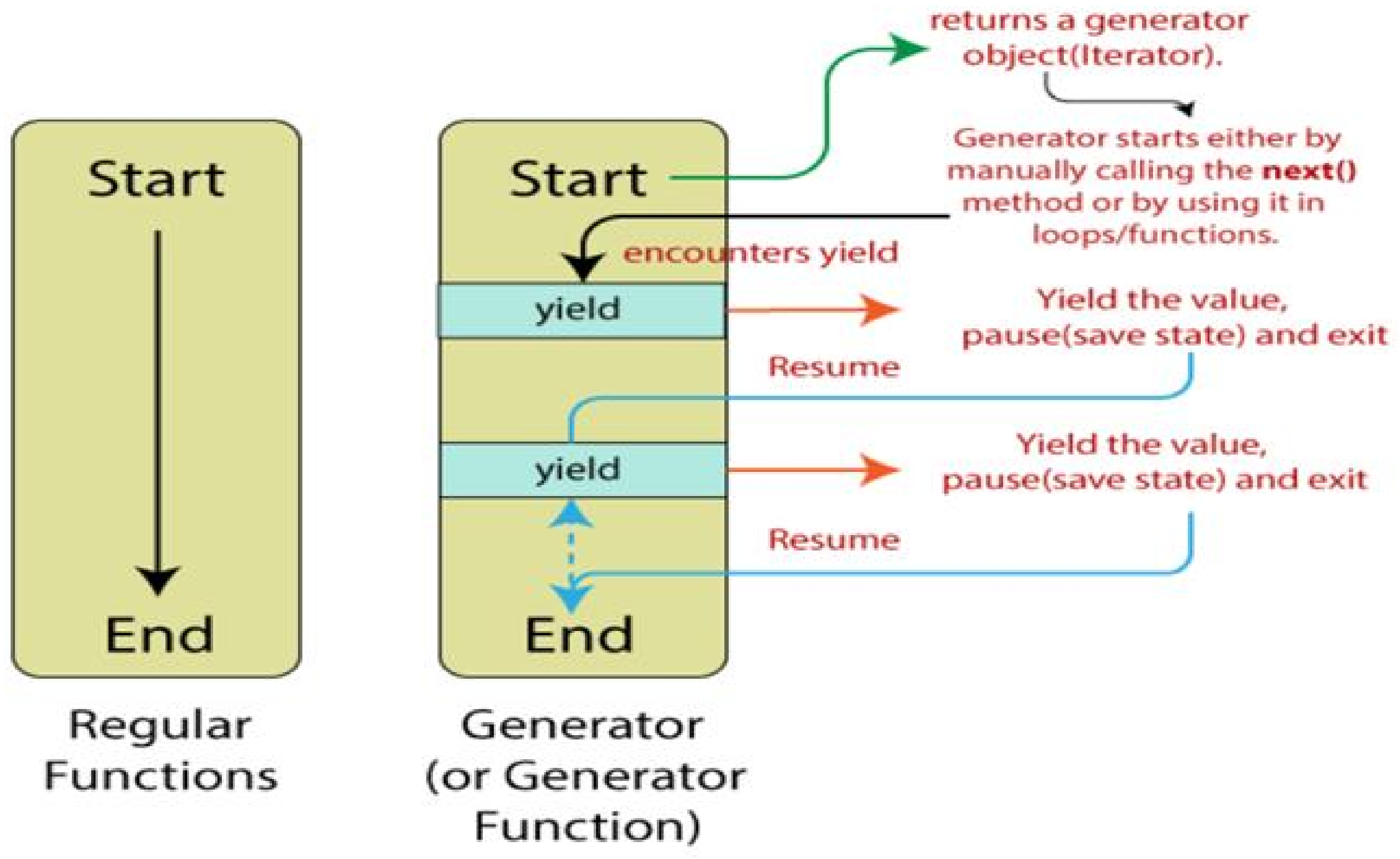
JavaScript Generators

Vijesh M. Nair
Assistant Professor
Dept. of CSE (AI-ML)

Introduction

- Generator functions in Javascript are special functions that can generate a sequence of values.
- Generator Functions in Javascript are used to generate value. Whenever called, they return a Generator Object.
- The generator object returned by the generator function follows the Iterable Protocol of ES6, so it works similarly to iterators.

