aan te spreken.

### 4.1. ROOT NODES

Binnen de DOM hebben we de ROOT nodes van een document. We hebben deze reeds gebruikt in de vorige oefeningen.

ROOT nodes zijn o.a.: window, document, ....

Met de volgende nodes kunnen we navigeren tussen de onderliggende nodes met javascript.

- parentNode
- childNodes[nodenummer]
- firstChild
- lastChild
- nextSibling
- previousSibling

Voorbeeld:

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

<p>Full Stack Developers!</p>

<div>
<p>DOM ROOT NODES!</p>
<p>ophalen van alle nodes (tags) van een document met
javascript</p>
</div>

<script>
alert(document.body.innerHTML);
alert(document.documentElement.innerHTML);
</script>

</body>
</html>
```

Voorbeeld:

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

<h1 id="h101">My First Page</h1>
<p id="p02"></p>

<script>
document.getElementById("p02").innerHTML =
document.getElementById("h101").firstChild.nodeValue;
</script>

</body>
</html>
```

Voorbeeld:

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

<div id="div1">De text hierboven werd dynamisch
toegevoegd.</div>

<script>
document.body.onload = addElement;

function addElement () {
    // Een nieuwe div creëren met de property createElement
    var newDiv = document.createElement("div");

    // Toevoegen van tekst aan de nieuwe div
    var newContent = document.createTextNode("Hallo, Full Stack
Developers!");

    // voeg de tekst toe aan de nieuwe div
    newDiv.appendChild(newContent);

    // voeg de DIV VOOR de reeds bestaande div toe in de DOM
    var currentDiv = document.getElementById("div1");
    document.body.insertBefore(newDiv, currentDiv);
}
</script>

</body></html>
```

## 5. JAVASCRIPT OUTPUT METHODES

We hebben reeds 3 output methodes kort aangehaald:

`alert()`, `console.log()` en `document.getElementById("div").innerHTML`.

Er zijn er echter nog heel wat andere die we hier zullen opsommen. De meest gebruikte geven we hier dan ook terug.

### 5.1. DOCUMENT WRITE EN DOCUMENT.WRITELN

**document.write(variabele)** = print een variabele op het scherm

**document.writeln(variabele)** = print op 2 lijnen ENKEL wanneer het script omgeven is door html tags zoals bijvoorbeeld pre.

Voorbeeld:

```
<pre>
  <script>
    document.write("Hallo iedereen!");
    document.write("Klaar?");
  </script>
</pre>
<pre>
  <script>
    document.writeln("Hallo iedereen!");
    document.writeln("Klaar?");
  </script>
</pre>
<pre>
  <script>
    document.write("\t\tHallo\nworld!\n");
    document.writeln("\nHallo iedereen, Welkom .\n");
    document.writeln('Smiley face:\n');
  </script>
</pre>
```

Escape characters	Betekenis
\'	Single quote afdruk
\"	Double quote afdruk
\t	tabulator afstand
\n	Nieuwe regel
\r	Return
\f	Form feed
\b	Backspace
\e	Escape
\\	Backslash

Referenties:

Html entities : <https://unicode-table.com/en/html-entities/>

## 5.2.INNERHTML, TEXTCONTENT, INNERTEXT

index.html:

```
<body>
  <h1>Javascript Basis</h1>
  <h2>Source:</h2>
  <p id="source">
    <span>HIDDEN TEXT</span>
  </p>

  <h3>Resultaat van innerhtml:</h3>
  <p>Neemt alles mee binnen de div, nl. tags en values</p>
  <textarea id="innerHTML" rows="6" cols="30"
readonly>...</textarea>
  <h3>Resultaat van textContent:</h3>
  <p>Neemt enkel de values</p>
  <textarea id="textContentOutput" rows="6" cols="30"
readonly>...</textarea>
  <h3>Result van innerText:</h3>
  <p>Innertext is zich bewust van css</p>
  <textarea id="innerTextOutput" rows="6" cols="30"
readonly>...</textarea>
  <script src="js/script.js">
  </script>
</body>
```

css:

```
p span{display:none;}
```

script.js:

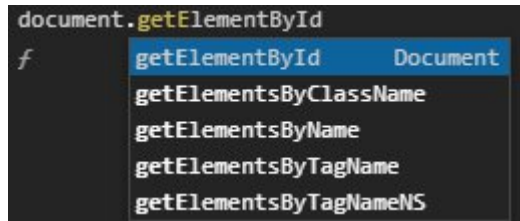
```
var source = document.getElementById("source");
var innerhtmlOutput = document.getElementById("innerHTML")
var textContentOutput =
document.getElementById("textContentOutput");
var innerTextOutput =
document.getElementById("innerTextOutput");
```

```
innerHTMLOutput.value = source.innerHTML;
textContentOutput.value = source.textContent;
innerTextOutput.value = source.innerText;
```

### 5.3. DOCUMENT.GETELEMENT

Er zijn nogal wat mogelijkheden om tags in HTML aan te spreken. Eén ervan `getElementById` hebben we reeds gezien.

De andere zijn:



De twee belangrijkste bespreken we, nl. `getElementsById` en `getElementsByClassName`.

#### 5.3.1. DOCUMENT.GETELEMENTSBYCLASSNAME

Er zijn nogal wat mogelijkheden om tags in HTML aan te spreken. Eén ervan is `getElementsByClassName`. Ipv het id van een tag aan te spreken, spreken we één of meerdere classes aan.

html:

```
<span class="bmw x5">BMW X5</span>
<span class="bmw x6">BMW X5</span>
<span class="mercedes c200">mercedes c200</span>
<span class="mercedes cla">mercedes cla</span>
<textarea id="resultArea"></textarea>
```

css:

```
#resultArea{
  width:80%;
  height:7em;
}
```

js:

```
var alleBMW = document.getElementsByClassName('bmw');
var result = "document.getElementsByClassName('bmw')";
for (var i=0, len=alleBMW.length; i<len; i=i+1) {
  result += "\n " + alleBMW[i].textContent;
}
```

```
document.getElementById("resultArea").value = result;
```

Opmerking: een nieuwe lijn wordt als volgt gebruikt.

`\n` = new line;



## 5.4. DOCUMENT.QUERYSELECTOR EN DOCUMENT. QUERYSELECTORALL

De queryselector retourneert het eerste element die voldoet.

Voorbeeld wanneer je een tag met een classnaam zou willen aanspreken en die komt 5 keer voor in je html document, dan zal hij enkel de eerste nemen.

Wanneer je ze allemaal zou willen aanspreken dan gebruiken we queryselectorAll.

html:

```
<span class="bmw x5">BMW X5</span>
<span class="bmw x6">BMW X5</span>
<span class="mercedes c200">mercedes c200</span>
<span class="mercedes cla">mercedes cla</span>
<textarea id="resultArea"></textarea>
```

CSS:

```
#resultArea{
    width:80%;
    height:7em;
}
```

js:

```
var alleBMW = document.querySelectorAll('.bmw');
var result = "document.querySelectorAll('.bmw')";
for (var i=0, len=alleBMW.length; i<len; i=i+1) {
    result += "\n " + alleBMW[i].textContent;
}
```

