

Cursus Robbe Magerman Web Development II

Robbe Magerman

08/06/2024

Inhoudstafel

1	Hoofdstuk 1: JavaScript, de basis	4
1.1	Declareren van variabelen	4
1.2	Alert	4
1.3	Use Strict	4
1.4	parseInt	4
1.5	toString	4
1.6	subString	4
1.7	toFixed	4
1.8	Math	4
1.8.1	Math.random	4
1.8.2	Math.floor en Math.ceil	4
1.8.3	Math.min en Math.max	5
1.8.4	Math.pi	5
1.9	Controlestructuren	5
1.9.1	if - else - if else	5
1.9.2	switch	5
1.9.3	while - do while	5
1.9.4	conditionele operator	5
2	Hoofdstuk 2: Functies - Arrays - Modules	6
2.1	Functieparameters	6
2.2	Arrays	6
2.2.1	declaratie	6
2.2.2	Selecteren eerste en laatste element	6
2.2.3	Itereren over een array	6
2.2.4	Destrucuring met elementen	7
2.2.5	Elementen toevoegen en verwijderen	7
2.2.6	delete	7
2.2.7	Andere handige functies van een array	7
2.3	Modules	7
2.3.1	script src aanpassing	7
2.3.2	default export en import	8
2.4	Handige functies	8
2.4.1	Functieberekening van array binnenin een array	8
3	Hoofdstuk 3: Objecten en Functies	9
3.1	Objecten	9
3.1.1	Creeëren van een object	9
3.1.2	Overlopen van het object	9
3.1.3	Een element toevoegen	9
3.1.4	Verwijderen van een element	9
3.1.5	Aanpassen van een elemtn	9
3.1.6	Functies in Objecten	9
3.1.7	Object destructuring	10
3.2	Functies	10
3.2.1	Rest operator	10
3.2.2	Arrow function	10
3.3	Events	10
3.3.1	Een element in html ophalen	10
3.3.2	onclick	10
3.3.3	addEventListener	11
4	Hoofdstuk 4: Object Oriënted Programming	12
4.1	Standaardvorm van een Class	12
4.1.1	window.onload index.js	12

5	Hoofdstuk 5: Functioneel Programmeren	13
5.1	Arrays	13
5.1.1	ForEach	13
5.1.2	Filter	13
5.1.3	Map	13
5.1.4	Reduce	13
5.1.5	Find(FindIndex)	14
5.1.6	Sort	14
5.2	Maps en Sets	14
5.2.1	Maps	14
5.2.2	Sets	15
6	Hoofdstuk 6: Webopslag	17
6.1	Data opslaan in de browser	17
6.1.1	Theorie	17
6.1.2	Hulpmiddel eenvoudige versie Localstorage	17
6.1.3	set...InStorage() via een id	17
6.1.4	set...InStorage() via een id	17
6.1.5	get...InStorage() via een array	18
6.1.6	set...InStorage() via een array	18
6.1.7	Een leuke sidenote in verband met localStorage	18
6.1.8	toJSON methode	18
6.2	Handige methodes	19
6.2.1	Volledig wissen van een lijst	19
6.2.2	Wissen van slechts één element uit een lijst	19
6.2.3	Een lijst volledig opvullen via de localStorage	19
7	Hoofdstuk 7: DOM (Document Object Model)	20
7.1	HTML tags maken via JavaScript	20
7.1.1	Aan een bestaande tag extra htmlcode toevoegen	20
7.1.2	In een repository iets toevoegen	20
7.1.3	Inhoud van een bestaande volledig vervangen van een htmlElement	20
7.1.4	Een volledig nieuwe tag aanmaken - alles apart aanbrengen	20
7.1.5	Een volledig nieuwe tag aanmaken - insertAdjacentHTML	20
7.1.6	Css classe toevoegen	21
7.1.7	QuerySelector	21
7.2	Handige methodes	21
7.2.1	Maken van een filtersysteem	21
8	Hoofdstuk 8: Asynchroon programmeren	22
8.1	Synchroon vs Asynchroon	22
8.2	setTimeout(function() => {}, 3000)	22
8.3	AJAX (Asynchrhooon JavaScript And XML)	22
8.4	Data ophalen via een API	22
8.5	Promises - Single	22
8.6	Promises - Single Moeilijker	23
8.7	Promises - Promise.all	23
8.8	Async - Await	23
8.9	Voorbeeld van data ophalen en als een nieuw object omtoveren via Asyn - Await	24

