Unit-1 JSON FSD-2

What is JSON

JSON is an acronym for JavaScript Object Notation, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange. It is a language-independent data format. It supports almost every kind of language, framework, and library.

JSON is an open standard for exchanging data on the web. It supports data structures like objects and arrays. So, it is easy to write and read data from JSON.

In JSON, data is represented in key-value pairs, and curly braces hold objects, where a colon is followed after each name. The comma is used to separate key-value pairs. Square brackets are used to hold arrays, where each value is comma-separated

- JSON stands for JavaScript Object Notation.
- JSON is an open standard data-interchange format.
- JSON is lightweight and self-describing.
- JSON originated from JavaScript.
- JSON is easy to read and write.
- JSON is language independent.
- JSON supports data structures such as arrays and objects.

Basic data types supported by json are:

- **Strings**: Characters that are enclosed in single or double quotation marks.
- **Number**: A number could be integer or decimal, positive or negative.
- **Booleans**: The Boolean value could be either true or false without any quotation marks.
- **Null**: Here, null means nothing without any quotation marks.

In addition to basic data types, json has arrays and objects.

For example:

```
{"employees":[
  {"name":"Sonoo", "email":"sonoojaiswal1987@gmail.com"},
  {"name":"Rahul", "email":"rahul32@gmail.com"},
  {"name":"John", "email":"john32bob@gmail.com"}  ]}
```

What is XML?

XML stands for an extensible markup language. It is like HTML, where HTML stands for Hypertext Markup language. HTML is used for creating websites, whereas XML can be used for any kind of structured data.

XML has two ways of handling data, i.e., Tags and Attributes. The tags work as HTML. The start tags start with the <_> and end with the </_>. The start and end tags must match. The names must only be letters, numbers, and underscore, and the tag name must start with a letter only.

For example:

```
<color>

 <red> 1 </red>

<yellow> 2 </yellow>

<green> 3 </green>

</color>
```

JSON Vs XML

JSON stands for JavaScript Object Notation, whereas XML stands for Extensive Markup Language. Nowadays, JSON and XML are widely used as data interchange formats, and both have been acquired by applications as a technique to store structured data.

The following are the differences between the json and xml:

JSON	XML
JSON stands for javascript object notation.	XML stands for an extensible markup language.
The extension of json file is .json.	The extension of xml file is .xml.
The internet media type is application/json.	The internet media type is application/xml or text/xml.
The type of format in JSON is data interchange.	The type of format in XML is a markup language.
It is extended from javascript.	It is extended from SGML.
It is open source means that we do not have to pay anything to use JSON.	It is also open source.

The object created in JSON has some type.	XML data does not have any type.
The data types supported by JSON are strings, numbers, Booleans, null, array.	XML data is in a string format.
It does not have any capacity to display the data.	XML is a markup language, so it has the capacity to display the content.
JSON has no tags.	XML data is represented in tags, i.e., start tag and end tag.
	XML file is larger. If we want to represent the data in XML then it would create a larger file as compared to JSON.
JSON is quicker to read and write.	XML file takes time to read and write because the learning curve is higher.
JSON can use arrays to represent the data.	XML does not contain the concept of arrays.
It can be parsed by a standard javascript function. It has to be parsed before use.	XML data which is used to interchange the data, must be parsed with respective to their programming language to use that.
It can be easily parsed and little bit code is required to parse the data.	It is difficult to parse.
File size is smaller as compared to XML.	File size is larger.
JSON is data-oriented.	XML is document-oriented.
It is less secure than XML.	It is more secure than JSON.

JSON Example

JSON example can be created by object and array. Each object can have different data such as text, number, boolean etc. Let's see different JSON examples using object and array.

JSON Object Example

A JSON object contains data in the form of key/value pair. The keys are strings and the values are the JSON types. Keys and values are separated by colon. Each entry (key/value pair) is separated by comma.

The **{** (curly brace) represents the JSON object.

```
{
  "employee": {
     "name":      "sonoo",
     "salary":     56000,
     "married":   true
  }
}
```

JSON Array example

The **[** (square bracket) represents the JSON array. A JSON array can have values and objects.

example of JSON array having objects.

```
{   "employees":[
   {"name":"Shyam", "email":"shyamjaiswal@gmail.com"},
   {"name":"Bob", "email":"bob32@gmail.com"},
   {"name":"Jai", "email":"jai87@gmail.com"}
  ]
}
```

JSON Object

JSON object holds key/value pair. Each key is represented as a string in JSON and value can be of any type. The keys and values are separated by colon. Each key/value pair is separated by comma. The curly brace **{** represents JSON object.

example of JSON object.

```
    {
  "employee": {
         "name":      "sonoo",
         "salary":     56000,
        "married":   true
  }
 }
```

In the above example, employee is an object in which "name", "salary" and "married" are the key. In this example, there are string, number and boolean value for the keys.

JSON Object with Strings

The string value must be enclosed within double quote.

```
    {
     "name":      "sonoo",
    "email":    "sonoojaiswal1987@gmail.com"
   }
```

JSON Object with Numbers

JSON supports numbers in double precision floating-point format. The number can be digits (0-9), fractions (.33, .532 etc) and exponents (e, e+, e-,E, E+, E-).

```
{
"integer": 34,
"fraction": .2145,
"exponent": 6.61789e+0
}
```

JSON Object with Booleans

JSON also supports boolean values *true* or *false*.

```
 {
   "first": true,
  "second": false
   }
```

JSON Nested Object Example

A JSON object can have another object also. Let's see a simple example of JSON object having another object.

```
{
  "firstName": "Sonoo",
  "lastName": "Jaiswal",
  "age": 27,
  "address" : {
    "streetAddress": "Plot-6, Mohan Nagar",
    "city": "Ghaziabad",
    "state": "UP",
    "postalCode": "201007"
  }
}
```

JavaScript JSON.parse() method

The JavaScript **JSON.parse()** takes a JSON string and transforms it into a JavaScript object.

Syntax

```
JSON.parse(text[, reviver])
```

Parameters

text: The string to parse as JSON.

reviver: It is Optional. It prescribes how the value originally produced by parsing is transformed, before being returned.

Return value

An object corresponding to the given JSON text.

