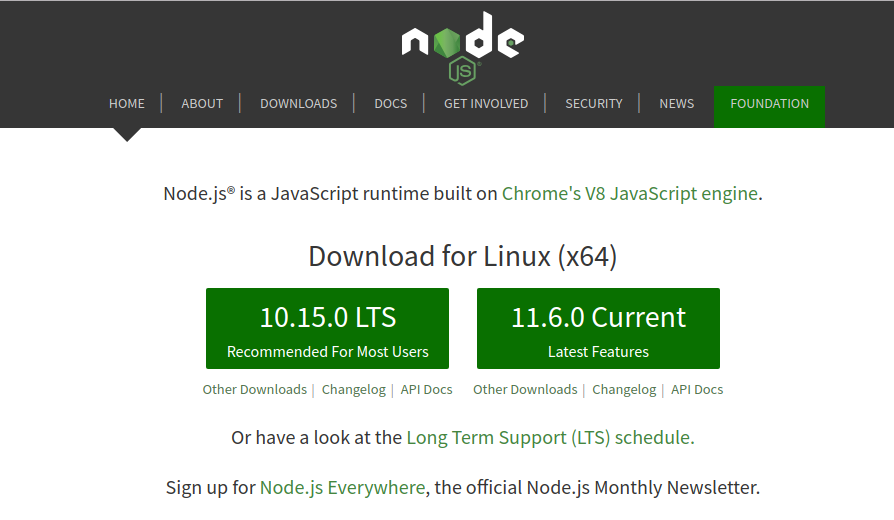
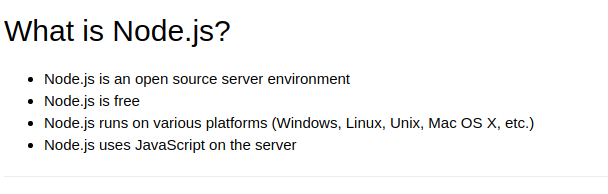
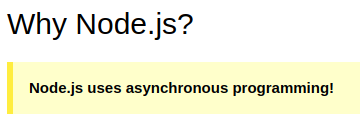
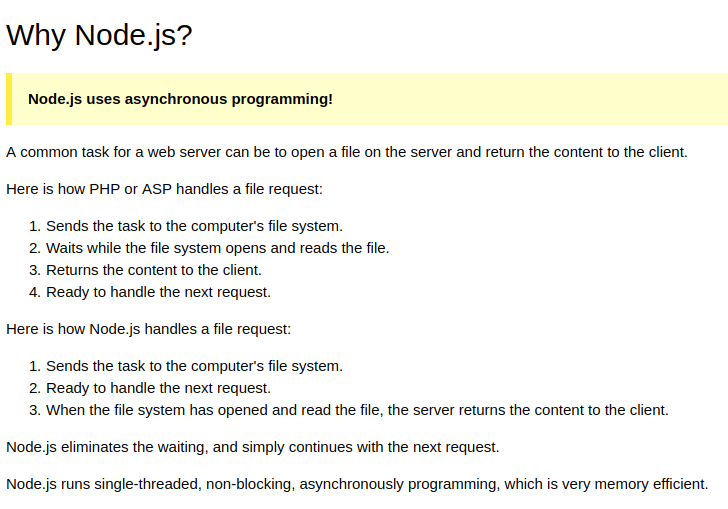


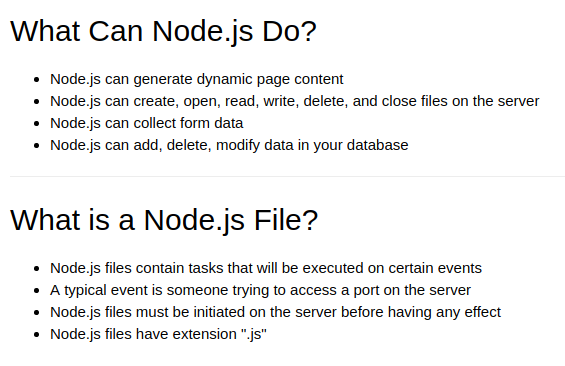
https://nodejs.org/en/



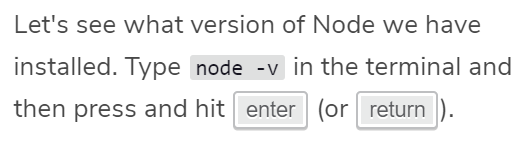




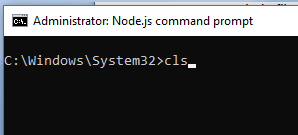


……………………………………………………………………….

