THOMAS MORE

JavaScript

Arrow functions, Arrays & Sets

Inhoudsopgave

THOMAS MORE

JavaScript

Arrow functies

Arrow functies

01_arrow.js

- Een arrow functie is een compact alternatief (**kortere syntax**) voor een traditionele functie.
- Arrow functies zijn, in tegenstelling tot een normale functie, niet gebonden aan het **this** keyword (zie volgende les).

```
// Klassieke functie
function foo() {
  return 'Hello Foo'
}

// Arrow functie op één lijn.
// Geen return nodig, een one-line array functie
// geeft het resultaat automatisch terug.
const bar = () => 'Hello Bar'

// Arrow functie met meerdere statements.
const baz = () => {
  const message = 'Hello Baz'
  // Een multi-line arrow functie heeft een
  // return statement nodig (als je iets wilt teruggeven).
  return message
}
```

Arrow functies

01_arrow.js

- Indien **geen parameters** dan zijn lege ronde haken verplicht.
  Met een simpele expressie is een return niet nodig.

```
const functionName = () => expression
```

- **Eén parameter** dan mogen de haken weggelaten worden.
  Met een simpele expressie is een return niet nodig:

```
const functionName = param => expression
```

- **Meerdere parameters** hebben ronde haken nodig.
  Met een simpele expressie is een return niet nodig.

```
const functionName = (param1, paramN) => expression
```

Arrow functies

01_arrow.js

- **Meerdere statements** hebben **accolades** nodig:

```
const functionName = param => {
  const a = 1;
  return a + param;
}
```

- **Meerdere parameters** hebben **ronde haken** nodig en **meerdere statements** hebben **accolades** nodig:

```
const functionName = (param1, paramN) => {
  const a = 1;
  return a + param1 + paramN;
}
```

THOMAS MORE

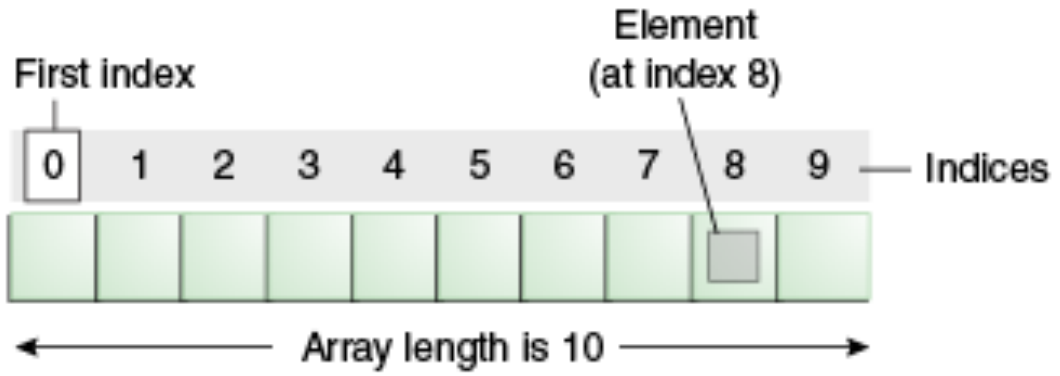
JavaScript

Arrays

Arrays - Declareren en initialiseren

02_array.js

- Array = **geordende** verzameling elementen
  - Elk element heeft een numerieke positie (index)
  - Index start bij 0
  - Notatie: blokhaken []




```
// Voorbeelden van arrays
const leeg = []; // Een lege array.
const leeg2 = new Array(5); // Een lege array met 5 lege plaatsen.
const tientallen = [10, 20, 30, 40, 50]; // Een array met 5 numerieke waarden.
const diversen = ['Hello', 'World', 10, 20, true]; // Een array met verschillende types. Nooit gebruiken!
const reeks = [1, , 3] // Een array met een lege plaats tussen 1 en 3.
```

Arrays uitlezen en toekennen

02_array.js

- Opvragen: [x]
  - x = postitie index
  - array[] rechts van = → uitlezing waarde (get) a = tientallen[2];
  - array[] links van = → toekenning waarde (set) tientallen[5] = 60;
  - Bestaande index: overschrijving

```
// Vervang het element op positie 0 in de array "tientallen" door 0
// De array heeft vijf numerieke waarden, index van 0 t.e.m. 4
tientallen[0] = 0;

// Voeg een nieuw element toe aan de array tientallen.
// Posities 5, 6, 7, 8 en 9 worden opgevuld met lege waarden.
tientallen[10] = 100;

// Lees de waarde van de array tientallen uit op positie 2.
const waarde = tientallen[2];
```

length (property)

