

# Geïndexeerde collecties

Een Array of geïndexeerde collectie is een geordende verzameling van waarden waarnaar we refereren met een naam en een index. We kunnen bijvoorbeeld een array genaamd `students` definiëren dat een lijst van studenten bevat. De eerste student kan opgevraagd worden via `students[0]` de tweede student via `students[1]` ...

Net zoals alle andere objecten bevat een array een reeks van eigenschappen (lengte ...) en methoden (filteren, zoeken, omkeren ...)

## Aanmaak van een array

Een array kan aangemaakt worden op drie manieren:

```
1  const arr1 = new Array(element0, element1 ..., elementN);
2  const arr2 = Array(element0, element1 ..., elementN);
3  const arr3 = [element0, element1 ..., elementN];
```

`element0, element1 ..., elementN` is een lijst van waarden voor de elementen uit de array. De lengte van een array is gelijk aan het aantal aanwezige elementen in deze array. De lengte van een array kan opgevraagd worden via de eigenschap `arrayName.length`. De bracket ( `[...]` ) notatie, ook gekend als een **array literal** of **array initializer**, heeft de voorkeur op de andere notaties.

Om een lege (zonder elementen) array met een bepaalde lengte (positief geheel getal) te maken kunnen we het volgende implementeren:

```
1  const arrayLength = 6;
2  const arr1 = new Array(arrayLength);
3  const arr2 = Array(arrayLength);
4  const arr3 = [];
5  arr3.length = arrayLength;
```

Al deze arrays bevatten de volgende elementen `[undefined, undefined, undefined, undefined, undefined, undefined]`. Lege elementen ( `undefined` ) worden niet weergegeven in een `for...in` -lus. Indien een array lengte

wordt gedefinieerd met een floating point waarde bijv. `Array(78.9)` dan resulteert dit in de foutboodschap

`Uncaught RangeError: Invalid array length`.

`Array.of()` laat toe om een array aan te maken bestaande uit één element, bijv. `const arr1 = Array.of('John');`.

## Bevolken van een array

Een array **bevolken** (*Eng. populating*) kan op verschillende manieren gebeuren. We kunnen een lege array bevolken door waarden toe te kennen op een specifieke index.

```
1  const persons = [];  
2  persons[0] = 'Philippe';  
3  persons[1] = 'Evelien';  
4  persons[2] = 'Olivier';  
5  console.log(persons); // Output: ["Philippe", "Evelien", "Olivier"]
```

Definiëren we een lege array en kennen we vervolgens een waarde toe aan een element op index `99` dan resulteert dit in een array met een lengte van `100` elementen. De elementen voor dit element bevatten de waarde `undefined`.

```
1  const persons = [];  
2  persons[99] = 'Philippe';  
3  console.log(persons.length);
```

Definiëren we een index met een **decimaal getal** (*Eng. floating point number*), dan zal dit getal gezien worden als een eigenschap van de array, en dus geen element.

```
1  const persons = [];  
2  persons[8.4] = 'Philippe';  
3  console.log(persons.length); // Output: 0  
4  console.log(persons.hasOwnProperty(8.4)); // Output: true
```

### Opgelet

Door de lengte van de array op `0` in te stellen, verwijderen we alle aanwezige elementen.

```
1 const cats = ['Dusty', 'Misty', 'Twiggy'];
2 console.log(cats.length); // Output: 3
3 cats.length = 0;
4 console.log(cats); // Output: []
```

Met de methoden `push`, `unshift`, `splice` en `concat` kan de array ook uitgebreid worden met elementen.

Met de `push` methode kunnen we één of meerdere elementen toevoegen op het einde van de array. De `push` methode geeft de lengte van de aangepaste array terug.

```
1 let fruit = ['Apple', 'Banana', 'Grape'];
2 fruit.push('Pear'); // Array fruit is now ['Apple', 'Banana', 'Grape', 'Pear']
```

Met de `unshift` methode kunnen we één of meerdere elementen toevoegen aan het begin van de array. De `unshift` methode geeft de lengte van de aangepaste array terug.

```
1 let weather = ['Wind', 'Rain', 'Fire'];
2 weather.unshift('Snow', 'Hail'); // Array weather is now ['Snow', 'Hail', 'Wind', 'Rain', 'Fire']
```

Met de `splice` [🔗](#) methode kunnen we één of meerdere elementen toevoegen binnen een opgegeven positie uit de array. De `splice` methode laat ook toe om elementen te verwijderen uit de array. De `splice` methode geeft een array terug van verwijderde elementen.

```
1 let myArray = new Array('1', '2', '3', '4', '5');
2 myArray.splice(1, 0, 'a', 'b', 'c', 'd'); // Array myArray contains ['1', 'a', 'b', 'c', 'd', '5']
```

Met het eerste element bepalen we de plaats (index) in de array waarna we elementen zullen toevoegen. Het tweede argument met de waarde `0` betekent dat we geen elementen zullen verwijderen uit de array.

```
1 let myArray = new Array('1', '2', '3', '4', '5');
2 myArray.splice(1, 3); // Array myArray contains ['1', '5']
```