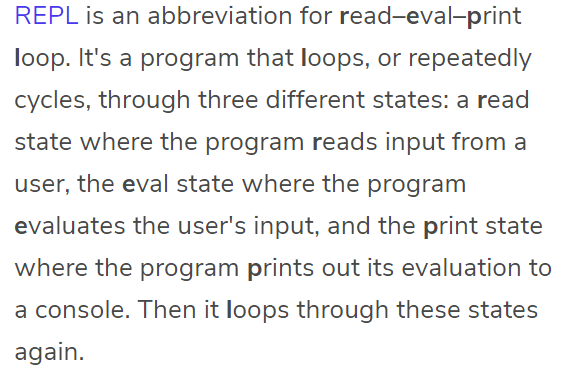
Node js version

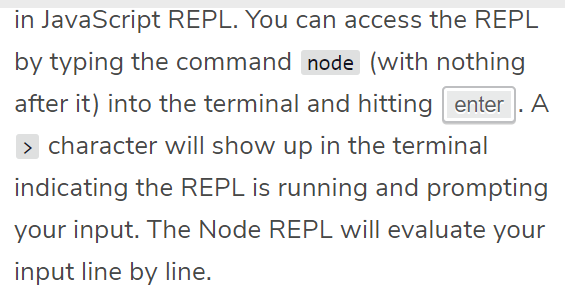


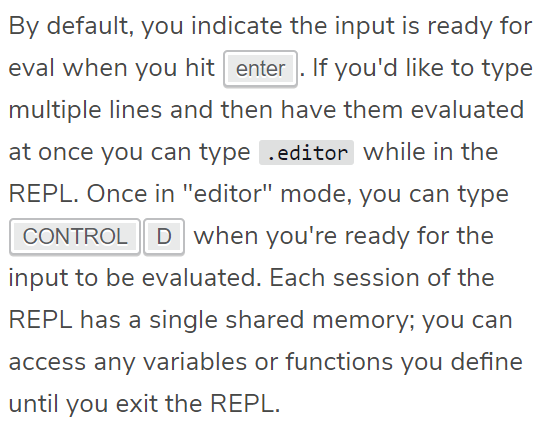
cls to clear terminal via windows

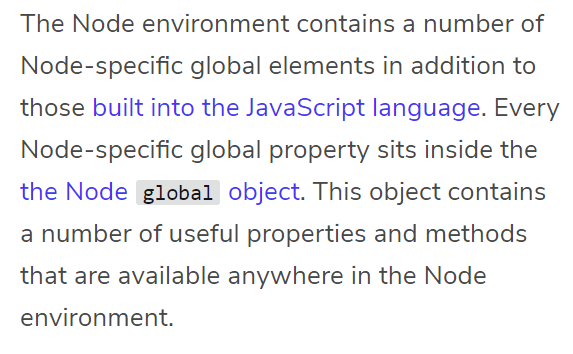


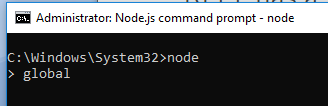
The Node REPL

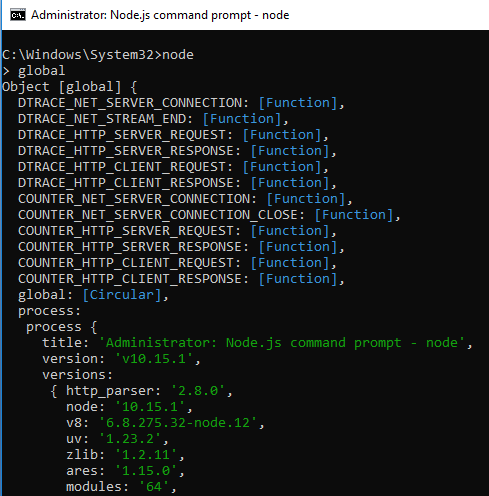


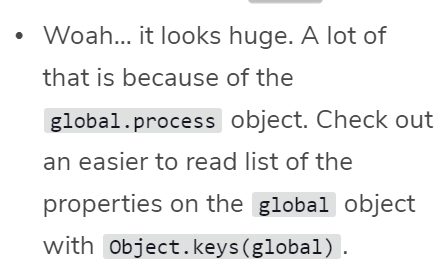


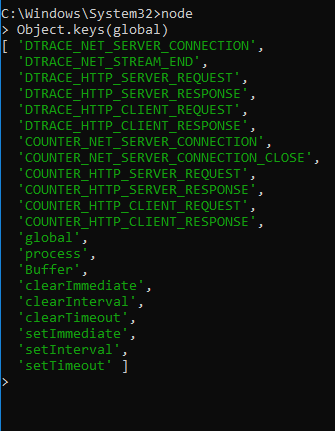


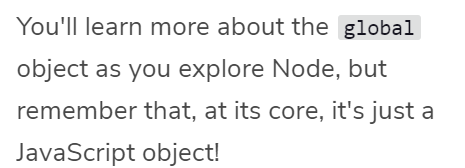






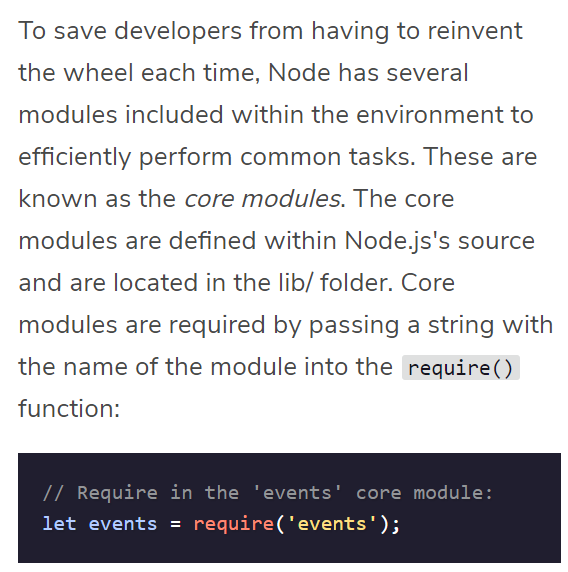




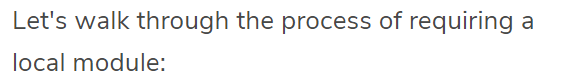


# Core Modules and Local Modules

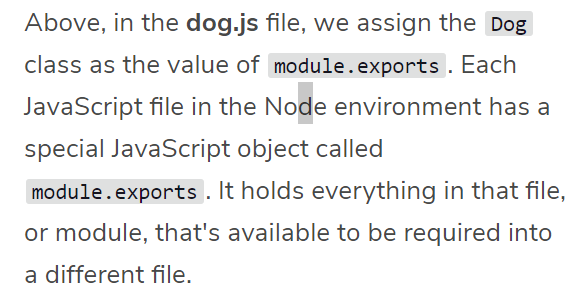
Essentially, a module is a collection of code located in a file. Instead of having an entire program located in a single file, code is organized into separate files and combined through requiring them where needed using the require() function.



the require()function includes some interesting logic "under the hood." The require() function will first check to if its argument is a core module, if not, it will move on to different attempts to locate it.

