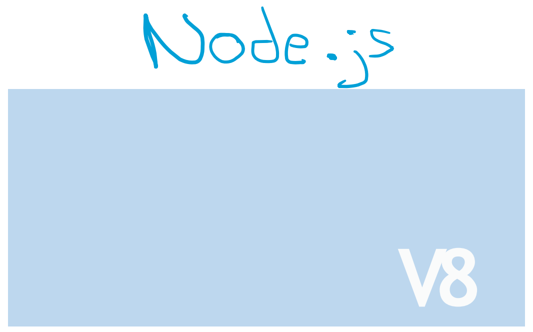
Node.js Modules

Node.js

- Είπαμε ότι το Node.js αποτελεί ένα **περιβάλλον εκτέλεσης JavaScript** βασισμένο στην ανοιχτού κώδικα v8 Javascript engine της Google.

  (asynchronous event-driven JavaScript runtime environment)



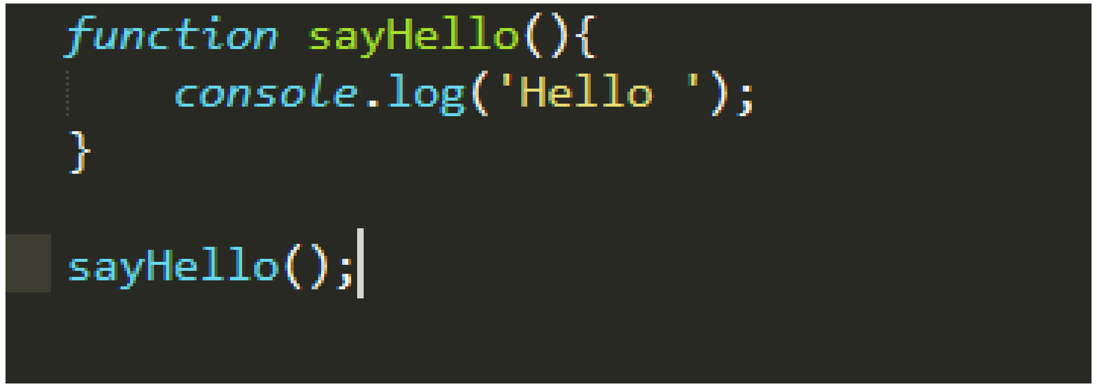
Node.js is like a container where javascript can be executed. Outside of any browser.

Node.js

- Nodejs είναι βέβαια κάτι πολύ παραπάνω από ένα «δοχείο» εκτέλεσης της V8 engine
- To Nodejs μας δίνει περισσότερες δυνατότητες από αυτές που μας έδινε η javascript στον browser
- Ανάγνωση/δημιουργία εγγραφή σε αρχεία, σύνδεση σε βάση δεδομένων, δημιουργία server κλπ.
- Το Node.js μας παρέχει μια πληθώρα από JavaScript modules που απλοποιούν την ανάπτυξη εφαρμογών
- More about V8 engine: https://v8.dev/

Lets see node.js in action

- Lets create a javascript file, lets say app.js



```
function sayHello(){
    console.log('Hello ');
}

sayHello();
```

Lets see node.js in action

- In normal javascript in the browser we would include the aforementioned file (or code) in an html file and then open up the html file in the browser…

- With node js in the terminal we run the following code:

- node app.js

Lets see node.js in action

- Node.js allows us to do things that we are not able to do with the browser: ie. read & write to files

Node.js modules

- Node.js is built around this concept of Modules

- **Modules** in Node.js can be considered to be the same as JavaScript libraries.
  - ➢A **set of functions** you want to include in your application.

- Node.js comes with a set of built-in modules (no further installation required)

Node.js modules

- include a module-**> require()** function + **name** of module

Examples:

Node.js includes an **HTTP module** that allows it to transfer data over the Hyper Text Transfer Protocol (HTTP is an application protocol, is the set of rules for transferring files -- such as text, images, sound, video, and other multimedia files over the web)

to **access the HTTP module** & create a server:

- var http = require('http’);

Node.js modules

- include a module-**> require()** function + **name** of module

fs module enables interacting with the file system

- var filesystem=require(‘fs’);