Example 1

```
<script>
var json = '{ "firstName":"ASHU", "lastName":"BHATI", "studentCode":7 }';
var student = JSON.parse(json);
document.write(student.firstName + " " + student.lastName);
</script>
```

Example 2

```
<script>
var j = '["C++","JavaScript","Python","HTML"]';
var data = JSON.parse(j);
document.write(data);
 //expected output: C++,JavaScript,Python,HTML
</script>
```

JavaScript JSON.stringify() method

The JavaScript **JSON.stringify()** method converts a JavaScript value to a JSON string.

Syntax

```
Json.stringify(value)
```

Example

```
<script>
var json = { firstName:"ASHU", lastName:"BHATI", studentCode:7 };
var student = JSON.stringify(json);
document.write(student);
</script>
```

Output:

```
{"firstName":"ASHU","lastName":"BHATI","studentCode":7}
```

Write code to fetch the “Give again” value from user object.

```
const user = {
      "name": ["ABC", "DEF", "GHI"],
      "age": "28",
      "course": ["FSD-1", "DE", "FSD-2"],
      "adress": ["T1", "T2", { "t3": "Give again" }]
  }
  console.log(user.adress[2].t3)

/*output
Give again */
</script>
</html>
```

From below JSON object fetch the values as asked.

```
<html>
<body>
  <script type="module" src="index.js">
   // `import a from '.\JSONEX' assert {type:'json'}`
   const a= {
     'Datastructures':
     [
       {
         'Name': 'tree',
         'course':'Intro',
         'content':['1','B','C']
       },
       {
         "Name": "tree1",
         "course":"Intro1",
         "content":["1","B","C","d"]
       }
     ],
     "xyz":
     {
       "Name":"Graphics",
       "Topic":["BFS","CDF","Sorting"],
     }
    }
    console.log(Datastructures[1].Name); //tree1
    console.log(a.Datastructures[0].Name); //tree
    console.log(a.xyz.Name);//Graphics
    console.log(a.xyz.Topic); //['BFS', 'CDF', 'Sorting']
    console.log(a.xyz.Topic[0]); //BFS
    console.log(a.Datastructures[1]); // { "Name": "tree1", "course":"Intro1",
"content":["1","B","C","d"] }
    console.log(a.Name); //undefined
    console.log(a.xyz); //{Name: 'Graphics', Topic: Array(3)}
    </script>
</body>
</html>
```

Write one JSON string with date property (yyyy-mm-dd) and print date in India standard time.

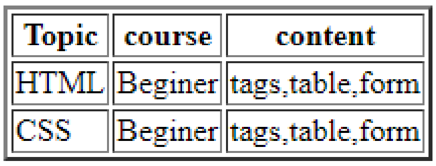
```
<html>
  <script>
    const obj=JSON.parse(`{"name":"xyz","age":"17","dob":"1997-03-14"}`);
    console.log(obj.dob);
    a = new Date(obj.dob);
    console.log(a);
  </script>
</html>

/*
output
Fri Mar 14 1997 05:30:00 GMT+0530 (India Standard Time)
*/
========================On click==================================

<html>
<script>
function call()
{
  obj=JSON.parse('{"Name":"ABC","Age":18,"City":"Ahmedabad","DOB":"1990-08-24"}');
  obj.DOB=new Date(obj.DOB);
  document.write(obj.DOB);
}
</script>
<body>
  <P id="demo"> </P>
  <button onclick="call()"> ClickMe </button>
</body>
</html>
```

Get JSON object values using for loop

```
<html>
<script>
 const sub =
 {
   "FSD": [
     {
       "Topic": "HTML",
       "course": "Beginer",
       "content": ["tags", "table", "form"],
     },
     {
       "Topic": "CSS",
       "course": "Beginer",
       "content": ["tags", "table", "form"]
     }
   ]
 };
 for (x in sub.FSD) {
   console.log(x)
   for (y in sub.FSD[x]) {
     console.log("Y" + y)
     document.write(sub.FSD[x][y] + "<br>")
     console.log(sub.FSD[x][y] + "<br>")
   }
 }
</script>
</html>
```

Get JSON object values using for loop and make HTML table to display these values.

```
const sub =
 {

   "FSD": [
     {
       "Topic": "HTML",
       "course": "Beginer",
       "content": ["tags", "table", "form"],
     },
     {
       "Topic": "CSS",
       "course": "Beginer",
       "content": ["tags", "table", "form"]
     }
   ]
 };
 var temp = "<table border='2px solid red'>";
 temp += "<tr>";
 for (x in sub.FSD[0]) {
   temp += "<th>" + x + "</th>";
 }
 temp += "</tr>";
 for (x in sub.FSD) {
 console.log(x);
 temp += "<tr>";
   for (y in sub.FSD[x]) {

     temp += "<td>" + sub.FSD[x][y] + "</td>";
   }
   temp += "</tr>";
 }
 temp += "</table>";
 //console.log(temp);
 document.write(temp);
```



Topic	course	content
HTML	Beginer	tags,table,form
CSS	Beginer	tags,table,form

Programs

Write a JS to store an array of objects having height and name. display names by sorting an array according to height.

```
const student =
  [
    {
      name: "NAS",
      height: 5.7
    },
    {
      name: "ABC",
      height: 6.0
    },
    {
      name: "XYZ",
      height: 6.2
    }
  ]

  for (let x = 0; x < student.length; x++){
    for (let y = 0; y < student.length; y++){

      if (student[x].height > student[y].height) {
        var temp;
        temp = student[y];
        student[y] = student[x];
        student[x] = temp;
      }
    }
  }
  for (let i = 0; i < student.length; i++){
    console.log(student[i].name, student[i].height);
  }
```

Output:

```
XYZ 6.2
```

ABC 6

NAS 5.7

Write a script to define two JSON objects named as “division1” and “division2” having an array to store names of students. These name should be sorted alphabetically in the object and should merge in one JSON object

```
var test = {
   "division1": {
      "name":["Z","B","H"]
   },
   "division2": {
      "name" :["Y","A","G"]
   }
}
const div1_data = test.division1.name;
const div2_data = test.division2.name;

var combine_data = div1_data.concat(div2_data).sort();

console.log(combine_data);
```

Output:

```
[ 'A', 'B', 'G', 'H', 'Y', 'Z' ]
```

Write a function “KeyandValue” that takes in an array and returns an object with

The first element of the array as the object’s key

The last element of the array as the key’s value.

