**Node JS-1**

**Introduction**

**What is Node.js?**

* Node.js is an open source server environment. It allows to run Java Script on the server. ( It is not a framework or programming language)
* Node.js is free
* **Runtime environment** for building highly scalable server-side applications using JS.
* Node.js often use for building back-end services like APIs, Web app, Mobile app.
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)

**Why Node.js?**

**Node.js uses asynchronous programming!**

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

**What Can Node.js Do?**

* Node.js can generate dynamic page content.
* Node.js can create, open, read, write, delete, and close files on the server.
* Node.js can collect form data.
* Node.js can add, delete, modify data in your database.

**What is a Node.js File?**

* Node.js files contain tasks that will be executed on certain events.
* A typical event is someone trying to access a port on the server.
* Node.js files must be initiated on the server before having any effect.
* Node.js files have extension ".js".

**Setup**

## Download Node.js

The official Node.js website has installation instructions for Node.js: [https://nodejs.org](https://nodejs.org/)

* The Node.js installer includes the NPM(Node Package Manager). **NPM is the package manager** for the Node JS platform. It puts modules in place so that node can find them, and manages dependency conflicts intelligently.

PS D:\NODE JS> npm -v

9.5.0

PS D:\NODE JS> node -v

v18.14.2

**REPL**

**REPL stands for**

* **R Read**
* **E Eval**
* **P Print**
* **L Loop**

It represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

It performs the following tasks −

* **Read** − Reads 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.

## REPL Commands

* **ctrl + c** − terminate the current command.
* **ctrl + c twice** − terminate the Node REPL.
* **ctrl + d** − terminate the Node REPL.
* **Up/Down Keys** − see command history and modify previous commands.
* **tab Keys** − list of current commands.
* **.help** − list of all commands.
* **.break** − exit from multiline expression.
* **.clear** − exit from multiline expression.
* **.save *filename*** − save the current Node REPL session to a file.
* **.load *filename*** − load file content in current Node REPL session.

### **Starting REPL**

REPL can be started by simply running **node** on shell/console without any arguments as follows.

$ node

You will see the REPL Command prompt > where you can type any Node.js command −

PS C:\Users\priyen> node

Welcome to Node.js v18.15.0.

Type ".help" for more information.

>

**> (> indicates that you are in REPL Node)**

### **Simple Expression**

Let's try a simple mathematics at the Node.js REPL command prompt −

$ node

> 1 + 3

4

> 1 + ( 2 \* 3 ) - 4

3

>

### **Use Variables**

You can make use variables to store values and print later like any conventional script. If **var** keyword is not used, then the value is stored in the variable and printed. Whereas if **var** keyword is used, then the value is stored but not printed. You can print variables using **console.log()**.

$ node

> x = 10

10

> var y = 10

Undefined =🡺 ( repl.repl.ignoreUndefined = true) will ignore undefined error.

> x + y

20

> console.log("Hello World")

Hello World

undefined

**To ignore undefined write this command:** repl.repl.ignoreUndefined = true

### **Multiline Expression**

Node REPL supports multiline expression similar to JavaScript. Let's check the following do-while loop in action −

$ node

> var x = 0

undefined

> do {

... x++;

... console.log("x: " + x);

... }

while ( x < 5 );

x: 1

x: 2

x: 3

x: 4

x: 5

undefined

>

**...** comes automatically when you press Enter after the opening bracket. Node automatically checks the continuity of expressions.

### **Underscore Variable**

You can use underscore **(\_)** to get the last result −

$ node

> var x = 10

undefined

> var y = 20

undefined

> x + y

30

> var sum = \_

undefined

> console.log(sum)

30

undefined

>

**> .editor // type .editor to enter in editor mode (Block wise execution only)**

// Entering editor mode (**Ctrl+D to finish**, Ctrl+C to cancel)

const fun=(a,b)=>

{console.log("Hello");

return a+b;

}

console.log("Addition is =",fun(10,20));

//Output:

***Hello***

***Addition is = 30***

***Undefined***

**Task: Print Number between 56 to 61 with constant t=55**

PS D:\NODE JS> node

Welcome to Node.js v18.14.2.