Introduction



Start
End
Regular Functions
Start
encounters yield
yield
yield
End
Generator (or Generator Function)
returns a generator object(Iterator).
Generator starts either by manually calling the next() method or by using it in loops/functions.
Yield the value, pause(save state) and exit
Resume
Yield the value, pause(save state) and exit
Resume

Introduction

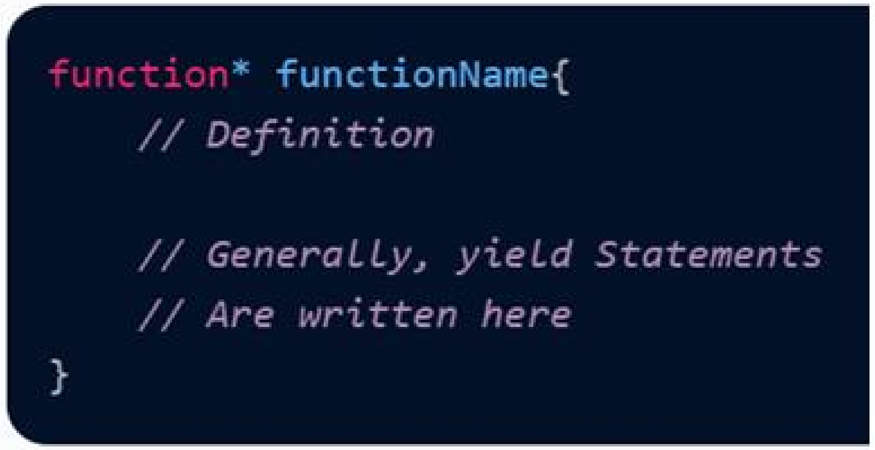
- Calling the *next() method* on the generator object only executes the function till the first yield statement and the yield value is returned to the caller.

- When we repeatedly call the next() method, we can access a sequence of the objects containing two properties;

    - one is the **value**, the value associated with the yield statement, and

    - the other is a boolean flag **done**, to indicate whether there is something remaining in the function to execute or not.

Syntax



```
function* functionName{
    // Definition

    // Generally, yield Statements
    // Are written here
}
```

Occasionally, you will see the asterisk next to the function name, as opposed to the function keyword, such as `function *generatorFunction()`. This works the same, but `function*` is a more widely accepted syntax.

Examples

```
function* mygenfun()    // Valid
{
yield 1;
yield 2;
...
...
}
```

```
function *mygenfun()    // Valid
{
yield 1;
yield 2;
...
...
}
```

