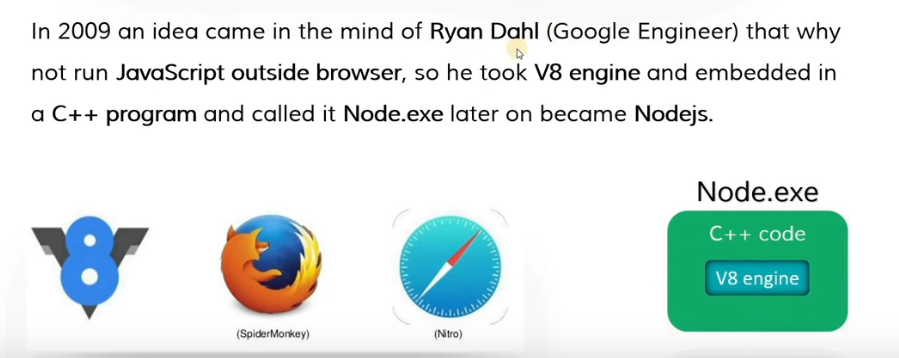
**NODE JS (nodejs.org)**

Node.js is a JavaScript runtime built on Chrome’s V8 JavaScript Engine (Built on Google Chrome, Just-in-Time compiler, No Interpreter)

Node.js is a JavaScript running on the server.



**When to use Node.js**

I/O Bound

Data Streaming Applications

Realtime Chat Applications



**Install NodeJS and NPM (nodejs.org):**  
Download and install NodeJS. After installing check whether it will properly installed or not:

Command Prompt: **C:\Users\ap\_aa>node –v** (it will give your version installed onto your system)

NodeJS automatically install NPM and to check whether it will install or not:

Command Prompt: **C:\Users\ap\_aa>npm –v** (it will give your version installed onto your system)

**To Create Node App**  
  
Create a **folder** for your project in your system and open it in VS Code.

**To Run any File;**

D:\nodelearning>node index.js (filename)

**PREREQUISITES FOR NODEJS**

1: Basic knowledge of JavaScript (with Advanced JavaScript)

2: Basic understanding of ES6 features

3: Client Server Model

**REPL IN NODEJS | READ, EVAL, PRINT & LOOP IN NODEJS**

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with output in an interactive mode. Node.js or Node comes bundled with a REPL environment. It performs the following tasks –

**Read** − Reads the user's input, parses the input into JavaScript data-structure, and stores in memory.

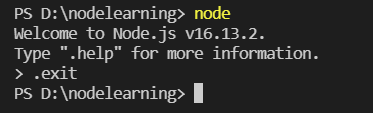
**Eval** − Takes and evaluates the data structure.

**Print** − Prints the result.

**Loop** − Loops the above command until the user presses ctrl-c twice.

The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

**REPL Environment:**



**To create a new file under NODEJS Project through terminal:**

D:\nodelearning> type nul > index.js

**REPL Functionality:**

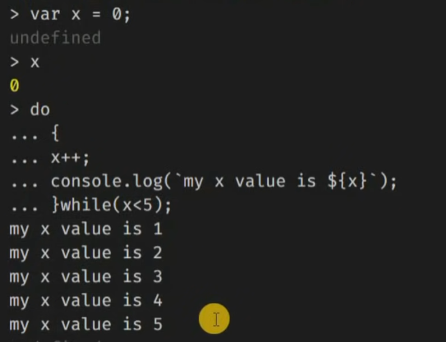
**JS Expression**: > 3+3

**Use Variables**: > var a = ‘Aayush’

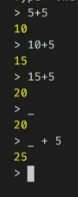
> var b = ‘Agarwal’

> a+b

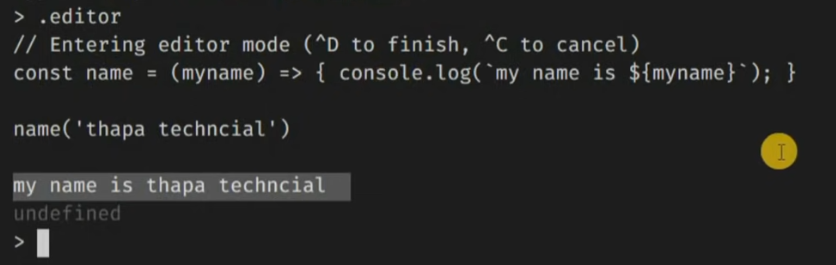
**Multiline Code / Loops:**



**Use (\_) to get the last result:**



**Use Editor Mode:**



# FILE SYSTEM NODE.JS MODULE (Synchronous)

# There are many core modules in Node.js.

# The Node.js file system module allows you to work with the file system on your computer. To include the File System module, use the require() method: var fs = require('fs'); A common use for the File System module:

# Read files, Create files, Update files, Rename files

# Creating a new Folder:

const fs = require('fs');

fs.mkdirSync(‘folder\_name’);

# Creating a new File:

fs.writeFileSync(“read.txt”,"Welcome"); **//provide a file path where you want to create**

**If again same function executes, then it update (overwrite) the existing data:**

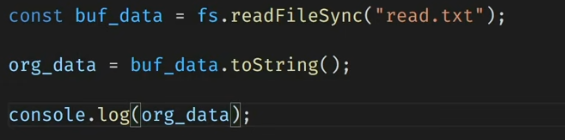
fs.writeFileSync(“read.txt”,"Welcome to World");

**If you want to add some extra data to existing file, then:**

fs.appendFileSync(“read.txt”, “How are you?”);

# Read data from File:

# 



**OR**

const data = fs.readFileSync(“read.txt”, “utf8”);

**Rename existing File:**

fs.renameSync(‘read.txt’, ‘readwrite.txt’);  **//(oldname, newname)**

**Delete existing File:**

fs.unlinkSync(‘read.txt’);

**Delete Folder:**

fs.rmdirSync(‘folder\_name’);

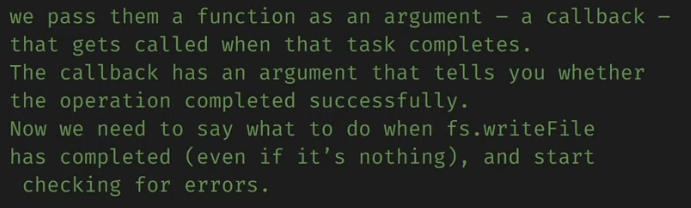
# FILE SYSTEM NODE.JS MODULE (Asynchronous)

# Creating a new Folder:

const fs = require('fs');

fs.mkdir(‘folder\_name’, (err) => {console.log(“task completed”)});

**Note: Callback function must be given in case of Asynchronous File System.**



**Create a new File:**

fs.writeFile(‘path with file name’, (err) => {console.log(“task completed”)});

**Append data on existing File:**

fs.appendFile(“read.txt”, “Hello Brother”, (err) => {console.log(“task completed”)});

**Read data from File:**

fs.readFile(“read.txt”, “UTF-8”, (error, data)=>{console.log(data)}); **//in case of read operation, we need to pass two parameters in which error should be first parameter.**

**Rename existing File:**

fs.rename(‘read.txt’, ‘readwrite.txt’, (error, data)=>{console.log(‘task completed’)}); **//(oldname, newname)**

**Delete existing File:**

fs.unlink(‘read.txt’, (error, data)=>{console.log(‘task completed’)});

**Delete Folder:**

fs.rmdir(‘folder\_name’, (error, data)=>{console.log(‘task completed’)});

**Synchronous VS Asynchronous**

|  |  |
| --- | --- |
|  |  |

# NODE.JS OS MODULE TO GET OPERATING SYSTEM INFORMATION

# The os module provides operating system-related utility methods and properties. It can be accessed using: const os = require('os');

### **os.arch()**

* Returns: [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)

Returns the operating system CPU architecture for which the Node.js binary was compiled. Possible values are 'arm', 'arm64', 'ia32', 'mips', 'mipsel', 'ppc', 'ppc64', 's390', 's390x', 'x32', and 'x64'.

### **os.freemem():** Returns: [<integer>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#Number_type)

Returns the amount of free system memory in bytes as an integer.

### **os.totalmem():** Returns: [<integer>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#Number_type)

Returns the total amount of system memory in bytes as an integer.

### **os.hostname():** Returns: [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)

Returns the host name of the operating system as a string.

### **os.platform():** Returns: [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)

Returns a string identifying the operating system platform. The value is set at compile time. Possible values are 'aix', 'darwin', 'freebsd', 'linux', 'openbsd', 'sunos', and 'win32'.

### **os.tmpdir():** Returns: [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)

Returns the operating system's default directory for temporary files as a string.

### **os.type():** Returns: [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)

Returns the operating system name as returned by [uname(3)](https://linux.die.net/man/3/uname). For example, it returns 'Linux' on Linux, 'Darwin' on macOS, and 'Windows\_NT' on Windows.