1 Hoofdstuk 1: JavaScript, de basis

1.1 Declareren van variabelen

- Een variabele wordt gemaakt met **let** of **const**
 - **let**: de waarde van de variabele **kan** veranderen
 - **const**: de waarde van de variabele kan **niet** worden veranderd + moet worden geïnitieerd

1.2 Alert

- `alert()` opent een venster dat enkel op pc kan weergegeven worden

1.3 Use Strict

- Wanneer je niet gebruik maakt van OOP (Object Oriënted Programming) gebruik dan `"use strict"` als eerste lijn in je javascript bestand

1.4 parseInt

- Aangezien je in JavaScript geen datatypes geeft aan je variabelen kan het voorvallen dat je input vraagt aan de gebruiker (een getal), maar dat je er nie mee kan rekenen omdat het nog in String vorm staat.
- Dit kunnen we oplossen met:

```
let getal = alert(parseInt(prompt("Geef een getal op: ")));
```

1.5 toString

- Je kan ook het omgekeerde doen, van een getal overgaan naar een String

```
let getal = alert(parseInt(prompt("Geef een getal op: ")));
getal.toString();
```

1.6 subString

- Laat toe om een bepaald aantal karakters van een zin te verwijderen via een nummer dat jij opgeeft

1.7 toFixed

- Laat toe om een kommagetal tot een bepaald aantal cijfers na de komma te laten zien;

```
let getal = 5.9664;
getal.toFixed(2); // 5.97
```

1.8 Math

1.8.1 Math.random

- Genereert een getal tussen een bepaalde waarde

```
const randomGetal = Math.floor(Math.random() * 6 + 1); // getal tussen 1 en 6
const randomGetal2 = Math.floor(Math.random() * 50 + 26); // getal tussen 25 en
75, want 76 is exclusief
```

1.8.2 Math.floor en Math.ceil

- Afronden tot een geheel getallen naar beneden en afronden tot een geheel getal naar boven

1.8.3 Math.min en Math.max

- Geef de maximum waarde weer van bepaalde waardes

```
const hoogsteGetal = Math.max(1000, 200, 1001, 199);  
const kleinsteGetal = Math.min(1000, 200, 1001, 199);
```

1.8.4 Math.pi

- Geeft het volledige getal Pi weer

```
const pi = Math.pi();  
const r = 5;  
const diameter = r * r * pi;
```

1.9 Controlestructuren

1.9.1 if - else - if else

1.9.2 switch

1.9.3 while - do while

1.9.4 conditionele operator

- Dit werkt als een oneliner voor een if else functie

```
const getal = 100;  
const resultaat =  
  getal >= 100  
    ? "Het resultaat is hoger dan 100"  
    : getal == 100  
      ? "Het resultaat is 100"  
      : "Het getal is kleiner dan 100";  
console.log(`${resultaat}`); // of console.log(resultaat);
```

2 Hoofdstuk 2: Functies - Arrays - Modules

2.1 Functieparameters

- Wanneer een functie wordt aangeroepen kan het zijn dat je soms een parameter moet opgeven.
- Via de parameter wordt er dan met de opgegeven data gewerkt
- Indien de gebruiker niks meegaf, kan je een **default waarde** ingeven (**parameter = iets**)

```
function myName(naam = null) {
  return `Mijn naam is ${naam}`;
}

console.log(myName);
```

2.2 Arrays

2.2.1 declaratie

- Ofwel geef je arrays mee met één enkele waarde, of je geeft de objecten ook eigenschappen
- Arrays maken gebruik van zero-based indexing

```
const namen = ["Jeff", "Pieter", "Paterposter"];

//

const namenMetEigenschappen = [
  { naam: "Jeff", leeftijd: 18, sex: "male" },
  { naam: "Pieter", leeftijd: 19, sex: "male" },
];
```

