Informatietechnologie 2

JavaScript: examen bootcamp

De dertien belangrijkste technieken

Kristof Michiels

De dertien belangrijkste technieken (1/2)

- Werken met variabelen
- Beslissingsstructuren
- Loops
- Arrays
- Functies
- DOM-element(en) vastnemen
- DOM-elementen aanmaken en plaatsen

Belangrijk: de informatie in deze presentatie is GEEN vervanging van de te kennen leerstof. In principe is alle leerstof die op de slides staat te kennen. Het is ENKEL een hulpmiddel met prioriteiten.

De dertien belangrijkste technieken (2/2)

- De attributen van DOM-elementen
- De styling van DOM-elementen
- DOM-elementen en hun class(es)
- Events
- Intervallen
- Willekeurig getallen

Belangrijk: de essentie om het goed te doen bij oefeningen en examen is de vaardigheid om al deze technieken snel met elkaar te kunnen combineren.

1. Werken met variabelen

Variabelen

- Variabelen zijn containers die ons helpen informatie voor later te op te slaan en te onthouden
- We maken hen aan met het woord *let* gevolgd door een naam, "identifier" genoemd
- We geven hen een waarde met de toekenningsoperator =
- We kunnen de waarde tijdens de looptijd van ons programma veranderen
- We kunnen ook meerdere variabelen aanmaken binnen hetzelfde statement

```
let naam = "Ben";
let leeftijd = 23;
```

```
let naam = "Ben", leeftijd = 23;
```

Variabelen

- Je mag zoveel variabelen gebruiken als je er nodig meent te hebben
- Vaak zal je zelf bij het maken van een oefening moeten beseffen: hier moet ik een extra variabele aanmaken

```
let somVanAlleGetallenDeelbaarDoorDrie = 0;
let i = 1;
while (i < 101) {
    if (i % 3 === 0) {
        somVanAlleGetallenDeelbaarDoorDrie += i;
    }
}
console.log(somVanAlleGetallenDeelbaarDoorDrie);
```

Variabelen

- Variabelen kunnen van een verschillend type zijn: string, numbers (integer of float), boolean, array, function,...
- JavaScript regelt zelf welk datatype het ziet in een variabele. Je moet dat dus niet gaan declareren
- Je mag let maar eenmaal gebruiken: bij het declareren van de variabele

```
let i = 0;
i = 2;
i = i + 1;
let moedertaal = "Nederlands";
let spreektNederlands = true;
```

Variabelen: wiskundige bewerkingen

- We kunnen rekenen met de getalwaarden in onze variabelen
- We kunnen +, -, * en / gebruiken
- We noemen dit wiskundige operatoren
- We gebruiken ze om wiskundige operaties of bewerkingen uit te voeren
- % modulus geeft je de rest bij deling door (4%3 zal 1 zijn)

```
let r = 4 + 2;
let pi = 3.141592653589793;
console.log(2 + r * p);
```

2. Beslissingsstructuren

Beslissingsstructuren

- Deze moet je met je ogen dicht kunnen schrijven
- Waarom? ze zitten vaak binnen andere constructies: while-loop, functie,... Twijfelen niet toegelaten.
- Je hebt ze maar in enkele gedaantes...

```
if (voorwaarde) {
    // code die wordt uitgevoerd indien de voorwaarde waar is
}
```

Beslissingsstructuren

```
if (voorwaarde) {
// code die wordt uitgevoerd indien de voorwaarde waar is
} else {
// code die wordt uitgevoerd indien de voorwaarde onwaar is
}
```

Beslissingsstructuren

```
if (voorwaarde1) {
// code die wordt uitgevoerd indien de voorwaarde1 waar is
} else if (voorwaarde2) {
// code die wordt uitgevoerd indien de voorwaarde2 waar is
} else if (voorwaarde3) {
// code die wordt uitgevoerd indien de voorwaarde3 waar is.
// Je mag zoveel else if-en toevoegen als je nodig hebt
} else {
// code die wordt uitgevoerd indien de alle andere voorwaarden onwaar zijn
}
```

Beslissingsstructuren

- Schrijf ze op zoals hier te zien en werk daarop naar binnen verder uit...

```
if () {}
if () {} else {}
if () {} else if () {} else if () {} else {}
```

De logische en (&&)