# PATH MODULE IN NODE.JS

# The path module provides utilities for working with file and directory paths. It can be accessed using: const path = require('path');

### **path.dirname(path):** The path.dirname() method returns the directory name of a path.

### **path.extname(path):** The path.extname() method returns the extension of the path, from the last occurrence of the . (period) character to end of string in the last portion of the path. If there is no . in the last portion of the path, or if there are no . characters other than the first character of the basename of path (see path.basename()) , an empty string is returned.

### **path.basename(path[, ext]:** The path.basename() method returns the last portion of a path.

### **path.parse(path):** The path.parse() method returns an object whose properties represent significant elements of the path.

The returned object will have the following properties:

* dir [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)
* root [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)
* base [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)
* name [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)
* ext [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type)

**Example**: path.parse('/home/user/dir/file.txt');

**// Returns:**

// { **root**: '/',

// **dir**: '/home/user/dir',

// **base**: 'file.txt',

// **ext**: '.txt',

// **name**: 'file' }

# 

# HOW TO CREATE AND EXPORT OUR OWN MODULES IN NODE JS

# In nodejs, every individual file is referred to as module. If you have define any function or variable in one file, you cannot call or use it in other module of file. You have to export that function or variable and then import it in other file in which you want to call it.

# Example: with one function

|  |  |
| --- | --- |
| index.js | operator.js |
| const addition = require(./operator); //you have to import module like any other fs, os etcconsole.log(addition(5,5)); | const add = (a,b) => { return a+b; };module.exports = add; //create a module like os, fs etc |

# Example: with two function

|  |  |
| --- | --- |
| index.js | operator.js |
| const operator = require(./operator); //you have to import module (which contain more than one function) like any other fs, os etcconsole.log(operator.add(5,5));console.log(operator.sub(10,5));OR | const add = (a,b) => { return a+b; };const sub = (a,b) => { return a-b; };module.exports.add = add; //create a module like os, fs etcmodule.exports.sub = sub; //create a module like os, fs etc |
| index.js | operator.js |
| const {add, sub, name} = require(./operator);console.log(add(5,5));console.log(sub(10,5));console.log(name); | const add = (a,b) => { return a+b; };const sub = (a,b) => { return a-b; };const name = “Aayush”;module.exports = {add, sub, name} |

# IMPORT NPM MODULE IN NODE.JS

**NOTE:** Go to your package.json and add **"type":"module"** above debug option. And use **import chalk from 'chalk' instead of the require() line.**

# npm is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js

# <https://www.npmjs.com/>

# We need to initialize npm for our node project:

# D:\nodelearning\npmModule> npm init

# These would be asked while installing npm

package name: (npmmodule)

version: (1.0.0)

description: learning

entry point: (index.js)

test command:

git repository:

keywords:

author: Aayush Agarwal

license: (ISC)

About to write to D:\nodelearning\npmModule\package.json:

{

"name": "npmmodule",

"version": "1.0.0",

"description": "learning",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Aayush Agarwal",

"license": "ISC"

}

After this, one file is created as **package.json (this file contain all packages dependencies which you install using npm). Now you can install any package using npm.**

**CHALK:** This is a package which we need to install using npm (<https://www.npmjs.com/package/chalk>)

**Installation**: D:\nodelearning\npmModule> **npm i chalk**

**Usage:** import chalk from ‘chalk’;

**Example**: console.log(chalk.blue("Hello World!"));

**NOTE:** when you install first package using npm, it will create a folder as **node\_modules** (chalk is one of the module) under your project which contain all code of your package.

**VALIDATOR:** This is a package which we need to install using npm (<https://www.npmjs.com/package/validator>)

**Installation**: D:\nodelearning\npmModule> **npm i validator**

**Usage:** import validator from ‘validator;

**Example**: console.log(validator.isEmail('abc@gmail.com'));

# NODEMON IN NODE.JS | IMPORT GLOBAL NPM MODULE IN NODE JS

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

nodemon does **not** require *any* additional changes to your code or method of development. nodemon is a replacement wrapper for node. To use nodemon, replace the word node on the command line when executing your script.

**Installation**: D:\nodelearning > **npm install -g nodemon**

**Usage:** donot have to import anything

**Example**: D:\nodelearning\npmModule> **nodemon index.js**

**ctrl+c:** it is used to get out of current terminal.

# MODULE WRAPPER FUNCTION IN NODE.JS

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

# const name = “aayush”;

# console.log(name);

# })

# 

# CREATING OUR OWN WEB SERVER IN NODE.JS | HTTP MODULE IN NODE JS

# <https://nodejs.org/dist/latest-v16.x/docs/api/http.html>

# To access web pages of any web application, you need a web server. The web server will handle all the HTTP requests for the web application e.g IIS is a web server for ASP.NET web applications and Apache is a web server for PHP or Java web applications.

# Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but it is recommended to use Node.js web server.

# Create Node.js Web Server: Node.js makes it easy to create a simple web server that processes incoming requests asynchronously.

# 

# 

**Example: How to create Web Server in NodeJS**

const http = require('http'); **//import http module**

const server = **http.createServer**((req, res) => {  **//two parameters for request and response**

    res.end('Hello! Server is Created'); **//if server start, give some response**

});

server.listen(8000, "127.0.0.1", () => { **//8000 is port no we have to give with localhost**

    console.log("Listening to Port no 8000");

});

# NODE JS ROUTING | HANDLED HTTP REQUESTS IN NODE.JS

# <https://nodejs.org/dist/latest-v16.x/docs/api/url.html>

# req.url; //this can be used to get the url which user has typed in browser.

# Example:

const http = require('http');

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

    if (req.url == '/'){

        res.end('Hello! Welcome to Home Page');

    }else if(req.url == '/about'){

        res.end('Hello! Welcome to About Page');

    }else{

res.writeHead(404, {“Content-type”: “text/html”}); **//to provide an error using status codes and tell the browser that it is of html document**

        res.end('**<h1>**Hello! Welcome to Error Page**</h1>**');

    }

});

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

    console.log("Listening to Port no 8000");

});

# 

# COMPLETE JSON IN NODE JS

# JavaScript Object Notation, or JSON, is a lightweight format for storing and transporting data. It is often used when data is sent from a server to a web page. JSON can be represented as either a list of values, e.g. an Array, or a hash of properties and values, e.g. an Object.

# Encoding and Decoding in JSON in NODEJS

# JavaScript provides 2 methods for encoding data structures to JSON and encoding JSON back to JavaScript objects and arrays. They are both available on the JSON object that is available in the global scope.

# JSON.stringify takes a JavaScript object or array and returns a serialized string in the JSON format. JSON.parse takes a JSON string and decodes it to a JavaScript data structure.

# Example: JSON.stringify

const bioData = {

    name: "aayush",

    age: 30,

    qual: "mtech"

};

const jsonData = JSON.**stringify**(bioData); **//object to json**

const objData = JSON.**parse**(jsonData); **//json to object**

console.log(**jsonData**);

console.log(**objData**);

# OUTPUT: {"name":"aayush","age":30,"qual":"mtech"}

# console.log(jsonData.qual); //gives an error, cannot access json data in this way

# console.log(objData.qual); //returns “mtech” as output

# Example:

const fs = require("fs"); **//import file system module**

const bioData = { **//create js object**

    name: "aayush",

    age: 30,

    qual: "mtech"

};

const jsonData = JSON.stringify(bioData); **//convert object into json format**

fs.writeFile("jsonData.json", jsonData, (err) => { **//write json data into a file**

     console.log("Done");

 });

fs.readFile("jsonData.json", "utf-8", (err, data) => { **//read json data from file**

    console.log(data);

    const orgData = JSON.parse(data); **//convert json data into js object format and show**

    console.log(orgData);

});

# HOW TO CREATE API IN NODEJS (api file is in json format eg UserApi.json)

# C:\Users\ap_aa\OneDrive\Desktop\Untitled.png

# 

# Steps to use API data into your project:

# Create a folder under which an api file is to be added eg UserApi.json

# Copy your api data into this file eg: <https://jsonplaceholder.typicode.com/users>

# Now, read api file where you want to access that file using fs module. This file is in json format but if you want to access any particular data, then you have to convert it into js object using JSON.parse() method. This procedure should be written in top and in sync mode as:

# 

# Eg: const orgData = JSON.parse(data);

# res.end(orgData[indexValue].keyvalue); //access object value

# res.end(orgData[0].name);

# EVENTS MODULE | HANDLING EVENTS IN NODE.JS WITH EVENTEMITTER

# Node.js has a built-in module called as Events where you can create-, fire- and listen- for your own events. You can call multiple functions with one event only.

# The EventEmitter class is a built-in class that resides in the events module.

# Creating EventEmitters: This can be done either via creating an instance of the class itself, or by implementing it through a custom class and then creating an instance of that class.

