**NodeJs**

Environment to run JS outside Browser

Built on Chrome’s 8 Engine

Big Community

Full Stack  
**Prerequisites**

HTML, CSS, JS, ES6

Callbacks, Promises, Async-Await

Youtube – Coding Addict

Playlist – Js Nuggets

**Architecture of Node:**

**Nodejs is not the programming language or framework, it is the environment to run JavaScript code outside browser.**

Node is a run-time environment for executing JavaScript code.

Before node we use JavaScript only to build applications run inside of our web browser.

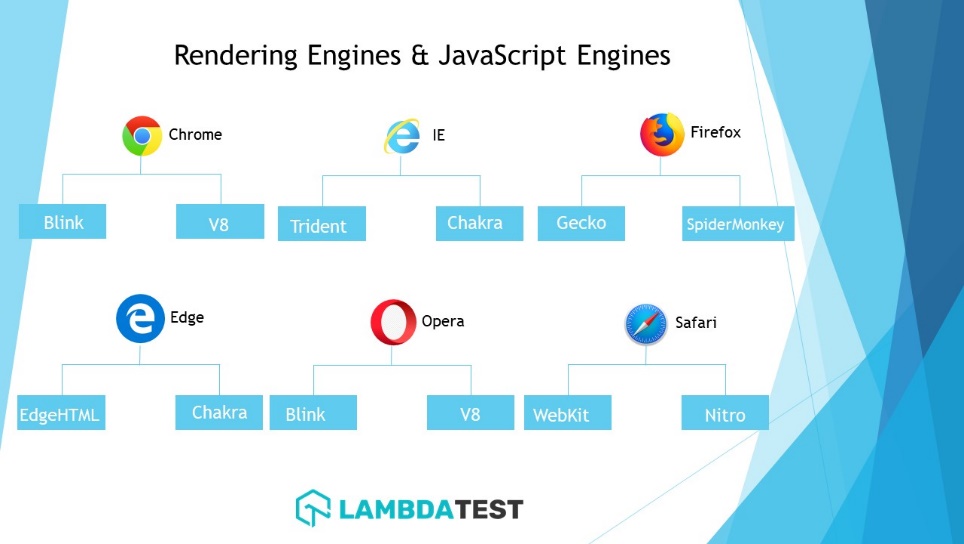
So, every browser out there has what we called a JavaScript engine, it takes JavaScript code and converts to code that computer can understand (Machine code).

Chrome uses V8 Engine and Blink

Opera uses V8 and Blink

IE was using Chakra and Trident  
Edge using EdgeHTML and Chakra

Firefox using Spider Monkey and Gecko

Safari using webkit and Nitro  
  


Because of these varieties of engines sometimes JScode can behave like differently from one browser to another. Browser provide runtime environment to JScode.

Ex: document.getElementById(‘header’);

**Ryan Dahl creator of NodeJS.**

He took google chrome V8 engine which is fastest JavaScript engine out there and embedded it inside C++ program and call that program node.exe

Like browser we have window object but in node we don’t have window or document object.

Node is a program that contain v8 engine plus some additional modules that give us capabilities not available inside browser.

NodeJS  
V8 + Modules

Chrome  
v8

We can work with file system, network and so on.

NodeJS is not programming language.

NodeJS is not a framework

It is runtime environment for executing JavaScript.

**How Node works:**

Asynchronous – Non-blocking – Single thread – utilize resources

request

response

Kitchen

Synchronous – Blocking – Multi thread – not utilize resource

Kitchen

**Note:** Node applications are asynchronous by default. In node we have single thread to handle all requests.

DB

Request

Request

Event Queue

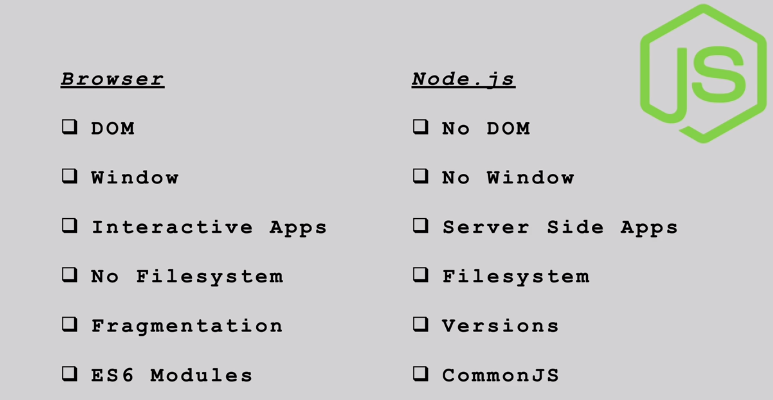
**Note:** Whenever you are using asynchronous mode you need to write call-back function. Once the complete task then it will call call-back function.

