



## Backend Tutorial: Blocking vs Non-Blocking execution | Web Development Tu...



Show Course Contents (+)

[Overview](#) [Q&A](#) [Downloads](#) [Announcements](#)

## Backend Tutorial: Blocking vs Non-Blocking execution | Web Development Tutorials #65

In the last tutorial, we have discussed the basics of Node.Js and how to run programs with the help of modules. Now, we will talk about, what are Blocking and Non-Blocking codes in Node.Js. Make a new file as *tut65.js*.

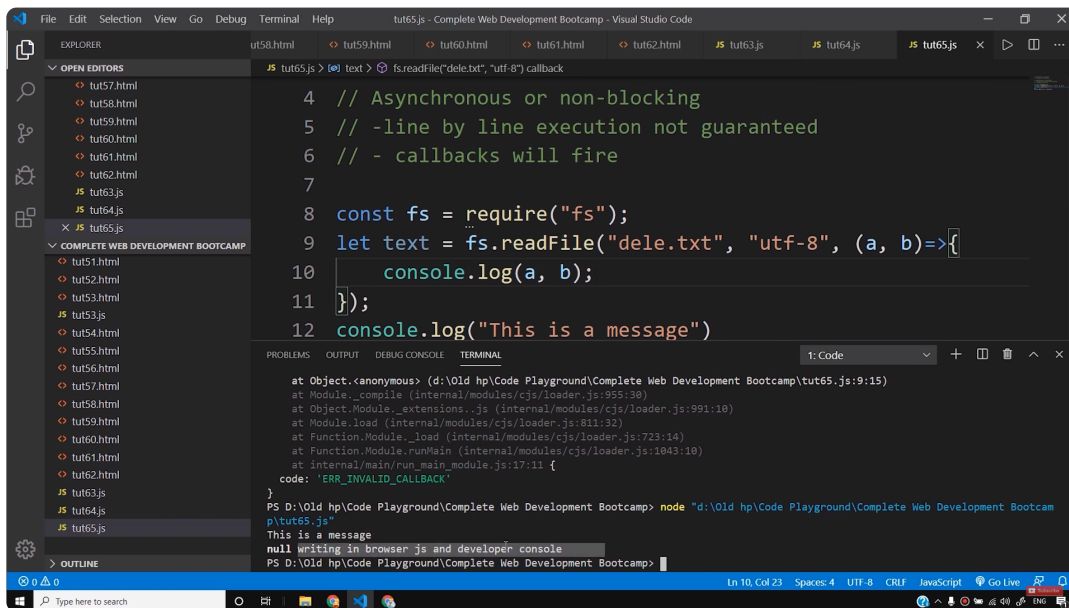
The code which runs via line by line execution is known as **synchronous** or **blocking code**. It means the line of code written first, will be executed first. On the other hand, a block of code where line by line execution is not guaranteed is known as **asynchronous** or **non-blocking** code. These types of codes accept a call-back function.

In the last tutorial, we made a file to read the contents with the help of **fs.readFileSync()** function. The *Sync* here stands for synchronous execution. Let us modify the same example and made the code only to read the file as follows-

```
const fs = require("fs");
fs.readFile("dele.txt", "utf-8", (a, b)=>{
  console.log(a, b);
});
```

```
});
console.log("This is a message");
```

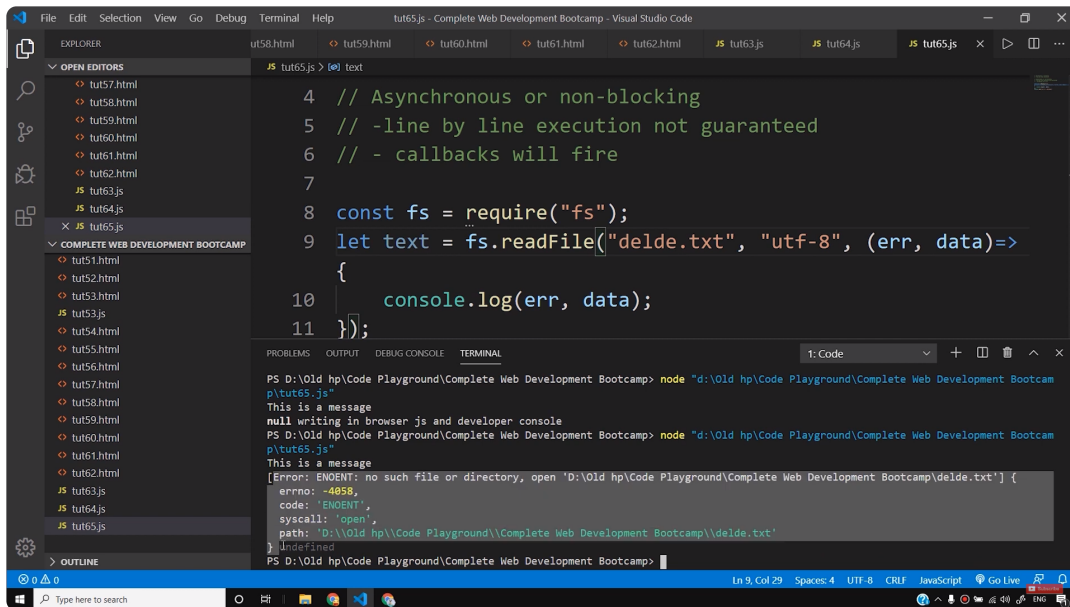
After executing the above code, we see the output as follows-