Type ".help" for more information.

> var t=55;

undefined

> do{

//console.log(t); // gives 55---60

... t++

... console.log(t); // gives 56---61

... } while(t<=60)

56

57

58

59

60

61

Undefined

**Callback using simple JS functions like setInterval(), setTimeout()**

**Why do we need Callback Functions?**

JavaScript runs code sequentially in top-down order. However, there are some cases that code runs (or must run) after something else happens and also not sequentially. This is called asynchronous programming.

Callbacks make sure that a function is not going to run before a task is completed but will run right after the task has completed. It helps us develop asynchronous JavaScript code and keeps us safe from problems and errors.

In JavaScript, the way to create a callback function is to pass it as a parameter to another function, and then to call it back right after something has happened or some task is completed.

**How to create a Callback?**

To understand what I’ve explained above, let me start with a simple example. We want to log a message to the console but it should be there after 3 seconds.

const message = function() {

console.log("This message is shown after 3 seconds");

}

setTimeout(message, 3000);

There is a built-in method in JavaScript called “setTimeout”, which calls a function or evaluates an expression after a given period of time (in milliseconds). So here, the “message” function is being called after 3 seconds have passed. (1 second = 1000 milliseconds)

In other words, the message function is being called after something happened (after 3 seconds passed for this example), but not before. So the message function is an example of a callback function.

**JavaScript setInterval() Method:** The setInterval() method repeats a given function at every given time interval.

**JavaScript setTimeout() Method:** This method executes a function, after waiting a specified number of milliseconds.

**What is an Anonymous Function?**

Alternatively, we can define a function directly inside another function, instead of calling it. It will look like this:

setTimeout(function() {

console.log("This message is shown after 3 seconds");

}, 3000);

As we can see, the callback function here has no name and a function definition without a name in JavaScript is called as an “anonymous function”. This does exactly the same task as the example above.

**Callback as an Arrow Function**

If you prefer, you can also write the same callback function as an ES6 arrow function, which is a newer type of function in JavaScript:

setTimeout(() => {

console.log("This message is shown after 3 seconds");

}, 3000);

**Some callback examples**

<html>

<head>

</head>

<body>

<p id="id"></p>

<script>

setTimeout(myfun,5000);

function myfun()

{

document.getElementById("id").innerHTML="LJU";

}

</script>

</body>

</html>

**Output**: LJU after 5 sec.

<html>

<head>

</head>

<body>

<p id="demo"></p>

<script>

function mydisplay(sum)

{

document.getElementById("demo").innerHTML="<b>"+ sum +"</b>";

}

function mycals(num1,num2,mycallback)

{

sum=num1+num2;

mycallback(sum);

}

mycals(13,15,mydisplay);

</script>

</body>

</html>

**Output : 28**

<html>

<head>

</head>

<body>

<p id="p1"></p>

<script>

function add(a,b)

{

obj=document.getElementById("p1");

obj.innerHTML=(a+b);

}

a=2;

b=5;

setInterval(

function()

{

add(++a,++b);

},1000

);

</script>

</body>

</html>

Output : Display 9 and then incremented

// Callback Function Example

function greet(name, myFunction) {

console.log('Hello world');

// callback function

// executed only after the greet() is executed

myFunction(name);

}

// callback function

function sayName(name) {

console.log('Hello' + ' ' + name);

}

// calling the function after 2 seconds

setTimeout(greet, 2000, ‘LJU’, sayName);

**Output:**

Hello world

Hello LJU

**Task**

**Write a js code that display ”Hello” with increasing font size in interval of 50ms in blue colour and it should stop when font size reaches to 50px.**

<html>

<body>

<p id="demo" style="color:blue"></p>

<script>

size = 15;

function add() {

obj = document.getElementById("demo");

obj.innerHTML = "hello";

obj.style.color ="red";

obj.style.fontSize = size;

if (size <= 50) {

size++;