We pass them a function as an argument a callback that gets called when that task completes.

The callbacks has an argument that tells you whether the operation completed successfully.

Now we need to say what to do when fs.writeFile has completed(even if its nothing) and starts checking for errors.

**Difference between Browser JavaScript and node.js**



**Installation**

Go to nodejs.org download and install latest version of NodeJS. Use LTS (Long time support version)

(> node -v) Check version

You have REPL and CLI to work with NodeJS (Read Evaluate Print Loop)

Read - Read user’s input, parses the input into javascript data-structure and stores in memory.

Eval - Takes and evaluates the data structure

Print - Print the result

Loop - Loops the above command until the user press ctrl + c

(> node ) hit enter - REPL started

(const name = “Surya”)

.help - get help

.break - sometimes you get stuck, this gets you out

.clear - Alias for .break

.editor - Enter editor mode

.exit - Exit the REPL

.help - Print the help message

.load - Load JS from a file into the REPL session

.save - Save all evaluated commands in this REPL session to a file

To clear the screen, we also have cls command

// REPL

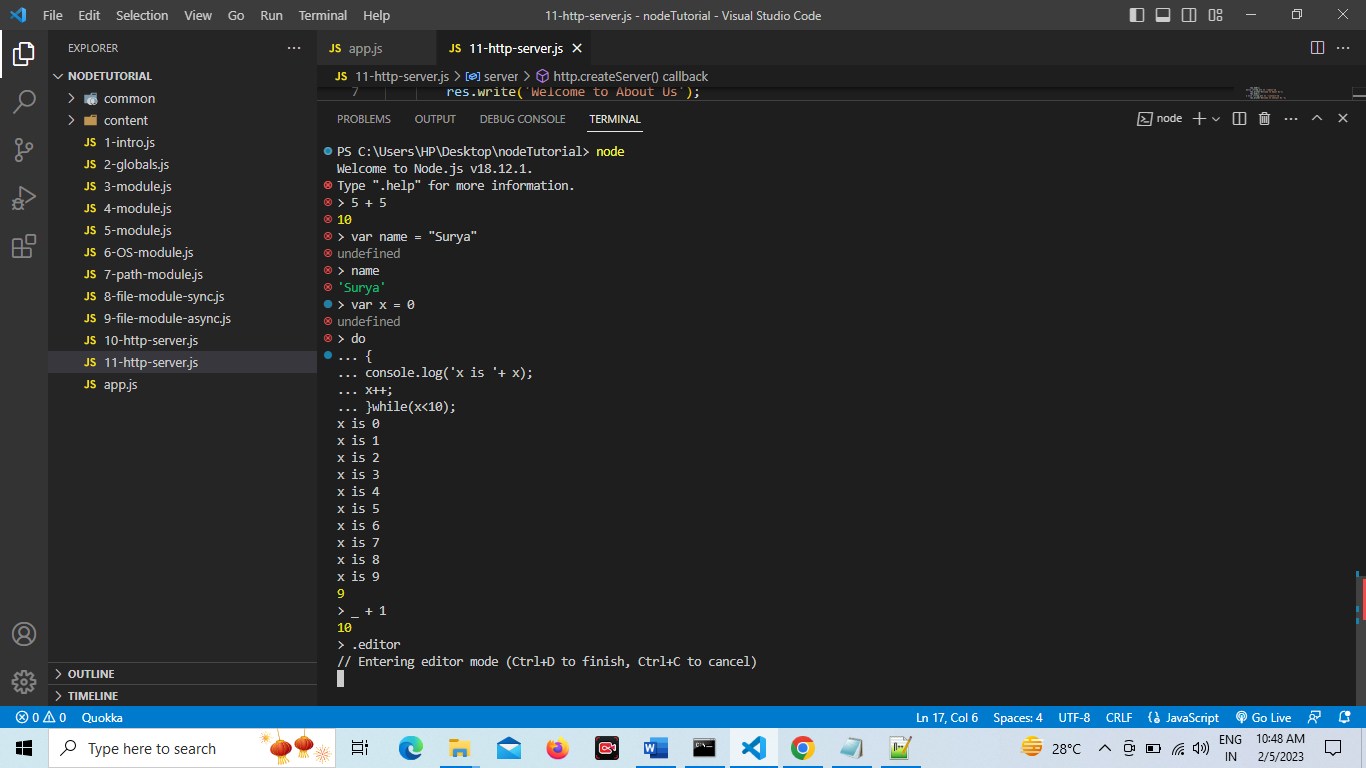
1: JS Expression

2: Use Variables

3: Multiline code

4: use (\_) to get last result

5: we can use editor mode



Create folder tutorial and create new file app.js

**Example: 1** [app.js]

let amount = 12;

if (amount < 10) {

console.log("Small Amount");