```
document.querySelector("#resultArea").value = result;
```

Zoals je kan zien is dit gemakkelijker in gebruik dan getElementById en getElementsByClassName. Het is echter de keuze van de developer om te gebruiken waar hij zich het best bij voelt.

## 5.5.ANDERE DOM ELEMENTEN

<b>focus()</b>		
Focus zorgt ervoor dat een veld de cursor dus de focus krijgt		
	html	<input type="text" id="myText" value="A text field">
	js	document.getElementById("myText").focus();
<b>blur()</b>		
Blur zorgt ervoor dat een veld de focus verliest.		
	html	<input type="text" id="myText" value="A text field">
	js	document.getElementById("myText").blur();
<b>childElement of children.length</b>		
childElement telt het aantal kinderen (childs) van een ouder (parent)		
	js	let numb = document.getElementById("myDIV").childElementCount of let numb = document.getElementById("myDIV").children.length;
<b>childNodes</b>		
childNodes worden gebruikt om de kinderen (childs) van een parents aan te spreken. In het voorbeeld hieronder geven we het aantal kinderen terug van een parent door length te koppelen.		
	js	let numb = element.childNodes.length
<b>children</b>		
onderstaand voorbeeld geef alle tagnames van de body weer op het scherm.		
	js	const collection = document.body.children; let text = ""; for (let i = 0; i < collection.length; i++) { text += collection[i].tagName + " "; } document.getElementById("demo").innerHTML = text;
<b>Add, remove en toggle classes</b>		
	html en js	Add: add zorgt ervoor dat de class myStyle uit je css wordt toegevoegd aan een div const list = document.getElementById("myDIV").classList; list.add("myStyle");  Remove: remove zorgt ervoor dat de class myStyle wordt verwijderd uit je html document.getElementById("myDIV").classList.remove("myStyle");  Toggle: een toggle heeft 2 statussen namelijk on/of, true/false De toggle zorgt ervoor dat de class myStyle wordt toegevoegd of weggelaten. Deze heeft dus de Add en Remove functionaliteit in huis. document.getElementById("myDIV").classList.toggle("myStyle");

Dit zijn slechts enkele voorbeelden die we kunnen gebruiken. Op de site van w3schools staan alle voorbeelden die we in javascript kunnen gebruiken. Bekijk deze lijst zorgvuldig. Je kan heel wat elementen gebruiken binnen je programmacode van javascript.

[https://www.w3schools.com/JSREF/dom\\_obj\\_all.asp](https://www.w3schools.com/JSREF/dom_obj_all.asp)

## 6. EVENTS EN FORMULIEREN

Maak een map JS2 aan telkens met een index.html bestand, styles.css bestand en een script.js bestand erin.

In formulieren (form tag) gebruiken we o.a. tekstvelden om waarden in te vullen. In HTML hebben jullie reeds gezien dat dit over input velden gaat. De button zal de tekst die wordt ingevuld in het tekstveld ophalen en via javascript weergeven in het resultaat.

Een event is een GEBEURTENIS. In dit geval is de gebeurtenis een KLIK op de button. In javascript dient een webpagina dus te gaan LUISTEREN wanneer er op die button wordt geklikt. Dit doen we door gebruik te maken van een methode EVENT LISTENER.

### 6.1. EVENT (LISTENERS)

#### 6.1.1. ONLOAD EVENT

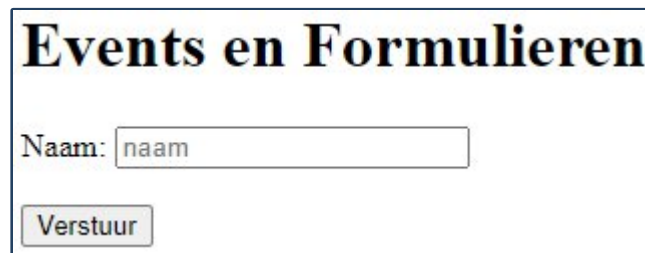
```
<body onload="mijnFunctie()">

</body>
<script>
    function mijnFunctie() {
        console.log("Hallo, ik werd ingeladen door de
browser");
    }
</script>
```

### 6.1.2. ONCLICK EVENT

Bij deze eerste optie gebruiken we BINNEN html een EVENT. In dit voorbeeld hieronder is het even **ONCLICK** met daarnaast een zelfgekozen functienaam `GETINPUTVALUE()` die we in ons js bestand zullen aanroepen.

#### RESULTAAT



#### HTML

```
<h1>Events en Formulieren</h1>
<form id="form">
  <label>Naam: <input id="textNaam" type="text"
placeholder="naam"></label>
  <br><br>
  <button id="btnSubmit" type="button"
onclick="getInputValue()">Verstuur</button>
</form>
<p id="resultaat"></p>
```

#### SCRIPT

```
function getInputValue() {
  var naam = document.getElementById("textNaam").value;
  document.getElementById("resultaat").innerHTML = naam;
}
```

#### OPMERKING:

Om op de pagina resultaten weer te geven dien je het **type=button** mee te geven. Wanneer je `type=submit` zou gebruiken dan voert hij dit ook uit, maar refreshed hij de pagina onmiddellijk waardoor je het resultaat niet ziet staan. De submit zullen we later nodig hebben wanneer we het formulier effectief dienen te versturen naar bijvoorbeeld een database.

### 6.1.3. GEEN EVENT

Bij de tweede optie gebruiken we BINNEN html NIETS van EVENTS. We zorgen ervoor dat tijdens het laden van de pagina alle EVENTS vanuit het JS bestand worden ingeladen. Dit doen we door een ADDEVENTLISTER methode te koppelen aan een bestaande HTML TAG. De ADDVENTLISTENER methode heeft vervolgens het type van event nodig als parameter. In dit geval is dit het CLICK EVENT.

#### HTML

```
<h1>Events en Formulieren</h1>
<form id="form">
  <label>Naam: <input id="textNaam" type="text"
placeholder="naam"></label>
  <br><br>
  <button id="btnSubmit" type="button">Verstuur</button>
</form>
<p id="resultaat"></p>
```

#### SCRIPT

```
window.onload = function(){

document.getElementById("btnSubmit").addEventListener('click',
function(){
  naam = document.getElementById("textNaam").value;
  document.getElementById("resultaat").innerHTML = naam;
  resultaat.innerHTML = '<h5>Welkom ' + naam + '</h5>';
},false);
}
```

#### 6.1.4. PREVENTDEFAULT()

Met de derde optie vangen we het volledige event op. In dit geval een click-event.

In deze pagina zal je zien dat we een checkbox zullen aanvinken. Met `event.preventDefault()` zullen we de standaard gedraging van het event uitschakelen. In dit geval is de standaard gedraging het aanvinken van een checkbox.

##### HTML

```
<form id="form">
  <label for="id-checkbox">Checkbox:</label>
  <input type="checkbox" id="id-checkbox"/>
</form>
<p id="resultaat"></p>
```

##### SCRIPT

```
document.querySelector("#id-
checkbox").addEventListener("click", function(event) {
  document.getElementById("resultaat").innerHTML += "Sorry!
<code>preventDefault()</code> verbied u dit aan te" +
    " vinken" + " this!<br>";
  event.preventDefault();
}, false);
```

Eén van de event listeners die we tot nu toe gezien hebben is het click event. Maar er zijn nog heel wat andere die we kunnen gebruiken zoals bijvoorbeeld een mouse event die zal reageren bij het aanklikken van de linkermuisknop of wanneer we over iets heen hoveren, ...

De volledige lijst kan je vinden onder deze link:

<https://developer.mozilla.org/en-US/docs/Web/Events>

De meest voorkomende zullen we samen van naderbij bekijken.

### 6.1.5. MOUSE EVENT

Maak een map JS4 aan telkens met een index.html bestand, styles.css bestand en een script.js bestand erin.

Er kunnen verschillende javascript events getriggerd worden wanneer je bijvoorbeeld over een html element heen zou bewegen, klikken, ...

Hieronder enkele mogelijk voorbeelden:

#### HTML

```
<!DOCTYPE html>
<html Lang="nl">

<head>
  <meta charset="UTF-8">
  <title>Javascript Basis</title>
  <link rel="stylesheet" href="styles.css">
</head>

<body>

  <h1>Javascript Basis</h1>
  <h2>DOM</h2>
  <h3>Events</h3>
  <h4>Mouse events</h4>
  <h5>Mouse over en mouse out</h5>
    <div id="binnen"></div>
    <div id="buiten"> </div>
    <script src="script.js"></script>
</body>

</html>
```

## CSS

```
#binnen{
  height: 200px;
  width: 200px;
  border: 2 px solid black;
  background: yellow;
}
```

## SCRIPT

De variabele **binnen** stelt het vierkant voor. Elk event hieronder spreekt voor zich. Probeer één voor één uit en bekijk het resultaat in je browser.