- Als we willen dat meerdere voorwaarden waar zijn vooraleer we een waar kunnen teruggeven dan gebruiken we de logische en operator
- De logische en gebruikt een dubbele ampersand als symbool
- Elke voorwaarde moet waar zijn om als geheel in waar te resulteren

```
let waarde = 5 < 10 && 10 < 15;
console.log(waarde); // true
waarde = 5 < 10 && 10 < 15 && 10 < 8;
console.log(waarde); // false
```

De logische of (||)

- Als het enkel noodzakelijk is dat één van twee vergelijkingen waar is, dan kunnen we de logische of operator gebruiken
- Van zodra één van beiden waar is geeft de operator waar terug
- Pas indien er geen enkele waar is krijgen we false
- Je kan zoveel vergelijkingen als je wil in de logische en/of operator gebruiken

```
let waarde = 5 > 10 || 10 > 15;
console.log(waarde); // false
waarde = 5 < 10 || 10 > 15;
console.log(waarde); // true
console.log(true || false || false) // true
```

3. Loops

Loops

- Kies een goede loopstructuur, beheers ze voor 100% en hou je eraan. Hier heb ik gekozen voor de while-loop
- Je hebt hiervoor een variabele nodig (hier gebruik ik teller) die je declareert en een startwaarde geeft
- Vervolgens gebruik je deze variabele in de voorwaarde, én je zorgt ervoor dat de teller verandert binnen in de loop uitgevoerde statements

```
let teller = 0;
while (teller < 10) {
    // hier komen statements die een aantal keer moeten uitgevoerd worden (hier 10x)
    // je kan gerust teller gebruiken in deze statements.
    teller = teller+1;
}
```

Loops

- Loops heb je ook nodig om door een array te gaan, of door een HTMLCollection

```
let mijnArray = ["aap","noot","mies"];
let teller = 0;
while (teller < mijnArray.length) {
    console.log(mijnArray[teller]);
    teller = teller+1;
}
```

```
let mijnElementen = document.querySelectorAll("p");
let teller = 0;
while (teller < mijnElementen.length) {
    mijnElementen[teller].style.color = "red";
    teller = teller+1;
}
```

4. Arrays

Arrays

Een array is een bijzonder type variabele, die verschillende waarden van een type of zelfs verschillende types kan bevatten. Arrays bieden een geweldige manier om gerelateerde waarden op te slaan.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
console.log(dingen); // ["raindrops","roses","whiskers","kittens"]
```

Index posities

Hoe kunnen we die waarden opvragen? Arrays hebben posities of index posities. Dus, om een bepaalde waarde op te vragen gaan we het overeenkomstige indexgetal tussen vierkante haakjes meegeven. Deze index posities starten bij 0.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
console.log(dingen[3]); // "kittens"
```

length

Zoals bij strings is er een eigenschap (of property) die ons vertelt hoeveel elementen een array telt.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
console.log(dingen.length); // 4
```

Arrays: push()

We kunnen de push() method gebruiken om een nieuwe waarde aan het einde van de array toe te voegen. Heeft hetzelfde resultaat als een waarde toekennen aan de index van dingen.length

```
let dingen = ["raindrops","roses","whiskers","kittens"];
dingen.push("kettles");
console.log(dingen); // ["kettles","roses","whiskers","kittens","kettles"]
```

Arrays: pop()

Waarden verwijderen uit arrays? De pop() method verwijdert de laatste waarde uit de array. Als we deze toekennen aan een variabele, dan ontvangt deze variabele de verwijderde waarde.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
let ding = dingen.pop();
console.log(dingen); // ["raindrops","roses","whiskers"];
console.log(ding); // kittens
```

Arrays: shift()

Wat als we de eerste waarde willen verwijderen? Dat doen we met de shift() method. Shift() verwijdert het eerste element uit de array en verhuist alle andere waarden naar een lagere index.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
dingen.shift();
console.log(dingen[0]); // roses
```

Arrays: unshift()

De unshift() method is het tegenovergestelde van shift(). Het voegt een waarde toe aan het begin van de array. Alle andere waarden krijgen een hoger indexgetal. We kunnen unshift() ook gebruiken met meerdere waardes tegelijk.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
dingen.unshift("guitars");
console.log(dingen); // ["guitars","raindrops","roses","whiskers","kittens"];
```

Arrays: splice()

De splice() method kan waarden toevoegen, vervangen en verwijderen. Splice() neemt twee getallen en daarna eventueel een variabel aantal waarden.

```
let fruit = ["banaan", "sinaasappel", "appel", "mango", "kiwi"];
fruit.splice(2, 2); // Op positie 2, verwijder 2 items
console.log(fruit); // ["banaan", "sinaasappel", "kiwi"]
fruit.splice(2, 1, "citroen", "kiwi"); // 2 items bij en 1 verwijderen op index 2
console.log(fruit); // ["banaan", "sinaasappel", "citroen", "kiwi"]
fruit.splice(2, 0, "peer", "pruim"); // 2 items bij op index 2, geen verwijderen
console.log(fruit); // ["banaan", "sinaasappel", "peer", "pruim", "citroen", "kiwi"]
```

5. Functies

Wat zijn functies?