2.2.2 Selecteren eerste en laatste element

```
// eerste element
console.log(namen[0]);

// laatste element
console.log(`${namen[namen.length - 1]});
```

2.2.3 Itereren over een array

- Itereren kan op 2 manieren:
 - Via een forlus met een index
 - Via een forlus met woorden
 - De forlus met woorden wordt in HF3 uitgebreid, hier is de **of** heel belangrijk. Bij objecten wordt dat later **in**

```
// manier 1
for (let i = 0; namen.length; i++) {
  console.log(namen[i]);
}

// manier 2
for (let naam of namen) {
  console.log(naam);
}
```

2.2.4 Destrucuring met elementen

- Destructuring betekent eigenlijk dat je elke index een naam een variabele toekent
- Met deze array wordt dat dan:

```
const fruiten = ["appelsienen", "appelen", "peren"];
const [fruitsapmiddel, appelmoesmiddel, gelei-ingredient];
console.log(`fruitsap maak je met ${fruitsapmiddel}`);
```

2.2.5 Elementen toevoegen en verwijderen

- push: voegt een element toe aan het **einde** van een array
- pop: verwijdert het **laatste** element van een array
- unshift: voegt een element toe aan het **begin** van de array
- shift: verwijdert het **eerste** element van een array

```
// push
namen.push("jouwNaam");

// pop
namen.pop();

// unshift
namen.unshift("jouwNaam2");

// shift
namen.shift();
```

2.2.6 delete

- Dit verwijdert een element op een plaats dat jij bepaalt.
- Deze verwijderde waarde wordt dan **undefined**

```
delete namen[1];
```

2.2.7 Andere handige functies van een array

- concat(): voegt 2 arrays samen
- reverse(): draait de array om
- slice(startIndex, eindIndex): splitst de array aan de hand van de 2 opgegeven indexen
- splice(startIndex, aantalTeDelete, ... teVervangenWaarden): verwijdert vanaf de opgegeven index een aantal waarden en jij kunt ze vervangen via waardex
- sort(): sorteert de array
 - sort((a,b) => a - b)
- indexOf(element): geeft de index weer van een bepaalt element
- lastindexOf(element): geeft de laatste index weer van een bepaald element indien het meerdere keren voorkwam
- join(): maakt van de array één grote String door de elementen te splitsen via het opgegeven teken in de join-functie

2.3 Modules

2.3.1 script src aanpassing

- Indien je js-bestanden uit meer dan één bestand bestaan gebruik je best modules

```
<script src="index.js" type="module"></script>
```

2.3.2 default export en import

- Default export gebruik je als je maar één ding wil exporteren
 - Als je meerdere dingen gaat exporteren mag je default export niet gebruiken, maar simpelweg meerdere exports
- Import gebruik je als je die export wil ophalen

```
// zegHallo.js
default export function zegHallo(naam){
  return `Hallo, ik ben ${naam}`;
}

// index.js
import zegHallo as halloFunctie from "./zegHallo.js"

// index.js met meerdere exports
import {zegHallo as halloFunctie, zegSallu as salluFunctie} from "./zegHallo.js"
```

2.4 Handige functies

2.4.1 Functieberekening van array binnenin een array

```
const shot1 = [
  ["single", 18],
  ["triple", 20],
  ["double", 20],
];

function giveTotal(shot) {
  return shot.reduce(
    (pv, [soortSchot, punten]) =>
      pv +
      (soortSchot === "triple"
        ? punten * 3
        : soortSchot === "double"
        ? punten * 2
        : punten),
    0
  );
}
```


3 Hoofdstuk 3: Objecten en Functies

3.1 Objecten

3.1.1 Creeëren van een object

```
const myBicycle = {
  speed: 30,
  gear: 1,
  frameMaterial: "carbon fiber",
};

// of

const speed = 30;
const gear = 1;
const frameMaterial = "carbon fiber";

const myBicycle = {
  speed: ${speed};
  frameMaterial: ${frameMaterial};
}
```