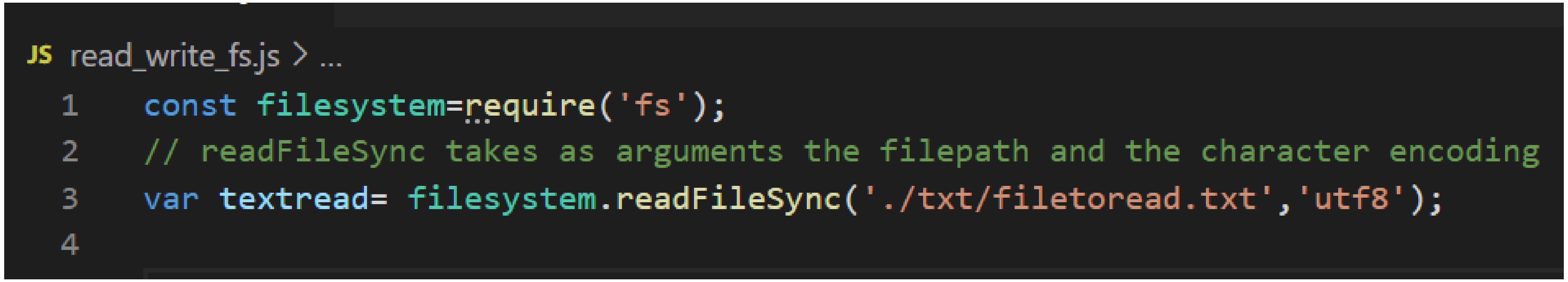
Node.js modules

- The **variable** stores the **result** of the module.

- That way we **gain access to different functions** based on the module used.

- Actually, ***require()*** function returns an **object** that has a number of **methods** that we can easily access

Node.js modules ie.



read_write_fs.js > ...

```javascript
const filesystem=require('fs');
// readFileSync takes as arguments the filepath and the character encoding
var textread= filesystem.readFileSync('./txt/filetoread.txt','utf8');

```

fs.readFileSync() method is used to :

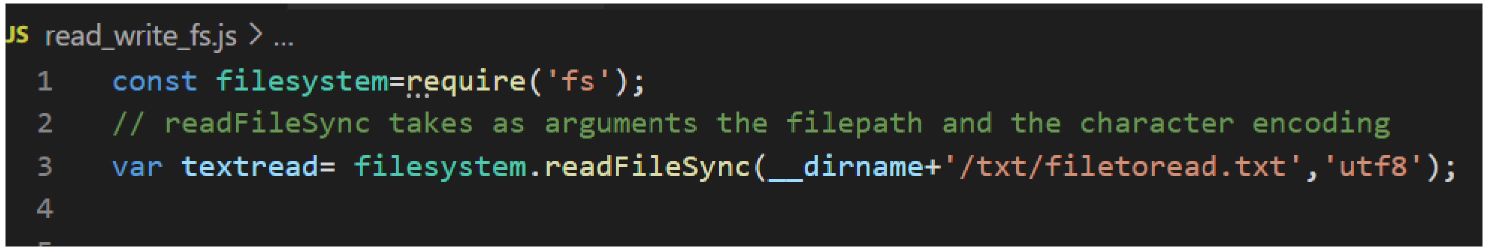
- read files and return their contents
- read files **synchronously**, instructing node.js to halt all concurrent processes
  - Thus the original node program **pauses** while it waits for the fs.readFileSync() function to complete. Once it does, remaining node program is executed.

Note that…

In general, files and directories maintain a tree structure for easy access.

There are 2 ways of getting the current directory in Node.js.

- **__dirname ->** returns the path of the folder where the **current JavaScript file resides**



read_write_fs.js > ...

```javascript
const filesystem=require('fs');
// readFileSync takes as arguments the filepath and the character encoding
var textread= filesystem.readFileSync(__dirname+'/txt/filetoread.txt','utf8');

```

Note that...

In general, files and directories maintain a tree structure for easy access.

There are 2 ways of getting current directory in Node.js .

- **./   ->** gives the **current working directory**. Current working directory is the path of the folder where the **node command** is executed and may change during the execution of the script.

- If for example we run node from the desktop, ./ will refer to the desktop.

Note that…

- The **only case** when ./ gives the **path of the currently executing file** is when it is used with the **require() command**

- Both __dirname and ./ give similar results when the node is running in the same directory as the currently executing file but produce different results when node.js run from some other directory.