Kort gezegd zijn functies blokken met code statements die een specifieke taak vervullen. Maar waarom zijn ze nuttig? Klinkt het logisch dat we telkens we de oppervlakte van een cirkel willen berekenen we deze statements moeten ingeven? Neen, daarvoor hebben we functies.

```
let straalCirkel = 4;
let oppervlakteCirkel = straalCirkel * straalCirkel * Math.PI;
console.log(oppervlakteCirkel); // 50.26548245743669
```

Het function keyword

Om functies te maken gebruiken we het function keyword, een unieke naam en een paar haakjes/accolades. Eens we de functie hebben gecreëerd, kunnen we ze hergebruiken telkens we ze nodig hebben. Alle code die we binnen de accolades plaatsen wordt een onderdeel van de functie. Om een functie uit te voeren moeten we ze aanroepen. We doen dit door de functienaam gevolgd door haakjes in te geven. Je mag dit doen op verschillende plaatsen in je code, en zo vaak je wenst.

```
function zegDag() {
    console.log("Hallo wereld!");
}
zegDag(); // Hallo wereld!
...
zegDag(); // Hallo wereld!
```

Functie-argumenten

Net zoals bij bestaande methods als console.log() kan je waarden meegeven binnen die haakjes. Die waarden noemen we parameters (of argumenten). Parameters zijn tijdelijke variabelen die we definiëren binnen de haakjes na de functienaam en gebruiken om toegang te krijgen tot waarden die met de functie-aanroep zijn meegekomen.

```
function berekenOppervlakteCirkel(straalCirkel) {
    let oppervlakteCirkel = straalCirkel * straalCirkel * Math.PI;
    console.log(oppervlakteCirkel);
}
berekenOppervlakteCirkel(4); // 50.26548245743669
```

Functies: meerdere argumenten

We kunnen ook functies met meerdere argumenten maken. In de functie-definitie scheiden we ze met een komma. We moeten de argumenten bij het aanroepen dan steeds in de juiste volgorde meegeven.

```
function voegToeAanArray(array, waarde, indexGetal) {
    array.splice(indexGetal, 0, waarde)
}
let mijnArray = ["peer", "appel", "kiwi"];
voegToeAanArray(mijnArray,"banaan",2);
console.log(mijnArray); // ["peer","appel","banaan","kiwi"]
voegToeAanArray(mijnArray,"kers",0);
console.log(mijnArray); // ["kers", "peer", "appel", "banaan", "kiwi"]
```

Functies: waarden teruggeven

Net zoals sommige bestaande methods waarden teruggeven, kunnen we dat ook laten gebeuren met onze eigengemaakte functies. We doen dit door gebruik te maken van het return keyword, gevolgd door de waarde die we wensen terug te geven.

```
function berekenOppervlakteCirkel(straalCirkel) {
    let oppervlakteCirkel = straalCirkel * straalCirkel * Math.PI;
    return oppervlakteCirkel;
}
let oppervlakte = berekenOppervlakteCirkel(4);
console.log(oppervlakte); // 50.26548245743669
```

Functie-expressies

We kunnen ook (anonieme) functies definiëren en opslaan in variabelen. Nadat we de functie aan de variabele hebben toegekend, kunnen we de variabele net als een functie gebruiken. Dit soort statement staat ook gekend als functie-expressies.

```
let optellen = function(getal1,getal2) {
    return getal1 + getal2
};
let som = optellen(4,6);
console.log(som);
```