Input = [‘abc’,’def’,’ghi’,’jkl’] Output = { abc: ‘jkl’ }

```
Function KeyandValue (a)
 { var temp = {};
   temp[a[0]] = a[a.length - 1];
   return temp;
}

var data = ['abc', 'def', 'ghi', 'jkl'];
console.log(KeyandValue (data));
```

Write a JSON object which contains array of 3 objects. Each object contains 2 properties name and age. Print details of elder person.

```
const person =
   [
      {
         name: "NAS",
         age: 38
      },
      {
         name: "ABC",
         age: 35
      },
      {
         name: "XYZ",
         age: 47
      }
   ]
for (let x = 0; x < person.length; x++){
      var t = 0;
      for (let y = 0; y < person.length; y++){

         if (person[x].age > person[y].age) {
            t++;
             console.log(t);
         }
      }
      if (t == person.length-1) {
         var t1;
         t1 = person[x];
         console.log(t1);
      }
}
```

Output:

```
{ name: 'XYZ', age: 47 }
```

What is Node.js?

- Node.js is an open source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!

Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.

A Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behaviour the exception rather than the norm.

When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.

This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.

Features of Node.js

Following is a list of some important features of Node.js that makes it the first choice of software architects.

1. **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
2. **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
3. **Single threaded:** Node.js follows a single threaded model with event looping.
4. **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.

5. **No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
6. **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License:** Node.js is released under the MIT license.

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 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.

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

Download Node.js

The official Node.js website has installation instructions for Node.js: https://nodejs.org

Command Line Interface

Node.js files must be initiated in the "Command Line Interface" program of your computer.

How to open the command line interface on your computer depends on the operating system. For Windows users, press the start button and look for "Command Prompt", or simply write "cmd" in the search field.

Navigate to the folder that contains the file "myfirst.js", the command line interface window should look something like this:

```
C:\Users\Your Name>_
```

Initiate the Node.js File

The file you have just created must be initiated by Node.js before any action can take place.

Start your command line interface, write `node myfirst.js` and hit enter:

Initiate "myfirst.js":

```
C:\Users\Your Name>node myfirst.js
```

Now, your computer works as a server!

Node.js REPL

The term REPL stands for **Read Eval Print** and **Loop**.

Each part of the REPL environment has a specific work.

Read: It reads user's input; parse the input into JavaScript data-structure and stores in memory.

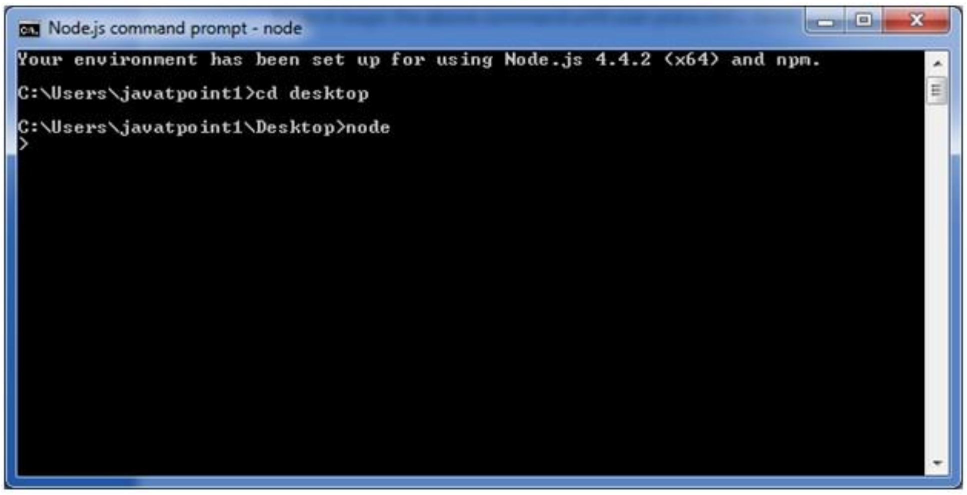
Eval: It takes and evaluates the data structure.

Print: It prints the result.

Loop: It loops the above command until user press ctrl-c twice.

How to start REPL

You can start REPL by simply running "node" on the command prompt. See this:



```
Your environment has been set up for using Node.js 4.4.2 (x64) and npm.

C:\Users\javatpoint1>cd desktop

C:\Users\javatpoint1\Desktop>node
>
```

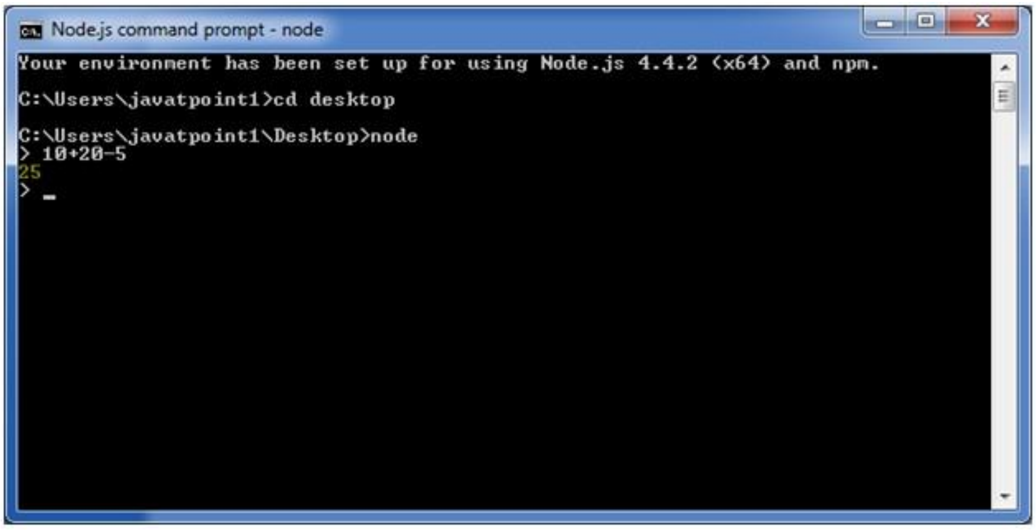
You can execute various mathematical operations on REPL Node.js command prompt:

Node.js Simple expressions

After starting REPL node command prompt put any mathematical expression:

Example1:

```
>10+20-5
25
```



```
Your environment has been set up for using Node.js 4.4.2 (x64) and npm.

C:\Users\javatpoint1>cd desktop

C:\Users\javatpoint1\Desktop>node
> 10+20-5
25
> _
```

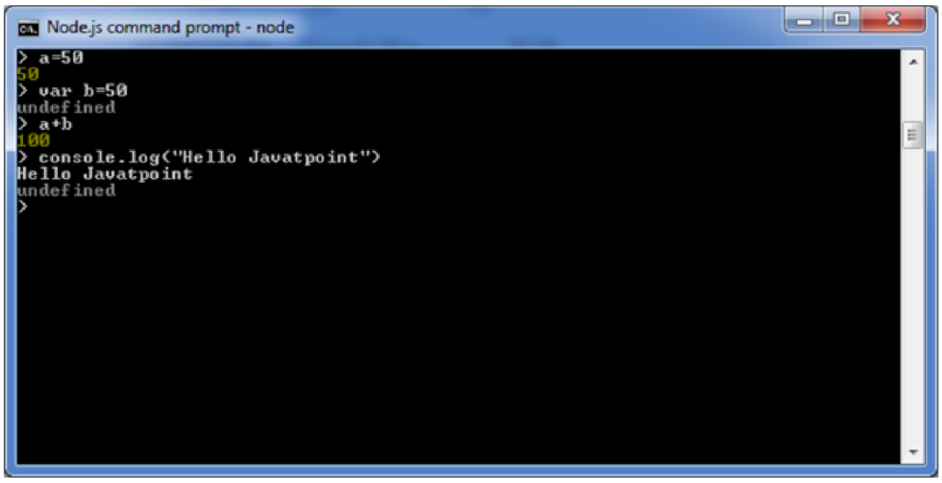
Example2:

```
>10+12 + (5*4)/7
  24.8571428571
```

Using variable

Variables are used to store values and print later. If you don't use **var** keyword then value is stored in the variable and printed whereas if **var** keyword is used then value is stored but not printed. You can print variables using console.log().

Example:



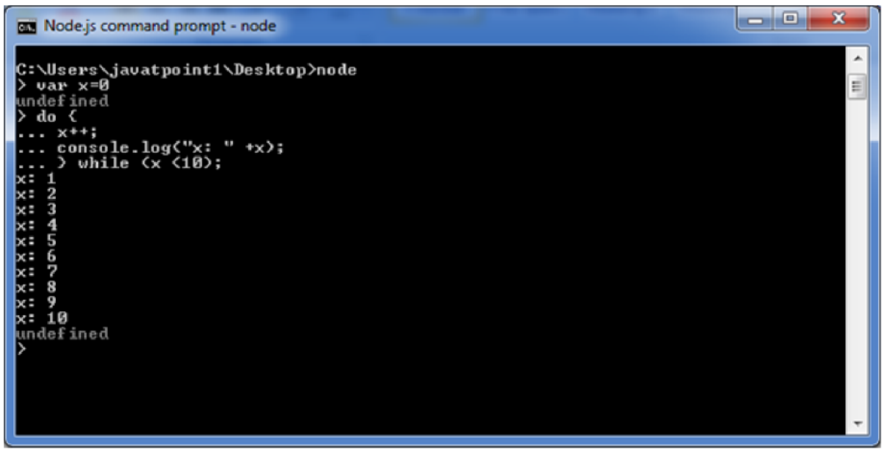
Node.js command prompt - node

```
> a=50
50
> var b=50
undefined
> a+b
100
> console.log("Hello Javatpoint")
Hello Javatpoint
undefined
>
```

Node.js Multiline expressions

Node REPL supports multiline expressions like JavaScript. See the following do-while loop example:

```
var x = 0
undefined   >
do {
... x++;
... console.log("x: " + x);
... } while ( x < 10 );
```



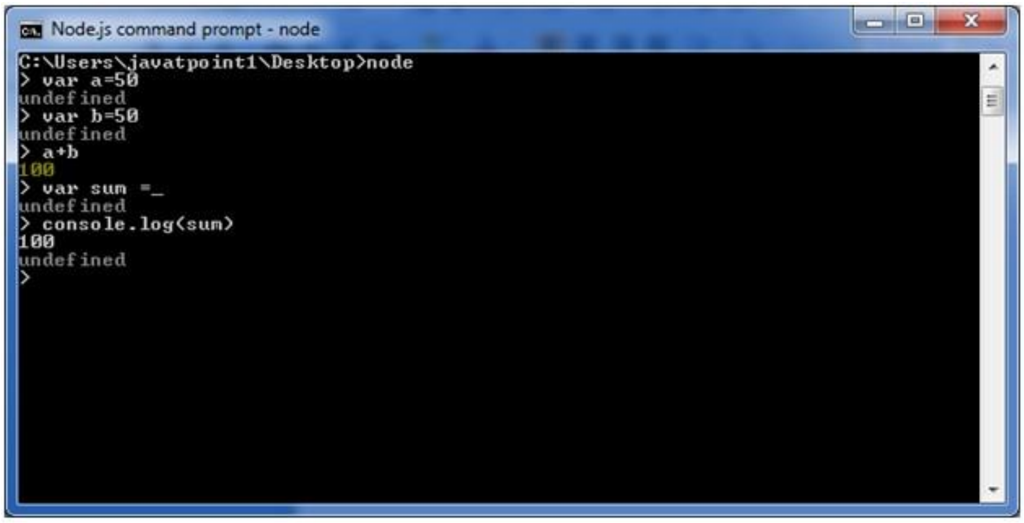
Node.js command prompt - node

```
C:\Users\javatpoint1\Desktop>node
> var x=0
undefined
> do {
... x++;
... console.log("x: " +x);
... } while (x <10);
x: 1
x: 2
x: 3
x: 4
x: 5
x: 6
x: 7
x: 8
x: 9
x: 10
undefined
>
```

Underscore Variable

You can also use underscore _ to get the last result.