02_array.js

- De eigenschap **length**
  - Lengte kan worden uitgelezen of ingesteld
  - Bij instellen worden eventueel elementen verwijderd!

```
// Eigenschap .length als toekenning gebruiken
const z = ['a', 'b', 'c', 'd']; // z heeft vier elementen
z.length = 2; // z is nu ['a', 'b']
z.length = 0; // z is nu [] (nul elementen, lege array)
z.length = 4; // De lengte is vier, maar de array bevat geen elementen. Dit is hetzelfde als new Array(4).
```

Itereren door Arrays op 5 manieren

- Herhalingen waarin we break of continue kunnen gebruiken
  - for
  - while
  - do…while
  - for…in
    - *Geeft geen lege elementen weer!*
  - for *… of*

- Herhalingen waarin we GEEN break of continue kunnen gebruiken
  - forEach
    - *Geeft geen lege elementen weer!*
    - *Gebruiken wanneer op ieder element een bewerking moet uitgevoerd worden*
    - *Later in de les*

Itereren met for, while & do while

03_arrray_loops.js

```
const countries = ['Brazil', 'Mexico', , 'Belgium', 'Germany', 'Sweden'];

// FOR
console.log('FOR\n-----------')
for (let i = 0; i < countries.length; i++) {
  console.log(countries[i]);
}

// WHILE
console.log('\nWHILE\n-----------')
let i = 0;
while (i < countries.length) {
  console.log(countries[i]);
  i++;
}

// DO WHILE
console.log('\nDO WHILE\n-----------')
i = 0;
do {
  console.log(countries[i]);
  i++;
} while (i < countries.length);
```

```
FOR
-----------
Brazil
Mexico
undefined
Belgium
Germany
Sweden

WHILE
-----------
Brazil
Mexico
undefined
Belgium
Germany
Sweden

DO WHILE
-----------
Brazil
Mexico
undefined
Belgium
Germany
Sweden
```

Itereren met for…in & forEach

03_arrray_loops.js

```
// FOR...IN
console.log('\nFOR...IN\n------------')
for (const index in countries) {
  console.log(countries[index]);
}

// FOR...OF
console.log('\nFOR...OF\n------------')
for (const country of countries) {
  console.log(country);
}
```

```
FOR...IN
------------
Brazil
Mexico
Belgium
Germany
Sweden
```

```
FOR...OF
------------
Brazil
Mexico
undefined
Belgium
Germany
Sweden
```

Array-methoden

- Arrays hebben een groot aantal built-in functies (methods)
  - Sorteren (**sort**)
  - Volgorde omdraaien (**reverse**)
  - Samenvoegen of splitsen van een string (**join <--> split**)
  - Elementen toevoegen/verwijderen (**push <--> pop**)
  - Elementen opvragen met negatieve index (**at**)
    - Het laatste element heeft index -1 voor deze methode, het element daarvoor -2

- Compleet overzicht
  - https://developer.mozilla.org/nl/docs/Web/JavaScript/Reference/Global_Objects/Array

join() - split()

04_array_join_split.js

- De methode **join** converteert alle elementen in de array naar één string.
  - Optie scheidingsteken meegeven (*)
- De methode **split** zet elementen om naar een array.

```
// Array.join - voegt elementen samen tot een string.
const mijnArray =
['info@tm.be','admin@tm.be','support@tm.be'];
console.log(mijnArray.join());
console.log(mijnArray.join(";"));

// Omgekeerd: string.split(), splits een string naar een
array
const emails  =
"info@tm.be;admin@tm.be;support@tm.be";
const mijnMails = emails.split(";");
console.log(mijnMails);
```

```
// Uitvoer
info@tm.be,admin@tm.be,support@tm.be
info@tm.be;admin@tm.be;support@tm.be
```

```
// Uitvoer
[ "info@tm.be", "admin@tm.be", "support@tm.be" ]
```

reverse()