6. DOM-element(en) vastnemen

Bestaande DOM-elementen opvragen

- Elementen opvragen: je kan er meerdere opvragen, of één enkele
- De vraag die je je moet stellen: is het element uniek?
- Uniek = werken met ID, dus altijd eentje
- Met klassenamen of elementnamen = er kunnen er meerdere zijn van dezelfde klasse, dus altijd meerdere!

```
<h1>Hoofding 1</h1>
<h2 id="uniekehoofding">Hoofding 2</h2>
<h3 class="tussentitel">Hoofding 3</h3>
<h3 class="tussentitel">Andere hoofding 3</h3>
<h2 id="andereuniekehoofding">Hoofding 2</h2>
<h3 class="tussentitel">Hoofding 3</h3>
<h3 class="tussentitel">Andere hoofding 3</h3>
```

document.querySelector()

Deze document method is een modern alternatief voor .getElementById(). Ze geeft het eerste (!) element terug dat overeenkomt met een als argument meegegeven CSS selector. Je kunt hiervoor alle selectors gebruiken zoals je ze ook binnen een stylesheet gebruikt. Je schrijft ze op dezelfde manier: .mijnlink, #mijnnav, h1, ...

```
<h1>Dit is een voorbeeld</h1>
<nav id="mijnnav">
    <a href="#" class="mijnlink">Home</a>
</nav>
```

```
document.querySelector("h1").innerHTML = "Nieuwe tekst voor titel";
document.querySelector(".mijnlink").innerHTML = "Nieuwe tekst voor link";
```

document.querySelectorAll()

Deze document method is een modern alternatief zowel voor .getElementsByTagName() als .getElementsByClassName(). Ze geeft ALLE elementen terug die overeenkomen met een als argument meegegeven CSS selector. Ook hier weer kan je alle selectors gebruiken zoals je ze binnen een stylesheet gebruikt.

```
<p class="voorbeeld">Dit is een paragraaf</p>
<p class="voorbeeld">Dit is een andere paragraaf</p>
```

```
let mijnElementen = document.querySelectorAll(".voorbeeld");
let i = 0;
while (i < mijnElementen.length) {
    mijnElementen[i].innerHTML= "We veranderen de tekst binnen de paragraaf";
    i++;
}
```

Tekst van bestaande DOM-elementen opvragen

Dit doen we met de innerHTML-eigenschap die beschikbaar is voor elk element

```
<h1 id="hoofdtitel">Je titel staat hier</h1>
```

```
var hoofdTitel = document.querySelector("#hoofdtitel");
console.log(hoofdTitel.innerHTML); // "Je hoofdtitel staat hier"
```

Toegang tot formuliervelden

De inhoud van een tekstveld kunnen we opvragen met de value eigenschap. Geldt ook voor checkboxes en radiobuttons.

```
<input type="text" id="mijnveld" naam="mijnveld">
```

```
let mijnVeld = document.querySelector("input");
let mijnTekst = mijnVeld.value;
```

7. DOM-elementen aanmaken en plaatsen

Nieuwe DOM-elementen aanmaken

- Doen we met de createElement() en appendChild() methods
- Deze techniek laat toe om vanuit de code je pagina's aan te passen terwijl de pagina reeds ingeladen is
- Hoe? Eerst creëren we het nieuw DOM element met createElement(), daarna voegen we het toe aan het document met appendChild()

```
let mijnNieuwElement = document.createElement("li"); // nieuw element
let mijnNietgeordendeLijst = document.querySelector("#mijnlijstje"); // de moeder
mijnNietgeordendeLijst.appendChild(mijnNieuwElement); // vastmaken in de DOM
```

Nieuwe DOM-elementen voorzien van informatie

We kunnen dit nieuwe element van tekst voorzien via de innerHTML method (zie code-regel 2)

```
<div id="trivia">...</div>
```

```
let nieuweHoofding = document.createElement("h1"); // nieuw element
let nieuweParagraaf = document.createElement("p"); // nog een nieuw element
nieuweHoofding.innerHTML = "Wist je?"; // tekst toevoegen
nieuweParagraaf.innerHTML = "Weetje komt hier..."; // tekst toevoegen
let moederElement = document.querySelector("#trivia"); // moederelement vastnemen
moederElement.appendChild(nieuweHoofding); // kind toevoegen aan moederelement
moederElement.appendChild(nieuweParagraaf); // kind toevoegen aan moederelement
```

8. De attributen van DOM-elementen

.getAttribute()

Met de .getAttribute() method vraag je de inhoud op van een bestaand attribuut. Deze method neemt één argument, namelijk de naam van het attribuut, tussen aanhalingstekens.

```
<nav>
    <a href="https://google.com" class="extern" style="color: red;">Google</a>
</nav>
```

```
let mijnElement = document.querySelector("a");
console.log(mijnElement.getAttribute("href")); // https://google.com
console.log(mijnElement.getAttribute("class")); // extern
console.log(mijnElement.getAttribute("style")); // color: red;
```