Example:



```
C:\Users\javatpoint1\Desktop>node
> var a=50
undefined
> var b=50
undefined
> a+b
100
> var sum =_
undefined
> console.log(sum)
100
undefined
>
```

> .editor mode // 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
```

Node.js REPL Commands

Commands	Description
ctrl + c	It is used to terminate the current command.
ctrl + c twice	It terminates the node repl.
ctrl + d	It terminates the node repl.
up/down keys	It is used to see command history and modify previous commands.

tab keys	It specifies the list of current command.
.help	It specifies the list of all commands.
.break	It is used to exit from multi-line expressions.
.clear	It is used to exit from multi-line expressions.
.save filename	It saves current node repl session to a file.
.load filename	It is used to load file content in current node repl session.

Node.js Callbacks

Node.js callbacks are a special type of function passed as an argument to another function.

They're called when the function that contains the callback as an argument completes its execution, and allows the code in the callback to run in the meantime.

syntax of a callback in Node:

```
functionfunction_name(argument,callback)
```

How to Use Callbacks in Node

The callback is used to define what happens when the function containing the callback as an argument completes its execution.

For example, we can define a callback to print the error and result after the function execution.

```
functionfunction_name(argument,function(error, result)
{if(error)
{ console.log(error)}
                              else
{ console.log(result)}
}
)
```

You can write a callback as a regular function. You do that using a function keyword then the argument inside round brackets. Then you use curly brackets where you can define the callback body. It is not required to define the function name since it is automatically called.

Let's see an example of a callback using the setTimeout function. You can use this method to define a callback function that runs after a definite time.

```
setTimeout(function(){
console.log('Callback as Standard Function');
},1000);
```

Here we define a callback that runs after 1000 milliseconds which is equivalent to 1 second.

Let's rewrite the same example we wrote in the above section using an arrow function. Here we change the string to "Callback as Arrow Function".

```
setTimeout(()=>{
console.log('Callback as Arrow Function');
},1000);
```

Program : Write a script to display sum of two numbers which are passed as an argument in function named Total. the sum should display in any html element. Use callback as third argument in Total function.

```
<html>
<head>
</head>
<body>
<p id="demo"></p>
<script>        function
mycallback(sum)
        {
document.getElementById("demo").innerHTML="<b>"+ sum +"</b>";
        }
function total(value1,value2,callback)
        {
            sum= value1+value2; callback(sum);
        }
total(25,15,mycallback);
</script>
</body>
</html>
```

Program: Write a script to Initialize two variables and increment both the variables each time and display the addition of both the variables at interval of 1 second.

```
<html>
<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>
```

Program: Write a script to Display the text “LJU” on browser after 5 seconds

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

Program : Write Node.Js code that display "Hello" with increasing font-size in interval of 50 milliseconds in blue color. When font-size reaches to 50 pixel it should stop.

```
<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; if (size
<= 50) { size++;
}
}
setInterval(add, 1000);
</script>
</body>
</html>
```

Nodejs Global Module

Nodemon :

The nodemon Module is a module that develop node.js based applications by automaticallyrestarting the node application when file changes in the directory are detected. Nodemon doesnot 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 andagain.

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 yourapplication.

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 dependencyon 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:

- Read files
- Create files
- Update files
- Delete files
- Rename files

Every method in fs module has synchronous and asynchronous forms.

Asynchronous methods take a last parameter as completion function callback. Asynchronous method is preferred over synchronous method because it never blocks the program execution where as the synchronous method blocks.

Synchronous function for file access.

Read files -fs.readFileSync("filename" ,"utf-8")

Create files-fs.writeFileSync("filename", data)

Update files-fs.appendFileSync("filename",data)

Delete files- fs.unlinkSync("filename")

Rename files-fs.renameSync("oldfilename","newfilename")

Create directory: fs.mkdirSync("foldername")

Remove directory: fs.rmdirSync("foldername")

Asynchronous function for file access.

Read files -fs.readFile("filename","utf-8",callback)

Create files -fs.writeFile("filename",callback)

Update files-fs.appendFile("filename",callback)

Delete files -fs.unlink("filename",callback)

Rename files-fs.rename("oldfilename","newfilename",callback)

Let's take an example to create a JavaScript file named "main.js"

having the following code: **File: main.js** var ps = require("fs");

// Asynchronous read

```
ps.writeFile('input.txt', 'Hello World!', function (err) {
```

```
if (err) console.log(err); else

console.log('Write operation complete.');

});




ps.readFile('input.txt', function (err, data) {

   if (err) {          return

console.error(err);

   }

console.log("Asynchronous read: " + data.toString());

});
```

// Synchronous read

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

var data=ps.readFileSync("Hello.txt");
console.log(data.toString()); console.log("Program
ended");
```

Program: Write a Node.js program to CRUD operation of file management.

1)Create folder named "Hello".
2) Create file in it named abc.txt and enter data in to it.
3) Add more data at last in file.
4)Read data without getting buffer data at first.
5)rename file
6)Delete both file and folder.

```
var ps=require("fs"); ps.mkdirSync("Hello");
ps.writeFileSync("Hello/abc.txt","Hello world");
ps.appendFileSync("Hello/abc.txt ","Hi");
data=ps.readFileSync("Hello/abc.txt ",”utf-8”); console.log(data);
ps.renameSync("Hello/abc.txt "," Hello/readwrite.txt")
ps.unlinkSync("Hello/readwrite.txt");
```

Program: Write a Node.Js program to sort an interger array, where all element are available in a file separated by white space. Print sorted array elements on node.js server

```
var ps=require("fs");
ps.writeFileSync("task.txt","50 -1 -44 23 7");
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]);
}
console.log(data);
for(i=0;i<5;i++) {
for(j=0;j<5;j++)
{   if(data[i]<data[j])
{temp=data[i];
data[i]=data[j];
data[j]=temp;}
   }
}

console.log(data);
```

Program: 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","Todayis 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
```

Program: Write node.js script to copy content of one file to the other 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
```

Program : 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 } ]
```

Program :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 write in shape.txt 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
```

Program: 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);
constjsonstr=JSON.stringify(p3); var
ps=require("fs");
ps.writeFileSync("d2.txt",jsonstr);
```

Output:

```
person { age: 34, name: 'abc' }
```

Program: Write node.js script to create a class named timeand 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);

constjsonstr=JSON.stringify(t3); var
ps=require("fs");
ps.writeFileSync("time.txt",jsonstr);
```

Output:

Program: 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("\r");  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":"CE"}');
var data=fs.readFileSync("xyz.json");  var
data1=JSON.parse(data);  console.log(data1.name);
```

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");

Program: 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
```

Program: Write node.js script to create a folder named “AAAA” at temp folder. Also, create file named “temp1.txt” inside “AAAA” folder. Now, check your system is working on which platform . then print message “`You are working on windows 32 bit`” if your system is 32 bit , 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 pm=require("path");

program: 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

What is Web Server

Web Server is a software program that handles HTTTP requests sent by HTTP clients like web browsers, and returns web pages in response to the clients. Web servers usually respond with html documents along with images, style sheets and scripts.