} else {

console.log("Large Amount");

}

console.log("It is first NodeJS ");

In VSCode editor you can open terminal by (ctrl + ~)

You can save and run above program with the command > node app.js OR > node app

Some are important points  
> node (press enter will be in REPL – If you double press tab node module list will be shown)  
To exit from REPL double press ctrl + c.

If you are in REPL editor you can finish coding with ctrl + d

**Note:** console.log(window); // window is not defined in NodeJS.  
In node we don’t have window or document object.

**Global Objects in NodeJS**

// GLOBALS - NO WINDOW

// \_\_dirname - path to current directory

// \_\_filename - file name

// require - function to use modules (CommonJS)

// module - it gives info about current module(file)

// process - It gives info about env where the program is being executed.

console.log(\_\_dirname);

console.log(\_\_filename);

console.log(module);

console.log(process);

setInterval(function () {

console.log("Hello World");

}, 1000);

**Note:** You can use > node app.js OR > node app (Use up and down arrow key previous command)

setTimeOut() and clearTimeOut()

setInterval() and clearInterval()

**Module in Node**

Client-side JavaScript runs inside client browser when we declare a variable or function, it added to global scope.

var sayHello(){

console.log(‘Hello’);

}

window.sayHello();

There is problem with this behavior. In real world application we often slit js code into multiple files so it is possible that we have two files and in both these files we define a function sayHello() with the exact same name because this function added to global scope when we define this function in another file. New definition is going to overwrite so this is the problem with global scope. So, in order to build reliable and maintainable application we should avoid defining variables and functions in the global scope.

Instead, we need module. We need to create a small building blocks or modules where we define a variables and functions. So, two variables or functions with the same name don’t overwrite another variable or function defined somewhere else.

Encapsulate inside the module.

Every file in node application considers as a module. Variables or functions defined in the file or that module is scope to that file/module.

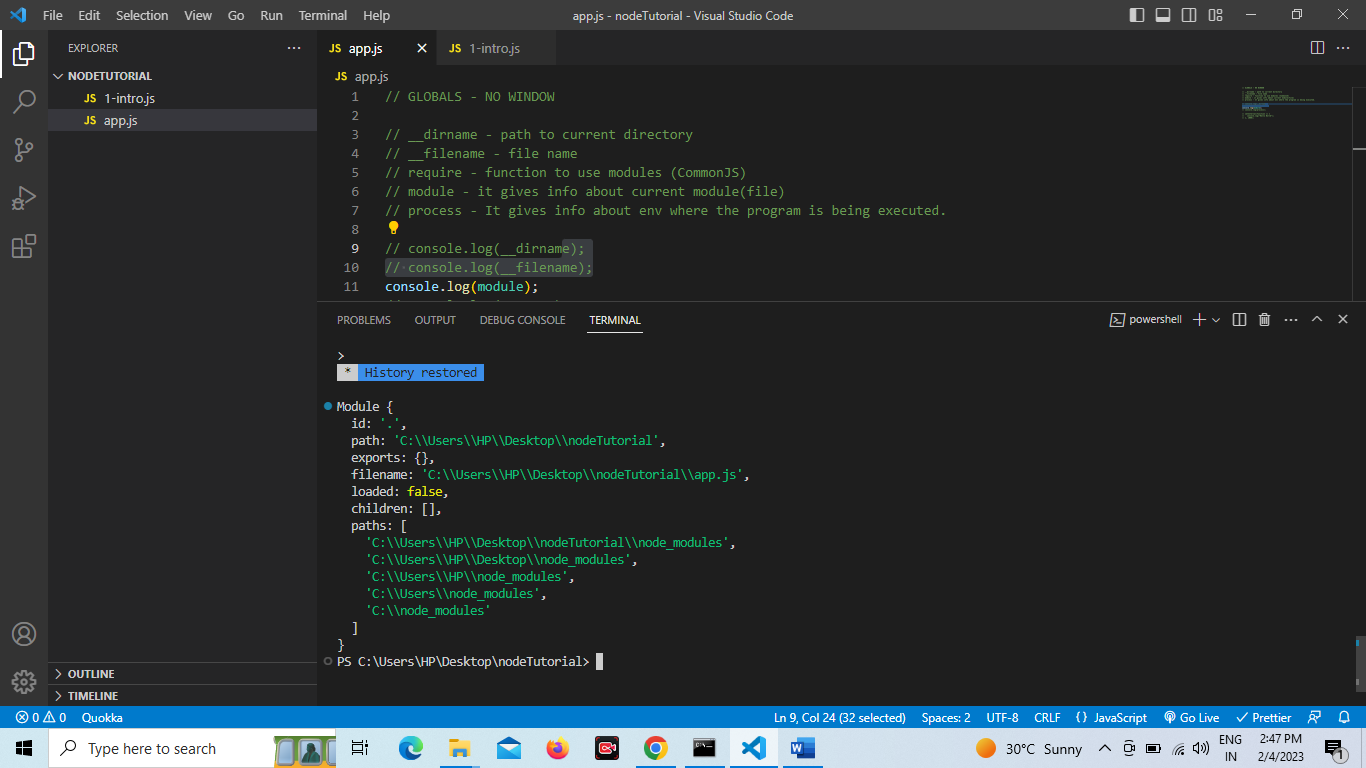
In object-oriented programming language we can say that they are private or not available outside that container. If wanna you use it outside you need to explicitly export it and make it public.