05_array_reverse.js

- De methode **reverse** keert de volgorde om van de elementen in de array en retourneert de omgekeerde array.
  - Gebruik de methode “in place” => geen nieuwe array, maar bestaande array wordt aangepast dan!

```
// Array.prototype.reverse - draait de volgorde van elementen in de
array om.
const getallen = [10, 20, 30, 40, 50];
console.log('Origineel: ', getallen);

const omgekeerd = getallen.reverse();
console.log('Reverse: ', omgekeerd);
getallen.reverse(); // De originele array wordt "in place" aangepast!
```

```
// Uitvoer
Origineel:  [ 10, 20, 30, 40, 50 ]
Reverse:  [ 50, 40, 30, 20, 10 ]
```

sort()

06_array_sort.js

- De methode **sort** sorteert elementen op eenvoudige wijze.
  - Standaard van A-Z, daarna van a-z
  - Indien nodig worden elementen omgezet naar strings

```
// Array.sort - sorteert de elementen in de array op ALFABETISCHE volgorde.
// Eerst hoofdletters A-Z, dan a-z
const kleuren = ["rood", "geel", "blauw", "groen", "zwart", "wit"];
kleuren.sort();
console.log('Gesorteerde kleuren:\n  ', kleuren);

// Cijfers worden ook vergeleken op basis van de ASCII tabel.
// Daardoor is 1 < 10000 < 20 < 32 < 4 < 5
const getallen = [5, 4, 10000, 20, 1, 32, 4];
getallen.sort();
console.log('Gesorteerde getallen:\n  ', getallen);
```

```
// Uitvoer
Gesorteerde kleuren:
   [ "blauw", "geel", "groen", "rood", "wit", "zwart" ]
Gesorteerde getallen:
   [ 1, 10000, 20, 32, 4, 4, 5 ]
```

push()

07_array_push.js

- De methode **push** voegt element(en) aan het einde van de array toe.

```
// Array.push - voegt achteraan een element toe.
const kleuren = ["rood", "geel"];
console.log(kleuren);

kleuren.push("blauw");
console.log('Push 1 extra: ', kleuren);
kleuren.push("wit","zwart");
console.log('Push 2 extra: ', kleuren);
```

```
// Uitvoer
[ "rood", "geel" ]
Push 1 extra:  [ "rood", "geel", "blauw" ]
Push 2 extra:  [ "rood", "geel", "blauw", "wit", "zwart" ]
```

pop()

08_array_pop.js

- De methode **pop** verwijderd het laatste element in de array.

```
// Array.pop - verwijder laatste element uit de array.
const kleuren = ["rood", "geel", "blauw", "zwart", "wit"];
console.log(kleuren);

kleuren.pop();
console.log(kleuren);
```

```
// Uitvoer
[ "rood", "geel", "blauw", "zwart", "wit" ]
[ "rood", "geel", "blauw", "zwart" ]
```

fill()

09_array_fill.js

- De **fill** methode wijzigt alle elementen (of een aantal) met een waarde en geeft de gewijzigde array terug.
- Syntax => arr.fill(value[, start[, end]])
  Start range is inclusive and end range is exclusive.

```
// We initialiseren een array van 10 elementen met nullen
const simpelArray = new Array(10).fill(0);
console.log(simpelArray);

// We initialiseren een array van 10 elementen met een lege string.
// Daarna vullen we deze vanaf de 3e index (incl.) tot de laatste (excl.) op met de string 'test'.
const stringArray = new Array(10).fill('');
stringArray.fill('test', 3, stringArray.length);
console.log(stringArray);


// Uitvoer
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
[ "", "", "", "test", "test", "test", "test", "test", "test", "test" ]
```

You Don’t Need Loops in JavaScript

We kunnen i.p.v. een iteratie een **HIGHER ORDER FUNCTION** gebruiken zoals **map**, **filter** of **reduce**.

In eenvoudige bewoordingen is een functie van hogere orde een functie die een functie als argument ontvangt of de functie als uitvoer retourneert.