Met de `concat` [🔗](#) methode kunnen we meerdere arrays met elkaar te verbinden tot één array. De `concat` methode geeft de resulterende array terug.

```
1  const array1 = ['a', 'b', 'c'];
2  const array2 = ['d', 'e', 'f'];
3  let arrayResult = array1.concat(array2); // Array arrayResult contains ['a', 'b', 'c',
```

Meerdere arrays kunnen verbonden worden:

```
1  const num1 = [1, 2, 3];
2  const num2 = [4, 5, 6];
3  const num3 = [7, 8, 9];
4
5  const numbers = num1.concat(num2, num3);
6
7  console.log(numbers); // Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

## Itereren over een array

Een array kan doorlopen worden met een for lus:

```
1  const colors = ['red', 'green', 'blue'];
2  for (let i = 0; i < colors.length; i++) {
3    const currentColor = colors[i];
4    console.log(currentColor);
5  }
```

Een element kan binnen de lus opgevraagd worden door de index te specificeren bijv. `const currentColor = colors[i];`. De index van een geïndexeerde collectie begint altijd vanaf `0`.

Het `for...of` statement creëert een lus die itereert over iterabele objects (zoals: `Array`, `Map`, `Set` ...) en roept vervolgens een variabele aan met een waarde die kan verschillen bij iedere iteratie.

```
1  const colors = ['red', 'green', 'blue'];
2  for (let element of colors) {
3    const currentColor = element;
4    console.log(currentColor);
5  }
```

Hebben we de index van een element nodig binnenin een `for...of` lus, dan moeten we gebruik maken van de `entries()` [🔗](#) methode uit een Array object.

```
1  const colors = ['red', 'green', 'blue'];
2  for (let [index, element] of colors.entries()) {
3    const currentColor = element;
4    console.log(`Index ${index} with color ${currentColor}`);
5  }
```

Met de `forEach()` [🔗](#) methode van een array kunnen we eveneens itereren over een array:

```
1  const colors = ['red', 'green', 'blue'];
2  colors.forEach(function(currentColor) {
3    console.log(currentColor);
4  });
```

We kunnen deze code nog korter schrijven door gebruik te maken van een **arrow function**:

```
1  const colors = ['red', 'green', 'blue'];
2  colors.forEach((currentColor) => console.log(currentColor));
```

De functie als argument in de `forEach` methode wordt bij iedere iteratie uitgevoerd. Deze functie bevat zelf een argument, namelijk het huidige element uit de array. **Niet-toegewezen** (*Eng. unassigned*) waarden worden niet geïtereerd:

```
1  const colors = ['red', 'green', , 'blue'];
2  color.forEach((currentColor) => console.log(currentColor));
3  // red
4  // green
5  // blue
```

## Opgelet

Gebruik de `for...in` -lus niet om te itereren over de elementen uit de array, want deze lus lijst ook alle eigenschappen op.

# Multidimensionale array

Arrays kunnen genest worden, wat betekent dat een array een andere array kan bevatten als element.

De volgende code creëert een tweedimensionale array:

```
1  const a = new Array(4);
2  for (i = 0; i < 4; i++) {
3    a[i] = new Array(4);
4    for (j = 0; j < 4; j++) {
5      a[i][j] = '[' + i + ', ' + j + ']';
6    }
7  }
```

Genereert de volgende array inhoud:

```
1  Row 0: [0, 0] [0, 1] [0, 2] [0, 3]
2  Row 1: [1, 0] [1, 1] [1, 2] [1, 3]
3  Row 2: [2, 0] [2, 1] [2, 2] [2, 3]
4  Row 3: [3, 0] [3, 1] [3, 2] [3, 3]
```

Het volgende voorbeeld bevat een array van auto's. Een auto is een array bestaanden uit 3 elementen, namelijk type, stock en sold. De code is als volgt:

```
1  const cars = [
2    ['Volvo', 22, 18],
3    ['BMW', 15, 13],
4    ['Saab', 5, 2],
5    ['Land Rover', 17, 15],
6  ];
7
8  cars.forEach(function(car) {
9    console.log(`${car[0]}: ${car[1]} in stock and ${car[2]} sold.`)
10 });
```

## Array methoden

We hebben reeds elementen aan een bestaande array kunnen toevoegen via de methoden `push` , `unshift` , `splice` en `concat` . Er zijn nog een hele reeks andere methoden die nuttig kunnen zijn.

Met de `pop` methode kunnen we het laatste element uit een array verwijderen. De `return` waarde van deze methode is het verwijderd element of `undefined` (indien de array leeg was).

```
1  const fruit = ['Apple', 'Banana', 'Grape'];
2  const removedElement = fruit.pop(); // Contains "Grape"
3  console.log(fruit.length); // Output: 2
```

Met de `shift` [↗](#) methode kunnen we het eerste element uit een array verwijderen. De `return` waarde van deze methode is het verwijderd element of `undefined` (indien de array leeg was).