```
window.onload = function(){
  var binnen = document.getElementById('binnen');
  var buiten = document.getElementById('buiten');
  var x = 0;
  var z = 0

  binnen.onmousemove = function(){ //bewegen van de muis
    binnen.innerHTML = x+=1;
  }
  binnen.onmouseover = function(){ //bewegen van de muis over een html element
    binnen.style.backgroundColor = '#FAC';
  }
  binnen.onmouseout = function(){ //verlaten van de muis van een html element
    buiten.innerHTML += 'De muis beweegt uit het vierkant<br>';
    binnen.style.backgroundColor = '#FFF';
  }
  binnen.onmousedown = function(){ //Linkermuis ingedrukt
    binnen.innerHTML += z+=1;
  }
  binnen.onmouseleave = function(){ //verlaten html element
    binnen.innerHTML += z-=1;
  }
}
```

## RESULTAAT

### Javascript Basis

#### DOM

#### Events

#### Mouse events

#### Mouse over en mouse out

810

De muis beweegt uit het vierkant



Er zijn heel wat events in javascript.  
Hieronder een gedeelte van de lijst van events in javascript.

abort	hashchange	resize
afterprint	input	reset
animationend	invalid	scroll
animationiteration	keydown	search
animationstart	keypress	seeked
beforeprint	keyup	seeking
beforeunload	load	select
blur	loadeddata	show
canplay	loadedmetadata	stalled
canplaythrough	loadstart	storage
change	message	submit
click	mousedown	suspend
contextmenu	mouseenter	timeupdate
copy	mouseleave	toggle
cut	mousemove	touchcancel
dblclick	mouseover	touchend
drag	mouseout	touchmove
dragend	mouseup	touchstart
dragenter	mousewheel	transitionend
dragleave	offline	unload
dragover	online	volumechange
dragstart	open	waiting
drop	pagehide	wheel
durationchange	pageshow	
ended	paste	
error	pause	
focus	play	
focusin	playing	
focusout	popstate	
fullscreenchange	progress	
fullscreenerror	ratechange	

Het is onmogelijk om bovenstaande lijst helemaal te overlopen. Veel van deze elementen zul je ook nooit gebruiken.

Op de site van w3schools kan je deze lijst met voorbeelden ook bekijken.

Afhankelijk van wat je nodig hebt tijdens het schrijven van je code, geeft deze lijst telkens terug een voorbeeld.

[https://www.w3schools.com/JSREF/dom\\_obj\\_event.asp](https://www.w3schools.com/JSREF/dom_obj_event.asp)

### 6.1.6. VOORBEELD: TODO LIST

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width,
initial-scale=1">
  <title>Todo App</title>
  <style>
    .strikethrough{
      text-decoration:line-through;
    }
  </style>
</head>
<body>
<form id="new_task_form">
  <input id="new_task" type="text" placeholder="Create New
Task...">
  <button id="new_task_button" type="submit"></button>
</form>
<hr>
<section id="taskList">

</section>
<script>
  /** Toevoegen van een event listener op de form tag */
  var taskform = document.getElementById("new_task_form");
  taskform.setAttribute("style","background:red;");
  var taskList = document.getElementById("taskList");
  taskform.addEventListener("submit",function(e){
    e.preventDefault();//standaard gedraging van de submit
button uitschakelen.
    //var newTaskInputValue =
document.getElementById("new_task").value;
    var newTaskInputValue = taskform.elements.new_task;
    addTask(newTaskInputValue.value);
    //reset van de Inputvalue
    newTaskInputValue.value = "";
  });
  function addTask(newTaskInputValue){
    /** opbouw van de tags */
    const div = document.createElement("div");
    const trash = document.createElement("button");
    const checkbox = document.createElement("input");
    checkbox.type="checkbox";
    const label = document.createElement("label");
```

```

        /** Toevoegen van de waarden van de button en de
ingevulde taak door de gebruiker */
        trash.innerHTML = "DEL";
        label.innerText = newTaskInputValue;

        /**injectie volgorde HTML elementen*/
        div.appendChild(trash);
        div.appendChild(checkbox);
        div.appendChild(label);

        /** injectie van de volledig div in de tasklists */
        taskList.appendChild(div);

        /** addEventListener aan het checkbox element
verbinden met het click event */
        checkbox.addEventListener("click", function(){
            if(checkbox.checked == true){
                div.style.textDecoration = "line-through";
            } else{
                div.style.textDecoration = "none";
            }
        });

        /** addEventListener aan het trash element verbinden
met een click event. */
        trash.addEventListener("click", function(){
            div.remove();
        });
    }

</script>

</body>
</html>

```

OEFENINGEN:  
COUNTER, REKENLIJST, CAROUSSEL.

## 7. OBJECT

### 7.1.BUILT-IN OBJECTEN

#### 7.1.1. DATE - NEW

Het date object wordt steeds gecreëerd met het keyword NEW. Hieraan kun je herkennen dat je niet met een gewone variabele te maken hebt maar met een OBJECT VARIABLE. Het is dus geen functie! Een OBJECT VARIABLE bevat alle eigenschappen en functies (PROPERTIES EN METHODS) van het object.

##### 7.1.1.1. GET METHODES

De get methodes worden gebruikt om een systeemdatum op te halen als object. Het date object is één van de meest gebruikte objecten binnen javascript. Hieronder zie je een voorbeeld en al zijn mogelijke methodes.

```
<p id="demo"></p>
```

```
<script>  
    const d = new Date();  
    document.getElementById("demo").innerHTML = d.getTime();  
</script>
```

Method	Description
getFullYear()	Get the <b>year</b> as a four digit number (yyyy)
getMonth()	Get the <b>month</b> as a number (0-11)
getDate()	Get the <b>day</b> as a number (1-31)
getHours()	Get the <b>hour</b> (0-23)
getMinutes()	Get the <b>minute</b> (0-59)
getSeconds()	Get the <b>second</b> (0-59)
getMilliseconds()	Get the <b>millisecond</b> (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

#### 7.1.1.2. SET METHODES

SET methodes van het date object worden gebruikt om date variabelen op te vullen.

```
<p id="demo"></p>
```

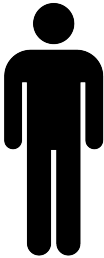
```
<script>
    const d = new Date();
    d.setFullYear(2020);
    document.getElementById("demo").innerHTML = d;
</script>
```

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

## 7.2. EEN INSTANTIE VAN EEN OBJECT

Een object heeft alle karakteristieken van het object. Deze karakteristieken worden bepaald door properties en methods.

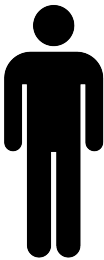
Properties zijn de eigenschappen die een object bepalen. Methods zijn de functies die een object kan uitvoeren.

OBJECT:persoon	PROPERTIES	METHODS
	persoon.naam	persoon.spreken()
	persoon.leeftijd	persoon.stappen()
	persoon.geslacht	persoon.stoppen()
	...	...

## 7.3. EEN INSTANTIE VAN EEN OBJECT

De karakteristieken en methodes van een object kunnen meerdere malen worden herbruikt! Dit noemen we telkens een instantie van een object.

**Instantie 1** van object persoon:

OBJECT:persoon	PROPERTIES	METHODS
	persoon.naam = <b>Tom</b>	persoon.spreken()
	persoon.geboortedatum = 1973	persoon.stappen()
	persoon.geslacht = <b>m</b>	persoon.stoppen()
	...	persoon.berekenLeeftijd() ...

**Instantie 2** van object persoon:

OBJECT:persoon	PROPERTIES	METHODS
	persoon.naam = <b>Tim</b>	persoon.spreken()
	persoon.geboortedatum = <b>1980</b>	persoon.stappen()
	persoon.geslacht = <b>m</b>	persoon.stoppen()
	...	persoon.berekenLeeftijd() ...

**Instantie 3 ...**

Zoals je ziet wordt telkens hetzelfde raamwerk, nl. het object gebruikt om deze te vullen met andere waarden. Dit noemen we het instantiëren van objecten. Eén

instantie bevat telkens de waarden in dit geval van één persoon met zijn gegevens.

```
var persoonA = new Object();
persoonA.naam = "Tom";
persoonA.geboortedatum = 1973;
persoonA.geslacht="m";
```

```
var persoonB = new Object();
persoonB.naam = "Tom";
persoonB.geboortedatum = 1973;
persoonB.geslacht="m";
```