Every node application has at least one file – one module which we call main module.

app.js is our main module.

app.js

console.log(module);



Node uses commonJS and every file is module (by default).

Module encapsulate the code (Only share minimum)  
  
**Example: 3**

// Node uses commonjs and every file is module by default in node application

// Module - Encapsulate the code

const fname = "Surya";

const sname = "Peter";

function sayHello(name) {

console.log(`Hello there ${name}`);

}

sayHello(fname);

sayHello(sname);

Run code: > node app

Hello there Surya

Hello there Peter

Variables and functions defined in the file they are not added to global object. They are only scope to the file. They are not available outside of the file.

Note:

Window.console.log(‘Hello’);

Var message;

Console.log(window.message);

This is fine when we run JS code into the browser. But in the node we don’t have window or document object.

In nodejs we have global object.

Global.console.log(‘Hello’);

**Example: 4**

[common/names.js]

// Node uses commonjs and every file is module by default in node application

// Module - Encapsulate the code

// local

const secrete = "Secreate variable";

// share

const fname = "Surya";

const sname = "Peter";

module.exports = { fname, sname };

[common/greeting.js]

function sayHello(name) {

console.log(`Hello there ${name}`);

}

module.exports = sayHello;

[common/another.js]

// This example shows you how you can export data.

module.exports.items = ["Item 1", "Item 2"];

const person = {

name: "Suryakant",

};

module.exports.singlePerson = person;

[app.js]

const names = require("./common/names");

const sayHello = require("./common/greeting");

console.log(names);

sayHello(names.fname);

sayHello(names.sname);

const { fname, sname } = require("./common/names");

sayHello(fname);

sayHello(sname);

const data = require("./common/another");

console.log(data);

**Example: 5 (This example shows how to add third party module)**

[common/addition.js]

const num1 = 5;

const num2 = 10;

function addVal() {

console.log(`The addition is ${num1 + num2}`);

}

addVal();

[app.js]

require("./common/addition");

Output: The addition is 15

**Module Wrapper Function:**

Node.js is an open-source platform for operating both backend and frontend. Node.js does not run the code directly but first wraps it inside a function. This wrapping is done in the form of a function, and this function is called the node.js module wrapper function. This blog gives an overview of the node.js module wrapper function.

Node.js will wrap a module's code with a wrapper function before it is executed. The wrapper function looks like the following:

(function(exports, require, module, \_\_filename, \_\_dirname) {

// Module code

});

The variables and functions in the node.js are private to the module that has it. The variables and functions inside the module are private and not visible to the outside. The variables and functions are made private in node.js using the node.js module wrapper function.

The arguments in the wrapper function are as follows: -

|  |  |
| --- | --- |
| **Arguments** | **Description** |
| exports | It refers to module.exports which are shorter to types. |
| requires(id) | The parameter is a string that refers to the module name or path. |
| module | It implies the current module that is executed. |
| \_\_filename | It implies the file name of the current module. |
| \_\_dirname | It refers to the directory name of the current module. |

**Build In Modules in NodeJS / Core NodeJS module**

Consider modules to be the same as JS library. A set of functions you want to include in your application.

