Web technology

Les 12: JS - Functies, objecten en toegang tot de DOM

Kristof Michiels

Functies

Wat zijn functies?

Functies zijn blokken met code statements die een specifieke taak vervullen. Waarom zijn ze nuttig? Klinkt het logisch dat we telkens we de oppervlakte van een cirkel willen berekenen we onderstaande statements telkens opnieuw 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 declareren (we spreken hier over *functie-declaraties*) gebruiken we het function keyword, een unieke naam en een paar haakjes/accolades. Eens 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);
```

ps: JS kan je functies ook creëren met de Function() constructor

Objecten

Verschillende variabelen

Herinner je de naam, leeftijd en lengte variabelen die we in de eerste les hebben aangemaakt? Deze variabelen 'horen' duidelijk samen. Wat nu als we bvb de leeftijd van een andere persoon wensen bij te houden?

```
let naam = "Benny";
let leeftijd = 35;
let lengte = 1.78;
console.log(naam); // Benny
```

Variabelen binnen een object: eigenschappen

Objecten zijn variabelen die bestaan uit verschillende waarden. We noemen die waarden eigenschappen (of properties). Eerst tonen we hoe we een object dienen aan te maken. We doen dit (hier voor een leeg object) met een paar accolades.

```
let persoon = {};
console.log(typeof(persoon)); // object
```

Objecten en hun eigenschappen

Vergeleken met de waarden van arrays hebben de eigenschappen van objecten een belangrijk voordeel: ze hebben een naam. We maken deze eigenschappen aan met een naam en een waarde, die we van elkaar scheiden met een dubbelpunt. De eigenschappen zijn op hun beurt van elkaar gescheiden door een komma.

```
let persoon = {
    naam: "Benny",
    leeftijd: 35,
    lengte: 1.78
};
console.log(persoon); // {naam: "Benny", leeftijd: 35, lengte: 1.78}
```

Opvragen van de waarden van de eigenschappen

We kunnen hun namen gebruiken om toegang te krijgen tot de waarden. We kunnen ofwel vierkante haakjes gebruiken, ofwel een dot-notatie (met een puntje dus).

```
let persoon = {
    naam: "Benny",
    leeftijd: 35,
    lengte: 1.78
};
console.log(persoon.naam); // Benny
console.log(persoon["leeftijd"]); // 35
```

De waarden van de eigenschappen wijzigen

We kunnen de waarden van de eigenschappen aanpassen door er een nieuwe waarde aan toe te kennen.

```
let persoon = {
    naam: "Benny",
    leeftijd: 35,
    lengte: 1.78
};
persoon.leeftijd = 36;
console.log(persoon); // {naam: "Benny", leeftijd: 36, lengte: 1.78}
```

Eigenschappen verwijderen

Wat als we een eigenschap willen verwijderen? Hiervoor gebruiken we het keyword delete. Met delete verwijder je het hele naam-waarde koppel die de eigenschap omvatte.

```
let persoon = {
    naam: "Benny",
    leeftijd: 35,
    lengte: 1.78
};
delete persoon.lengte;
console.log(persoon); // {naam: "Benny", leeftijd: 35}
```

Object methodes

Methodes (of methods) zijn niets anders dan functies die verbonden zijn met een object. Methods doen acties op het object. We definiëren een method door een functie-expressie aan een eigenschap te koppelen. We roepen de method aan op dezelfde wijze we dat zouden doen met een normale functie.

