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

The official Node.js website has installation instructions for Node.js: 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
- o **R Read**
- o **E Eval**
- o **P Print**
- o **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** − resets the REPL context to an empty object and clears any multi-line expression currently being input.
- **.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

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.

Syntax: function function_name(argument, callback)

How to create a Callback? - use of setTimeout()

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

To understand what explained above, lets 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.

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:

Write js code that display hello in capital after 5 second

```js
setTimeout(() => {
console.log("hello".toUpperCase())
}, 5000);
```

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

---

Display clock using setInterval Method

```js
function updateTime() {
  // Get the current time in HH:MM:SS format for India timezone
  const timeString = new Date().toLocaleTimeString('en-IN', { timeZone: 'Asia/Kolkata',
hour12: false });
  console.log(timeString);
}
updateTime();

// Call updateTime every second (1000 milliseconds)
setInterval(updateTime, 1000);
```

Some callback examples

1. **Display content on browser after 5 seconds**

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

2. **Display addition of two number on browser using callback function**

```html
<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

3. **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>
```

Output : Display 9 and then incremented

4. 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 ="blue";
          obj.style.fontSize = size + "px";
          if (size <= 50) {
              size++;
          }
```

```
        }
      setInterval(add, 50);
    </script>
  </body>
</html>
```

5. 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>
  <head>
    <style>
      p{
         color:blue;
      }
    </style>
  </head>
  <body>
    <p id="p1"> Hello</p>
    <button onclick="fun2()">font-size</button>
    <script>
    font="15";
    function fun(font)
    {
      document.getElementById("p1").style.fontSize=font;
    }
    function fun2()
    {
      setInterval(
        function()
        {
          if(font<=50)
          {
            fun(font++);
          }
        },50 );
    }
  </script> </body></html>
```

Nodejs Core Modules

Node.js is a lightweight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and loaded automatically

when the Node.js process starts. However, you need to import the core module first in order to use it 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 System 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

1. Create files fs.writeFile()

It is used to asynchronously write the specified data to a file. By default, the file would be replaced if it exists. The 'options' parameter can be used to modify the functionality of the method.

Syntax:
> fs.writeFile(file, data, options, callback)

Parameters:
This method accepts four parameters as mentioned above and described below:
- **file:** It is a string, Buffer, URL or file description integer that denotes the path of the file where it has to be written. Using a file descriptor will make it behave similar to fs.write() method.
- **data:** It is a string, Buffer, TypedArray or DataView that will be written to the file.
- **options(optional):** It is an string or object that can be used to specify optional parameters that will affect the output. It has three optional parameter:
  - **encoding:** It is a string value that specifies the encoding of the file. The default value is 'utf8'.
  - **mode:** It is an integer value that specifies the file mode. The default value is 0o666.
  - **flag:** It is a string value that specifies the flag used while writing to the file. The default value is 'w'.
- **callback:** It is the function that would be called when the method is executed.
  - **err:** It is an error that would be thrown if the operation fails.

2. Read files fs.readFile()

This method is an inbuilt method that is used to read the file. This method read the entire file into the buffer. To load the fs module we use **require()** method. For example: var fs = require('fs');

Syntax:

> fs.readFile(filename, encoding, callback_function)

Parameters: The method accepts three parameters as mentioned above and described below:

- **filename:** It holds the name of the file to read or the entire path if stored at another location.
- **encoding:** It holds the encoding of the file. Its default value is **'utf8'**.
- **callback_function:** It is a callback function that is called after reading of file. It takes two parameters:
    - **err:** If any error occurred.
    - **data:** Contents of the file.

Return Value: It returns the contents/data stored in file or error if any.

3. Update files fs.appendFile()

The **fs.appendFile() method** is used to asynchronously append the given data to a file. A new file is created if it does not exist. The options parameter can be used to modify the behaviour of the operation.

Syntax:

> fs.appendFile(path, data, options, callback)

Parameters:

This method accepts four parameters as mentioned above and described below:

- **path:** It is a String, Buffer, URL or number that denotes the source filename or file descriptor that will be appended to.
- **data:** It is a String or Buffer that denotes the data that has to be appended.
- **Options (optional):** It is a string or an object that can be used to specify optional parameters that will affect the output. It has three optional parameters:
    - **encoding:** It is a string which specifies the encoding of the file. The default value is 'utf8'.
    - **mode:** It is an integer which specifies the file mode. The default value is '0o666'.
    - **flag:** It is a string which specifies the flag used while appending to the file. The default value is 'a'.
- **callback:** It is a function that would be called when the method is executed.
    - **err:** It is an error that would be thrown if the method fails.

4. Delete files fs.unlink()

`fs.unlink()` in Node.js removes files or symbolic links, while for directories, it's advisable to use `fs.rmdir()` since it doesn't support directory removal.