}

}

setInterval(add, 1000);

</script>

</body>

</html>

**Write a js code that display ”Hello” with increasing font size in interval of 50ms in blue color after clicking on button and it should stop when font size reaches to 50px.**

<html>

<head>

<style>

p{

color:blue;

}

</style>

</head>

<body>

<p id="p1"> Hello</p>

<button onclick="fun2()">font-size</button>

<script>

font="2";

function fun(font)

{

document.getElementById("p1").style.fontSize=font;

}

function fun2()

{

setInterval(

function()

{

if(font<=50)

{

fun(font++);

}

},50

);

}

</script>

</body>

</html>

**Nodejs Module**

**Nodemon :**

The nodemon Module is a module that develop node.js based applications by automatically restarting the node application when file changes in the directory are detected. Nodemon does not require any change in the original code and method of development.

Advantages of Using nodemon Module:

It is easy to use and easy to get started.

It does not affect the original code and no instance require to call it.

It help to reduce the time of typing the default syntax node <file name> for execution again and again.

Installation: Install the module using the following command and check version:

**npm install -g nodemon**

**Core modules**

Modules are same as JavaScript libraries. It is a set of functions you want to include in your application.

To include a module, use the require() function with the name of the module:

var module = require('module\_name');

The require() function will return an object, function, property or any other JS type dependency on what the specified module returns.

* **File Module**

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');

Common use for the File System module:

* Read files **fs.readFile()**
* Create files **fs.writeFile()**
* Update files **fs.appendFile()**
* Delete files **fs.unlink()**
* Rename files **fs.rename()**
* **Synchronous-Blocking**

var ps=require("fs");

ps.writeFileSync("Hello.txt","Hello World")

var data=ps.readFileSync("Hello.txt");

//console.log(data)

console.log(data.toString());

console.log("Program ended");

**Output:**

Hello World

Program ended

* **Reading the file data from file using Asynchronous mode (Asynchronous-nonblocking)**

By using callbacks, we can write asynchronous code in a better way. The following example creates a new file called test.txt and writes "Hello World" into it asynchronously.

var ps=require("fs");

ps.readFile("Hello.txt", function(e,data)

{

if(e)

{

return console.error(e);

}

console.log(data.toString()); // if you want buffer data then remove to string

console.error("complete");

}

);

console.log("Program ended");

**Output:**

Program ended

Hello World

complete

* **File Module (CRUD)**

**Write node Example with File system methods.**

1. **To create folder**
2. **Create one file inside that folder**
3. **Append some data to that file.**
4. **Read data from the file**
5. **Rename that file**
6. **Delete File**

var ps=require("fs");

ps.mkdirSync("Hello");

ps.writeFileSync("Hello/user.txt","Hello");

ps.appendFileSync("Hello/user.txt","\nWorld");

var data=ps.readFileSync("Hello/user.txt","utf-8");

ps.renameSync("Hello/user.txt","Hello/user1.txt");

console.log(data.toString());

ps.unlinkSync("Hello/user1.txt");

ps.rmdirSync("Hello");

**Output:**

Hello

World

* **File module with callback**

ps=require("fs");

ps.writeFile("a2.txt","Today is cold day",

()=>

{

console.log("completed");

});

ps.readFile("a2.txt","utf-8",(err,data)=>

{

console.log(data); //use data.toString() if not using utf-8

});

**Output:**

completed

Today is cold day

* **Write a Nodejs script to take 5 elements separated by white space in .txt file. Print sorted array of these 5 elements on Node Js server.**

**//string format**

var ps=require("fs");

ps.writeFileSync("s1.txt","50 -1 99 20 0 56 78 59");

data=ps.readFileSync("S1.txt","utf-8");

data=data.split(" ");

data.sort();

console.log(data);

Output:

[

'-1', '0', '20',

'50', '56', '59',

'78', '99'

]