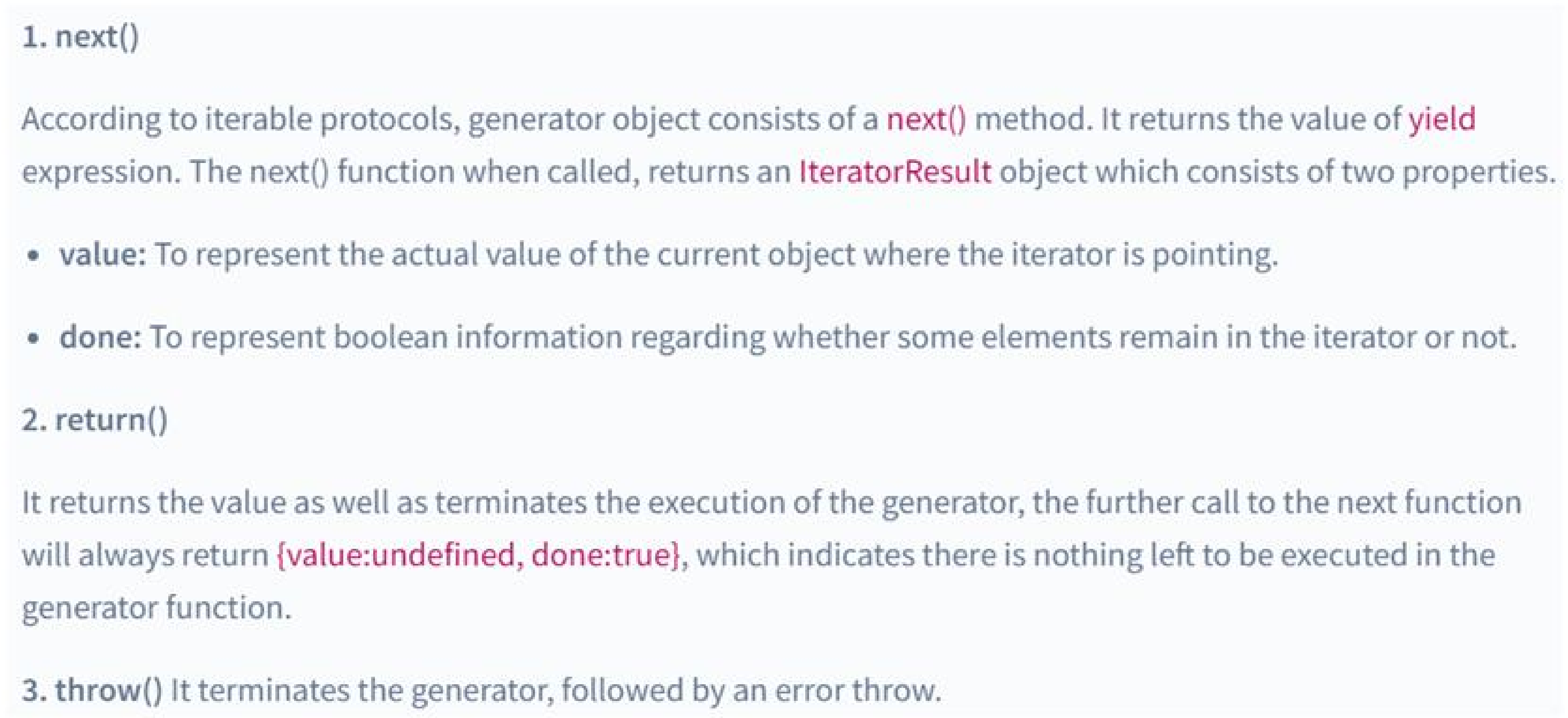
```
function*mygenfun() // Valid
{
yield 1;
yield 2;
...
...
}
```

```
function* gen()
{
yield 100;
yield;
yield 200;
}
// Calling the Generator Function
var mygen = gen();
console.log(mygen.next().value);
console.log(mygen.next().value);
console.log(mygen.next().value);
```

Output

```
100
undefined
200
```

Methods of the Generator Object



1. next()

According to iterable protocols, generator object consists of a next() method. It returns the value of yield expression. The next() function when called, returns an IteratorResult object which consists of two properties.