.setAttribute()

Met de .setAttribute() method kan je de inhoud van een bestaand attribuut wijzigen of een nieuw attribuut aanmaken. Je geeft aan deze method 2 argumenten mee: de eerste verwijst naar de naam van het attribuut, de tweede naar de waarde ervan. Beiden horen tussen aanhalingstekens te staan. Geef je een attribuutnaam mee die nog niet aan het element is gekoppeld, dan wordt dit attribuut aangemaakt.

```
<a href="https://google.com" class="extern" style="color: red;">Google</a>
```

```
let mijnElement = document.querySelector("a");
mijnElement.setAttribute("href","https://docs.google.com");
mijnElement.setAttribute("style","color: blue;");
mijnElement.setAttribute("id","mijnlink");
```

HTML Data-* attributen

HTML biedt ons een speciaal soort attributen aan die we kunnen gebruiken om data op te slaan binnen een element. Je kan die data dan via JavaScript gaan gebruiken in je code. Elk element kan er zoveel bevatten als nodig. Data-* attributen bestaan uit 2 delen:

- De attribuut-naam: gebruik enkel kleine letters, en minstens 1 karakter lang na de prefix "data-"
- De attribuut-waarde: kan elke string bevatten (steeds string, ook bij getal!).

```
<ul>
     <li data-dier-type="vogel" data-aanwezig-in-zoo="true">Uil</li>
     <li data-dier-type="vis" data-aanwezig-in-zoo="false">Zalm</li>
     <li data-dier-type="spin" data-aanwezig-in-zoo="true">Tarantula</li>
</ul>
```

HTML Data-* attributen

Deze attributen zijn niet zichtbaar in je browser. Deze negeert deze data elementen volledig bij het weergeven van de pagina. Ze dienen enkel om data op te slaan die jij vanuit Javascript wenst te gebruiken. Je kan ze in HTML handmatig aan je elementen toevoegen of je kan ze meegeven vanuit JavaScript. Dat doe je met .setAttribute(). Het opvragen van de informatie in een data-* attribuut doe je met .getAttribute().

```
<div id="reisaanbieding" data-bestemming="Parijs"></div>
```

```
let mijnDiv = document.querySelector("div");
let bestemming = mijnDiv.getAttribute("data-bestemming");
console.log(bestemming); //Parijs
mijnDiv.setAttribute("data-periode","3 dagen");
mijnDiv.setAttribute("data-prijs","250 eur");
mijnDiv.setAttribute("data-vantot","3 april tot 7 april 2019");
```

9. De styling van DOM-elementen

Elementen van styling voorzien

HTML elementen van de nodige styling voorzien: het is iets dat we vaak gaan doen met behulp van JavaScript. We kunnen dit doen op 4 verschillende manieren:

- met het style attribuut
- met de style eigenschap
- met de className-eigenschap
- met de classList-eigenschap

Stylen met de style eigenschap

De style eigenschap biedt een andere mogelijkheid om een vastgenomen element te stylen. Let op: niet te verwarren met het style attribuut. Je doet het zoals te zien in onderstaand voorbeeld. Maar let op: de JavaScript syntax is lichtelijk anders dan bij CSS voor de style properties. Een voorbeeld: background-color wordt backgroundColor.

```
<h1>Dit is een hoofding</h1>
```

```
let mijnHoofding = document.querySelector("h1");
mijnHoofding.style.color = "red";
mijnHoofding.style.backgroundColor = "yellow";
mijnHoofding.style.borderBottom = "0.1rem solid black";
mijnHoofding.style.textAlign = "center";
console.log(mijnHoofding.style.backgroundColor);
```

10. DOM-elementen en hun classes

De classList eigenschap

De classList eigenschap van een element is een interessant alternatief voor className. classList geeft alle klasses terug voor een bepaald element. classList.length geeft het aantal klasses die aan het element zijn gekoppeld. Bovendien beschikt classList over enkele bijzonder krachtige methods die je in de voorbeelden aan het werk kan zien:

- classList.add() : een klasse toevoegen
- classList.remove() : een klasse verwijderen
- classList.contains() : checken of een klasse aanwezig is
- classList.toggle() : een klasse toevoegen indien afwezig, verwijderen indien aanwezig