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');
```

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:

Example

```
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(8080); //the server object listens on port 8080
```

The function passed into the `http.createServer()` method, will be executed when someone tries to access the computer on port 8080.

Save the code above in a file called "demo_http.js", and initiate the file:

Initiate demo_http.js:

```
C:\Users\Your Name>node demo_http.js
```

write : http://localhost:8080 on browser url and see the output

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('Hello World!');                               res.end();
                    }).listen(8180);
```

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.

Program: 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);
```

Program: 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);
```

Program: 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!");
```

Program: 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}

Program : Write a node.js script to jump on a specific code by specifying path on address bar of browser.

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

server=http.createServer(

                function(req,res)
                {
                  if(req.url=='/')

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

res.write("<h1>This is home page</h1>");                    res.end();

                  }

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

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

res.write("<h1>This is home page</h1>");                    res.end("thanks");

                  }

else

                  {
                     res.end("invalid request");

                  }

                }
).listen(5001);
```

Program: Write a script to define two JSON objects named as “division1” and “division2” having an array to store names of students. These names should be sorted alphabetically in the object and should be written to the file. At last, both division objects should be visible with names sorted alphabetically in file

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

test = {

"division1": {

"name":["Z","B","H"]

},
"division2": {

"name" :["Y","A","G"]

}
}

const div1_data = test.division1.name; const

div2_data = test.division2.name;

var combine_data = div1_data.concat(div2_data).sort();
ps.writeFileSync("json.txt",combine_data);
console.log(combine_data);
```

Program: Write node.js script to print “Welcome Admin” on home page of server. If user request for second page it display “This is second page” in italic font- style and if any other request is requested it shows “Page not found” message.

```
var http=require("http"); var server=http.createServer(

                    function(req,res)

                    {
```

```
                    if(req.url=='/')
                    {
                       res.writeHead(200,{"content-type":"text/plain"})
res.write("welcome admin");
res.end();
                    }
                    else if(req.url=='/second')
                    {   res.writeHead(200,{"content-type":"text/html"});
res.write("<i>This is second page</i>");                     res.end();
                    }
else
                    {
                       res.end("page not found");
                    }
                  }
).listen(5001);
```

Q-89 Write a node.js script to print 3 different JSON objects by specifying corresponding object name on address bar.

```
const http = require('http');
 var object1= { "name": "Object One", "type": "Type A", "value": 100 }
 var object2= { "name": "Object Two", "type": "Type B", "value": 200 }
 var object3= { "name": "Object Three", "type": "Type C", "value": 300 }

var server=http.createServer( function(req,res)
{
if(req.url=="/object1")
{
res.writeHead(200,{"content-type":"application/json"});
res.write(JSON.stringify(object1));
res.end();
}
else if(req.url=="/object2")
  {
  res.writeHead(200,{"content-type":"application/json"});
  res.write(JSON.stringify(object2));
  res.end();
  }
  else if(req.url=="/object3")
    {
    res.writeHead(200,{"content-type":"application/json"});
    res.write(JSON.stringify(object3));
    res.end();
    }
});
server.listen(3000);
```

Q-90 Write a node js script to write the text “You are creating a file” to help.txt file. After that append the text “you are appending data” to same help.txt file. After that read the file and print file contents on console. After finishing read operation , print the line “Thanks for using my program” on console. write ,append,read sequence must be maintain. all read ,write and append operations are asynchronous.

```
const fs = require('fs');
fs.writeFile('help.txt', 'You are creating a file\n', () => {
                    console.log('File created and written to.');
                    fs.appendFile('help.txt', 'You are appending data\n', () => {
                              console.log('Data appended to file.');
                              fs.readFile('help.txt', 'utf8', (err, data) => {
                                         if (err) {console.log(err)}
                                         else {console.log(data);}
                                        console.log('Thanks for using my program');
                                  });
                           });
                     });
```

Q-91 Write a node.js script to print 1st 10 prime numbers on browser in table of 10 rows. Odd no. of rows should render in blue color and even no. of rows should render in red color.

```
const http = require('http');
function isPrime(num) {
  if (num <= 1) return false;
  if (num <= 3) return true;
  if (num % 2 === 0 || num % 3 === 0) return false;
  for (let i = 5; i * i <= num; i += 6) {
    if (num % i === 0 || num % (i + 2) === 0) return false;
  }
  return true;
}
function generatePrimes(n) {
  const primes = [];
```

```
 let num = 2;
 while (primes.length < n) {
   if (isPrime(num)) {
     primes.push(num);
   }
   num++;
 }
 return primes;
}
const primes = generatePrimes(10);
const server = http.createServer((req, res) => {
 res.writeHead(200, { 'Content-Type': 'text/html' });
 let html = `
   <html>
     <head>
       <style>
         .odd { background-color: blue; color: white; }
         .even { background-color: red; color: white; }
         table { width: 50%; margin: auto; border-collapse: collapse; }
         td { padding: 10px; text-align: center; }
       </style>
     </head>
     <body>
       <h2 style="text-align: center;">First 10 Prime Numbers</h2>
       <table border="1">
 `;
 primes.forEach((prime, index) => {
   const rowClass = index % 2 === 0 ? 'odd' : 'even';
   html += `<tr class="${rowClass}"><td>${prime}</td></tr>`;
 });
```

```
  html += `
      </table>
    </body>
  </html>
  `;
  res.end(html);
}).listen(3000);
```

Q-94 Write a Node.Js program for following action 1.Write a file having five random elements separated by white space in .txt file. 2.append sorted array of these 5 elements in same file along with message : “Sorted array:” in new line. 3.Find maximum number from that and append with message “maximum number=” in same file.