# Creating an EventEmitter Object: We'll create an EventEmitter that will emit an event that contains information about the application's uptime, every second. First, import the EventEmitter class from the events modules: const { EventEmitter } = require('events');

# <https://nodejs.org/dist/latest-v16.x/docs/api/events.html>

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

const **EventEmitter** = require(‘events’); **//creating a class EventEmitter with events module**

const event = new **EventEmitter**(); **//create instance of class**

**event.emit**(“sayMyName”); **//create an event (sayMyName is event name)**

**event.on**("sayMyName", () => { **//defining event (must be define before calling)**

    console.log('My name is Aayush');

});

# 

# Example: Create an event emitter instance and register a couple of callbacks

**const** EventEmitter **= require(‘events’);** **//creating a class EventEmitter with events module**

**const event = new EventEmitter();** **//create instance of class**

**event.on**("sayMyName", () => { **//You can call multiple function through one event**

    console.log('My name is Aayush');

});

**event.on**("sayMyName", () => { **//You can call multiple function through one event** console.log('My name is Agarwal);

});

**event.emit(“sayMyName”);**  **//create an event (sayMyName is event name)**

# Example: Registering for the event with callback parameters

**const** EventEmitter **= require(‘events’);** **//creating a class EventEmitter with events module**

**const event = new EventEmitter();** **//create instance of class**

**event.on**("checkStatus", (status\_code, message) => {

    console.log(`Status Code is ${status\_code} and the page seems to be ${message}`);

});

**event.emit**(“checkStatus”, 200, ‘OK’); **//create an event (sayMyName is event name with two parameters i.e. 200 and OK)**

# STREAMS AND BUFFER IN NODEJS | READABLE AND WRITABLE STREAMS

Streams are objects that let you read data from a source or write data to a destination in a continuous fashion. Streaming means listening to music or watching videos in real time, instead of downloading a file to your computer and watching later. In Node.js, there are four types of streams:

**Readable** − Stream which is used for a read operation.

**Writable** − Stream which is used for a write operation.

**Duplex** − Stream which can be used for both reading and write operation.

**Transform** − A type of duplex stream where the output is computed based on input.

Each type of Stream is an EventEmitter instance and throws several events at a different instance of times. For example, some of the commonly used events are –

**data** − This event is fired when there is data is available to read.

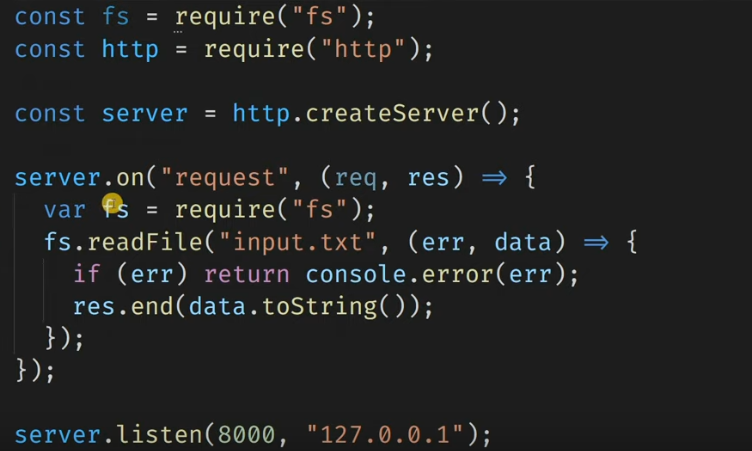
**end** − This event is fired when there is no more data to read.

**error** − This event is fired when there is an error receiving or writing data.

**finish** − This event is fired when all the data has been flushed to the underlying system

Streaming means listening to music or watching a video in ‘real-time’, instead of downloading a file to your computer and watching it later

**Example**: Old Method where all data is shown at once (without straming)



**Example**: Read data with Streaming Method

const fs = require('fs');

const http = require('http');

const server = http.createServer(); **//create server**

server.on("request", (req, res) => {

    const rstream = fs.**createReadStream**('input.txt'); **//read file with streaming**

    rstream.on('data', (chunkData) => {

        res.write(chunkData);

    });

    rstream.on('end', () => {

        res.end();

    });

    rstream.on('error', (err) => {

        console.log(err);

        res.end("File not FOund");

    });

});

server.listen(8000, '127.0.0.1');

**STREAM PIPE IN NODE JS**

**stream.pipe()** method is used to take a readable stream and connect it to a writable stream.

|  |  |
| --- | --- |
| **Difference** | |
| rstream.on('data', (chunkData) => {          res.write(chunkData);      });      rstream.on('end', () => {          res.end();      });      rstream.on('error', (err) => {          console.log(err);          res.end("File not FOund");      }); | rstream.pipe(res); |