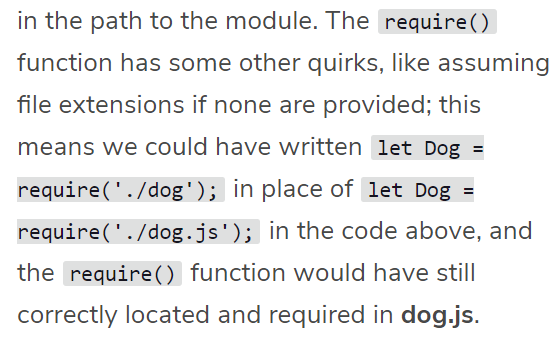


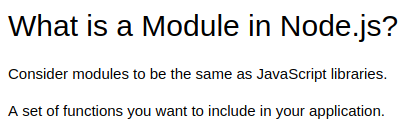


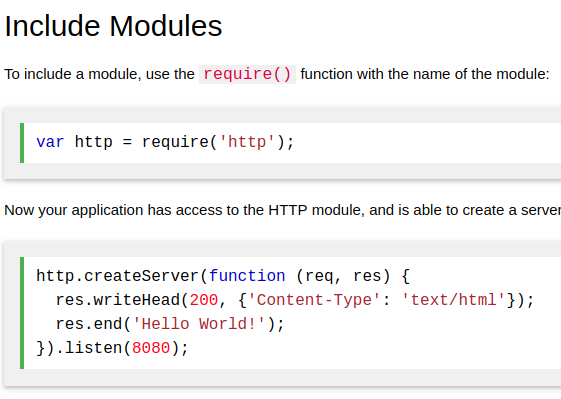


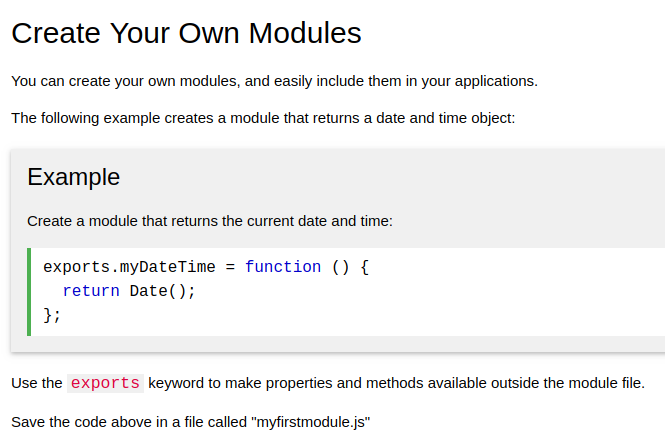


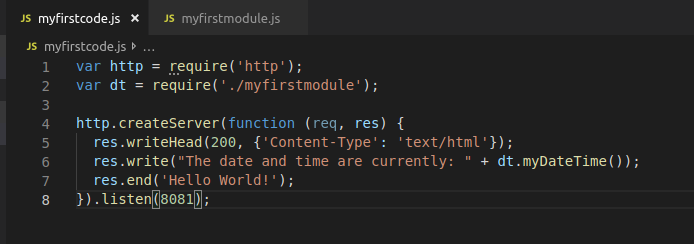
Note:







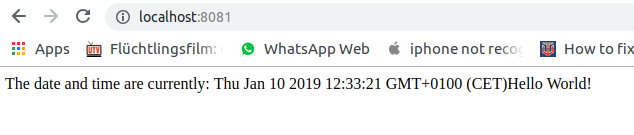


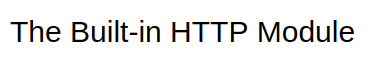


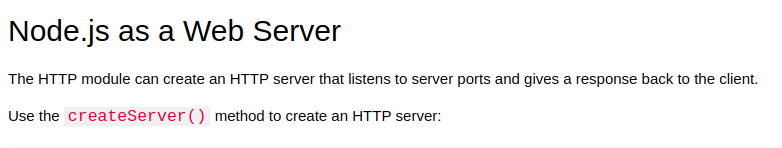
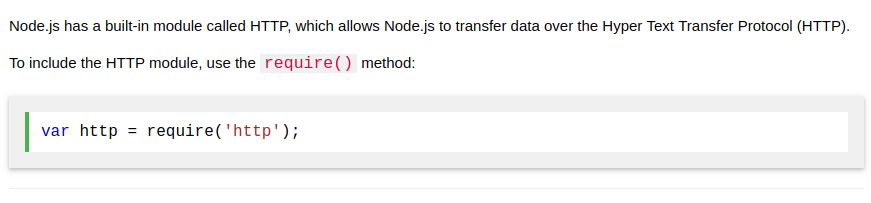


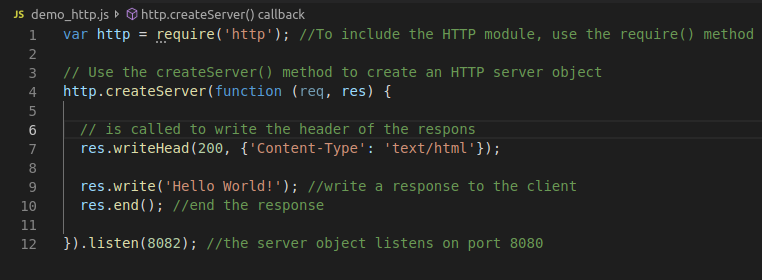
[http://localhost:8080](http://localhost:8080/)