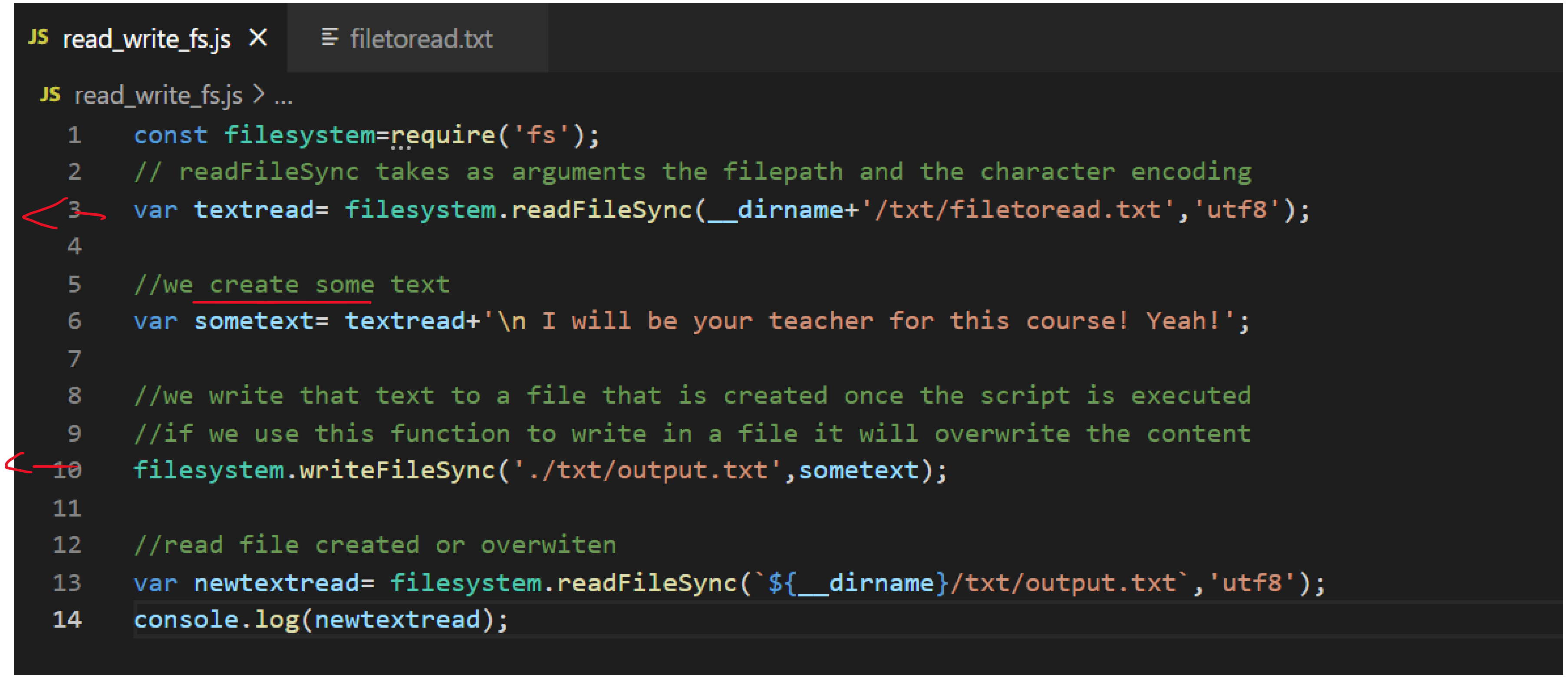
Lets see an example

- Lets see how we can read and write files with node.js

Code example

Read a file

Create or overwrite file



```
const filesystem=require('fs');
// readFileSync takes as arguments the filepath and the character encoding
var textread= filesystem.readFileSync(__dirname+'/txt/filetoread.txt','utf8');

//we create some text
var sometext= textread+'\n I will be your teacher for this course! Yeah!';

//we write that text to a file that is created once the script is executed
//if we use this function to write in a file it will overwrite the content
filesystem.writeFileSync('./txt/output.txt',sometext);

//read file created or overwiten
var newtextread= filesystem.readFileSync(`${__dirname}/txt/output.txt`,'utf8');
console.log(newtextread);
```

Create our own modules

- We may easily create and incorporate our own modules into our apps
- In order to do so we are going to use **module.exports**
- module.exports ->is used to export any var, function or object as a module

Create our own modules

- module.exports actually is a special object that is included in every JavaScript file in the Node.js application.

- It is an empty object by default, and its primary purpose is to define what **should be exported from a particular module**.

- When you create a module in Node.js, you can use module.exports to expose variables, functions, or objects from that module, making them accessible to other modules that require or import the module.

module.exports vs exports

module.exports:

- actual object returned when you require a module in another file.
- If you assign a new value to module.exports, it will **completely replace** the existing exports and become the **new exports object**.
- You can use module.exports to directly export a single object, function, or primitive value.

exports:

- It is a shorthand notation for module.exports.
- If you assign a new value to exports, it will **add properties to the existing module.exports object**.
- However, directly assign a new value to exports and expect it to replace the entire exports object.

To be continued...