In the above example, *a* is null and *b* is the content of the file. Or rather, if go according to the official website of Node.js, then **a** can be called as **error** and **b** can be called as the data. So the final code now goes like this-

```
const fs = require("fs");
fs.readFile("dele.txt", "utf-8", (err, data)=>{
  console.log(err, data);
});
console.log("This is a message");
```

If we give the wrong file name apart from *dele.txt*, then the pops out as follows-



```
4 // Asynchronous or non-blocking
5 // -line by line execution not guaranteed
6 // - callbacks will fire
7
8 const fs = require("fs");
9 let text = fs.readFile("delde.txt", "utf-8", (err, data)=>
10 {
11   console.log(err, data);
12 });
```

PS D:\Old hp\Code Playground\Complete Web Development Bootcamp> node "d:\Old hp\Code Playground\Complete Web Development Bootcamp\tut65.js"

This is a message

null writing in browser js and developer console

PS D:\Old hp\Code Playground\Complete Web Development Bootcamp> node "d:\Old hp\Code Playground\Complete Web Development Bootcamp\tut65.js"

This is a message

[Error: ENOENT: no such file or directory, open 'D:\Old hp\Code Playground\Complete Web Development Bootcamp\delde.txt'] {

errno: -4055,

code: 'ENOENT',

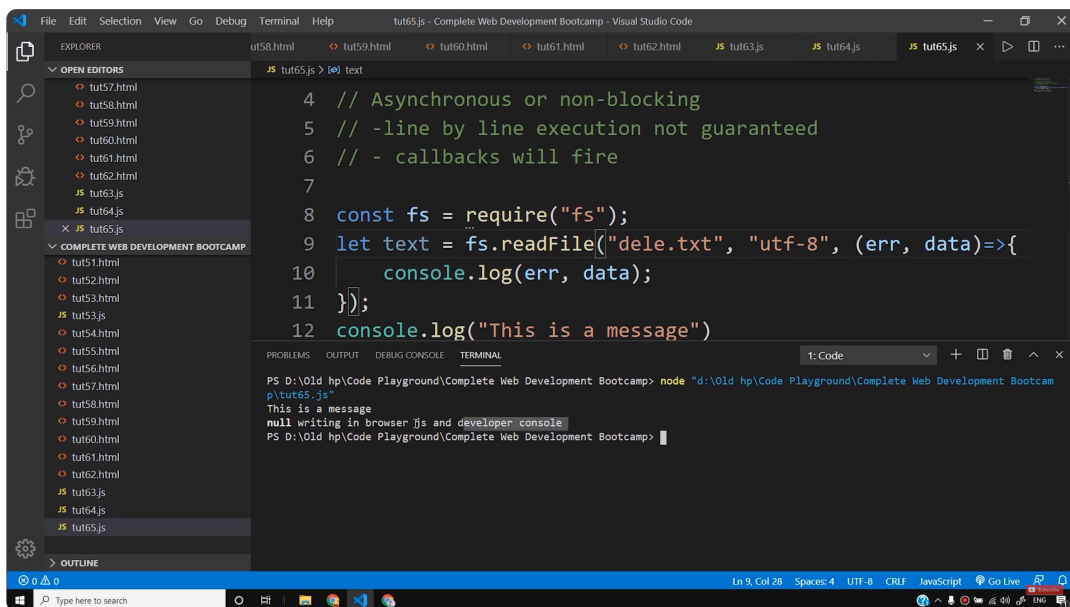
syscall: 'open',

path: 'D:\Old hp\Code Playground\Complete Web Development Bootcamp\delde.txt'

} Undefined

PS D:\Old hp\Code Playground\Complete Web Development Bootcamp>

The main thing to notice after executing the above code is, we are getting the output *"This is the message"* first, and then the callback function is returning the output as follows–



```
4 // Asynchronous or non-blocking
5 // -line by line execution not guaranteed
6 // - callbacks will fire
7
8 const fs = require("fs");
9 let text = fs.readFile("dele.txt", "utf-8", (err, data)={
10   console.log(err, data);
11 });
12 console.log("This is a message")
```

PS D:\Old hp\Code Playground\Complete Web Development Bootcamp> node "d:\Old hp\Code Playground\Complete Web Development Bootcamp\tut65.js"

This is a message

null writing in browser js and developer console

PS D:\Old hp\Code Playground\Complete Web Development Bootcamp>

The reason for this is, it is an asynchronous function. It allows the **readFile()** function to read the file completely. By the time the code is reading the file, the next block of code is executed. And when the reading is completed, it then prints the *data*. It does not mean that the code is not executed line by line. The code is executed synchronously but the output we get depends upon the time taken while reading the file. An asynchronous function is used here because it does not allow the code to **block** the user. If we use the asynchronous function here, then the file will be first read and then the next line will be executed.

So I hope you must have got an idea of blocking and non-blocking concepts used in Node.js. However, it is not a very important topic but it should be known to you.

## Code as described/written in the video

```
// Synchronous or blocking
// - line by line execution

// Asynchronous or non-blocking
// - line by line execution not guaranteed
// - callbacks will fire

const fs = require("fs");
fs.readFile("dele.txt", "utf-8", (err, data)=>{
  console.log(data);
});
console.log("This is a message");
```

[< < Previous](#)[Next > >](#)

CodeWithHarry

Copyright © 2023 CodeWithHarry.com