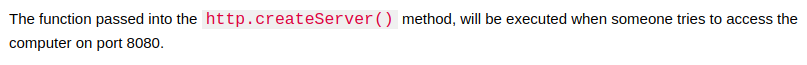


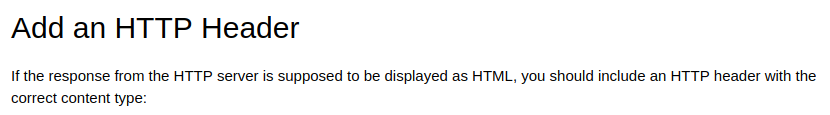


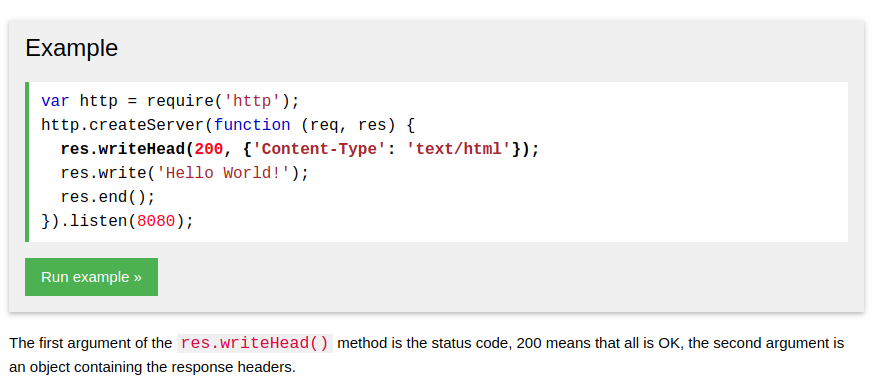


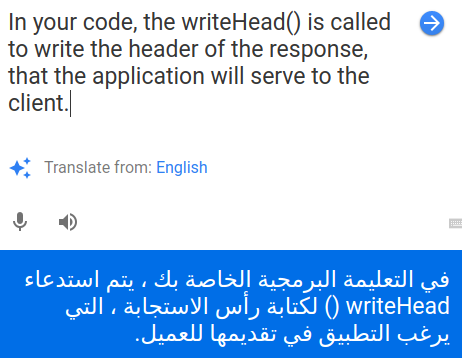


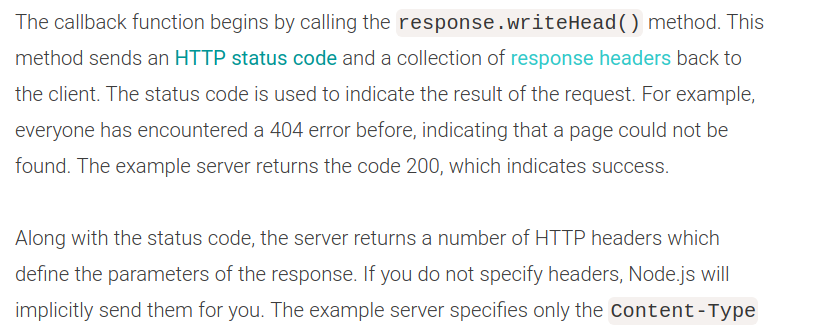


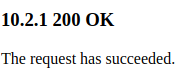


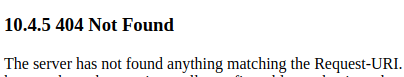
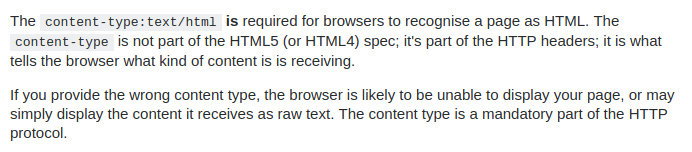








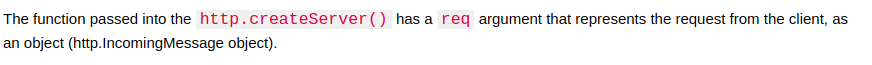






query=question=inquiry=request

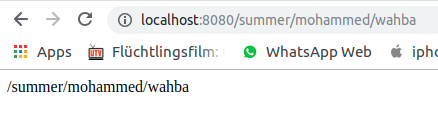
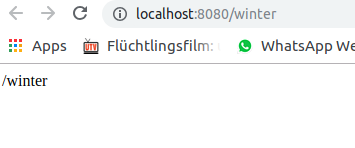






ex:





To update node.js

<https://github.com/coreybutler/nvm-windows/releases>

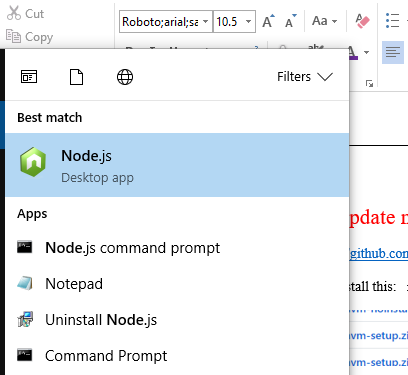
and install this: nvm-setup.zip



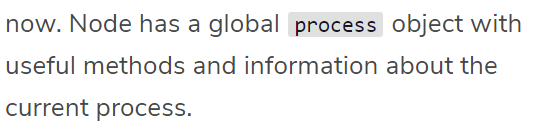
REPL Read Evaluate Print Loop

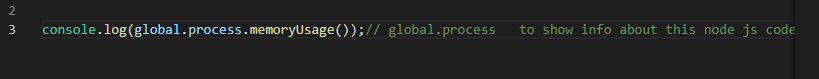
To execute the code of js (node js).

