Node JS

- Latest version of NODE JS is 20.0.0. Download node js from https://nodejs.org/en and install it. To check downloaded version type node -v in console.
- The Node.js installer includes the NPM(Node Package Manager). For version check npm –v.
- **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.
- Node.js is an open source server environment. Node.js allows you to run JavaScript on the server.

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.

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 −

```
$ 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
> x + y
20
> console.log("Hello World")
Hello World
undefined
```

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 generate output, Ctrl+C to cancel)

```
const fun=(a,b)=&gt;
{console.log(“Hello”);
return a+b;
}
console.log(”Addition is =”,fun(10,20));
//Output:
Hello
Addition is = 30
Undefined
```

```
var t=55;
undefined
 do{
... t++
... console.log(t);
... } while(t&lt;=60)
56
57
58
59
60
61
```

```
undefined
```

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.
- **.editor** – enable editor mode to perform tasks
- **.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.

```
To remove undefined error
“repl.repl.ignoreUndefined = true”
```

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 PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

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.

File System (fs) Module

The syntax for including the fs module in your application:
var fs=require("fs");

Write node Example with File system methods. (CRUD Operation)

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("node");
ps.writeFileSync("node/write.txt","Hello");
ps.appendFileSync("node/write.txt","Hi");
data=ps.readFileSync("node/write.txt");
console.log(data);
console.log(data.toString());  //Or add “utf-8”
ps.renameSync("node/write.txt"," node/readwrite.txt")
ps.unlinkSync("node/readwrite.txt");
```

Read data from file and sort that data in ascending order using .sort() .

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

```
data=ps.readFileSync("S1.txt","utf-8");
data=data.split(" ");
d=data.sort();
let p=[];
for(i=0;i<d.length;i++){
   p[i]=parseInt(d[i]);

}
console.log(p)
```

Output:

```
[
 -1,  0, 20, 50,
 56, 59, 78, 99
]
```

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(data1);
```

Output:

```
        ABCDEF
```

Asynchronous Programming Using Callbacks

Asynchronous programming is an approach to running multiple processes at a time without blocking the other part(s) of the code.

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

```
// Display content on browser after 5 seconds
<html>
   <head>

   </head>
   <body>
      <p id="id"></p>
      <script>
         setTimeout(myfun,5000);
         function myfun()
         {
            document.getElementById("id").innerHTML="LJU";
         }
      </script>
   </body>
</html>
```

```
//Display addition of two number on browser using callback function
<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>
```

//Initialize two variables and increment both the variables each time and display the addition of both the variables at interval of 1 second.

```
<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>
```

// Write code to increase the font size at interval of 50 ms and it should stop increasing when the font size reaches to 50px. This task should be performed when you click on "Increase button" on browser. (Default font size 15px)

```
<html>
  <body>
    <p id="p1"> Hello</p>
    <button onclick="fun1()">Increase</button>
    <script>
    font="15";
    function fun(font)
    {
      P1.style.fontSize=font;
      P1.style.color= "blue";
    }
    function fun1()
```

```
        {
            setInterval(
                function()
                {
                    if(font<=50)
                    {
                        fun(font++);
                    }
                },50
            );
        }
    </script>
    </body>
</html>
```

```
// Without using button

<html>
    <body>
        <p id="demo" style="color:blue"></p>
        <script>
            size = 15;
            function add() {

                demo.innerHTML = "hello";
                demo.style.color ="red";
                demo.style.fontSize = size;
                if (size <= 50) {
                    size++;
                }
            }
            setInterval(add, 1000);
        </script>
    </body>
</html>
```

==

File system with Non blocking(ASynchronous) concept

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 fs = require('fs');

fs.writeFile('test.txt', 'Hello World!', function (err) {
   if (err)
      console.log(err);
   else
      console.log('Write operation complete.');
});
```

For example, we can define a callback that prints the result after the parent function completes its execution. Then there is no need to block other blocks of the code in order to print the result.

```
var ps=require("fs");
ps.readFile("Hello.txt",function(err,data)
{
   if(err)
   {
      return console.error(err);

   }
   console.log(data.toString());
   console.error("completed");
}
);
console.log("Program ended");
```

Output:

```
Program ended
Hello. welcome to LJU
Completed
```

Writing data to file, appending data to file and then reading the file data using using ES6 callback.