## 7.4.CONSTRUCTOR

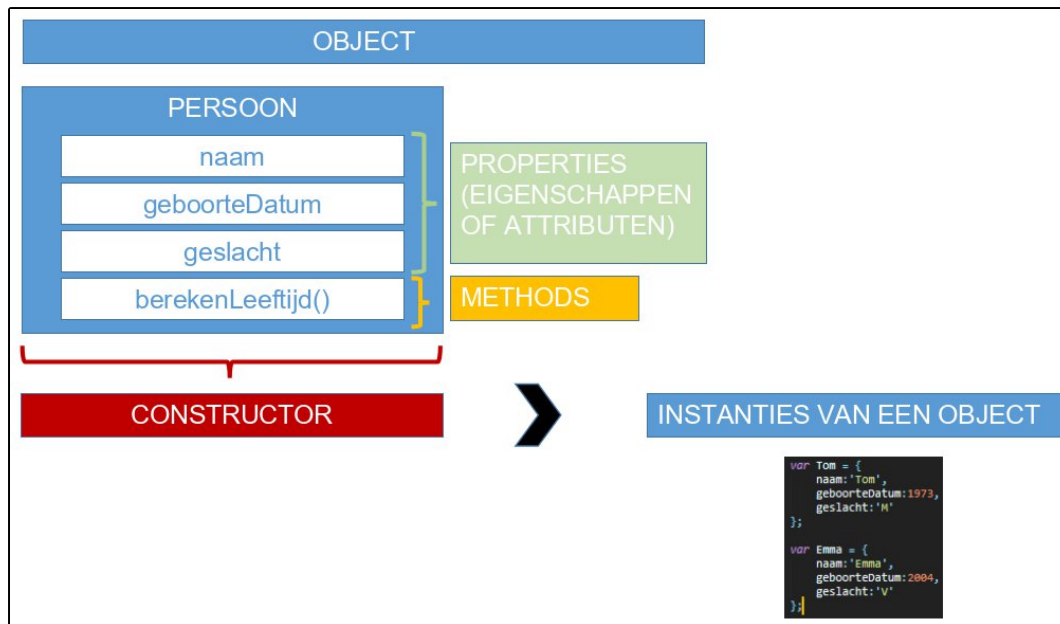
Het is niet de bedoeling dat we iedere instantie telkens opnieuw handmatig moeten opbouwen met telkens terug dezelfde eigenschappen en methodes. Wanneer er plots een nieuwe eigenschap toegevoegd zou worden, dan zouden we dit voor alle instanties moeten doen. Bij een database van 1000 personen zouden we dus 1000 aanpassingen moeten uitvoeren. Om dit op te lossen bestaat de constructor.

```
var Persoon = function(naam, geboortedatum, geslacht){
    this.naam = naam;
    this.geboortedatum = geboortedatum;
    this.geslacht = geslacht;
}
Persoon.prototype.berekenLeeftijd = function(){
    console.log(2020-this.geboortedatum);
}
var persoonA = new Persoon("Tom", 1973, "m");
var persoonB = new Persoon("Tim", 1980, "m");

console.log(persoonA.berekenLeeftijd());
```

In het object voegen we een functie toe die je geboortjaar aftrekt van het huidige Jaartal tijdens schrijven van deze cursus. Hier maken we ook voor het eerst gebruik van het **this** keyword. Wanneer het keyword this wordt gebruikt dan slaat dit steeds standaard op het window.object van de browser. Maar in dit geval gaat this over het item in code waar hij in staat, dus m.a.w. het object persoon .





## 7.5. OBJECT CLASS EN INHERITANCE

Als laatste stap dienen objecten als class te worden beschreven. Op die manier kunnen we grotere class libraries maken met herbruikbare objecten voor toekomstige projecten.

## 7.6. INHERITANCE

Eén van de krachtigste tools van object georiënteerd programmeren is het overerven van properties en methodes van een ander object. Dit noemen we ook het overerven van parent naar children.

Voorbeeld: Wij hebben eigenschappen overgeërfd van onze ouders.  
Bijvoorbeeld kleur ogen, vorm neus, haarkleur, lengte, ...

Het zou logisch zijn om deze eigenschappen te hergebruiken. Dus reeds bestaande code gebruiken van de ouders en niet te herschrijven.

In onderstaand voorbeeld tonen we het object persoon met daarin de verschillende properties die in de constructor worden meegegeven. De methode present die we schrijven zorgt voor de weergave van alle properties van deze class.

Werknemer is ook een class die wordt uitgebreid met alle eigenschappen en methodes van zijn parent, nl. Persoon dmv de extends functionaliteit. Met de super methode herhalen we eerst de overerving van de eerste class. Daarna voegen we de bijkomende eigenschappen toe voor werknemer. Dit is hier de soort.  
De show methode die we hier schrijven zal er dan in de child class voor zorgen dat de present methode van de parent en daarnaast de bijkomende soort wordt afgedrukt.

```

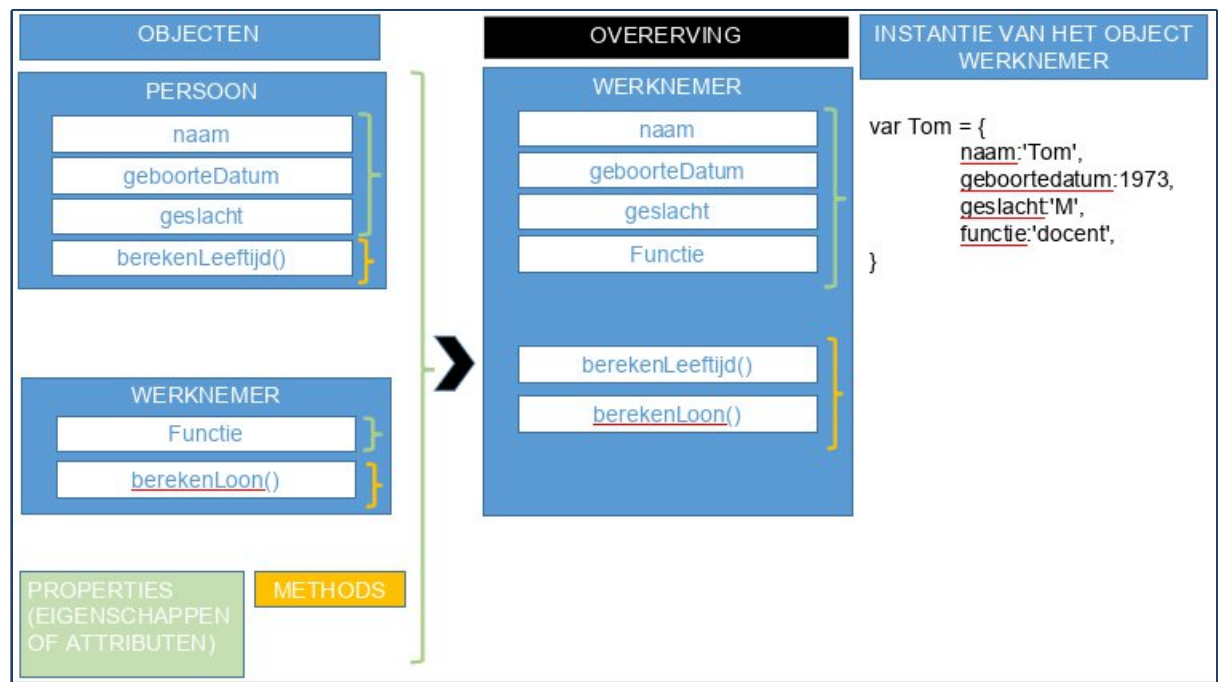
class Persoon {
    constructor(naam, geboortedatum, geslacht) {
        this.naam = naam;
        this.geboortedatum = geboortedatum;
        this.geslacht = geslacht;
    }
    berekenLeeftijd() {
        var huidigJaar = new Date();
        return huidigJaar.getFullYear() - this.geboortedatum;
    }
    present() {
        return 'Ik ben een (geslacht):' + this.geslacht + ' en
mijn naam is ' + this.naam;
    }
}

class Werknemer extends Persoon {
    constructor(naam, geboortedatum, geslacht, soort) {
        super(naam, geboortedatum, geslacht); //overerving
        this.soort = soort;
    }
    show() {
        return this.present() + ' en ik ben een ' +
this.soort;
    }
}

var Tom= new Werknemer("Tom",1973,"m","bediende");
document.getElementById("resultaat").innerHTML = Tom.show() +
" en is " + Tom.berekenLeeftijd() + " oud!";

```

Probeer onderstaand voorbeeld eens uit te werken:



## 8. BOM

BOM staat voor Browser Object model. Naast DOM die handelt over het document (document.), hebben we bovenliggend nog window liggen die eigenlijk tot de BOM behoort. Dit zijn eigenlijk objecten gerelateerd aan elke individuele browser. (google, firefox,...)