3.1.2 Overlopen van het object

```
for (let key in myBicycle) {
  alert(`${key}: ${myBicycle[key]}\n`);
}
```

3.1.3 Een element toevoegen

```
myBicycle.color = "red";
```

3.1.4 Verwijderen van een element

```
delete myBicycle.gear;
```

3.1.5 Aanpassen van een element

- Een element aanpassen doe je altijd met `myBicycle.speed += 6` nooit `myBicycle.speed = speed + 6`

3.1.6 Functies in Objecten

```
const myBicycle = {
  speed: 30,
  gear: 1,
  frameMaterial: "carbon fibre",
  accelerate: function (percentage) {
    this.speed += (this.speed * percentage) / 100;
  },
  snelheidBeoordelen: function () {
    let beoordeling = "";
    if (speed <= 10) {
      return (beoordeling = "Walking");
    } else if (speed <= 20) {
```

```
        return (beoordeling = "Stappen");
    }
},
};
```

3.1.7 Object destructuring

```
const {
  extras,
  gears: { make: gearMake },
} = myBicycle;

console.log(extras, gearMake);
```

3.2 Functies

3.2.1 Rest operator

```
function geefGetallen(...getallen) {
  for (let getal of getallen) {
    console.log(getal);
  }
}

console.log(geefGetallen(1, 2, 3, 4, 5, 6));
```

3.2.2 Arrow function

```
// vroegere declaratie
function functie(param) {}

// declaratie vanaf nu
const functie = (param) => {};
```

3.3 Events

3.3.1 Een element in html ophalen

```
<p id="pElement">Ik ben de inhoud van dit p-element</p>
```

```
const pElement = document.getElementById("pElement");
```

3.3.2 onclick

```
const knop = document.getElementById("buttonElement");
const uitslag = uitslag >= 10 ? "geslaagd" : "niet geslaagd";

knop.onclick = () => {
  document.getElementById("teVeranderenElement").innerText = `${uitslag}`;
};
```

3.3.3 addEventListener

- dankzij addEventListener kan je meerdere events instellen voor één object

```
const knop = document.getElementById("buttonElement");
const uitslag = uitslag >= 10 ? "geslaagd" : "niet geslaagd";

knop.addEventListener("click", () => {
  document.getElementById("teVeranderenElement").innerText = `${uitslag}`;
});
```

4 Hoofdstuk 4: Object Oriënted Programming

4.1 Standaardvorm van een Class

- Aanmaken van de klasse = `class Game`
- Elke klasse heeft properties die **private** zijn: `#titel | #rating`
- Een klasse heeft een **constructor** die zich bezig houdt met het oproepen van de properties en methodes instelt die automatisch moeten worden uitgevoerd bij oproep van de klasse
- Elke property heeft een getter om de waarde op te halen
- Elke property heeft een setter om de waarde aan te passen

```
class Game {
  #titel;
  #rating;

  constructor(titel, rating) {
    this.#titel = titel;
    this.#rating = rating;
    this.#initialiseerHTML();
  }

  #initialiseerHTML() {
    document.getElementById(
      "gameBeschrijving"
    ).innerText = `De naam van de game is ${this.#titel} met rating: ${
      this.#rating
    }`;
  }

  get titel() {
    return this.#titel;
  }
  set titel(titel) {
    this.#titel = titel;
  }

  get rating() {
    return this.#rating;
  }
  set rating(rating) {
    this.#rating = rating;
  }
}
```

4.1.1 window.onload | index.js

- Dit is meestal hoe het index.js-bestand er zal uitzien:

```
import Game from "../Game.js";

const init = () => {
  new Game();
};

window.onload = init;
```

5 Hoofdstuk 5: Functioneel Programmeren

5.1 Arrays

5.1.1 ForEach

- Retourneert een nieuwe array en verloopt alle elementen uit een array op een makkelijke manier
- Heeft 3 waarden: *element*, *index* en de *array* zelf

```
const animals = [
  { name: "cat", size: "small", weight: 5 },
  { name: "dog", size: "small", weight: 10 },
  { name: "lion", size: "medium", weight: 150 },
  { name: "elephant", size: "big", weight: 5000 },
];

// ===== //

animals.forEach((dier, i, array) => {
  console.log(
    `Van de array animals is dit dier ${i}/${array.length} met naam: ${dier}`
  );
});
```