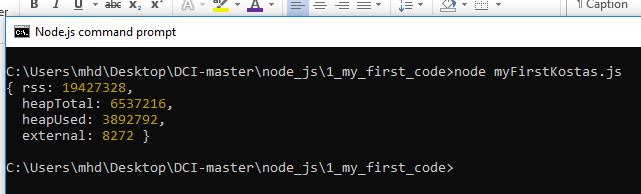
To test the code, like console log in Browser .



global.process

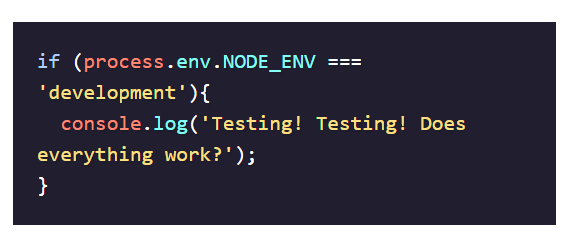






In computer science, a process is the instance of a computer program that is being executed. You can open Task Manager if you're on a Windows machine or Activity Monitor from a Mac to see information about the various processes running on your computer right now. Node has a global process object with useful methods and information about the current process.

The process.env property is an object which stores and controls information about the environment in which the process is currently running. For example, the process.env object contains a PWD property which holds a string with the directory in which the current process is located. It can be useful to have some if/else logic in a program depending on the current environment— a web application in a development phase might perform different tasks than when it's live to users. We could store this information on the process.env. One convention is to add a property to process.env with the key NODE\_ENV and a value of either production or development.



The process.memoryUsage() returns information on the CPU demands of the current process. It returns a property that looks similar to this:

{ rss: 26247168,

heapTotal: 5767168,

heapUsed: 3573032,

external: 8772 }

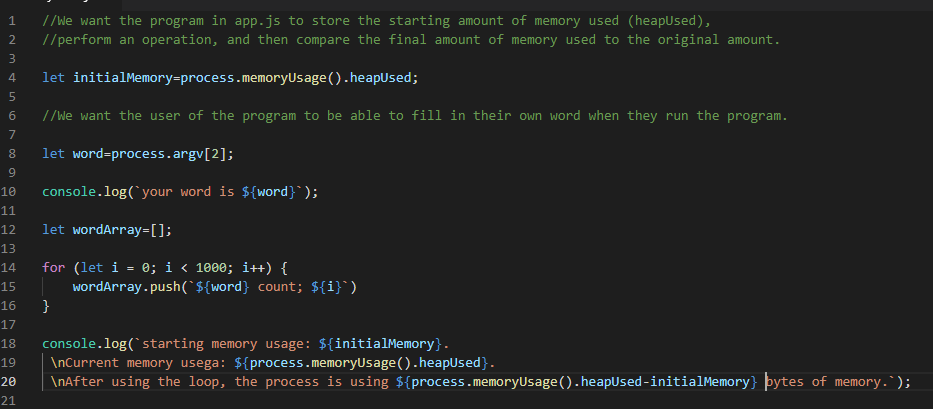
Heap can mean different things in different contexts: a heap can refer to [a specific data structure](https://en.wikipedia.org/wiki/Heap_(data_structure)), but it can also refer to the a block of [computer memory](https://en.wikipedia.org/wiki/Memory_management). process.memoryUsage().heapUsed will return a number representing how many bytes of memory the current process is using.

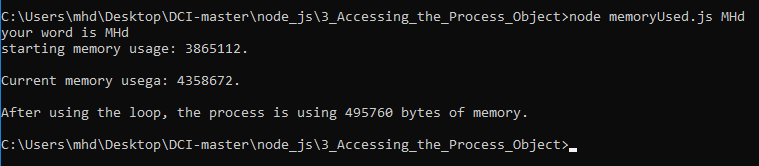
The process.argv property holds an array of command line values provided when the current process was initiated. The first element in the array is the absolute path to Node, which ran the process. The second element in the array is the path to the file that's running. The following elements will be any command line arguments provided when the process was initiated. Command line arguments are separated from one another with spaces.



We've only covered a few of the properties of the process object, so make sure to check out the [documentation on the process object](https://nodejs.org/api/process.html) to learn more about it and explore some of its other methods and properties.