```
let persoon = {
    naam: "Benny",
    leeftijd: 35,
    lengte: 1.78,
    spreek: function() {
        console.log("Hallo, ik ben " + this.naam + "!");
    }
};
persoon.spreek(); // Hallo, ik ben Benny!
```

Meerdere gelijkaardige objecten aanmaken

Door het aanmaken van een constructor functie kunnen we het keyword new gebruiken om zoveel instances van het object te creëren als we willen

```
function Persoon(naam, leeftijd, lengte) {
    this.naam = naam;
    this.leeftijd = leeftijd;
    this.lengte = lengte;
    this.spreek = function() {
        console.log("Hallo, ik ben " + this.naam + "!");
    }
}
let persoonBenny = new Persoon("Benny", 35, 1.78);
let persoonBjorn = new Persoon("Bjorn", 29, 1.84);
console.log(persoonBenny.leeftijd); // 35
console.log(persoonBjorn.leeftijd); // 29
```

Toegang tot de DOM met het Document object: de basis

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>
```

Bestaande DOM-elementen opvragen

- Een uniek element kan je opvragen met de wellicht meest gebruikte method van het document object: document.getElementById()
- Indien je wil opvragen op klassenaam of op elementnaam: document.getElementsByClassName() of document.getElementsByTagName()
- Het resultaat komt terug als een array-achtige HTML-collection: kan 0, 1 of meerdere elementen bevatten
- We stoppen het resultaat best in een variabele: dit geeft ons een houvast naar een plek in de DOM.

getElementById()

Ik heb er 1 en slechts 1 vast: getElement!

```
<h3 id="mijntitel">Een titel</h3>
```

```
let mijnOnderTitel = document.getElementById("mijntitel");
```

getElementsByTagName()

Ik heb er altijd meerdere vast (zelfs als er geen gevonden zijn, dan krijg ik een lege HTMLCollection): getElements!

```
<h1>Hoofdtitel</h1>
<h3>Een titel</h3>
<h3>Een titel</h3>
<h3>Een titel</h3>
```

```
let hoofdTitels = document.getElementsByTagName("h1");
let onderTitels = document.getElementsByTagName("h3");
```

getElementsByClassName()

Ik heb er altijd meerdere vast (zelfs als er geen gevonden zijn, dan krijg ik een lege array): getElements!

```
<h1 class="hoofdtitel">Hoofdtitel</h1>
<h3 class="ondertitel">Een titel</h3>
<h3 class="ondertitel">Een titel</h3>
<h3 class="ondertitel">Een titel</h3>
```

```
let hoofdTitels = document.getElementsByClassName("hoofdtitel");
let onderTitels = document.getElementsByClassName("ondertitel");
```

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.getElementById("hoofdtitel");
console.log(hoofdTitel.innerHTML); // "Je hoofdtitel staat hier"
```

Loopen door de array

Komt er een array terug, en je wil met elk element iets doen? Dan ga je loopen door de array.

```
<p>Mijn eerste paragraaf</p>
<p>Mijn tweede paragraaf</p>
<p>Mijn derde paragraaf</p>
<p>Mijn vierde paragraaf</p>
```

```
let mijnParagrafen = document.getElementsByTagName("p");
let i = 0;
while (i < mijnParagrafen.length) {
    console.log(mijnParagrafen[i].innerHTML);
    i++;
}
```

Heb je er maar eentje nodig?

Komt er een array terug en heb je er maar eentje nodig? Je kan net zoals bij een array verwijzen naar het indexgetal.

```
<a href="#" class="linkjes">Mijn eerste link</a>
<a href="#" class="linkjes">Mijn tweede link</a>
```

```
let mijnLinks = document.getElementsByClassName("linkjes");
console.log(mijnLinks[0].innerHTML);
```

Tekst van bestaande DOM-elementen aanpassen

Hiervoor gebruiken we net zoals bij het opvragen de innerHTML-eigenschap

```
<p id="demo">Tekst voor mijn paragraaf</p>
```

```
let demoElement = document.getElementById("demo");
console.log(demoElement.innerHTML); // "Tekst voor mijn paragraaf"
demoElement.innerHTML = "Nieuwe tekst, komt in de plaats!";
console.log(demoElement.innerHTML); // "Nieuwe tekst, komt in de plaats!"
```

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.getElementById("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.getElementById("trivia"); // moederelement vastnemen
moederElement.appendChild(nieuweHoofding); // kind toevoegen aan moederelement
moederElement.appendChild(nieuweParagraaf); // kind toevoegen aan moederelement
```