5.1.2 Filter

- Retourneert een nieuwe array met een bepaalde voorwaarde
- Heeft 3 waarden: *element*, *index* en de *array* zelf

```
const underTonAnimals = animals.filter(
  (a) => a.weight >= 0 && a.weight <= 1000
);
// OF //
const underTonAnimalsBis = animals.filter(
  ({ weight }) => weight >= 10 && weight <= 1000
);
```

5.1.3 Map

- Retourneert een nieuwe array met specifieke elementen/data uit een array
- Heeft 3 waarden: *element*, *index* en de *array* zelf

```
// VB - maak een array met de namen van alle animals
const namesOfAnimals = animals.map(({ name }) => name);
console.log(`Names of animals: ${namesOfAnimals}`);

// DIY - maak een array met 'naam: size' van alle animals
const namesOfAnimalsWithSize = animals.map(
  ({ name, size }) => `${name}: ${size}`
);
```

5.1.4 Reduce

- Retourneert een nieuwe array, wordt gebruikt voor een berekening
- Heeft 3 waarden: *element*, *index* en de *array* zelf

```
// VB - log het totale gewicht van alle animals naar de console
```

```

console.log(
  `Total weight of animals = ${animals.reduce(
    (pv, { weight }) => pv + weight,
    0
  )}`
);

// DIY - hoeveel animals hebben size die small is
const numberOfSmallAnimals = animals.reduce(
  (previousElement, element) => (element.size === "small" ? pv + 1 : pv),
  0
);

```

5.1.5 Find(FindIndex)

- Zoekt een specifiek iets in een array

```

const users = [
  {
    id: 1,
    firstname: "Jan",
    lastname: "Janssens",
  },
  {
    id: 2,
    firstname: "Eva",
    lastname: "De Smet",
  },
  {
    id: 3,
    firstname: "Pieter",
    lastname: "Martens",
  },
];

const user = users.find((item) => item.id === 1);
console.log(user);

const indexuser = users.findIndex((item) => item.id === 1);
console.log(indexuser); // 0

```

5.1.6 Sort

- Retourneert een nieuwe array en sorteert op alfabetische volgorde
- De manier hoe je sorteert kan je ook aanpassen

```

console.log(fruit.sort());
// OF //
fruit.sort((a, b) => a.length - b.length);
console.log(fruit);

```

5.2 Maps en Sets

5.2.1 Maps

- Een map houdt key-waarden bij (niet per se uniek)

```
// een map maken en waarden in stoppen
let mijnMap = new Map([
  ["a", 1],
  ["b", 2],
  ["c", 3],
]);

// waarden toevoegen aan een map
mijnMap.set("d", 4);

// Waarde ophalen uit een map op basis van de sleutel
console.log(mijnMap.get("a")); // geeft 1 terug

// Controleren of een sleutel in de map zit
console.log(mijnMap.has("b")); // geeft true terug

// Sleutel verwijderen uit een map
mijnMap.delete("c");

// De grootte van de map ophalen
console.log(mijnMap.size); // geeft het aantal sleutel-waardeparen in de map
                           terug

// Itereren over de sleutels van de map
for (let key of mijnMap.keys()) {
  console.log(key);
}

// Itereren over de waarden van de map
for (let value of mijnMap.values()) {
  console.log(value);
}

// Itereren over zowel de sleutels als de waarden van de map
for (let [key, value] of mijnMap.entries()) {
  console.log(key, value);
}
```