Ex:



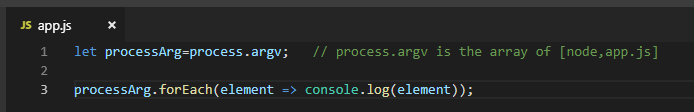


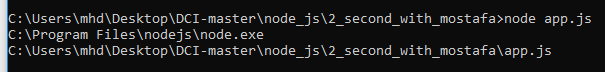
process.exit();

to stop to execute the code.

process.argv

Process.argv is the array of [node, app.js]





**Or from Kostas: slice + spread function**

3\_Accessing\_the\_Process\_Object

// another way from kostas

const myArgArray=process.argv.slice(2); // it will take copy from process.argv array, start from index 2. to new array myArgArray

const calculate=(salary, numOfMonth, percentDeducted)=>(percentDeducted/100)\*salary\*numOfMonth;

console.log(calculate(...myArgArray));





filesystem core module

to read and write files

const fs=require('fs'); // to work with filesystem core module

## fs.readFile(path[, options], callback)



fs.readFile('./ex.txt',(err,data)=>{

if (err) console.log(err);

else console.log(data.toString('utf8'));

});

## fs.writeFile(file, data[, options], callback)



fs.writeFile('./newFile.txt',`Hallo Mohammed`,err=>{

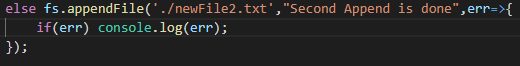
if (err) console.log(err);

else console.log('Writing done');

});

fs.appendFile(path, data[, options], callback)





Promise core module

1 create promise module

const util=require('util'); // to work with promise core module

2 const promiseVarible = util.promisify( function that returns callback)

const readPromiseWay=util.promisify(fs.readFile);

Whole code:

const fs=require('fs'); // to work with filesystem core module

const util=require('util'); // to work with promise core module

const readPromiseWay = util.promisify(fs.readFile);

const writePromiseWay = util.promisify(fs.writeFile);

const appendPromiseWay = util.promisify(fs.appendFile);

const dealWithFiles = async (file1, file2, file3) => {

try {

const data = await readPromiseWay(file1);

console.log('Read done!');

await writePromiseWay('./empty.txt', data.toString('utf8'));

console.log('Write done!');

await appendPromiseWay('./empty.txt', 'That was added by Jake, minutes before the break');

console.log('Append operation finished!');

const results = await readPromiseWay('./lorem.txt');

console.log('Read lorem ipsum');

await appendPromiseWay('./empty.txt', ' ' + results.toString('utf8'));

console.log('Appended the lorem ipsum');

} catch(error) {

console.log(error);

}

}

dealWithFiles('./example.txt', './empty.txt', './lorem.txt');

console.log('Execute first!');

3 promiseVarible(path)

.then(one argument arrow function + return asyn function)

.then(one argument arrow function + return asyn function)

.then(one argument arrow function + return asyn function)

.then(one argument arrow function)

.catch(err => console.log(err) )

\*When first then done,so the second then will take the previous return and executes.

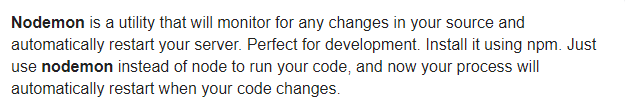
\* any then has error, so it will go next thens and go catch.

\* no then inside then

\* any error will happen in all thens, it will pass to catch(err=>)