```
         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 dayToday is a good day
```

Examples

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); // To access values of properties
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 two objects and a method 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);
const jsonstr=JSON.stringify(p3);
var ps=require("fs");
ps.writeFileSync("d2.txt",jsonstr);
```

Output:
```
person { 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 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 }
```

Write example as asked below

1. **Create one CSV(.csv) file with minimum two lines of data and copy the file content in JSON (.json) file. Read the json file data and print the data in console.**
2. **Write simple html code and create one file named “h1” with .html extension.**
3. **Write simple JSON string with two properties name and branch to .json file. Read the file data and print the value of name in console.**

```
const fs = require("fs");
//CSV file
csv = fs.readFileSync("test.csv","utf-8")

// csv to json
array = csv.split("\n");
let json = JSON.stringify(array);
fs.writeFileSync('test.json', json);
json_data = fs.readFileSync("test.json","utf-8");
json_parse = JSON.parse(json_data)
console.log(json_parse[1]);

// HTML file
fs.writeFileSync("h1.html","<html><body><h1
style='color:red'>Hello</h1></body></html>");
data= fs.readFileSync("h1.html","utf-8");
console.log(data);

// JSON file
fs.writeFileSync("xyz.json",'{"name":"LJU","branch":"CSE"}');
var data=fs.readFileSync("xyz.json");
var data1=JSON.parse(data);
console.log(data1.name);


/*
test.csv
A,B,C,D,E
we,are,students,of,LJU

test.json
["A,B,C,D,E","we,are,students,of,LJU"]

h1.html
```

```
<html><body><h1 style='color:red'>Hello</h1></body></html>

xyz.json
{"name":"LJU","branch":"CSE"}

Output:
we,are,students,of,LJU
<html><body><h1 style='color:red'>Hello</h1></body></html>
LJU

*/
```

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");

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 write "Sufficient Memory" in the file, else write "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 platform is "win32" or not and print message accordingly 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+"/AA/temp1.txt","You are working on windows 32 bit")
}
else{
   ps.writeFileSync(f+"/AA/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");

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

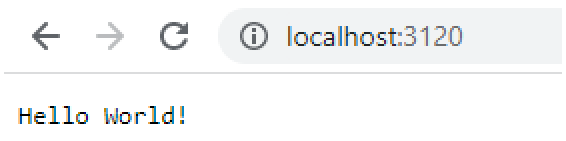
HTTP module allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
To include the HTTP module, use the require() method:
var http = require('http');

Node.js as a Web Server

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:

```
var http = require('http');

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(3120); //Server listening on port
```



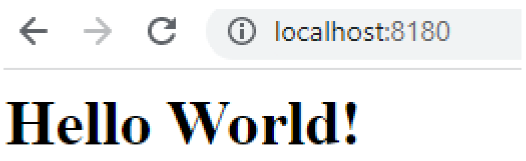
```
Hello World!
```

Add an HTTP Header

Name	MIME type
HyperText Markup Language (HTML)	text/html
Cascading Style Sheets (CSS)	text/css
JavaScript	application/javascript
JavaScript Object Notation (JSON)	application/json
JPEG Image	image/jpeg
Portable Network Graphics (PNG)	image/png

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<h1>Hello World!</h1>');
  res.end();
}).listen(8180);
```



Hello World!

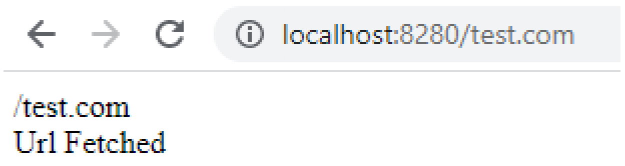
The first argument of the `res.writeHead()` method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

Request Url

The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).

This object has a property called "url" which holds the part of the url that comes after the domain name:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end("Url Fetched");
}).listen(8280);
```



```
/test.com
Url Fetched
```

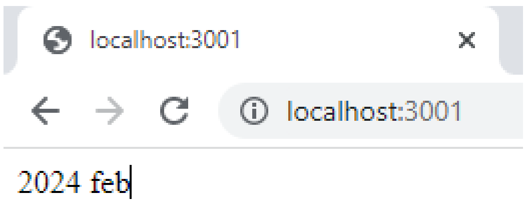
Get the query string

We can fetch the values from url query string as mentioned below using URL module.