**//integer format**

var ps=require("fs");

ps.writeFileSync("task.txt","0 1 99 20 33 -44 50");

data=ps.readFileSync("task.txt","utf-8")

console.log(data);

data=data.split(" ");

console.log(data);

for(i=0;i<data.length;i++)

{

data[i]=parseInt(data[i]);

}

d1=data.sort();

console.log(d1);

**Output:**

0 1 99 20 33 -44 50

['0', '1', '99', '20', '33', '-44', '50']

[-44, 0, 1, 20, 33, 50, 99]

* **Task: Write file using one JSON Object and read file which gives you Same JSON object in console.**

var ps=require("fs");

var data={"Name":"PSP"}

ps.writeFileSync("abc183.txt",JSON.stringify(data));

console.log("Entered data=")

console.log(data) //check same data will be stored in abc183.txt

var data2=ps.readFileSync("abc183.txt","utf-8");

console.log("Read data=")

var obj=JSON.parse(data2)

console.log(obj);

**Output**

Entered data=

{ Name: 'PSP' }

Read data=

{ Name: 'PSP' }

* **Task: Write file using having one JSON array of two Object and read file which gives you Same JSON object in console.**

var ps=require("fs");

var data={"Name":[{"Firstname":"Priyen"},{"Lastname":"Patel"}]}

ps.writeFileSync("h1.txt",JSON.stringify(data));

console.log(data) // Data in file

console.log(data.Name[0].Firstname)

var data2=ps.readFileSync("h1.txt","utf-8");

var obj=JSON.parse(data2) //stored data in string formate so require to convert in object form/

console.log(obj.Name[0].Firstname +" "+ obj.Name[1].Lastname);

Output:

{ Name: [ { Firstname: 'Priyen' }, { Lastname: 'Patel' } ] }

Priyen

Priyen Patel

* **Write a node.js script to copy contents of one file to another file. Data should be fetched from Source.txt and insert to destination.txt.**

var ps=require("fs");

ps.writeFileSync("source.txt","ABC");

ps.appendFileSync("source.txt","DEF");

data=ps.readFileSync("Source.txt","utf-8");

ps.writeFileSync("destination.txt",data);

data1=ps.readFileSync("destination.txt","utf-8");

console.log(data.toString());

**Output:**

ABCDEF

* **Writing data to file, appending data to file and then reading the file data using Asynchronous mode.**

var fs=require("fs");

fs.writeFile("abc.txt","Today is a good day",(err)=> {if(err){console.log("completed")}});

fs.appendFile("abc.txt","Today is a good day",function(err)

{

if(err){console.log("completed")

}

});

fs.readFile("abc.txt",(err,data)=>{

if(err){

console.error(err);

}

console.log(data.toString())

});

console.log("File Operations ended")

Output:

File Operations ended

Today is a good day

* **Defining an array of object with properties name and age. Write this object in a file named student.txt then read the file and display the object on console.**

const student =

[

{

name: "ABC",

age: 30

},

{

name: "XYZ",

age: 32

}

]

var ps=require("fs");

ps.writeFileSync("student.txt",JSON.stringify(student));

data=ps.readFileSync("student.txt","utf-8");

b=JSON.parse(data);

console.log(b);

Output:

[ { name: 'ABC', age: 30 }, { name: 'XYZ', age: 32 } ]

* **Create JSON object which contains array of objects. Calculate perimeter of square and perimeter of circle by using side value and diameter value respectively. And**

const shape =

[

{

name: "circle",

diameter: 8

},

{

name: "square",

side: 10

}

]

var ps=require("fs");

ps.writeFileSync("shape.txt",JSON.stringify(shape));

data=ps.readFileSync("shape.txt","utf-8");

b=JSON.parse(data);

if( b[0].name == 'circle'){

var perimeter = (b[0].diameter/2) \* 3.14 \* 2 ;

console.log(perimeter);

}

if ( b[1].name == 'square'){

var peri = (b[1].side) \*4 ;

console.log(peri);