```
var ps=require("fs");
 ps.writeFileSync("task.txt","50 -1 -44 23 7");
 data=ps.readFileSync("task.txt","utf-8");
 data=data.split(" ");
   for(i=0;i<data.length;i++)
    {    data[i]=parseInt(data[i]);
    }
    for(i=0;i<5;i++)
       { for(j=0;j<5;j++)
       { if(data[i]<data[j])
         {temp=data[i];
         data[i]=data[j];
         data[j]=temp;}
       }
    }
    ps.appendFileSync("task.txt",`\nSorted array:${data}`)
function findMax(numbers) {
  return Math.max(...numbers);
}
const maxNumber=findMax(data);
ps.appendFileSync("task.txt",`\nmaximum number=:${maxNumber}`)
```

Q-96 Write a function 'ArrayToObject' which takes in an array of arrays, and returns an object with each pair of elements in the array as a key-value pair and store the result in one arraytoobject.txt file. Input=[['Country', India'], ['State', 'Gujarat'], ['City', 'Ahmedabad']] Output= { Country : ' India ', State : ' Gujarat ', City : 'Ahmedabad' }

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

const input = [['Country', 'India'], ['State', 'Gujarat'], ['City', 'Ahmedabad']];

function ArrayToObject(arr) {

  const obj = {};

  arr.forEach(pair => {

    const [key, value] = pair;

    obj[key] = value;

  });

  return obj;

}

const output = ArrayToObject(input);

fs.writeFileSync('arraytoobject.txt', JSON.stringify(output));
```

Q-98 Write node.js script to print "Welcome to Home Page" with two links containing two pages named as "About Us" and "Contact Us" on home page of server. If user request for About Us page it should display "Welcome to LJ University" in bold font-style with blue color and if user request for Contact Us page it should display "Email:abc@ljinstitutes.edu.in" in italic font-style with red color if any other request is requested it shows "Page not found" message in plaintext.

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

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

  res.setHeader('Content-Type', 'text/html');

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

    res.writeHead(200);

    res.end(`

      <html>

        <body>

          <h1>Welcome to Home Page</h1>

          <ul>

            <li><a href="/about">About Us</a></li>
            <li><a href="/contact">Contact Us</a></li>
          </ul>
        </body>
      </html>
    `);
  } else if (req.url === '/about') {
    res.writeHead(200);
    res.end(`
      <html>
        <body>
          <h1 style="color: blue; font-weight: bold;">Welcome to LJ University</h1>
        </body>
      </html>
    `);
  } else if (req.url === '/contact') {
    res.writeHead(200);
    res.end(`
      <html>
        <body>
          <h1 style="color: red; font-style: italic;">Email: abc@ljinstitutes.edu.in</h1>
        </body>
      </html>
    `);
  } else {
    res.writeHead(404);
    res.end(`
      <html>
        <body>
          <h1>Page not found</h1>
        </body>
      </html>
    `);
  }
}).listen(3000);
```

Extra Program

Program: Write code to perform the tasks as asked below.

- **Add three buttons.**
- **Increase button to increase the fonts. It should stop increasing the fonts when the font size reaches to 200px or stop button is clicked.**
- **Decrease button to decrease the fonts. It should stop decreasing the fonts when the font size reaches to 20px or stop button is clicked.**
- **Stop button to stop increasing or decreasing the fonts.**
- **Increasing/decreasing interval is of 100 ms.**
- **(Default font size = 50px)**

```
<html>
<body>
<p id="p1" style="font-size: 50px;"> Hello</p>
<button onclick="inc()">Increase</button>
<button onclick="dec()">Decrease</button>
<button onclick="stop()">stop</button>
<script>
font=50;
function fun(font){
document.getElementById('p1').style.fontSize=font+'px';
}
function inc(){
test = setInterval(()=>{
if(font<100){ fun(++font);
}},100);
}
function dec()
{test =setInterval(()=>{
if(font>15){ fun(--font);}
},100);
}
```

```
function stop() { clearInterval(test); }

</script>

</body>

</html>
```

URL module

The URL module splits up a web address into readable parts.

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

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

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

Example:

Split a web address into readable parts:

```
var url = require('url');
var adr='http://localhost:8080/default.htm?year=2017&month=february'; var q =
url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
```

Program: Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error:

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

http.createServer(function (req, res) {
    var q = url.parse(req.url, true);
    var filename = "." +q.pathname;
   fs.readFile(filename, function(err, data)
        { if (err) {
          res.writeHead(404, {'Content-Type': 'text/html'});
          return res.end("404 Not Found");
       }
       res.writeHead(200, {'Content-Type': 'text/html'});
       res.write(data);
```

```
        return res.end();
    });
}).listen(8080);
```

Program: Write a nodejs program to load a html file on nodejs web server which is requested by client and print its 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"});

res.write(data);

res.end();

});

server.listen(6055);

console.log("Server Started");
```

Node.js NPM

What is NPM?

NPM is a package manager for Node.js packages, or modules if you like.

www.npmjs.com hosts thousands of free packages to download and use.

The NPM program is installed on your computer when you install Node.js

What is a Package?

A package in Node.js contains all the files you need for a module.

Modules are JavaScript libraries you can include in your project.

Download a Package

Downloading a package is very easy.

Open the command line interface and tell NPM to download the package you want.

I want to download a package called "upper-case":

Download "upper-case":

```
C:\Users\Your Name>npm install upper-case
```

Chalk Module

The **chalk** module is a third-party library that can be used for styling of texts. It allows the users to create their own themes in a Node.js project.

- This module helps the users to customize the response messages with different colors as per the preferences.
- It also improves the readability by providing colors and makes it easier to detect warnings and errors.

Installation

- Create one folder

- Set proxy if required: npm config set proxy http://192.168.10.252:808

- To install chalk: npm install chalk or npmi chalk

- After installation of chalk module, package-lock.json and package.json will be created.

- Add ["type": "module",] in package.json file as shown belowInpackage.json file:

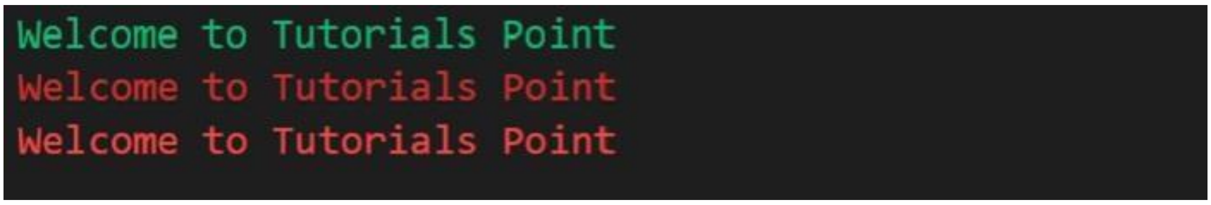
```
{
"type": "module",
"dependencies": {
"chalk": "^5.2.0"
}
```

Example 1

Create a file with the name "**chalk.js**" and copy the following code. After creating the file, use the command "**node chalk.js**" to run this code as shown in the example below −