1) **Add static url in code and request server to display data of query string on browser.**

```
var http = require('http');
var url = require('url');
var addr="http://localhost:8080/default.html?year=2024&month=feb";
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
```

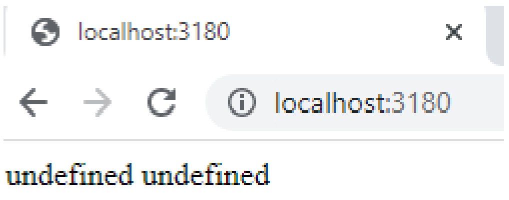
```
 /*Use the url module to get the querystring*/
 var q = url.parse(addr, true).query;

/*Return the year and month from the query object:*/
 var txt = q.year + " " + q.month;

res.end(txt);
}).listen(3001);
```



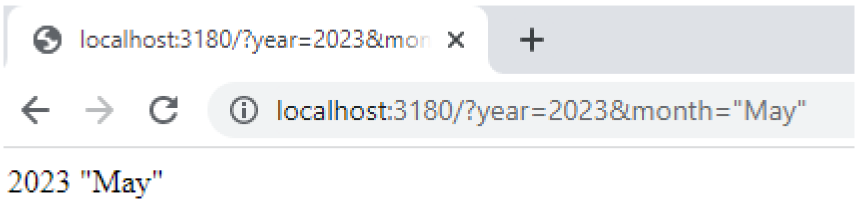
```
2024 feb
```

2) **Add query string in url at browser and request server to display data.**

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(3180);
```



```
undefined undefined
```


localhost:3180/?year=2023&mon
localhost:3180/?year=2023&month="May"

```
2023 "May"
```

Write node js script to perform tasks as asked.

1) **Create one page with two links (Home(/) and about(/about)).**
2) **Both pages must contain HTML type content and add required content on both the pages.**
3) **If user add any other URL path, then he/she will be redirected to page and plain message will be displayed of "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("<h1> Home page </h1><div><ul><li><a href='/'>Home</a></li><li><a
href='/about'>About</a></li></ul>");
         res.end();
      }
      else if(req.url=="/about")
      {
         res.writeHead(200,{"content-type":"text/html"});
         res.write("<h1> About Page </h1>");
         res.end();
      }
      else
      {
         res.writeHead(404,{"content-type":"text/plain"});
         res.write("Page not found");
         res.end("\nPlease check the url");
      /* res.write("Bye");*/ //display nothing if you add any content after res.end

      }
   });
server.listen(5051);
console.log("Thanks!");
```

Write node js script to request server to display JSON data on browser

```
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(6001);
```

Output: Thank you..!{"Name":"ABC","Age":35}

Write a nodejs program load a simple html file defined as static on nodejs web server a nd print its content as html content.

```
var h=require("http");
var ps=require("fs");
var u=require("url");
var addr="http://localhost:6051/7.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);
```

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 server=h.createServer(
    function(req,res)
    {
        var q=u.parse(req.url,true);
       data=ps.readFileSync("."+q.pathname);
            res.writeHead(200,{"content-type":"text/html"}); //text/plain gives program
            res.write(data);
            console.log(data);
            res.write("<h1> hello</h1>")
            res.end()
});
server.listen(6055);
console.log("Server Started");
```

Nodemon

Nodemon is a popular tool that is used for the development of applications based on node.js. It simply restarts the node application whenever it observes the changes in the file present in the working directory of your project.

To carry out the installation of Nodemon in your node.js-based project use the following steps for your reference.

```
npm install -g nodemon
```

To check version

```
Nodemon -v
```

Once nodemon is installed it might throw an error. We need delete the file as mention in error. Below is the file path.

C:/user/LJENG (This will be different) /Appdata/Roaming/npm/nodemon.ps1
(Follow the path and delete nodemon.ps1 file)

npm list -g command is used to check the path.

For example, if we created one file named “first.js” which contains code as below

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<h1>Hello World!</h1>');
  res.end();
}).listen(8180);
```

To run the file use below command

```
nodemon first.js
```

If we make any changes in the **first.js** file, it will automatically be reflected and the server will restart and the latest output will be displayed on the browser.