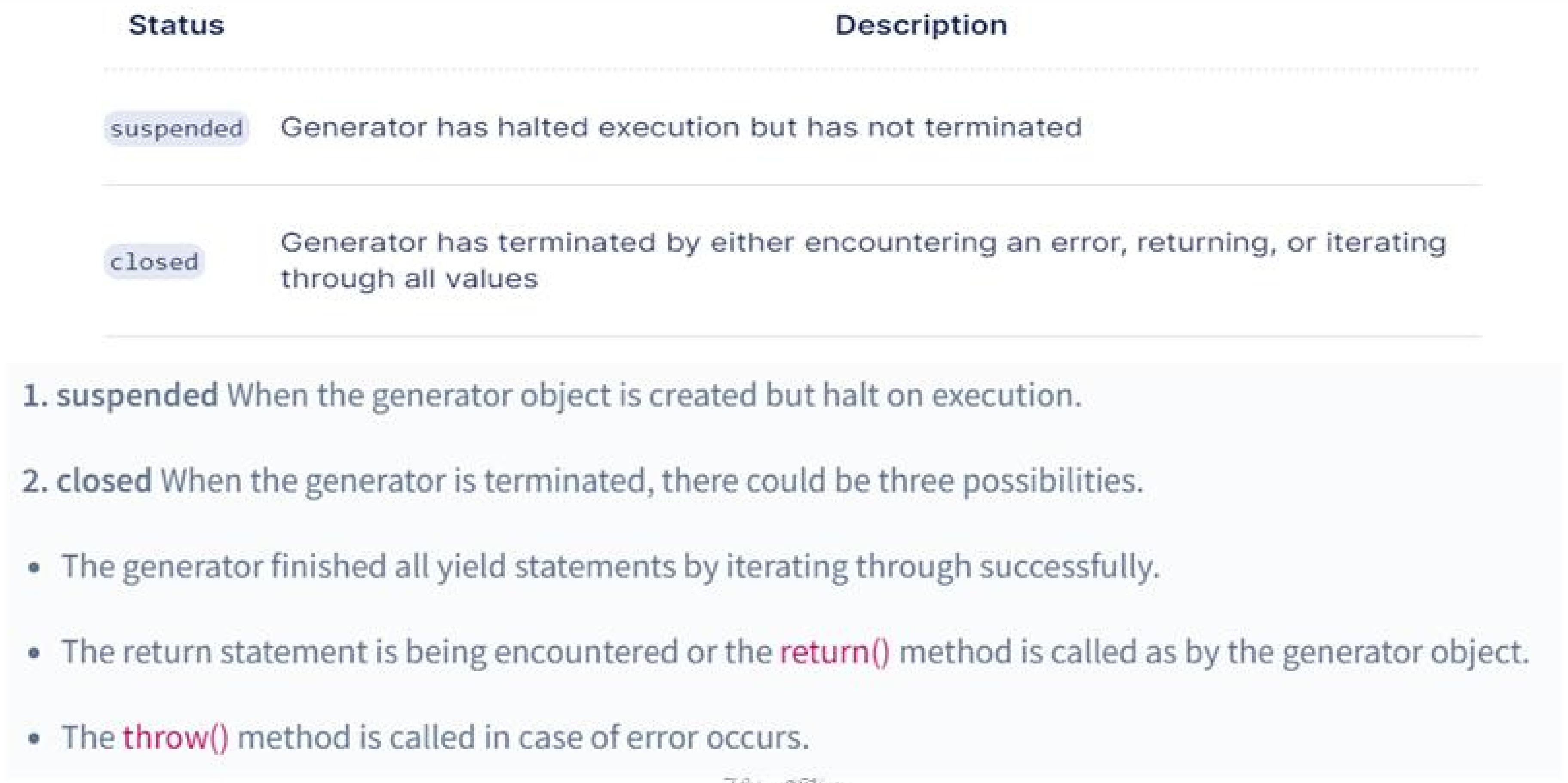
- **value:** To represent the actual value of the current object where the iterator is pointing.
- **done:** To represent boolean information regarding whether some elements remain in the iterator or not.

2. return()

It returns the value as well as terminates the execution of the generator, the further call to the next function will always return {value:undefined, done:true}, which indicates there is nothing left to be executed in the generator function.

3. throw() It terminates the generator, followed by an error throw.

Status of the Generator Object



Status	Description
`suspended`	Generator has halted execution but has not terminated
`closed`	Generator has terminated by either encountering an error, returning, or iterating through all values