Syntax:

> fs.unlink(path, callback)

Parameters: This method accepts two parameters as mentioned above and described below:
- **path:** It is a string, Buffer or URL that**,** represents the file or symbolic link that has to be removed.
- **callback:** It is a function that would be called when the method is executed.
- **err:** It is an error that would be thrown if the method fails.

Synchronous Approach -Blocking Mode

Use synchronous file system functions provided by the `fs` module to perform operations like reading, writing, and deleting files. These functions have the same names as their synchronous counterparts, but with 'Sync' appended to the function names.

For example fs.writeFile(file, data, **callback**) will become **"fs.writeFileSync(file, data)"**

```
var ps=require("fs");

// To Write File in Sync

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

// To Read File in Sync

var data=ps.readFileSync("Hello.txt");
//console.log(data) – Provides Buffer data
//Use .toString() Method to convert this Buffer object to a string
console.log(data.toString());
console.log("Program ended");
```

Output:

Hello World
Program ended

- **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**
7. **Delete Folder**

```
var fs=require("fs");

fs.mkdirSync("Hello");                    //Generate Folder named Hello

fs.writeFileSync("Hello/user.txt","Hello");    // Insert File user.txt in Hello folder with data

fs.appendFileSync("Hello/user.txt","\nWorld"); // append the data

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

fs.renameSync("Hello/user.txt","Hello/user1.txt"); // Rename file name

console.log(data.toString());

fs.unlinkSync("Hello/user1.txt");           // Delete File

fs.rmdirSync("Hello");                      // Delete Folder (Folder Must be empty)
```

Output:
Hello
World

What is UTF-8 encoding?

UTF-8 is a type of character encoding that is commonly used to represent text in computer systems. It stands for **"Unicode Transformation Format - 8 bits"** and is a variable-length encoding that can represent every character in the Unicode character set.

In Node.js, UTF-8 is the default encoding for reading and writing text files. When you read a text file using a method like fs.readFileSync() or fs.readFile(), you can specify the encoding as UTF-8 to ensure that the text is correctly interpreted.

It is widely used on the web and in many computer systems because it is efficient for handling ASCII characters (which use only one byte), but can also represent characters from non-Latin scripts (which require more than one byte

Asynchronous Approach – Non-blocking Mode

WriteFile Async

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.');
});
```

ReadFile Async

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(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  //Print first
Hello World
Complete
```

- **File module (Write & Read) with callback [Using ES6]**

```
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

- **Writing data to file, appending data to file and then reading the file data using ES6 Concept.**

```
var fs=require("fs");
fs.writeFile("abc.txt","Today is a good day .\n",(err)=>{
   if(err){
   console.log("completed")
}
fs.appendFile("abc.txt"," Is it???",(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.
Is it???
```

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

//string format

```
var ps=require("fs");
ps.writeFileSync("s1.txt","5 9 6 1 2 0");
data=ps.readFileSync("S1.txt","utf-8");
data=data.split(" ");
data.sort();
console.log(data);
```

Output:
['0', '1', '2', '5', '6', '9']

//integer format

```
var ps=require("fs");
ps.writeFileSync("task.txt","5 9 6 1 2 0");
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:

5 9 6 1 2 0
['5', '9', '6', '1', '2', '0']
[0, 1, 2, 5, 6, 9]

Note: For More than 2 digits along with sign weight.
```
const array = [1, 12, 2];
array.sort((a, b) => a - b);
console.log(array); // Output: [1, 2, 12]
```

In this example, the comparison function (a, b) => a - b subtracts b from a. If the result is negative, a comes before b in the sorted order. If it's positive, b comes before a. If it's zero, their order remains unchanged.
By providing this comparison function, the sort() method sorts the numbers in ascending order based on their numerical values, resulting in [1, 2, 12]. Else due to lexicographical order your answer will not correct.

- **Write a node.js script to write and 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

- **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

OS Module: Operating System

Get information about the computer's operating system:
OS is a node module used to provide information about the computer operating system.

Advantages:
It provides functions to interact with the operating system. It provides the hostname of the operating system and returns the amount of free system memory in bytes.