}

ps.appendFileSync("shape.txt","\nPerimeter of circle = "+ JSON.stringify(perimeter)+ "\nPerimeter of square = "+JSON.stringify(peri));

Output:

[{"name":"circle","diameter":8},{"name":"square","side":10}]

Perimeter of circle = 25.12

Perimeter of square = 40

* **Write node.js script to create a class named person by assigning name and age in form of members. Create one function named elder which returns elder person object. Details of elder person should be printed in console as well as in file.**

class person

{

constructor(name,age)

{

this.age=age;

this.name=name;

}

elder(P)

{

if(this.age>P.age)

{

return this;

}

else{

return P;

}

}

}

var p1= new person("xyz",23);

var p2= new person("abc",34);

var p3=p1.elder(p2);

console.log(p3)

console.log("elder Person is" + " " + p3.name)

const jsonstr=JSON.stringify(p3);

var ps=require("fs");

ps.writeFileSync("d2.txt",jsonstr);

**Output: person { age: 34, name: 'abc' }**

**elder Person is abc**

**//And in file** {"age":34,"name":"abc"}

**Write node.js script to create a class named time and assign members hour, minute and second. Create two objects of time class and add both the time objects so that it should return the value in third time object. The third time object should have hour , minute and second such that after addition if seconds exceed 60 then minute value should be incremented and if minute exceed 60 then hour value should be incremented. The value should be printed in console as well as in file.**

class time

{

constructor(hour,min,sec)

{

this.hour=hour;

this.min=min;

this.sec=sec;

}

timer(p)

{

var t=new time();

t.hour=this.hour+p.hour;

t.min=this.min+p.min;

t.sec=this.sec+p.sec;

if(t.sec>60)

{

t.sec%=60;

t.min++;

}

if(t.min>60)

{

t.min%=60;

t.hour++;

}

return t;

}

}

var t1= new time(1,50,50);

var t2= new time(2,30,50);

var t3=t1.timer(t2);

console.log(t3);

const jsonstr=JSON.stringify(t3);

var ps=require("fs");

ps.writeFileSync("time.txt",jsonstr);

**Output:** time { hour: 4, min: 21, sec: 40 }

**OS Module : Operating System**

**Get information about the computer's operating system:**

The syntax for including the os module in your application:

**var os=require("os");**

**Example:**

os=require("os");

console.log(os.arch());

console.log(os.hostname());

console.log(os.platform());

console.log(os.tmpdir());

console.log(os.freemem());

a1=os.freemem();

console.log(`${a1/1024/1024/1024}`);

**Output:**

x64

SYCEIT309A-115

win32

C:\Users\foram\AppData\Local\Temp

298242048

0.2777595520019531

* **Write node.js script to create a folder named “AA” at temp folder. Also, create file named “temp.txt” inside “AA” folder. Now, check if available physical memory of the system is greater than 1 GB then print message “Sufficient Memory” in the file, else print message “Low Memory” in file.**

var ps=require("fs");

var os=require("os");

console.log(os.arch());

console.log(os.hostname());

console.log(os.platform());

console.log(os.tmpdir());

f = os.tmpdir();

freemem=os.freemem()/1024/1024/1024;

ps.mkdirSync(f+"/AA");

if(freemem > 1){

ps.writeFileSync(f+"/AA/temp.txt","Sufficient memory")

}

else{

ps.writeFileSync(f+"/AA/temp.txt","Low memory")

}

**Output:**

x64

ITICT406-182

win32

C:\Users\LJIET\AppData\Local\Temp

* **Write node.js script to create a folder named “AA” at temp folder. Also, create file named “temp1.txt” inside “AA” folder. Now, check if System is Win 32 then print message You are working on Windows 32 bit in file else print message You are working on windows 64 bit in file.**

var ps=require("fs");

var os=require("os");

console.log(os.platform());

f = os.tmpdir();

p = os.platform();

if(p == "win32"){

ps.writeFileSync(f+"/AAAA/temp1.txt","You are working on windows 32 bit")

}

else{

ps.writeFileSync(f+"/AAAA/temp.txt","You are working on windows 64 bit")

}

**Path Module**

The Path module provides a way of working with directories and file paths.