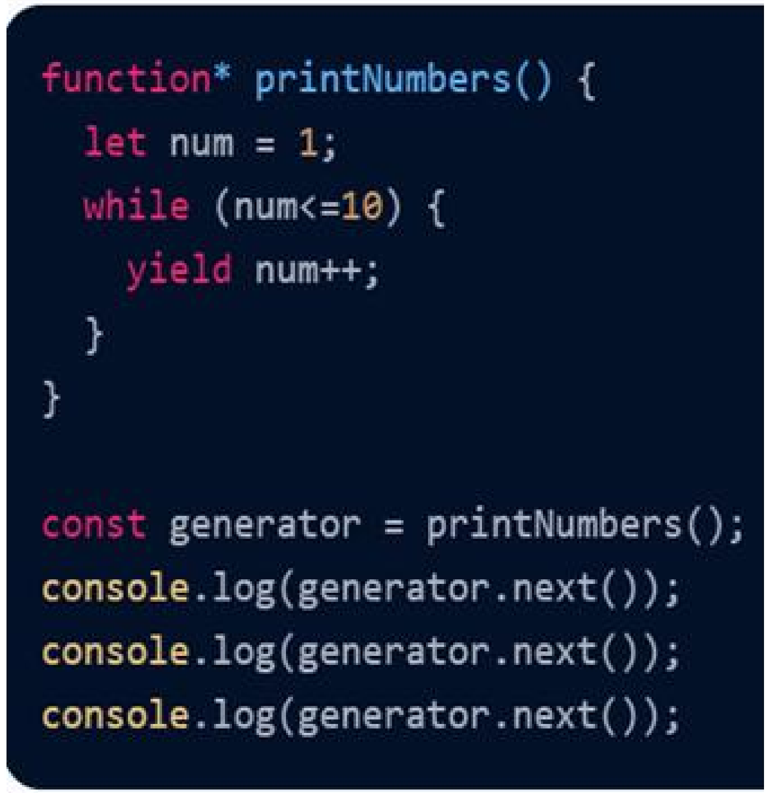
1. **suspended** When the generator object is created but halt on execution.

2. **closed** When the generator is terminated, there could be three possibilities.

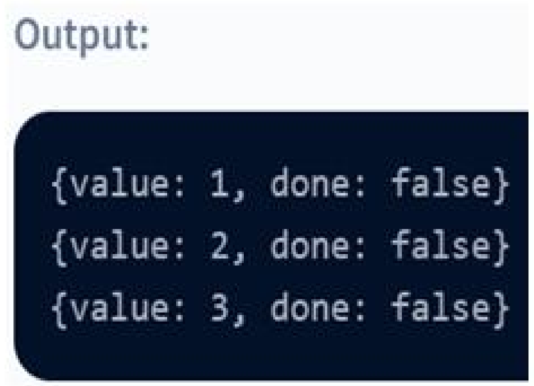
- The generator finished all yield statements by iterating through successfully.
- The return statement is being encountered or the return() method is called as by the generator object.
- The throw() method is called in case of error occurs.

Vijesh Nair

Example: Create a Generator function that can maintain its state and provide us numbers based on auto-increment on subsequent calls to the next() method.