### 8.1. SCREEN

De screen property van het window object heeft de mogelijkheid om na te gaan in welk venster de pagina wordt geladen. Wat met andere woorden de grootte, breedte, ... van het venster is waar hij wordt in weergegeven.

html:

```
<p id="demo"></p>
```

js:

```
document.getElementById("demo").innerHTML =  
"Screen width is " + screen.width + "<br>" +  
"Screen height is " + screen.height + "<br>" +  
"Screen width is " + screen.width + "<br>" +  
"Screen availWidth is " + screen.availWidth + "<br>" +  
"Screen availHeight is " + screen.availHeight + "<br>" +  
"Screen colorDepth is " + screen.colorDepth + "<br>" +  
"Screen pixelDepth is " + screen.pixelDepth + "<br>"
```

resultaat:

Afhankelijk van het beeldscherm en de resolutie waarop je werkt kan onderstaand resultaat dus verschillend zijn.

```
Screen width is 1920  
Screen height is 1080  
Screen width is 1920  
Screen availWidth is 1920  
Screen availHeight is 1040  
Screen colorDepth is 24  
Screen pixelDepth is 24
```

## 8.2. COOKIES

Cookies zijn stukjes data die bewaard worden in kleine tekstbestanden op de pc van de client die naar uw webpagina surft.

Cookies worden gebruikt om informatie van een gebruiker te onthouden, wanneer hij/zij de volgende keer terug naar diezelfde pagina surft.

Bijvoorbeeld: wanneer een gebruiker meerdere keren naar éénzelfde product kijkt binnen de 2 dagen, dan zouden we daar een cookie voor kunnen schrijven om dit bij te houden. Dit zou dus een “lead” kunnen zijn in marketing termen die tot een verkoop kan leiden.

Javascript kan cookies lezen, maken en deleten met de **document.cookie** property.

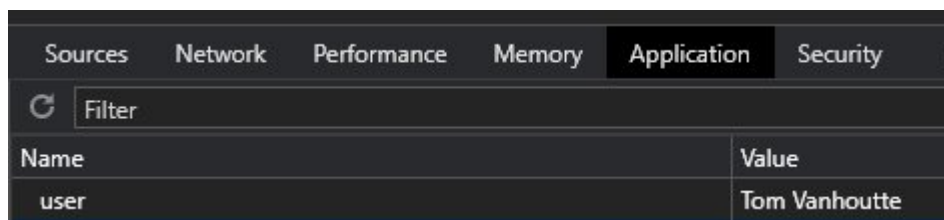
## 8.3. COOKIES MAKEN EN LEZEN

Hier voegen we de cookie user toe en geven we een value mee, nl. Tom Vanhoutte. De tweede parameter is de houdbaarheidsdatum van de cookie in de browser. Als derde parameter zal je meegeven op welk path van de website, dus m.a.w. welke pagina van de website de cookie betrekking heeft. In dit geval is dit de root van de website.

js:

```
/**cookie toevoegen**/  
document.cookie = "user=Tom Vanhoutte; expires=Thu, 27 Dec  
2020 12:00:00 UTC; path="/;
```

Wanneer je de cookie hebt toegevoegd door gewoon naar de pagina te surfen, dan kan je deze in het inspect element zien staan. Hier staan trouwens alle cookies die per pagina worden geladen.



The screenshot shows the Chrome DevTools Application tab. The 'Cookies' section is active, displaying a table of cookies. The table has two columns: 'Name' and 'Value'. One cookie is listed with the name 'user' and the value 'Tom Vanhoutte'.

Name	Value
user	Tom Vanhoutte

```
/**cookie lezen**/  
var decodedCookie = document.cookie;  
console.log(decodedCookie);
```

## 9. AJAX

### 9.1. WAT IS AJAX?

Asynchronous Javascript XML

Met het XMLHttpRequest OBJECT zijn we in staat om xml, json of text files te versturen en te ontvangen via het web.

Ajax is een webdev techniek.

#### 9.1.1. XMLHttpRequest Methodes

Een request is een vraag die gesteld wordt aan een webserver. Dit object bevat enkele methodes

Method	Description
<code>new XMLHttpRequest()</code>	Een nieuw XMLHttpRequest() object maken
<code>abort()</code>	Onderbreekt de huidige Request
<code>getAllResponseHeaders()</code>	Retourneert de header informatie
<code>getResponseHeader()</code>	Retourneert specifieke header informatie.
<code>open(method, url, async, user, psw)</code>	De detail van de request  <i>method</i> : request type GET of POST <i>url</i> : path van het xml bestand <i>async</i> : true (asynchronous) or false (synchronous) <i>user</i> : optioneel user name <i>psw</i> : optioneel password
<code>send()</code>	stuurt de vraag naar de webserver Wordt gebruikt voor de GET methode.
<code>send(string)</code>	stuurt de vraag naar de webserver Wordt gebruikt voor de POST methode.

### 9.1.2. XMLHttpRequest Properties

Een request is een vraag die gesteld wordt aan een webserver. Dit object bevat enkele properties/eigenschappen.

Property	Description
onreadystatechange	Wanneer de readystate van een request wijzigt, wordt een functie opgeroepen via onreadystatechange.
readyState	Status van het XMLHttpRequest. 0: request niet bekend 1: server connectie ok 2: request ontvangen 3: request wordt uitgevoerd 4: request in binnen en staat klaar.
responseText	Retourneert het antwoord van de request als een string
responseXML	Retourneert het antwoord van de request als XML data
status	Retourneert het statusnummer van de request 200: "OK" 403: "Verboden/Forbidden " 404: "Not Found/niet gevonden" 500.. webserver errors.
statusText	Retourneert de status-text "OK" of "Not Found")

### 9.1.3. TXT BESTAND VAN EEN WEBSERVER LADEN.

Voorbeeld:

Start je wamp server op. Onder de map wamp64/www maak je een nieuw folder aan jsvoorbeeld.

Maak vervolgens een nieuwe pagina aan: index.html en kopieer onderstaande code in de body tag.

index.html:

```
<p id="demo">AJAX zal hier de tekst wijzigen.</p>

<button type="button" onclick="loadDoc()">Wijzig
Content</button>

<script>
    function loadDoc() {
        var xhttp = new XMLHttpRequest();
        xhttp.open("GET", "test.txt", false);
        xhttp.send();
        document.getElementById("demo").innerHTML =
xhttp.responseText;
    }
</script>
```

test.txt:

```
<h3>Hoplahoi tekst ingelezen </h3>
<p>nie geweune!</p>
```

Probeer maar uit!



#### 9.1.4. XML BESTAND VAN EEN WEBSERVER LADEN.

Een xml bestand bestaand uit nodes/tags die gestructureerder zijn dan gewone txt-bestanden. Maak een nieuwe map aan op de wamp server onder de www folder met de naam xml.

Maak vervolgens 2 bestanden aan in de root van xml, nl. index.html en books.xml

books.xml:

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
      with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate
zombies,
an evil sorceress, and her own childhood to become
queen
of the world.</description>
  </book>
  <book id="bk103">
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
    <description>After the collapse of a nanotechnology
society in England, the young survivors lay the
foundation for a new society.</description>
  </book>
  <book id="bk104">
    <author>Corets, Eva</author>
    <title>Oberon's Legacy</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-03-10</publish_date>
    <description>In post-apocalypse England, the
mysterious
```

```

life          agent known only as Oberon helps to create a new
              for the inhabitants of London. Sequel to Maeve
              Ascendant.</description>
</book>
<book id="bk105">
  <author>Corets, Eva</author>
  <title>The Sundered Grail</title>
  <genre>Fantasy</genre>
  <price>5.95</price>
  <publish_date>2001-09-10</publish_date>
  <description>The two daughters of Maeve, half-sisters,
to            battle one another for control of England. Sequel
              Oberon's Legacy.</description>
</book>
<book id="bk106">
  <author>Randall, Cynthia</author>
  <title>Lover Birds</title>
  <genre>Romance</genre>
  <price>4.95</price>
  <publish_date>2000-09-02</publish_date>
  <description>When Carla meets Paul at an ornithology
ruffled.</description>
  </book>
  <book id="bk107">
    <author>Thurman, Paula</author>
    <title>Splish Splash</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>A deep sea diver finds true love twenty
    thousand leagues beneath the sea.</description>
  </book>
  <book id="bk108">
    <author>Knorr, Stefan</author>
    <title>Creepy Crawlies</title>
    <genre>Horror</genre>
    <price>4.95</price>
    <publish_date>2000-12-06</publish_date>
    <description>An anthology of horror stories about
roaches,
              centipedes, scorpions  and other
insects.</description>
  </book>
  <book id="bk109">
    <author>Kress, Peter</author>

```