The syntax for including the path module in your application:

**var os=require("path");**



**Example:**

var pm=require("path");

path1=pm.dirname("D:/FSD-2/node/addon.txt");

console.log("Path: " + path1);

path2=pm.extname("D:/FSD-2/node/addon.txt");

console.log("Extension: "+path2);

path2=pm.basename("D:/FSD-2/node/addon.txt");

console.log("Basename: "+ path2);

path2=pm.parse("D:/FSD-2/node/addon.txt");

console.log(path2);

console.log(path2.root);

console.log(path2.dir);

console.log(path2.base);

console.log(path2.ext);

console.log(path2.name);

**Output:**

Path: D:/FSD-2/node

Extension: .txt

Basename: addon.txt

{

root: 'D:/',

dir: 'D:/FSD-2/node',

base: 'addon.txt',

ext: '.txt',

name: 'addon'

}

D:/

D:/FSD-2/node

addon.txt

.txt

addon

// PS D:\Priyen\_Patel>

* **Write node.js script to check whether the file extension is .txt or not.**

var pm=require("path");

path=pm.dirname("D:/LJ/abc.html");

console.log(path);

path=pm.basename("D:/LJ/abc.txt");

console.log(path);

ext = pm.extname("D:/LJ/abc.txt")

console.log(ext);

path=pm.parse("D:/LJ/abc.html");

console.log(path);

if(path.ext == ".txt"){

    console.log("Text Document");

}else{

    console.log("Not a text Document");

}

**Output:**

D:/LJ

abc.txt

.txt

{

root: 'D:/',

dir: 'D:/LJ',

base: 'abc.html',

ext: '.html',

name: 'abc'

}

Not a text Document

HTTP Module

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

To include the HTTP module, use the require() method:

var http = require('http');

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Use the createServer() method to create an HTTP server:

**Example1:**

var http = require('http');

var server = http.createServer( //create a server object

function (req, res) {

res.write('Hello World!'); //write a response to the client

res.end(); //end the response can be empty or include string

}).listen(8080); //the server object listens on port 8080

//(or server.listen(5051) instead of listen());

**Output on** <http://localhost:8080/>

Hello World!

* **Create HTTP webpage on which home page display “Home page”, student page shows “Student page” and any other page shows “Page Not found”.**

var h=require("http");

var server=h.createServer(

function(req,res)

{

if(req.url=="/")

{

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

res.write("<b> Home page </b>");

res.end();

}

else if(req.url=="/student")

{

res.writeHead(200,{"content-type":"text/plain"}); //plain shows code as it is

res.write("<i> Home page1 </i>");

res.end();

}

else

{

res.writeHead(404,{"content-type":"text/html"});

res.write("<h1> Page Not found </h1>");

res.end("Thanks");

res.write("Bye");

}

}

);

server.listen(5001);

console.log("Thanks for run");

* **Create http webpage and pass JSON object on webpage.**

var http=require("http");

var server=http.createServer(

function(req,res)

{

if(req.url=="/")

{

const a={"Name":"ABC", "Age":35};

res.writeHead(200,

{"content-type":"application/json"

});

res.write("Thank you..!");

res.write(JSON.stringify(a));

res.end();

}

});

server.listen(6008);

Render Response, Read HTML File Server, Routing, JSON Response

**Write a nodejs program load a simple html file on nodejs web server and print its content as html content.**

var h=require("http");

var ps=require("fs");

var u=require("url");

var addr="http://localhost:8080/16.html";

var q=u.parse(addr,true);

data=ps.readFileSync("."+q.pathname);

var server=h.createServer(

function(req,res)

{

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

*//res.writeHead(200,{"content-type":"text/plain"}); gives content of file(Whole program will display in port)*

res.write(data);

res.end();

});

server.listen(6051);