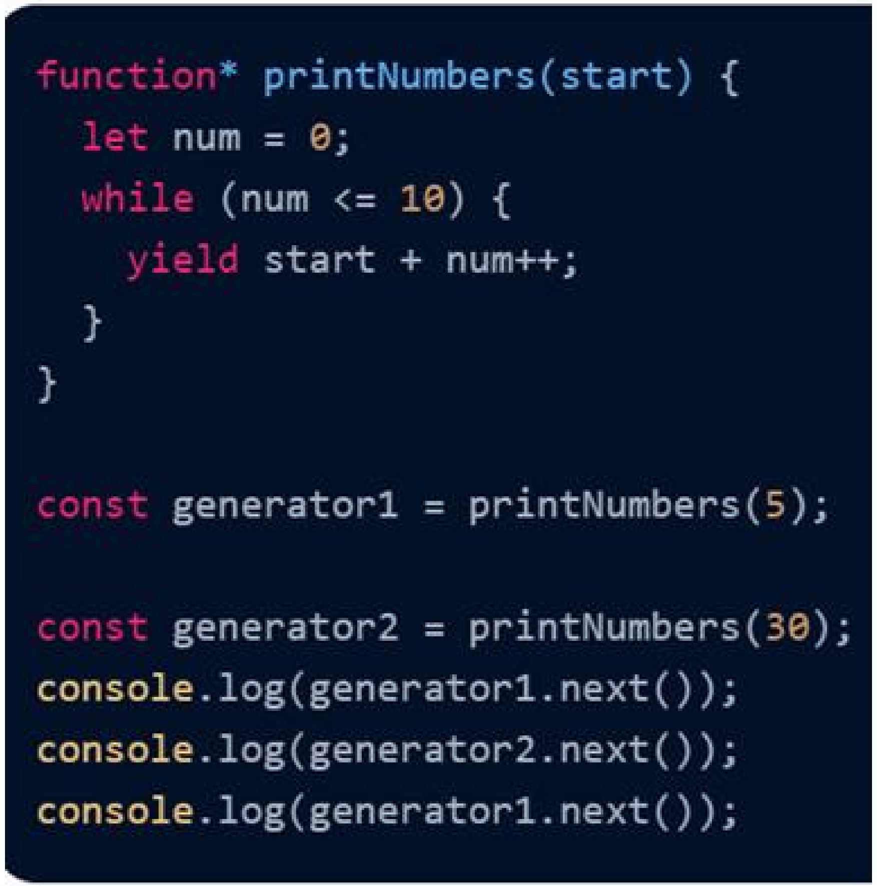
```javascript
function* printNumbers() {
  let num = 1;
  while (num<=10) {
    yield num++;
  }
}

const generator = printNumbers();
console.log(generator.next());
console.log(generator.next());
console.log(generator.next());
```



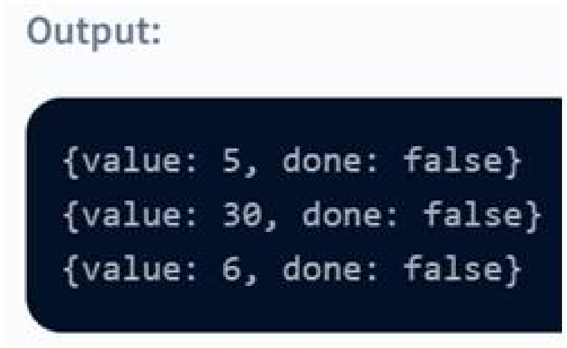
Output:

```
{value: 1, done: false}
{value: 2, done: false}
{value: 3, done: false}
```

Example: Passing Arguments into Generators



```
function* printNumbers(start) {
  let num = 0;
  while (num <= 10) {
    yield start + num++;
  }
}

const generator1 = printNumbers(5);

const generator2 = printNumbers(30);
console.log(generator1.next());
console.log(generator2.next());
console.log(generator1.next());
```



Output:

```
{value: 5, done: false}
{value: 30, done: false}
{value: 6, done: false}
```