```

    <title>Paradox Lost</title>
    <genre>Science Fiction</genre>
    <price>6.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>After an inadvertant trip through a
Heisenberg
        Uncertainty Device, James Salway discovers the
problems
        of being quantum.</description>
</book>
<book id="bk110">
    <author>O'Brien, Tim</author>
    <title>Microsoft .NET: The Programming Bible</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-09</publish_date>
    <description>Microsoft's .NET initiative is explored
in
        detail in this deep programmer's
reference.</description>
</book>
<book id="bk111">
    <author>O'Brien, Tim</author>
    <title>MSXML3: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-01</publish_date>
    <description>The Microsoft MSXML3 parser is covered in
processing,
        detail, with attention to XML DOM interfaces, XSLT
SAX and more.</description>
</book>
<book id="bk112">
    <author>Galos, Mike</author>
    <title>Visual Studio 7: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>49.95</price>
    <publish_date>2001-04-16</publish_date>
    <description>Microsoft Visual Studio 7 is explored in
depth,
        looking at how Visual Basic, Visual C++, C#, and
ASP+ are
        integrated into a comprehensive development
environment.</description>
</book>
</catalog>

```

Voorbeeld:  
index.html:

```
<h2>XMLHttpRequest Object</h2>

<p id="demo">AJAX zal hier de tekst wijzigen.</p>

<button type="button" onclick="loadDoc()">Change
Content</button>

<script>
    function loadDoc() {
        var xhttp, xmlDoc, txt, x, i;
        xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                xmlDoc = this.responseXML;
                txt = "";
                x = xmlDoc.getElementsByTagName("title");
                for (i = 0; i < x.length; i++) {
                    txt = txt + x[i].childNodes[0].nodeValue +
"
"
                    }
                document.getElementById("demo").innerHTML =
txt;
            }
        };
        xhttp.open("GET", "books.xml", true);
        xhttp.send();
    }
</script>
```

### 9.1.5. XML - METHODE GETALLRESPONSEHEADERS.

Naast de gewone tags krijg je heel wat informatie terug van je request naar de webserver. Eigenlijk is het data die ook een header heeft met informatie over de server die heeft geantwoord.

index.html:

```
<h2>XMLHttpRequest Object</h2>

<p id="demo">AJAX zal hier de tekst wijzigen.</p>

<button type="button" onclick="loadDoc()">Change
Content</button>

<script>
    function loadDoc() {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("demo").innerHTML =
                    this.getAllResponseHeaders();
            }
        };
        xhttp.open("GET", "books.xml", true);
        xhttp.send();
    }
</script>
```

resultaat (is afhankelijk van de server):

accept-ranges: bytes content-length: 4824 content-type: application/xml date: Tue, 27 Oct 2020 12:38:35 GMT etag: "12d8-5b2a625630658" last-modified: Tue, 27 Oct 2020 12:25:45 GMT server: Apache/2.4.46 (Win64) PHP/7.3.21

### 9.1.6. JSON BESTAND VAN EEN WEBSERVER LADEN.

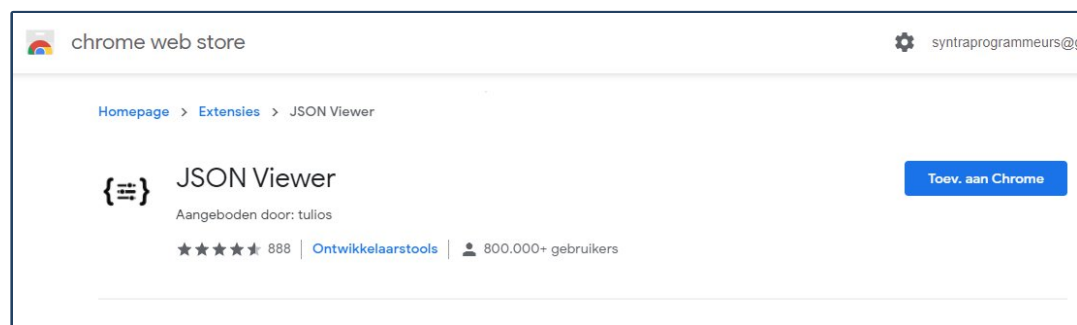
Een json bestand is een array bestaande uit nodes/tags.

Een voorbeeld kan je online vinden op: <http://dnd5eapi.co/api/monsters>

Resultaat:

```
{
  "count": 322,
  "results": [
    {
      "index": "aboeth",
      "name": "Aboeth",
      "url": "/api/monsters/aboeth"
    },
    {
      "index": "adult-blue-dragon",
      "name": "Adult Blue Dragon",
      "url": "/api/monsters/adult-blue-dragon"
    },
    {
      "index": "adult-bronze-dragon",
      "name": "Adult Bronze Dragon",
      "url": "/api/monsters/adult-bronze-dragon"
    },
    {
      "index": "adult-copper-dragon",
      "name": "Adult Copper Dragon",
      "url": "/api/monsters/adult-copper-dragon"
    },
    {
      "index": "adult-green-dragon",
      "name": "Adult Green Dragon",
      "url": "/api/monsters/adult-green-dragon"
    },
    {
      "index": "adult-silver-dragon",
      "name": "Adult Silver Dragon",
      "url": "/api/monsters/adult-silver-dragon"
    },
    {
      "index": "adult-white-dragon",
      "name": "Adult White Dragon",
      "url": "/api/monsters/adult-white-dragon"
    },
    {
      "index": "ancient-black-dragon",
      "name": "Ancient Black Dragon",
      "url": "/api/monsters/ancient-black-dragon"
    },
    {
      "index": "ancient-bronze-dragon",
      "name": "Ancient Bronze Dragon",
      "url": "/api/monsters/ancient-bronze-dragon"
    },
    {
      "index": "ancient-copper-dragon",
      "name": "Ancient Copper Dragon",
      "url": "/api/monsters/ancient-copper-dragon"
    },
    {
      "index": "ancient-green-dragon",
      "name": "Ancient Green Dragon",
      "url": "/api/monsters/ancient-green-dragon"
    },
    {
      "index": "ancient-silver-dragon",
      "name": "Ancient Silver Dragon",
      "url": "/api/monsters/ancient-silver-dragon"
    },
    {
      "index": "ancient-white-dragon",
      "name": "Ancient White Dragon",
      "url": "/api/monsters/ancient-white-dragon"
    },
    {
      "index": "black-dragon",
      "name": "Black Dragon",
      "url": "/api/monsters/black-dragon"
    },
    {
      "index": "blue-dragon",
      "name": "Blue Dragon",
      "url": "/api/monsters/blue-dragon"
    },
    {
      "index": "bronze-dragon",
      "name": "Bronze Dragon",
      "url": "/api/monsters/bronze-dragon"
    },
    {
      "index": "copper-dragon",
      "name": "Copper Dragon",
      "url": "/api/monsters/copper-dragon"
    },
    {
      "index": "green-dragon",
      "name": "Green Dragon",
      "url": "/api/monsters/green-dragon"
    },
    {
      "index": "silver-dragon",
      "name": "Silver Dragon",
      "url": "/api/monsters/silver-dragon"
    },
    {
      "index": "white-dragon",
      "name": "White Dragon",
      "url": "/api/monsters/white-dragon"
    }
  ]
}
```

Dit is echter niet leesbaar in je browser. We kunnen hiervoor een browser extensie installeren van de google webstore.



Nadat je deze chrome extensie geïnstalleerd hebt, surf terug naar:

<http://dnd5eapi.co/api/monsters>

Resultaat: onderstaande maakt de json array al veel leesbaarder.