NodeJS has a set of build-in modules which you can use without any further installation.

1. **OS Module**

const os = require("os");

// Get the user information

console.log(os.userInfo());

// Method return system uptime in seconds

console.log(`The system uptime is ${os.uptime()} seconds`);

const currentOs = {

"OS Type": os.type(),

"OS Release": os.release(),

"Total Memory": os.totalmem(),

"Free Memory": os.freemem(),

"Host Name": os.hostname(),

"OS Platform": os.platform(),

"Temp dir": os.tmpdir(),

};

console.log(currentOs);

1. **Path module**

const path = require("path");

const pathObj = require("path");

// It returns path separator character

console.log(pathObj.sep);

// join method join the path

const filePath = pathObj.join("/content/", "sub-folder", "test.txt");

console.log(filePath); // content/sub-folder/test.txt

// get file name

const fileName = pathObj.basename(filePath);

console.log(fileName); // test.txt

// get file extenstion

const ext = pathObj.extname(filePath);

console.log(ext); // .txt

// Get absolute path

const absolute = pathObj.resolve(

\_\_dirname,

"content",

"sub-folder",

"test.txt"

);

console.log(absolute); // C:\Users\HP\Desktop\nodeTutorial\content\sub-folder\test.txt

// Get directory name

const dirName = pathObj.dirname(absolute);

console.log(dirName); // C:\Users\HP\Desktop\nodeTutorial\content\sub-folder\

1. **File Module** // This is synchronous

const fs = require("fs");

// Create file and add content to it

fs.writeFileSync("./content/sub-folder/surya.txt", "Welcome to NodeJS");

//It will overwrite if file is already present.

fs.writeFileSync("./content/sub-folder/surya.txt", "Welcome to new world of NodeJS");

// Append file content

fs.appendFileSync("./content/sub-folder/surya.txt",". I am loving nodejs.");

// Read file content

let buf\_data = fs.readFileSync("./content/sub-folder/surya.txt");

console.log(buf\_data);

console.log(buf\_data.toString());

**[Note: nodejs include an additional datatype called buffer. It is not available in browser javascript. Buffer is mainly used to store binary data]**

// Read file content using utf-8

let fileContent = fs.readFileSync("./content/sub-folder/surya.txt","utf-8");

console.log(fileContent);

// rename file

fs.renameSync("./content/sub-folder/surya.txt","./content/sub-folder/suryakant.txt");

// delete file

fs.unlinkSync("./content/sub-folder/suryakant.txt");

// create new folder

fs.mkdirSync("./content/sub-folder/demo");

// Delete folder

fs.rmdirSync("./content/sub-folder/demo");

**Note:** While running node app.js command it compile all operations line by line. We are doing file operation it will take few fractions of seconds to complete. You might be getting error because previous operation is in process and next operation is depend on first operation.

**/\* Note you can also do it \*/**

const {readFileSync, writeFileSync} = require('fs'); // extracting read and write method

const first = readFileSync("./content/first.txt","utf-8");

const second = readFileSync("./content/second.txt","utf-8");

console.log(first);

console.log(second);

const fileCont = writeFileSync("./content/result-sync.txt",`Cool loving node js. ${first} and ${second}`)

**File system module: Aynchronous**

const fs = require("fs");

// Create new file and add content to id

fs.writeFile("./content/sub-folder/surya.txt","Welcome to NodeJS", (err)=>{

console.log("File is created");

});

// append new content into the file

fs.appendFile("./content/sub-folder/surya.txt"," I am loving NodeJS", (err) => {

console.log("New file content added");

});

// Read file content

const buf\_data = fs.readFile("./content/sub-folder/surya.txt",(err, data) => {

console.log(data.toString()); // data has buffer data

});

// Read file content

const fileContent = fs.readFile("./content/sub-folder/surya.txt","utf-8",(err,data)=>{

console.log(data);

});

// rename file

fs.rename("./content/sub-folder/surya.txt","./content/sub-folder/suryakant.txt",(err) => {

console.log("File renamed");

})

// delete file (unlink)

fs.unlink("./content/sub-folder/suryakant.txt",(err)=>{

console.log("File get Deleted");

});

// create Folder

fs.mkdir('./content/sub-folder/demo',(err)=>{

console.log("Folder created");

});