You Don't Need Loops in JavaScript

- Higher order functions krijgen andere parameters als argument (**callback**)
  - Anonieme functie of **arrow functie**

```
function higherOrderFunction(callback) {
  // Doe iets met de callback functie
}

// Met anonieme functie
higherOrderFunction(function(){
  // Doe iets.
})
```

```
// Met arrow functie, meest gebruikt in moderne code
higherOrderFunction(() => {
  // Doe iets.
})
```

filter()

10_array_filter.js

- De methode **filter** verwijderd elementen in een array die niet aan een voorwaarde voldoen (heeft **callback** parameter!)
  - De parameter geeft een bolean terug en wordt daarom een **predicaat** genoemd.

```
const leeftijden = [32, 33, 16, 40, 7, 56];

function isMeerderjarig(leeftijd) {
  return leeftijd >= 18;
}

// Beide opties zijn equivalent.
const meerderjarigen = leeftijden.filter(isMeerderjarig);
const meerderjarigen2 = leeftijden.filter(leeftijd => leeftijd >= 18);

console.log('Meerderjarigen:', meerderjarigen);
console.log('Meerderjarigen2:', meerderjarigen2);
```

```
// Uitvoer
Meerderjarigen: [ 32, 33, 40, 56 ]
Meerderjarigen2: [ 32, 33, 40, 56 ]
```

map()

11_array_map.js

- De methode **map** voert een bewerking uit op ieder element van de array, met als uitvoer een array met dezelfde lengte (heeft **callback** parameter!)

```
const getallen = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const kwadraten = getallen.map(getal => getal * getal);

console.log('Originele getallen:', getallen);
console.log('Kwadraten:', kwadraten);

const evenKwadratenOnevenHelft = getallen.map((getal, index) => {
  return index % 2 === 0 ? getal * getal : getal / 2;
})

console.log('Even indexen kwadrateren, oneven indexen halveren:', evenKwadratenOnevenHelft);

// Uitvoer
Originele getallen: [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
Kwadraten: [ 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 ]
Even indexen kwadrateren, oneven indexen halveren: [ 1, 1, 9, 2, 25, 3, 49, 4, 81, 5 ]
```

reduce()

12_array_reduce.js

- De methode **reduce** reduceert een array tot één waarde en gaat van links naar rechts (heeft **callback** parameter!)

```
const nummer = [1, 2, 3, 4, 5];

const som = nummer.reduce((accumulator, currentValue) => {
  return accumulator + currentValue;
}, 0)

console.log('Som: ', som)

const product = nummer.reduce((accumulator, currentValue) => {
  return accumulator * currentValue;
}, 0)

console.log('Product: ', product)
```

```
// Uitvoer
Som:  15
Product:  0
```

Nog andere array-methoden

concat(): voegt twee arrays samen tot een nieuwe array

slice(): retourneert een subarray uit de aangegeven array

splice(): voegt elementen in of verwijdert elementen uit de array op de aangegeven positie

unshift(): zoals push(), maar voegt element toe aan het begin van de array

shift(): zoals pop(), maar verwijdert element aan het begin van de array

toString(): zet alle elementen in de array om naar een string en retourneert deze

toLocaleString(): idem, maar houdt rekening met het scheidingsteken dat voor de huidige taalversie wordt gebruikt

THOMAS MORE

JavaScript

Sets

Sets - Declareren en initialiseren

13_sets.js

- Set = **niet** geordende verzameling elementen
  - Een element kan slechts één keer voorkomen
  - Veel performanter (sneller) dan een array als je wilt controleren of iets in de set zit

```
// Een lege set
const emptySet = new Set();

// Een set met een aantal elementen
const languages = new Set(['JavaScript', 'Python', 'Java', 'C++', 'JavaScript']);
console.log(languages);
```

```
// Uitvoer
Set(4) {
  "JavaScript",
  "Python",
  "Java",
  "C++",
}
```

Sets – Toevoegen, lezen en aanpassen

13_sets.js

```
// Voeg een element toe aan de set
languages.add('C#');

// Verwijder een element uit de set
languages.delete('Java');

// Controleer of een element in de set zit
console.log('Python in set: ', languages.has('Python'));

// Itereren over de set
console.log('Talen in de set:')
for (const language of languages) {
  console.log(language);
}
```

```
// Uitvoer
Python in
set:  true
Talen in de set:
JavaScript
Python
C++
C#
```

Nog andere set-methoden

clear(): maak de set volledig leeg

difference(): bepaal de elementen die in één set zitten, maar niet in een andere

has(): controleer of element in de set zit

intersection(): bepaal de doorsnede van twee sets (elementen die in beide sets zitten)

union(): combineer twee sets

forEach(): zoals de forEach methode van arrays, loop door alle elmenten in de set

Oefeningen