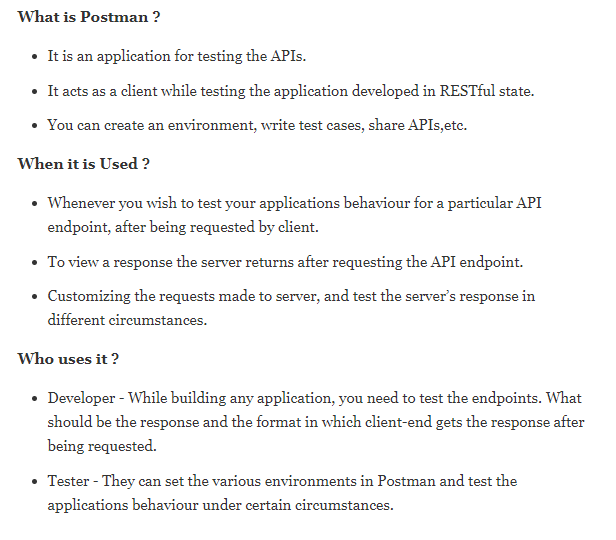
Nodemon

npm install -g nodemon



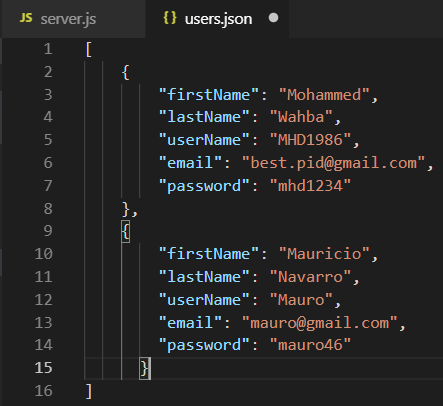
Postman

<https://www.getpostman.com/downloads/>



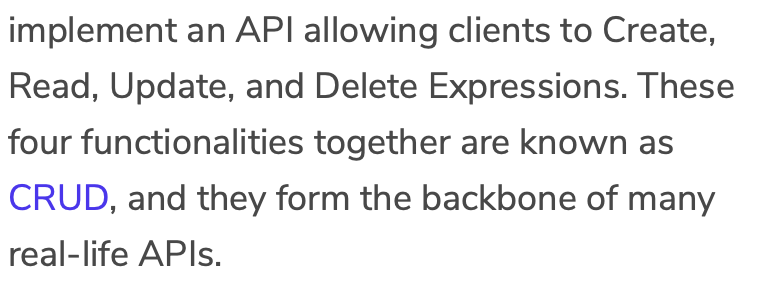
Restful API

To get, update, and delete data.



**Express**

**CRUD**



**13\_geoLoction Map**

Css Example

**css Example**

Css Example

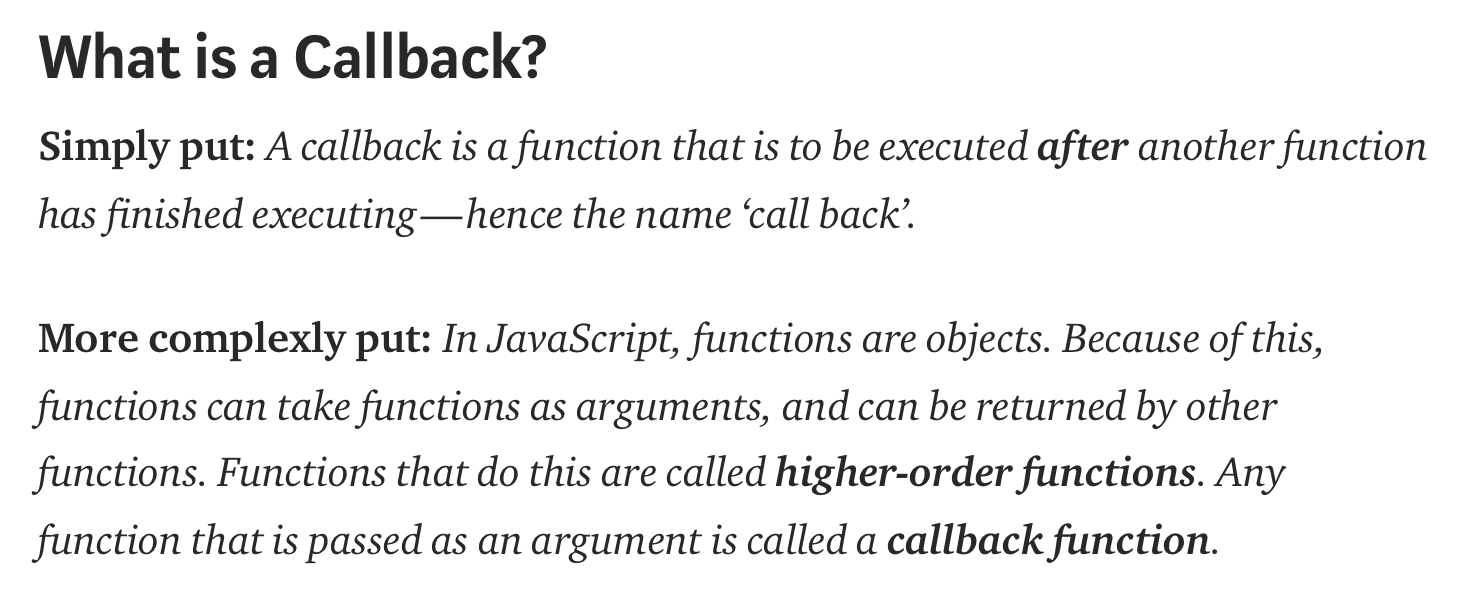
**css Example**

Css Example

Understanding ‘this’ in arrow function

**callback function**

**https://codeburst.io/javascript-what-the-heck-is-a-callback-aba4da2deced**

****

Callbacks VS Promises VS Async/Await

https://medium.com/@ThatGuyTinus/callbacks-vs-promises-vs-async-await-f65ed7c2b9b4

https://www.codecademy.com/learn/asynchronous-javascript

Axios , as fetch in js

Npm install –save

Axios.get

Create stream

**API-JSON**

What is API

https://www.w3schools.com/js/js\_json\_intro.asp