// delete folder

fs.rmdir("./content/sub-folder/demo",(err)=>{

console.log("Folder Deleted");

});

**Note: While running node app.js command it compile all operations line by line. We are doing file operation it will take few fractions of seconds to complete. You might be getting error because previous operation is in process and next operation is depend on first operation.**

**Example: another way**  
  
const {readFile, writeFile, write} = require('fs');

readFile('./content/first.txt','utf-8',(err,result) => {

if(err){

console.log(err);

}

console.log(result);

const first = result;

readFile('./content/second.txt','utf-8',(err,data)=>{

if(err){

console.log(err);

}

const second = data;

writeFile('./content/result-async.txt',`I am loving NodeJS. ${first} and ${second}`,(err,data)=>{

if(err){

console.log(err);

}

});

const fileCont = data;

console.log(fileCont)

})

})

**What is the difference between synchronous and asynchronous**

Note: At the starting of the program and after the ending of the program write console.log and see the difference. You will get idea what is synchronous and asynchronous.

Example: console.log(‘Start’);

Your code goes here read write operation

Console.log(‘Ends’);

Output: Start  
 Ends

Some file content.

Asynchronous is faster as it is single threads. While synchronous execute line by line if it took time for some process, it will not execute other operation it waits until it finishes current process.

Callback function run when the particular operation is finished but it is not good.

Your code looks complicated. With the help of promises and Async wait our code will be clean. That we will see in upcoming section.

1. **https module**

https module allows us to set-up web server.  
  
To access web pages of any web application you need a web server. The web server will handle all the http request for the web application.

For example: IIS web server for ASP.NET web application and Apache is web server for PHP or java web application.

NodeJs provides capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run NodeJS web application but it’s recommended to use NodeJS web server.

The http.createServer() method includes request and response parameters which is supplied by NodeJS.

The object (request object) can be used to get information about the current HTTP request

e.g url, request header and data.

The response object can be used to send a response for a current HTTP server is supposed to be displayed as HTML you should include an HTTP header with the correct content type.

**Create server:**

const http = require('http');

const server = http.createServer((req,res)=>{

res.write('Welcome to webpage');

res.end();

});

server.listen(8000,"127.0.0.1",()=>{

console.log('Listening from the port 8000');

});

**NodeJS routing / handle http request in nodejs**

const http = require('http');

const server = http.createServer((req,res)=>{

if(req.url == '/') {

res.write('Welcome to webpage');

res.end();

}else if(req.url == '/about'){

res.write('Welcome to About Us');

res.end();

} else if(req.url == '/contact'){

res.write('Welcome to Contact Us');

res.end();

}else{

res.write('Page Not Found');

res.end();

}

});

server.listen(8000,"127.0.0.1",()=>{

console.log('Listening from the port 8000');

});

Here you can try about page urls. Most important if you inspect element and go to the network tab it shows status code pending if you don’t write else condition.  
Status code showing 200

Information response (100 - 199)

Successful response (200 - 299)

Redirects (300 – 399)

Client errors (400 – 499)

Server errors (500 - 599)

You can also write like this   
  
res.write('Welcome to Contact Us ');

res.end('Stay connected with Us');

You can write response head

else{

res.writeHead(400,{'content-Type':'text/html'});

res.write('Page Not Found');

res.end();

}

1. **event module:**

**Handling Events in nodejs**

Node.js has build in module, called events, where you can create, fire, and listen for your own events.

Example: Registering for the event to be fired only one time using once.

Example: Create an event emitter instance and register a couple of callbacks registering for the events with callback parameter.

const EventEmitter = require('events');

const eventEmitter = new EventEmitter();

/\* arrow function \*/

eventEmitter.on('start',()=>{

console.log('Event start is called');

})

eventEmitter.emit('start') // event name start calling

/\* function with parameter \*/

eventEmitter.on('myCust',(number)=>{

console.log('Number: '+number);

})

eventEmitter.emit('myCust', 10); // event myCust calling

/\* function with \*/

eventEmitter.on('haha',(num1,num2)=>{

console.log(`Hello addition is ${num1 + num2}`);

});

eventEmitter.emit('haha', 5, 2); // event haha calling

**Import NPM module is nodejs**

[ > npm init ]

[ > npm init -y] // Everything is setup as default

**Install Chalk package**

Create new folder and inside that folder initialize NPM.

> npm init

It will create package.json file. This is important file that store all dependencies. You can also see one more file is created package.lock.json

This file has extra information of the installed packages.