```
// https://www.dnd5eapi.co/api/monsters
```

We kunnen dit json bestand nu inlezen.  
index.html:

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

<script>
  function jsonLink() {
    const xhr = new XMLHttpRequest(); // een constante van het
    type object variabele XMLHttpRequest();
    xhr.onload = onLoad;
    //dit object heeft een methode onload die uiteindelijk een
    extern bestand moet kunnen laden.
    xhr.open("GET", "http://dnd5eapi.co/api/monsters",
    true); //dit is de link die verwijst naar het json bestand
    xhr.send(null); //null wordt verstuurd bij de request
    (vraag), omdat de status van de server en het json bestand
    //nog niet bekend is op dit ogenblik.
  }

  function onLoad() {
    const overzicht = document.getElementById("overzicht"); //
    we trekken de div overzicht in de const overzicht
    let ul = document.createElement("ul"); // ul variabele die
    een ul tag dient te bevatten.
    let naam = JSON.parse(this.responseText);
    //doordat de functie onLoad verbonden is met het object
    xhr, hebben wij onmiddellijk toegang tot de data.
    //het keyword this staat voor de function onLoad(). parse
    zorgt ervoor dat de data als een array object wordt
    // binnengehaald.

    for (i = 0 ; i < naam["count"]; i ++){ //naam bevat alle
    data. naam["count"] = veldnaam count uit het json object
      let li = document.createElement("li");
      if (naam["results"][i]["name"] !== ""){
        li.innerHTML = naam["results"][i]["name"] + " <a
href='"+ naam["results"][i]["url"]+ "'> link </a>" ;
      }
      else{
        li.innerHTML= naam["results"][i]["name"];
      }
      ul.appendChild(li);
    }
    overzicht.appendChild(ul);
    let test = document.getElementById("counter");
    test.innerHTML = naam["count"];
  }
}
```

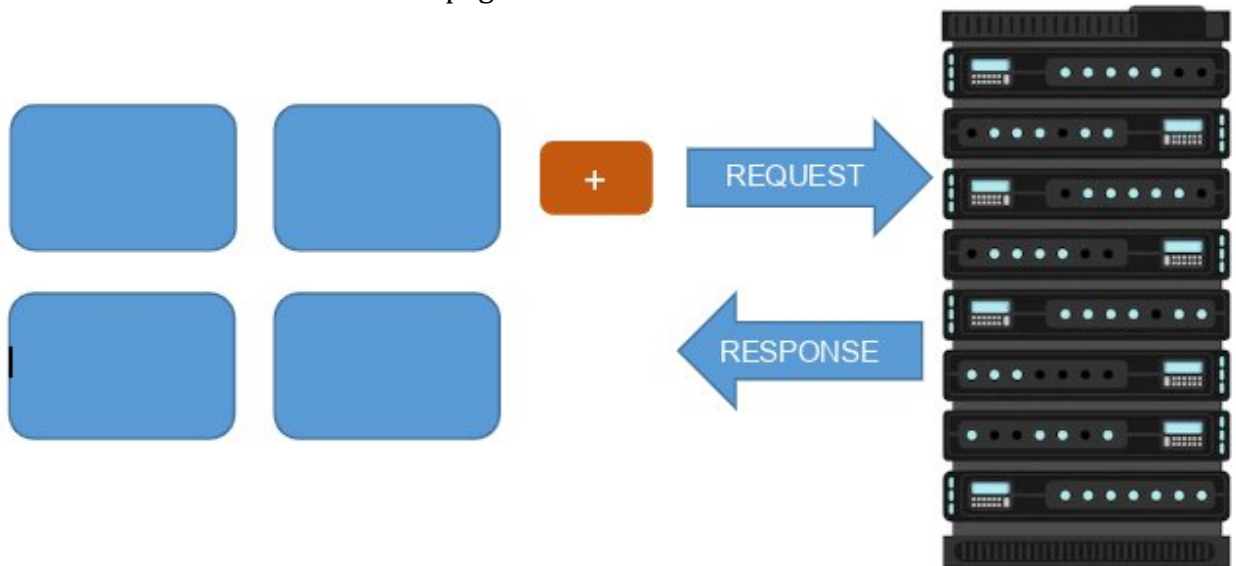
```

window.addEventListener("load", jsonLink);

</script>

```

Asynchronous = een webpagina is in staat data op te halen en te versturen zonder het refreshen van de webpagina zelf.



Voorbeeld code:

```

<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jI
W3" crossorigin="anonymous">
  <title>Ajax</title>
</head>
<body>
  <button onclick="toonLanden()">Landen</button>
  <div id="countryFeed"></div>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/boot
strap.bundle.min.js" integrity="sha384-

```



```

ka7Sk0Gln4gmtz2MlQnikT1wXgYs0g+OMhuP+I1RH9sENB00LRn5q+8nbTov4+
1p" crossorigin="anonymous"></script>
<script>
    function toonLanden(){
        let xhr = new XMLHttpRequest()
        /*REQUEST*/
        xhr.open('GET',
'https://restcountries.com/v2/all', true)
        xhr.send()
        /* RESPONSE*/
        /*callback function*/
        xhr.onload = function(){
            if(xhr.status == 200){
                let countries = JSON.parse(this.response)
                countries.forEach(country => {
                    const countryCard =
document.createElement('div')
                    countryCard.classList.add('card');
                    countryCard.style.cssText = 'width:
18rem;

                    const countryCardImage =
document.createElement('img')
                    countryCard.innerHTML = country.name
                    countryCardImage.src = country.flag

countryCard.appendChild(countryCardImage)

document.getElementById('countryFeed').appendChild(countryCard
)
                })
            }
        }
    }
</script>
</body>
</html>

```

UITLEG CODE:

```

<button onclick="toonLanden()">Landen</button>
<div id="countryFeed"></div>

```

We maken een button met daarin een onclick event die de functie toonLanden() zal uitvoeren.  
Het resultaat van deze functie zullen we uiteindelijk in de div van countryfeed injecteren.

```

let xhr = new XMLHttpRequest()

```

```

/*REQUEST*/
xhr.open('GET', 'https://restcountries.com/v2/all', true)
xhr.send()

```

In de functie van `toonLanden()` gebruiken we het object `XMLHttpRequest` om een extern bestand in te laden op onze pagina. Vooreerst dienen we de `REQUEST` uit te voeren.

We maken hiervoor een object variabele met de naam `xhr` aan. `Xhr` bevat binnen de `REQUEST` 2 methodes die we zullen gebruiken, nl. `open` en `send`.

De `open` methode telt 3 parameters:

- `GET` = de link/endpoint wordt opgehaald
- <https://restcountries.com/v2/all> = de link/endpoint
- `True` = we vragen javascript `ASYNCHRONOUS` data op te halen

```

/* RESPONSE*/
/*callback function*/
xhr.onload = function(){
  if(xhr.status == 200){
    let countries = JSON.parse(this.response)
    countries.forEach(country => {
      const countryCard = document.createElement('div')
      countryCard.classList.add('card');
      countryCard.style.cssText = 'width: 18rem;';
      const countryCardImage = document.createElement('img')
      countryCard.innerHTML = country.name
      countryCardImage.src = country.flag
      countryCard.appendChild(countryCardImage)

document.getElementById('countryFeed').appendChild(countryCard)
    })
  }
}

```

In het response gedeelte, nl. het antwoord die we terugkrijgen van de server, zien we een `onload` methode die we ook binnen de object variabele `xhr` kunnen aanspreken. Daar verbinden we een `callback function` aan die het endpoint zal doorlezen. We controleren vooreerst dat we een correcte response hebben teruggekregen zonder fouten. Dit controleren we met de methode `status` die gelijk dient te zijn aan `200`. `200` staat voor een correcte response voor de pagina.

Vervolgens zullen we de reponse die puur text is bij ontvangst parsen naar een `JSON` formaat. Het resultaat stoppen we in de variabele `countries`.

We hebben nu een array in `JSON` formaat die we kunnen doorlezen met een `foreach`.

Eénmaal in de `foreach` bouwen we onze layout op die we dan vullen met onze data. We beginnen met `countrycard` die een `div` is en waaraan we respectievelijk een class met de naam `card` aan koppelen en daarnaast ook een style attribuut met `css text`. Deze 2 items zijn een class en style van het card component van bootstrap die we injecteren binnen de `div`.

Daarna kunnen we de data binnen deze `div` van `countryCard` vullen. We dienen een `img` tag te voorzien en halen vervolgens de image en de name uit het `JSON` bestand op.

We voegen vervolgens alles toe aan countryCard die op zijn beurt alles zal toevoegen aan de div countryFeed.

## 9.2.ASYNCHRONOUS VS SYNCHRONOUS?

### 9.2.1. SYNCHRONOUS

Synchronous javascript is het sequentieel opvolgen van commando's via code. Javascript is wat men noemt "SINGLE THREADED". Iedere operatie kan pas uitgevoerd worden als de voorgaande werd uitgevoerd.

Voorbeeld 1:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Javascript synchronous</title>
</head>
<body>
<script>
  function partOne(){
    console.log('part one')
  }
  function partTwo(){
    console.log('part two')
  }
  function partThree(){
    console.log('part three')
  }
  function partFour(){
    console.log('part four')
  }
  partOne()
  partTwo()
  partThree()
  partFour()
</script>
</body>
</html>
```

Wanneer je de console.log bekijkt worden bovenstaande functies in VOLGORDE uitgevoerd.

Voorbeeld 2:

Hier laten we een for loop lange tijd lussen, pas daarna zal de h1 tag geïnjecteerd worden. Dus ook hier wordt de code sequentieel uitgevoerd.