5.2.2 Sets

- Een set houdt unieke key-waarden bij

```
// Een lege set maken
let mijnSet = new Set();

// Een set maken met enkele waarden
let mijnAndereSet = new Set([1, 2, 3, 4, 5]);

// Elementen toevoegen aan een set
mijnSet.add(1);
mijnSet.add(2);
mijnSet.add(3);

// Elementen verwijderen uit een set
mijnSet.delete(2);

// Controleren of een element in de set zit
```

```
console.log(mijnSet.has(1));

// De grootte van de set ophalen
console.log(mijnSet.size);

// De set omzetten naar een array
return [...new Set(this.#producten.map((el) => el.categorie))];
```


6 Hoofdstuk 6: Webopslag

6.1 Data opslaan in de browser

6.1.1 Theorie

- Om data op te slaan in de browsers gebruiken we `localStorage`
- In een klasse slaan we dit op als een eigenschap:

```
#storage = localStorage;
```

6.1.2 Hulpmiddel eenvoudige versie Localstorage

- Lijken deze storagemiddelen je wat moeilijk om te leren?
- Dit is de vereenvoudigde versie met alles wat te maken heeft met storage

```
let getal = Number(localStorage.getItem("getal")) || 0;;
getal.onclick = () => {
  getal++;
  this.#setGetal();
}
// waarde zetten in de storage
#setGetal(){
  this.#storage.setItem("getal", getal);
}

// waarde uit de storage halen. Hier niet per se nodig aangezien let getal dat
// ook al doet
#getGetal(){
  this.#storage.getItem(getal);
}
```

6.1.3 set...InStorage() via een id

- We kijken of het item in de storage zit via een if
- We halen in dit geval het HTML-id op en stoppen de waarde van de storage er in
- Het woord “jaarGefietsteKilometer” heb je zelf bepaald in `setJaarFromStorage`

```
#getJaarFromStorage() {
  if (this.#storage.getItem("jaarGefietsteKilometer")) {
    document.getElementById("jaar").value = this.#storage.getItem(
      "jaarGefietsteKilometer"
    );
  }
}
```

- We kijken of het item in de storage zit via een if
- We halen in dit geval het HTML-id op en halen de waarde er uit van de storage
- Het stoppen van een waarde in de storage werkt zo:
 - `setItem("zelfTeKiezenStorageNaam", teStoppenVariabeleInStorage)`

6.1.4 set...InStorage() via een id

```
#setJaarInStorage() {
  this.#storage.setItem(
    "jaarGefietsteKilometer",
    document.getElementById("jaar").value
  );
}
```

```
);
}
```

6.1.5 get...InStorage() via een array

- Het verschil met een woord is dat je dit moet omvormen naar één grote String
- Om dit te realiseren gebruiken `JSON.parse()`

```
#getGelezenBoekenFromStorage() {
  if (this.#storage.getItem("bogelesenBoekeneken")) {
    this.#gelezenBoeken = JSON.parse(this.#storage.getItem("gelezenBoeken"));
  }
}
```

6.1.6 set...InStorage() via een array

- Het verschil met een woord is dat je dit moet omvormen naar één grote String
- Om dit te realiseren gebruiken `JSON.stringify()`

```
#setGelezenBoekenInStorage() {
  this.#storage.setItem(
    "gelezenBoeken",
    JSON.stringify([...this.#gelezenBoeken])
  );
}
```

6.1.7 Een leuke sidenote in verband met localStorage

- Je kan ook de bezoeker laten weten hoeveel keer hij de website heeft bezocht

```
#getAantalBezoekenFromStorage() {
  if (this.#storage.getItem("aantalBezoeken")) {
    this.#aantalBezoeken =
      parseInt(this.#storage.getItem("aantalBezoeken")) + 1;
  } else {
    this.#aantalBezoeken = 1;
  }
}

#setAantalBezoekenInStorage() {
  this.#storage.setItem("aantalbezoeken", this.#aantalBezoeken);
}
```

6.1.8 toJSON methode

```
toJSON() {
  return {
    name: this.name,
    date: this.date,
  };
}
```

6.2 Handige methodes

6.2.1 Volledig wissen van een lijst

```
#clearMilestones() {  
  this.#milestones = [];  
  this.#storage.removeItem("milestones");  
  this.#toHTML();  
}
```