[ > npm I chalk –save ]

You need to include this package into your project.

const chalk = require('chalk');

console.log(chalk.blue('Hello'));

console.log(chalk.red.underline('Hello'));

console.log(chalk.green.underline.inverse('Hello World')); // flip bg and color

/\*

If you have installed latest version and you are getting error

then install old version.

Error: const chalk = require("chalk");

> npm install chalk@4.1.2

\*/

**Install validator package:**

const validator = require('validator');

const res = validator.isEmail('Surya@gmail.com');

console.log(res);

Install nodemon in nodejs

Nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are deleted.

**Global means you can use that module any project.**

Every time you need to run node index.js file but if you install nodemon it will show output at run-time (at the time of file modification).

[> node install nodemon -g]

[> nodemon -v]

[> nodemon ] // It will automatically run root file (index file)

[> nodemon app.js] // It will run app.js file.

To exit simply write ctrl + c and ctrl + c

https://www.npmjs.com/  
  
This will have many open source packages/modules designed and developed by some third party companies or the developers like you. Any one can write there own module or package and submit it and make it available for others. There is no quality check here. You can analysis good packages by seeing weekly downloads.

[> npm install bootstrap –save]

**Project Deployment on git:**

Deploy this on github repository. How we don’t want to upload node\_modules folder.

Create .gitignore file and add /node\_modules

[> git init]

Open gitbash and type git status command. You will see the untracked files in red color.

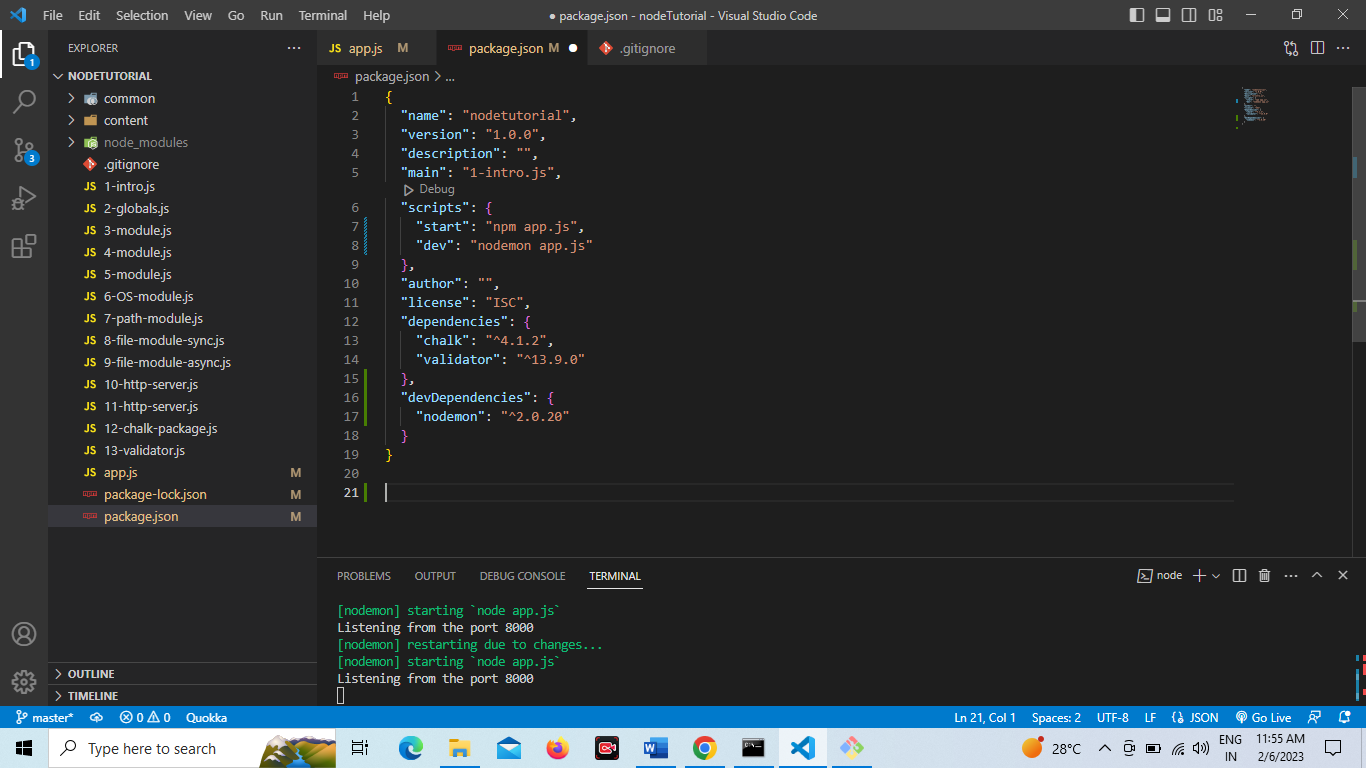
Now type git add . command, untracked files will be added to git. To check whether it is added or not type git status command again. You will see the all files in green color it means they are cached and ready for commit.