De classList eigenschap

```
<p class="klasse1 klasse2">Ik ben een paragraaf</p>
```

```
let elementKlasses = document.querySelector("p").classList;
console.log(elementKlasses.value); // klasse1 klasse2
console.log(elementKlasses[0]); // klasse1
console.log(elementKlasses.length); // 2
elementKlasses.add("klasse3"); // "klasse3" toegevoegd
elementKlasses.add("klasse4","klasse5"); // "klasse4" en "klasse5" toegevoegd
if (elementKlasses.contains("klasse3")) {
elementKlasses.remove("klasse3");
}
```

11. Events

Wat zijn events?

Op een webpagina vinden constant gebeurtenissen plaats. Een gebruiker beweegt met de muispijl over een bepaald gedeelte van een site, een pagina laadt in, een gebruiker klikt op een knop of op een link ... De lijst van events is lang. Kijk hier maar eens: https://www.w3schools.com/jsref/dom_obj_event.asp.

Sommige zijn algemeen (op pagina-niveau bvb), anderen kunnen op een specifiek element plaatsgrijpen. Voorbeelden: click, mouseover, mouseout, focus, blur, keyup, keydown, keypress...

Wat we met JavaScript gaan doen is met code vertellen dat we willen "luisteren" wanneer bepaalde van die events gebeuren. We kunnen er dan een functie aan koppelen die moet uitgevoerd worden telkens het event gebeurt.

Event listeners

We gaan bij voorkeur gebruik maken van event listeners.

```
<div id="mijndiv">Klik op mij als je kan :-)</div>
```

```
mijnDiv = document.querySelector("#mijndiv");
function veranderVanKleur() {
  if (mijnDiv.style.backgroundColor === "yellow") {
        mijnDiv.style.backgroundColor = "orange";
    } else {
        mijnDiv.style.backgroundColor = "yellow";
    }
}
mijnDiv.addEventListener("click", veranderVanKleur);
```

Event listeners

Events kunnen aan meerdere elementen tegelijk worden gekoppeld. Stel je een pagina voor met verschillende divs waaraan je een mouseover event en een mouseout event wil koppelen:

```
let mijnDivs = document.querySelectorAll("div");
let i = 0;
while (i < mijnDivs.length) {
    mijnElement[i].addEventListener("mouseover", function(){
        this.style.backgroundColor = "blue";
    });
    mijnElement[i].addEventListener("mouseout", function(){
        this.style.backgroundColor = "white";
    });
    i++;
}
```

Werken met "this"

Bij een functie die je aan een event koppelt verwijs je naar het element zelf met "this".

```
<button type="button">Klik op mij!</button>
```

```
let mijnKnop = document.querySelector("button");
function mijnFunctie() {
    this.classList.add("gekleurd");
}
mijnElement.addEventListener("click", mijnFunctie);
```

12. Intervallen

setInterval()

setInterval() voert een bepaalde functie constant uit, met een tussenpauze van een aangegeven interval in milliseconden.

```
function simpeleBoodschap() {
     console.log("Ping ping!");
}

setInterval(simpeleBoodschap,5000);
```

Timers neutraliseren

setInterval heeft een neutraliserende tegenhanger: clearInterval().

```
function doeElkeVijfSeconden() {
 console.log("Ping ping!");
}
let intervalBehandeling = setInterval(doeElkeVijfSeconden,5000);
let mijnKnop = document.querySelector("#mijnknop");
mijnknop.addEventListener("click", function(){ clearInterval(intervalBehandeling); });
```

13. Willekeurige getallen

Willekeurig geheel getal tussen 2 waarden (inclusief)

Volgende functie geeft een willekeurig getal terug tussen 2 meegegeven waarden. Deze zullen ze heel vaak in onze oefeningen nodig hebben.

```
function willekeurigGetal(min, max) {
  min = Math.ceil(min);
  max = Math.floor(max);
  return Math.floor(Math.random() * (max - min + 1)) + min;
}

console.log(willekeurigGetal(1,6)); // 5
console.log(willekeurigGetal(1,6)); // 6
console.log(willekeurigGetal(1,6)); // 1
console.log(willekeurigGetal(1,6)); // 1
console.log(willekeurigGetal(1,6)); // 2
```

IT2 - JavaScript: examen bootcamp

De dertien belangrijkste technieken

kristof.michiels01@ap.be