# Constructors, iterators and generators

**Constructors** in JavaScript-klassen zijn speciale methoden die worden gebruikt voor het maken en **initialiseren** van objecten.

Elke class (klasse) kan slechts **één methode** hebben met de naam "constructor". Als er meer dan één constructormethode is gedefinieerd voor een klasse, wordt er een **syntax error** (syntaxisfout) gegenereerd.

Hoe de class productObject eruit zou zien, met een method getDescription()

```
class productObject {
    constructor(id, item, description, price, image){
        this.id = id;
        this.item = item;
        this.description = description;
        this.price = price;
        this.image = image;
    }

    getDescription()
    {
        return this.description;
    }
}
```

## *Iteratoren en generatoren*

Het verwerken van items in een verzameling data komt veel voor bij het schrijven van JavaScript-code. Er zijn verschillende looping-methoden beschikbaar voor ontwikkelaars om een verzameling te herhalen. Iterators en generators zijn JavaScripts manier om iteratie in scripts te introduceren om het gedrag van loops aan te passen.

## *Definitie iterators*

**Iterators** zijn objecten die toegang hebben tot items in een verzameling (bijvoorbeeld een array), één voor één. En **iterables** zijn objecten (of gegevensstructuren) die hun eigen iteratiegedrag definiëren, d.w.z. hoe elk item in een verzameling wordt benaderd. Ingebouwde objecten, zoals Array en Map, hebben een standaard iteratiegedrag. Met iterators (of iterables) kunnen complexe containerobjecten als lijsten worden doorlopen. U kunt door de gebruiker gedefinieerde iterables maken met generatorfuncties. Later leert u meer over de generatorfuncties.

JavaScript biedt ook ingebouwde iterables, dit zijn arrays, strings en maps:

In arrays kan over de posities geïtereerd worden:

```
for (let name of ['Mark', 'Pete']) {
 console.log(name);
}
//Output:
//Mark
//Pete
```

Strings: Deze zijn óók itereerbaar, hierbij wordt over de tekens geïtereerd:

```
for (let val of 'Adam') {
 console.log(val);
}
//Output:
//A
//d
//a
//m
```

Maps: deze objecten zijn itereerbaar, waarbij elk item een [key, value] paar is:

```
const temp = new Map().set('Mark', 1).set('Pete', 2);
for (const map of temp) {
 console.log(map);
}
//Output:
// ["Mark", 1]
// ["Pete", 2]
```

Sets: Over deze elementen wordt in dezelfde volgorde geïtereerd als waarin ze aan de set zijn toegevoegd.

```
var temp = new Set().add('a').add('b');
for (var key of temp) {
 console.log(key);
}
// Output:
// 'a'
// 'b'
```

## De for ... of-lus

De for ... of -lus is nieuw in ECMAScript 6. Hier volgt de syntaxis:

```
var num = [1, 2, 3];
for (let x of num) {
  // at each iteration x assumes the value 1, 2, and 3
  console.log(x);
}

//Output:
// 1
// 2
// 3
```

`break` en `continue` werken ook binnen de for ... of loops. De for ... of loop biedt de gecombineerde voordelen van zowel de for loop als forEach () methode, door middel van break/continue; en beknopte syntaxis.

## Hoe iterators te gebruiken

Alle belangrijke ES6-gegevensstructuren (bijvoorbeeld Arrays, Strings en Sets) hebben drie methoden. Zij zijn:
`entries ():` Retourneert een [key, value]-paar voor elk item. Beschouw het volgende codeblok:

```
var example = ["Value1", "Value2", "Value3"];
var obj1 = example.entries();
for (let x of obj1) {
 console.log(x);
}
// Output:
// [0, "Value1"]
// [1, "Value2"]
// [2, "Value3"]
```

Deze methode retourneerde elke waarde in de voorbeeldarray als een [key, value]-paar.

`keys ():` Retourneert iterables over keys van de items voor een [key, value]-paar. Het volgende codeblok maakt het voor u duidelijker:

```
var example = ["Value1", "Value2", "Value3"];
var obj2 = example.keys();
for (let x of obj2) {
 console.log(x);
}
// Output:
// 0
// 1
// 2
```

De `keys ()` method retourneerde een array met keys voor elke index in de voorbeeldarray.
`values ():` geeft iteraties terug over de waarden van de posities voor een [key, value]-paar. Het volgende codeblok demonstreert het gebruik van deze methode:

```
var example = ["Value1", "Value2", "Value3"];
var obj3 = example.values();
for (let x of obj2) {
 console.log(x);
}
// Output:
// Value1
// Value2
// Value3
```

Zoals uit de uitvoer blijkt, retourneert deze methode alle waarden.
Opmerking: deze methode wordt alleen ondersteund in de Microsoft Edge-browser.

## *Definitie generatoren*

Eerder hebt u bestudeerd dat ontwikkelaars door de gebruiker gedefinieerde iterables kunnen maken. Dit kan worden bereikt door het gebruik van generatoren. Generatoren zijn speciale soorten functies die iterabele objecten retourneren. Een functie wordt een generatorfunctie wanneer deze wordt gedefinieerd met het trefwoord function *. Generatorfuncties gebruiken de `yield` operator om de functie te pauzeren of de output te verzenden en input te ontvangen.

Hoe generatoren worden gebruikt:
Laten we beginnen met het maken van een door de gebruiker gedefinieerde iteratie:

```
function* idGen() {
  var id = 0;
  while(true)
  yield id++;
}
var gen = idGen(); // calling a generator function
```

Deze code genereert een nieuwe iterator `idGen`. Het aanroepen van een generatorfunctie retourneert een generatorobject dat kan worden gebruikt om de generatorfunctie te besturen.

```
console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
```

In JavaScript retourneert de `next()` method het volgende item in de reeks. Bij elke iteratie retourneert de generatorfunctie een nieuwe waarde 0, 1, 2 enzovoort. Elk iterator-object moet zijn interne status behouden. Met generatoren kunnen ontwikkelaars een iteratief algoritme definiëren dat zijn eigen status kan behouden door het creëren van één enkele functie. U kunt ook proberen uw eigen aangepaste iterator te maken die zijn eigen `next ()` -methode heeft.

## *Yield*

De `yield` operator is alleen beschikbaar binnen generatorfuncties en wordt gebruikt om generatorfuncties te pauzeren en te hervatten. De syntaxis is:

```
[return_value] = yield [expression];
// expression value returned from the generator function
```

In het vorige voorbeeld onderbreekt de `yield` operator de uitvoering van de functie idGen () en retourneert de nieuwe generatorwaarde. De uitvoering van de functie wordt hervat de volgende keer dat de `next()` method (methode `gen.next ()`) wordt aangeroepen.