

Node.js - Callbacks Concept

What is Callback?

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Blocking Code Example

Create a text file named **input.txt** with the following content –

```
Tutorials Point is giving self learning content  
to teach the world in simple and easy way!!!!
```

Create a js file named **main.js** with the following code –

```
var fs = require("fs");  
var data = fs.readFileSync('input.txt');  
  
console.log(data.toString());  
console.log("Program Ended");
```

Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Tutorials Point is giving self learning content  
to teach the world in simple and easy way!!!!
```

Non-Blocking Code Example

Create a text file named input.txt with the following content.

```
Tutorials Point is giving self learning content  
to teach the world in simple and easy way!!!!
```

Update main.js to have the following code –

```
var fs = require("fs");  
  
fs.readFile('input.txt', function (err, data) {  
  if (err) return console.error(err);  
  console.log(data.toString());  
});  
  
console.log("Program Ended");
```

Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Program Ended  
Tutorials Point is giving self learning content  
to teach the world in simple and easy way!!!!
```

These two examples explain the concept of blocking and non-blocking calls.

The first example shows that the program blocks until it reads the file and then only it proceeds to end the program.

The second example shows that the program does not wait for file reading and proceeds to print "Program Ended" and at the same time, the program without blocking continues reading the file.

Thus, a blocking program executes very much in sequence. From the programming point of view, it is easier to implement the logic but non-blocking programs do not execute in sequence. In case a program needs to use any data to be processed, it should be kept within the same block to make it sequential execution.