Vijesh Nair

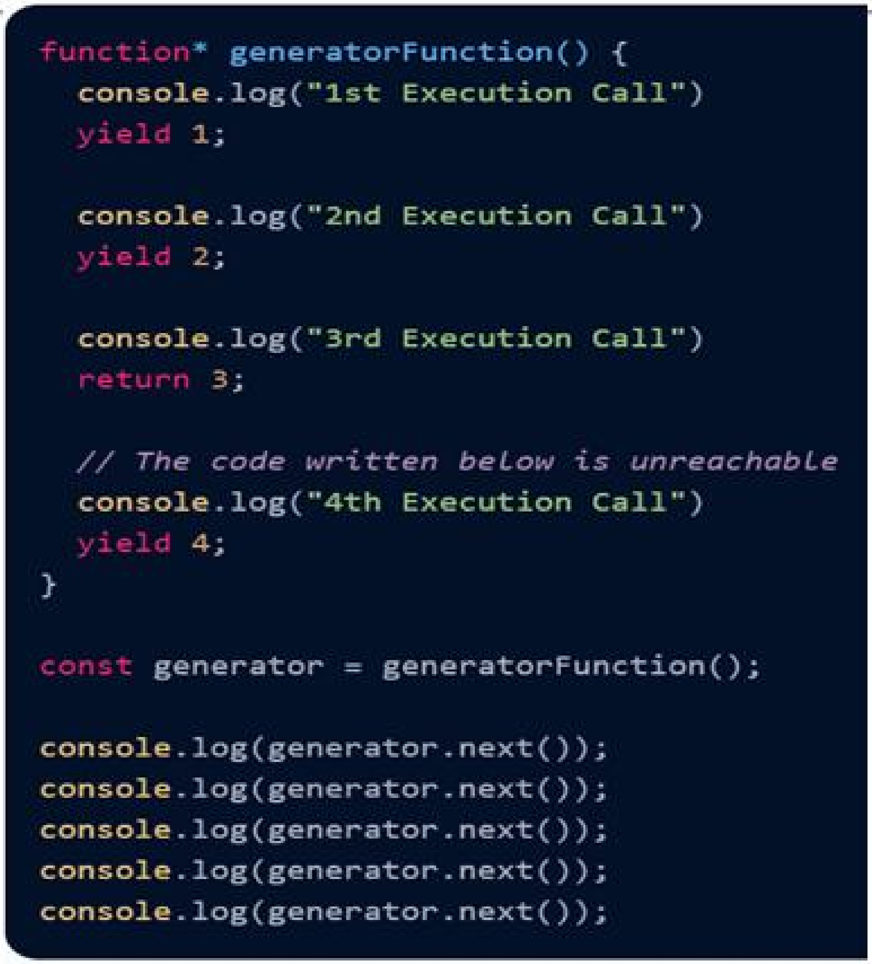
Return Statement in a Generator

- Whenever the generator function contains a return statement, the next() method can only execute the code written before that and only the yield statement written before the return statement will be executed.

- As soon as all yield statements execute and the next() encounters a return statement, the done boolean property is set to true.

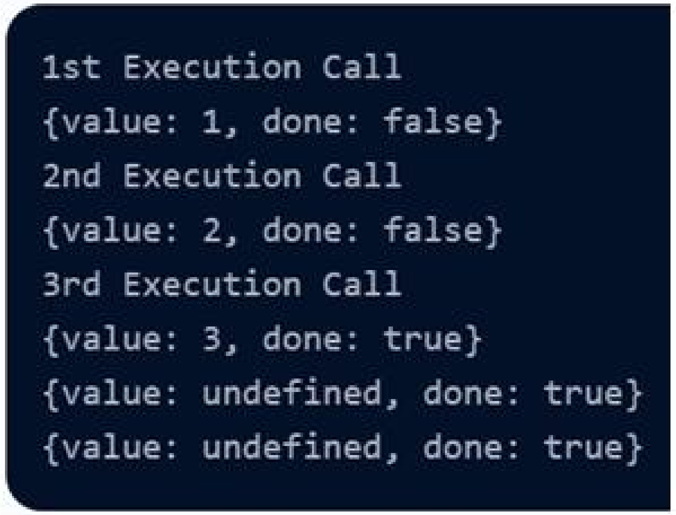
- A further call to the next method will not execute the code of the generator but always return the value as undefined and done as true.

