Cursus Robbe Magerman Web Development II

Robbe Magerman

08/06/2024

Inhoudstafel

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 geïnitialliseerd

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 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
```

1.7 Math

1.7.1 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
```

1.7.2 Math.floor en Math.ceil

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

1.7.3 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);
```

1.7.4 Math.pi

- Geeft het volledige getal Pi weer

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

1.8 Controlestructuren

1.8.1 if - else - if else

1.8.2 switch

1.8.3 while - do while

1.8.4 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 **(paramter = 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 uitgebreidt, 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 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();
```

2.2.5 delete

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

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

2.2.6 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, numberOfItemsToRemove, waarde1, waarde2, waardex): verwijdert vanaf de opgegeven index een aantal waarden en jij kunt ze vervangen via waardex
- sort(): sorteert de array
- 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
    "
```

3 Hoofdstuk 3: Objecten en Functies

3.1 Functies

3.1.1 Rest operator

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

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

3.1.2 Arrow function

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

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

3.2 Events

3.2.1 Een element in html ophalen

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

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

3.2.2 onclick

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

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

3.2.3 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 propertie heeft een getter om de waarde op te halen
- Elke propertie 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.getElmentById(
      "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 soorteert 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 browers gebruiken we **localStorage**
- In een klasse slaan we dit op als een propertie:

```
  #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 get...FromStorage() via een id van de HTML dat *niet* in een String moet worden gezetn

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

6.1.4 set...InStorage() via een id van de HTML dat *niet* in een String moet worden gezet

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

6.1.5 set...InStorage() via een id van de HTML dat *wel* in een String moet worden gezet

- 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 bepaalt 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)`

6.1.6 set...InStorage() via een id van de HTML dat *wel* in een String moet worden gezet

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

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);
  }
```

7 Hoofdstuk 7: DOM

7.1 HTML tags maken via JavaScript

7.1.1 Aan een bestande 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">
      <img src="images/${c.categorie}"/>
      <h4>${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

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

8.3 AJAX (Asyncrhoon 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

```
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 - 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]);
});
```

8.7 Async - Await

```
  async #getData() {
    try {
      const response = await fetch(this.#url);
      if (!response.ok) {
        throw new Error(
          `Failed fetching trivias, reason ${response.statusText}`
        );
      }
      const jsonResponse = await response.json();
      this.#triviaRepository.addTrivias(json);
      console.log(jsonResponse);
    } catch (error) {
      alert(error);
    }
  }
```