Now commit your changes and push by typing following command.

[git commit -m “First Commit”]

[git push origin master]

Now whenever you clone this project just you need to run [npm install] command it will install all packages required for the project.



npm run start  
npm run dev

Development dependencies

* npm uninstall nodemon –save-dev
* npm install -g nodemon // install globally – You can use it in any project

chalk 4.1.2 // 4 major change 1 minor change 2 bug fixes

**Event Loop:**

**Example: 1**

const readFile = require('fs');

console.log('First line started');

readFile.readFile('./content/sub-folder/test.txt','utf-8',(err,result)=>{

if(err){

console.log(err);

return

}

console.log(result);

console.log('complete the task');

})

console.log('Line Ended ');

/\*

Output:

First line started

Line Ended

Hello This is the simple test

complete the task

\*/

**Example: 2**

console.log('Hello');

setTimeout(function(){

console.log('Inside setTimeout');

},0);

console.log('World');

/\*

Output:  
Hello

World

Inside setTimeout

\*/

Note: setTimeOut is asynchronous

**Example: 3**

setInterval(function(){

console.log('Hello World');

},1000);

console.log('I am going to execute first');

Output:  
  
I am going to execute first

Hello World

Hello World

Hello World

**Example: 4**

const http = require('http');

const server = http.createServer((req,res)=>{

res.write(‘request event’);

res.end(‘Hello world’);

});

server.listen(8000,"127.0.0.1",()=>{

console.log('Listening from the port 8000');

});

**Complete JSON in NodeJS**

JSON stands for JavaScript Object Notation

JSON is a light weight format for storing & transforming data.

JSON is often used when data is sent from a server to a webpage.

**Javascript object:**

const bigData = {

name: "Suryakant",

age: 36,

city: "Pune"

}

console.log(bigData);

const jsonData = JSON.stringify(bigData);

console.log(jsonData);

const myObj = JSON.parse(jsonData);

console.log(myObj);

**Output:**

{ name: 'Suryakant', age: 36, city: 'Pune' }

{"name":"Suryakant","age":36,"city":"Pune"}

{ name: 'Suryakant', age: 36, city: 'Pune' }

If you want to create/convert javascript object to the JSON object then you need to know method. E.g stringify();

Const jsonData = JSON.Stringify(bioData); // javascript obj to JSON obj

Console.log(jsonData);

{“name”:”Suryakant”, “age”:”36”,”city”:”pune”}

Now, Again if you want to convert JavaScript object from the json then you have to use parse() method.

Const myObj = JSON.parse(‘jsonData’); // JSON obj to javascript obj

Console.log(myObj);

**Task:**

Convert to json

Add json data to another file

Read file

Again, convert back to json object

Console.log js object.

const fs = require('fs');

const bioData = {

name: "Suryakant",

age: 36,

city: "Pune"

}

const jsonData = JSON.stringify(bioData);

const fcontent = fs.writeFile('./content/sub-folder/bioData.json', jsonData ,(err,result)=>{

if(err){

console.log(err);

}else{

console.log("JSON file created successfully");

}

});

const fcont = fs.readFile('./content/sub-folder/bioData.json','utf-8',(err,result)=>{

if(err){

console.log(err)

}else{

console.log(result)

}

})

console.log(fcont);

**Create simple API in nodeJS**

API stands for Application Programming Interface. It is a way two or more computer programs to communicate

with each other

1) create folder userapi

2) create file userapi.json

const fs = require("fs");

const http = require('http');

const server = http.createServer((req,res)=>{

const fcont = fs.readFileSync('./content/sub-folder/bioData.json','utf-8');

const fdata = JSON.parse(fcont);

if(req.url == '/'){

res.write('Welcome to webpage');

res.end();

}else if(req.url == '/userapi'){

res.writeHead(200,{"content-type":"application/json"});

res.write(fdata.name);

res.end();

}else{

res.write('Inside else block');

res.end()

}

});

server.listen(8000,"127.0.0.1",()=>{

console.log('Listening from the port 8000');

});