Example



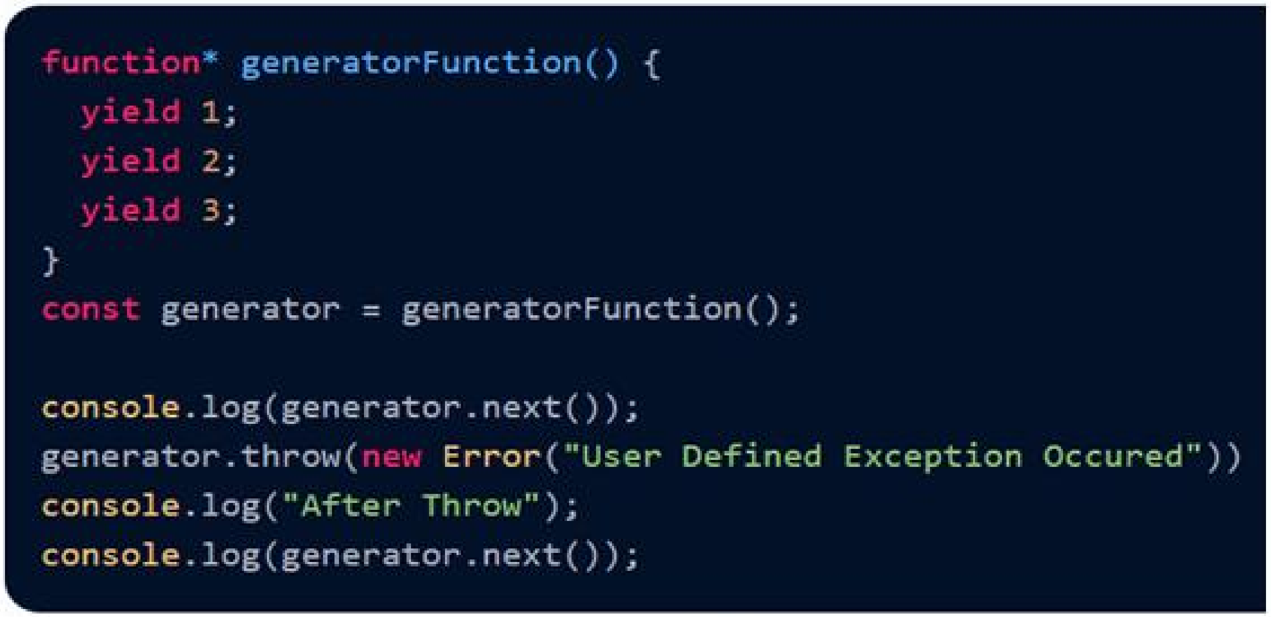
```
function* generatorFunction() {
  console.log("1st Execution Call")
  yield 1;

  console.log("2nd Execution Call")
  yield 2;

  console.log("3rd Execution Call")
  return 3;

  // The code written below is unreachable
  console.log("4th Execution Call")
  yield 4;
}

const generator = generatorFunction();

console.log(generator.next());
console.log(generator.next());
console.log(generator.next());
console.log(generator.next());
console.log(generator.next());
```

Output:



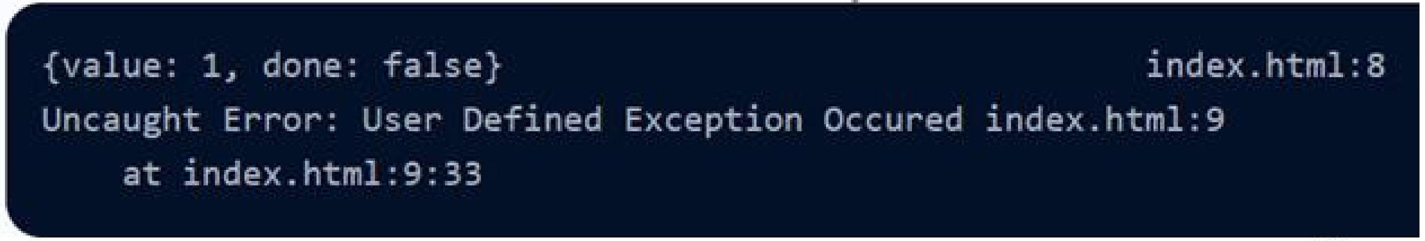
```
1st Execution Call
{value: 1, done: false}
2nd Execution Call
{value: 2, done: false}
3rd Execution Call
{value: 3, done: true}
{value: undefined, done: true}
{value: undefined, done: true}
```

Throw an Exception from Generator Object



```javascript
function* generatorFunction() {
  yield 1;
  yield 2;
  yield 3;
}
const generator = generatorFunction();

console.log(generator.next());
generator.throw(new Error("User Defined Exception Occured"))
console.log("After Throw");
console.log(generator.next());
```

Output:



```
{value: 1, done: false}                                   index.html:8
Uncaught Error: User Defined Exception Occured index.html:9
    at index.html:9:33
```

Thank You!