```
1  const fruit = ['Apple', 'Banana', 'Grape'];
2  const removedElement = fruit.shift(); // Contains "Apple"
3  console.log(fruit.length); // Output: 2
```

Met de `slice` [↗](#) methode kunnen we een portie van een array selecteren om daarmee een nieuwe array te maken startend van een bepaalde index ( `begin` ) en eindigend op een bepaalde index ( `end` ). Het element op deze laatste index maakt geen deel uit van de nieuwe gegenereerde array. De originele array wordt door de `slice` methode niet aangepast.

```
1  const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
2  const selectedAnimals = animals.slice(2, 4); // The array selectedAnimals contains ['camel', 'duck']
3  console.log(animals.length); // Output: 5
```

Met de `join` [↗](#) methode kunnen we alle elementen uit de array verbinden met elkaar tot een resulterende string waarin de elementen worden gescheiden door een komma ( `,` ). We kunnen als alternatief een eigen **scheidingsteken** (Eng. *delimiter, separator*).


```
1  const arr = ['macOs', 'Windows', 'Linux'];
2  const arrAsStr = arr.join();
3  console.log(arrAsStr); // Output "macOs,Windows,Linux"
```

Of met een eigen gedefinieerd scheidingsteken:


```
1 const arr = ['macOs', 'Windows', 'Linux'];
2 const arrAsStr = arr.join(' | ');
3 console.log(arrAsStr); // Output "macOs | Windows | Linux"
```

## Opgelet

Indien een element de waarde: `undefined`, `null` of een lege array `[]` bevat, dan zullen deze waarden geconverteerd worden naar een lege string. Indien een element een custom object is, dan voorzie je best een `toString` methode binnen dit object, zoniet resulteert dit in een string `[object Object]`.

Met de `reverse`  methode kunnen we de plaatse van elementen in de array verwisselen. Het eerste element wordt de laatste, het tweede element wordt de voorlaatste ... Bevat als return waarde de omgekeerde array. De originele array wordt ook beïnvloed!

```
1 const arr = ['one', 'two', 'three'];
2 const reversedArray = arr.reverse();
3 console.log(reversedArray); // Output: ['three', 'two', 'one']
4 console.log(arr); // Output: ['three', 'two', 'one']
```

De `sort`  methode sorteert de elementen van een array. De standaard sorteerorde werkt op basis “van geconverteerde elementen als een stringwaarde”. De `sort` methode kan ook een **callback function** bevatten om de sorteerorde te manipuleren.

```
1 const months = ['March', 'Jan', 'Feb', 'Dec'];
2 months.sort();
3 console.log(months); // Output: ["Dec", "Feb", "Jan", "March"]
```

Met numerieke waarden:

```
1 const numbers = [1, 30, 4, 21, 100000];
2 numbers.sort();
3 console.log(numbers); // Output: [1, 100000, 21, 30, 4]
```

De `sort` methode kan ook een **callback function** bevatten om te bepalen hoe de elementen in de array worden vergeleken met elkaar. Dit soort callback functies zijn beter gekend als iteratieve methoden, omdat ze itereren over



alle elementen in de array.

```
1  const myArray = new Array('Wind', 'Rain', 'Fire');
2  myArray.sort(function (a, b) {
3    if (a[a.length - 1] < b[b.length - 1]) return -1;
4    if (a[a.length - 1] > b[b.length - 1]) return 1;
5    if (a[a.length - 1] == b[b.length - 1]) return 0;
6  });
7  myArray.sort(sortFn); // sorts the array so that myArray = ["Wind","Fire","Rain"]
```

- als `a` kleiner is dan `b` dan geeft de functie de waarde `-1` terug
- als `a` groter is dan `b` dan geeft de functie de waarde `1` terug
- als `a` en `b` gelijk zijn dan geeft de de functie de waarde `0` terug

De `map` [🔗](#) methode voert voor elke element een callback functie uit en construeert hiermee een nieuwe array.

De volgende code bevat een array van numerieke waarde en creeert een nieuwe array bestaande uit de vierkantswortel van de numerieke waarden uit de eerste array.

```
1  const numbers = [1, 4, 9];
2  const roots = numbers.map(function (num) {
3    return Math.sqrt(num);
4  });
5  console.log(roots); // Output: [1, 2, 3]
```

De volgende code bevat een array van numerieke waarde en creeert een nieuwe array bestaande uit objecten met als key de index en als value het kwadraat van de corresponderende numerieke waarde uit de eerste array.

```
1  const numbers = [1, 4, 9];
2  const pows = numbers.map(function (num, index) {
3    return { key: index, value: num**2};
4  });
5  console.log(pows);
```

We kunnen deze code herschrijven met arrow functions:

```
1  const numbers = [1, 4, 9];
2  const pows = numbers.map((num, index) => ({ key: index, value: num**2 }));
3  console.log(pows);
```

De `filter` [↗](#) methode creeert een nieuwe array van elementen die een test doorstaan via de callback function.

```
1  const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
2  const filteredWords = words.filter(word => word.length > 6);
3  console.log(result); //Output: ["exuberant", "destruction", "present"]
```

Het Array object bevat nog heel wat andere nuttige methoden, zoals: `reduce` [↗](#), `reduceRight` [↗](#), `every` [↗](#), `flat` [↗](#) ...

[← Objecten](#)

[Time based stuff →](#)