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

Method	Description
arch()	Returns the operating system CPU architecture
hostname()	Returns the hostname of the operating system

platform()	Returns information about the operating system's platform
tmpdir()	Returns the operating system's default directory for temporary files
freemem()	Returns the number of free memory of the system in bytes

Example:

```
os=require("os");
console.log(os.arch());
console.log(os.hostname());
console.log(os.platform());
console.log(os.tmpdir());
console.log(os.freemem());
// os.freemem(): This method returns an integer value that specifies the amount of free
system memory in bytes.
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 file named "temp.txt". 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());
freemem=os.freemem()/1024/1024/1024;

if(freemem > 1){
ps.writeFileSync("temp.txt","Sufficient memory")
}
else{
    ps.writeFileSync("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 working on 32 bit plateform then print You are working on windows 32 bit else print You are working on windows 64 bit.**

```
var ps=require("fs");
var os=require("os");

console.log(os.platform());
f = os.tmpdir();
p = os.platform();
ps.mkdirSync(f+"/AA");
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")
}
```

Output:

win32 // File Generate at specific location.

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 pm=require("path");

Mehod	Description
basename()	Returns the last part of a path
dirname()	Returns the directories of a path
extname()	Returns the file extension of a path

Note: File path will be pass as a parameter inside method

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.txt");
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"); // It is not text file as changed to 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: Render Response, Read HTML File Server, Routing

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.

node js provides capabilities to create your own web-server. Which will handle http requests asynchronously.

http.createServer()

This method turns your computer into an HTTP server and creates an HTTP Server object. That includes request and response parameters.

Syntax : http.createServer(**requestListener**)

requestListener*() function*.

- The requestListener is a function that is called each time the server gets a request.
- The requestListener function is passed as a parameter to the http.createServer() method.
- The requestListener function handles *requests* from the user, and also the *response* back to the user

The **function(req,res){…} passed** into the http.createServer() method, will be executed when someone tries to access the computer on specified port.

- "req" parameter is the request object representing the HTTP request from the client.
- "res" parameter is the response object representing the HTTP response that will be sent back to the client.

server.listen(): method creates a listener on the specified port or path. You can specify any unused port no.

Example to create server and print "Hello world" message in **index,js** file

```
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());


Run file  by node index.js in terminal and hit http://localhost:8080 on browser


Output on browser →  http://localhost:8080/
Hello World!
```

Add an HTTP Header(To handle HTTP Requestes)

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:

Syntax: res.writeHead(status code, Type of header)

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.

Types of HTTP Header {"content-type":"MIME type"}

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

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. **Informational responses (100 – 199)** For ex. 100 – continue, 101- Switching protocols etc.
2. **Successful responses (200 – 299**) For ex. 200 – Ok, 201 – Created, 202 – Accepted etc.
3. **Redirection messages (300 – 399**) For ex. 301 - Move permanently, 304 - Not modified etc.
4. **Client error responses (400 – 499)** For ex. 400 – Bad request, 402 – payment require, 403 – forbidden, 404 – page not found etc.
5. **Server error responses (500 – 599)** For ex 500 – Internal Server Error, 502 – Bad Gateway , 503 – Service unavailable, 505 - HTTP Version Not Supported etc

Example to create server and print "Hello world" message in **h1 Tag**

```
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);
```
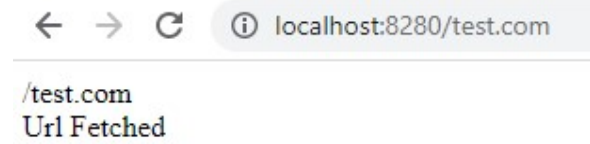
Request Url Approch

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
```

Note:
- If you change the text in res.write(),then you must run the code again to see the new text on the browser.
- If you don't specify res.end()then the browser will not stop progress.it shows continuously moving.
- res.writeHead() Method require to specify content type on your server. Else it will automatically set content-type. (**For example**: if First res.write() has plain text and second res.write() has HTML element. Then server will set plain content-type as text/plain and your html code will display as a plain text.)

- **Create HTTP webpage on which home page display "Home page", student page shows "Student page" and any other page shows "Page Not found".**
  (Render Response & Routing)

```
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");
}   });
server.listen(5001);
console.log("Thanks for run");
```

- **Create http webpage and pass JSON object on webpage. // JSON Response**

```
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);
```

- **Create http webpage and display message "Welcome to Priyen mam's class" in h1 tag after 10 seconds.**

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

// Create a server using the createServer method

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

  setTimeout(() => {

    res.writeHead(200, {'Content-Type': 'text/html'});

    res.end('<h2 style="color:tomato"> Welcome to Priyen mam\'s class</h2>');

  }, 10000); // 10000 milliseconds delay

});

const port = 3000;

server.listen(port, () => {

  console.log(`Server running at http://localhost:${port}/`);

});
```

Output Will Display after 10 second

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.

Advantage:
- 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.

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

Miscellaneous Programs

- **Create JSON object in file which contains array of objects. Calculate perimeter of square and perimeter of circle by using side value and diameter value respectively. And append final answers in file.**

```
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 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> Home page </h1><div><ul><li><a href='/'>Home</a></li><li><a
href='/about'>About</a></li></ul>");
```

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

- **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);
```

- **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 }