6.2.2 Wissen van slechts één element uit een lijst

```
#deleteMilestone(ind) {  
  this.#milestones.splice(ind, 1);  
  this.#setMilestonesInStorage();  
  this.#toHTML();  
}
```

6.2.3 Een lijst volledig opvullen via de localStorage

- Getter

```
#getMilestonesFromStorage() {  
  this.#milestones = [];  
  const milestonesAsJson = this.#storage.getItem("milestones");  
  if (milestonesAsJson) {  
    this.#milestones = JSON.parse(milestonesAsJson).map(  
      (m) => new Milestone(m.name, m.date)  
    );  
    this.#milestones = this.#milestones.filter(  
      (m) => m.daysUntilDeadline >= 0  
    );  
  }  
}
```

- Setter

```
#setMilestonesInStorage() {  
  this.#milestones.sort((a, b) => a.date - b.date);  
  this.#storage.setItem("milestones", JSON.stringify(this.#milestones));  
}
```

7 Hoofdstuk 7: DOM (Document Object Model)

7.1 HTML tags maken via JavaScript

7.1.1 Aan een bestaande tag extra htmlcode toevoegen

```
#categorieenToHtml(categorieen) {
  categorieen.forEach((c) =>
    document.getElementById("categorie").insertAdjacentHTML("beforeend", `<option
      value=${c}>${c}</option>`));
}
```

7.1.2 In een repository iets toevoegen

```
voegVacatureToe(id, titel, functieomschrijving, profiel, bedrijf, plaats) {
  return this.#vacatures.push(
    new Vacature(id, titel, functieomschrijving, profiel, bedrijf, plaats)
  );
}
```

7.1.3 Inhoud van een bestaande volledig vervangen van een htmlElement

```
#productenToHtml(producten) {
  document.getElementById(
    "aantalProducten"
  ).innerHTML = `<h4>Aantal producten: ${producten.length}</h4>`;
}
```

7.1.4 Een volledig nieuwe tag aanmaken - alles apart aanbrengen

- Je mag kiezen op het examen wat je gebruikt

```
producten.forEach((p) => {
  const divEl = document.createElement("div");
  divEl.setAttribute("id", p.id);
  const imgEl = document.createElement("img");
  imgEl.src = `images/${p.id}/thumbs/thumb_${p.afbeeldingen[0]}.jpg`;
  imgEl.alt = p.titel;
  divEl.appendChild(imgEl);
  const pEl = document.createElement("p");
  pEl.innerText = p.titel;
  divEl.appendChild(pEl);
  divEl.onclick = () => {
    this.#productDetailsToHtml(p);
    divEl.classList.add("tekstVet");
  };
  document.getElementById("overzichtProducten").appendChild(divEl);
});
```

7.1.5 Een volledig nieuwe tag aanmaken - insertAdjacentHTML

- insertAdjacentHTML heeft 4 waarden waar deze HTML-waarden kunnen worden toegevoegd
 - afterbegin
 - afterend
 - beforebegin

○ beforeend

```
this.#uitgavenRepository.uitgaven.forEach((c) => {
  document.getElementById("data").insertAdjacentElement(
    "beforeend",
    `<div class="aankoop">
      
      <h4>${c.titel} - ${c.bedrag}</h4>
      <p>${c.datum.datumNotatie()}</p>
    </div>
  `
  );
});
```

7.1.6 Css classe toevoegen

```
document.getElementById("productDetails").classList.add("verbergen");
```

7.1.7 QuerySelector

7.2 Handige methodes

7.2.1 Maken van een filtersysteem

- Dit komt in de respotory

```
filterOpZoekTermen(zoektermen) {
  return this.#vacatures.filter((v) => v.bevatZoekterm(zoektermen));
}
```

- Dit komt in de klasse van het object

```
bevatZoekterm(zoektermen) {
  return zoektermen.some((z) =>
    this.#titel.toUpperCase().includes(z.toUpperCase)
  );
}
```

8 Hoofdstuk 8: Asynchroon programmeren

8.1 Synchroon vs Asynchroon

- Synchroon:
 - De éne stap gebeurt na de andere
- Asynchroon:
 - Er gebeuren meerdere stappen tegelijk (API-requesten kunnen soms meerdere seconden duren waardoor de website lang zal laden en de gebruiker kan dan niks doen)