```
<!doctype html>
```

```

<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
<button>click</button>
<script>
  const button = document.querySelector('button')
  button.addEventListener('click', ()=>{
    let count = 0
    for (let i = 0; i < 5000000000; i++){
      count++
    }
    console.log(count)

    const text = document.createElement('h1')
    text.innerHTML = 'Sequentieel laden'
    document.body.appendChild(text)
  })
</script>
</body>
</html>

```

### 9.2.2. ASYNCHRONOUS

Wanneer je single threaded werkt zoals in javascript dan kan je soms fouten krijgen.

Stel je de volgende oefening voor:

De request is het ophalen van één of meerdere images van de server. Er zijn 2 regels code, nl:

- 1) Request en response van de image (deze werken asynchroon, zie eerste oefening)
- 2) Tonen van de image

Wanneer het ophalen van de request en response TE LANG zou duren, dan zal javascript de image toch proberen te tonen met ERRORS tot gevolg.

### 9.3. CALLBACK OF PROMISE

OPLOSSING: CALLBACK OF PROMISE

#### 9.3.1. CALLBACK

Een callback is het terug oproepen van een bestaande function als parameters in een function in de achtergrond van je applicatie. Pas wanneer deze in de achtergrond is uitgevoerd gaat de eigenlijk functie verder. Dit is een voorbeeld van een asynchronous function via callbacks.

```
function myFunction(callback){  
    //code  
    callback()  
}
```

Voorbeeld:

Het click event die je hier ziet is zo een voorbeeld. Het click event zal hier een anonymous function aanroepen. Dit is de callback. De callback bevat als uitvoering de console.log(test).

M.a.w. de callback function wordt enkel aangeroepen wanneer die nodig is, nl. bij het click event.

```
function callbackFunction(){  
    console.log(test)  
}  
button.addEventListener('click', callbackFunction)
```

Voorbeeld:

In onderstaand voorbeeld zie je dat de functie ShowImage eerst url, type en CALLBACK zal uitvoeren VOOR hij de rest binnen de functie zal uitvoeren. Op die manier zal je geen ERRORS genereren omdat de response te lang op zich laat wachten.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
<script>
  function ShowImage(url,type,callback){
    const xhr= new XMLHttpRequest()
    xhr.open('GET', url)
    xhr.responseType = type

    xhr.onload=function(){
      callback(xhr.response)
    }
    xhr.send()
  }
  function createImage(blob){
    const objectUrl = URL.createObjectURL(blob)
    const imageElement = document.createElement('img')
    imageElement.src = objectUrl
    document.body.appendChild(imageElement)
  }
  ShowImage('myImage.jpg', 'blob', createImage)
</script>
</body>
</html>
```

### 9.3.2. PROMISE

Promise zijn speciaal gemaakt voor ASYNC gebeurtenissen.

Promise functions gebruikt chaining methods. Callbacks gebruiken functions in functions als parameters. Dat is het verschil!

Met promise kunnen we de `fetch()` api gebruiken i.p.v. `XMLHttpRequest`

Voorbeeld promise:

```
getPerson()
  .then(getUser)
  .then(getName)
  .then(() => console.log('ok'))
```

Voorbeeld:

```
let response = fetch('https://restcountries.com/v2/all')
console.log(response)
```

De response geeft een promise terug.

De fetch functie hieronder is een promise/belofte dat al wat we meegeven aan deze functie zal uitgevoerd worden zonder dat de rest van de code geblokkeerd wordt. Fetch is dus een promise en wordt dusdanig asynchroon uitgevoerd.

Iedere "then" is op zich ook weer een promise.

Voorbeeld:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>

<script>
  const flag = document.createElement('img')
  document.body.appendChild(flag)

  fetch('https://restcountries.com/v2/all')
    .then(response => {
      return response.json()
    }).then(json => {
      flag.src = json[0].flag
    }).catch(err => {
      console.log('errors:' + err.message)
    })
</script>
```



```
</body>
</html>
```

### 9.3.2.1. ASYNC AWAIT

#### 9.3.2.1.1. ASYNC

Een async fuction retourneert automatisch een promise.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
        content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
<script>
  async function go(){
    return 'go'
  }
  go().then(response=> console.log(response))
</script>
</body>
</html>
```

#### AWAIT

Await in een promise zorgt ervoor dat er we asynchrone functies synchroon uit kunnen voeren.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
        content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>

<script>
```

```
  async function getData(){
```

```
        const response = await
fetch('https://restcountries.com/v2/all')
    const data = response.json()
    return data
    }
    getData().then(data => console.log(data))
    .catch(err => console.log('errors:' + err.message))
</script>
</body>
</html>
```

## 9.4.AXIOS?

Voorgaande voorbeelden werken perfect, maar de meest gebruikte manier om voor front-end programma's met servers te communiceren is axios. Net zoals bij fetch is axios gebaseerd op promises. Het biedt echter een krachtiger en flexibelere functieset.

Voordelen van axios t.o.v. fetch zijn:

- Bescherming tegen XSRF
- Reactie time-out
- Automatische json data transformatie
- Ondersteuning voor ouder browsers
- Gestroomlijnde foutafhandeling

Axios installer kunnen we doen we via npm, natuurlijk dienen we eerst een package.json te initialiseren.

Git init

npm axios install

De tweede manier is een cdn link:

```
< script src = "https://unpkg.com/axios/dist/axios.min.js" >
</ script >
```

Axios heeft de volgende mogelijke HTTP verzoeken in zijn bereik. Je kan er mee opvragen, bewaren, deleten en data wijzigen. Dit noemen we CRUD = create read update en delete.

Hier zitten we echter in het luik van backend omdat het bewaren, wijzigen en deleten zich op dit niveau zal bevinden. In het frontend gedeelte houden we ons bezig met het opvragen of lezen van data.

De HTTP-requests die we met axios kunnen uitvoeren zijn:

```
axios.request(config)
axios.get(url[, config])
axios.delete(url[, config])
axios.head(url[, config])
axios.options(url[, config])
axios.post(url[, data[, config]])
axios.put(url[, data[, config]])
axios.patch(url[, data[, config]])
```

Wanneer we een vraag of een verzoek voor data aan een webserver stellen dan gaan we iets vragen. We maken dus gebruik van axios.get

AXIOS.GET

Wanneer een HTTP GET werd uitgevoerd, zal axios een promise retourneren die fulfilled of rejected is net zoals bij fetch api. Hier gebruiken we ook method chaining.

Voorbeeld:

```
<script
src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
```

```

    axios.get('https://restcountries.com/v2/all')
      .then((response) => {
        console.log(response.data);
        console.log(response.status);
        console.log(response.statusText);
        console.log(response.headers);
        console.log(response.config);
      });
  </script>

```

In ons voorbeeld wordt alles door de response.data onmiddellijk in json formaat omgezet. Daarnaast kent response naast data nog enkele andere methodes die handig kunnen zijn zoals: status van de server, statutext, headers en config.

Een voorbeeld van simultane requests:

```

<script>
  axios.all([
    axios.get('https://restcountries.com/v2/all'),
    axios.get('https://restcountries.com/v3/all')
  ])
    .then(responseArr => {
      //onderstaande wordt uitgevoerd NADAT de requests
      (GET) volledig werd uitgevoerd
      console.log('Data: ', responseArr[0].data);
      console.log('Data: ', responseArr[1].data);
    });
</script>

```

Errors kan je als volgt opvangen

```
<script>
  axios.all([
    axios.get('https://restcountries.com/v2/all'),
    axios.get('https://restcountries.com/v3/all')
  ])
    .then(responseArr => {
      //onderstaande wordt uitgevoerd NADAT de requests
      (GET) volledig werd uitgevoerd
      console.log('Data: ', responseArr[0].data)
      console.log('Data: ', responseArr[1].data)
    })
    .catch(throw => {
      if (axios.isCancel(throw)) {
        console.log("Request cancelled",
throw.message);
      }
      else {
        //handle the error
      }
    })
</script>
```

## 10. GEBRUIK VAN API'S

API = Application programming interface.

Veel sites stellen hun API ten diensten van iedereen. Eigenlijk is dit gebruik maken van hun broncode binnen onze sites.

<https://developer.mozilla.org/en-US/docs/Web/API>