8.2 setTimeout(function() => {}, 3000)

- Deze *callback*-functie wordt dan 3 seconden uitgevoerd
- Nog een voorbeeld van een callbackfunctie:

```
function applyCallbackToActors(callback, ...actors) {
  actors.forEach((actor) => callback(actor));
}
```

8.3 AJAX (Asynchrhoon JavaScript And XML)

- Maakt gebruikt van HTML en CSS voor de presentatie

8.4 Data ophalen via een API

```
function getData() {
  fetch("https://v2.jokeapi.dev/joke/Any?")
    .then((response) => {
      if (!response.ok) {
        throw new Error("fetching joke failed");
      }
      return response.json();
    })
    .then((json) => {
      toHtml(json);
    });
}
```

8.5 Promises - Single

- De eerste stap is om de data te fetchen
- Daarna kijken we of de response niet ok is
 - indien niet, throwen we een Error
 - indien wel, returnen we de response.json()
- Indien de response dus ok was kunnen we doen met de data wat we willen, hier willen we de naam en de leeftijd
- En als allerlaatste indien er nog iets fout gaat, vangen we het op in een catch en gooien we een alert daarvan

```
fetch("./data/data1.json")
  .then((response) => {
    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }
    return response.json();
  })
  .then((json) => {
    console.log("name:", json.name);
    console.log("age:", json.age);
  });
```

```

})
.catch((error) => alert(error));

```

8.6 Promises - Single Moeilijker

- In deze API-fetch maken we van elk apart API-onderdeel een nieuw Bericht in dit geval

```

#getData() {
  fetch(
    "https://data.stad.gent//api/explore/v2.1/catalog/datasets/recente-
      nieuwsberichten-van-stadgent/records?limit=5"
  ).then((response) => {
    if (!response.ok){
      throw new Error = "API is gefaald";
    }
    return response.json;
  }).then((json) => {
    const berichten = json.results.map(bericht => new Bericht(bericht.
      publicatiedatum, bericht.titel, bericht.nieuwsbericht))
    this.#berichtenToHTML(berichten);
  }).catch(error => console.log(error));
}

```

8.7 Promises - Promise.all

```

const promiseArray = [
  fetch("../data/part1.txt").then((response) => response.text()),
  fetch("../data/part2.txt").then((response) => response.text()),
  fetch("../data/part3.txt").then((response) => response.text()),
];

Promise.all(promiseArray).then((array) => {
  boektekst.insertAdjacentText("beforeend", array[0]);
  boektekst.insertAdjacentText("beforeend", array[1]);
  boektekst.insertAdjacentText("beforeend", array[2]);
});

```

8.8 Async - Await

- De functie die de API ophaalt **moet** vanboven staan en moet ook een `async` zijn

```

async #initialiseHTML() {
  await this.#getData();
  // Volgende vraag (eerste vraag afbeelden)
  this.#showTrivia();
}

async #getData() {
  try {
    const response = await fetch(this.#url);
    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }
    const resultJSON = await response.json();
    this.#triviaRepository.addTrivias(resultJSON.results);
  } catch (error) {
    document.getElementById(

```

```
        'answer'
      ).innerText = `Something went wrong retrieving the quiz data: ${error}`;
    }
  }
}
```

8.9 Voorbeeld van data ophalen en als een nieuw object omtoveren via Async - Await

```
async haalDataOp() {
  try {
    const response = await fetch("./data/fruit.json");
    //const response = await fetch('https://www.fruityvice.com/api/fruit/all');
    if (!response.ok)
      throw new Error(`Fout ophalen fruit : ${response.status}`);
    const json = await response.json();
    json.forEach(
      ({ name, nutritions: { carbohydrates, protein, fat, calories } }) =>
        this.#fruitRepository.voegFruitToe(
          name,
          carbohydrates,
          protein,
          fat,
          calories
        )
    );
  } catch (error) {
    console.log(error);
  }
}
```