```
// Importing the chalk module Import

chalk from 'chalk';



// Coloring different text messages console.log(chalk.green("Welcome to Tutorials

Point")) console.log(chalk.red.underline("Welcome to Tutorials Point"))

console.log(chalk.red.underline.bold("Welcome to Tutorials Point"))
```

Output



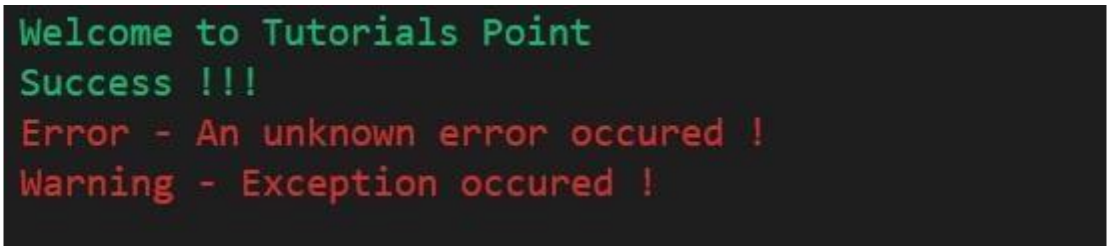
```
Welcome to Tutorials Point
Welcome to Tutorials Point
Welcome to Tutorials Point
```

Example 2

```
// Importing the chalk module Import

chalk from 'chalk';



// Coloring different text messages const

welcome=chalk.green;

const warning=chalk.red;



console.log(welcome("Welcome to Tutorials Point")) console.log(welcome("Success

!!!"))
```

```
console.log(warning("Warning - Exception occurred !"))
```

Output

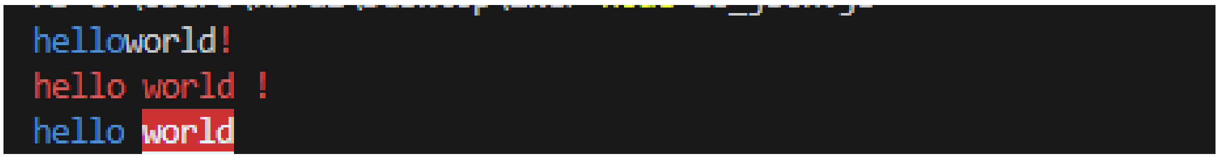


```
Welcome to Tutorials Point
Success !!!
Error - An unknown error occured !
Warning - Exception occured !
```

Example 3

```
import ch from "chalk";

console.log(ch.blue('hello') +'world' +ch.red('!'));

console.log(ch.red('hello','world','!')); console.log(ch.blue('hello',ch.underline.bgRed('world')));
```

OUTPUT



```
helloworld!
hello world !
hello world
```

validator Module

validator module is used to apply validation on data. validator module provide different functions for validate the data. These functions are Boolean functions, which returns answer in true or false.

For install:

```
Npm install validator
```

Example:

Import v from validator

Console.log(v.isEmail("hello@gmail.com");

Console.log(v.isHexadecimal("ABC");

Example1 : Check whether given email is valid or not

```
import v from "validator"

var email = 'test@gmail.com'

console.log(v.isEmail(email)) // true

email = 'test@'

console.log(v.isEmail(email)) // false
```

Example2 : Check whether string is in lowercase or not

```
import v from "validator"

var name = 'hello'

console.log(v.isLowercase(name)) // true

name = 'HELLO'

console.log(v.isLowercase(name)) // false
```

Example3: Check whether string is empty or not

```
import v from "validator"

var name = ''

console.log(v.isEmpty(name)) // true

name = 'hello'

console.log(v.isEmpty(name)) // false
```

Module Wrapper function

NodeJS does not run our code directly, it wraps the entire code inside a function beforeexecution. This function is termed as Module Wrapper Function. Before a module's code isexecuted, NodeJS wraps it with a function wrapper that has the following structure:

```
(function (exports, require, module, __filename, __dirname) {
//module code
});
```

The five parameters — exports, require, module, _filename, _dirname are available insideeach module in Node. Though these parameters are global to the code within a module yet theyare local to the module (because of the function wrapper as explained above).

These parametersprovide valuable information related to a module.

The variables like _filename and _dirname, that tells us the module's absolute filename andits directory path.

The meaning of the word 'anonymous' defines something that is unknown or has no identity. InJavaScript, an anonymous function is that type of function that has no name or we can say whichis without any name.

Example:

```
(
function()
{

console.log(__filename);
console.log(__dirname);
}
)();
Output:
C:\nodejs\ex1.js //returnd path of current file
C:\nodejs //returned path till current file (folder)
```

Own Module

Modules are the collection of JavaScript codes in a separate logical file that can be used in external applications on the basis of their related functionality. Modules are popular as they areeasy to use and reusable. To create a module in Node.js, you will need the exports keyword. This keyword tells Node.js that the function can be used outside the module.

Method 1

In test.js file:

```
const add=(a,b)=>

{return(a+b);

}

module.exports=add;
```

In another file where you want to use that module:

```
var d=require("./test.js");

console.log(d(10,15));
```

Method 2

In test.js file:

```
const sub=(a,b)=>

{return(a-b);

}

constmul=(a,b)=>

{return(a*b);

}

module.exports.s=sub;

module.exports.m=mul;
```

In another file:

```
var d1=require("./test.js");
```

```
console.log(d1.s(10,5));

console.log(d1.m(10,15));
```

Method 3

In test.js file:

```
const sub=(a,b)=>

{return(a-b);

}

constmul=(a,b)=>

{return(a*b);

}

module.exports.d2=sub;

module.exports.e2=mul;
```

In another file:

```
var {d2,e2}=require("./test.js");

console.log(d2(10,7));

console.log(e2(10,12));
```

Method 4

In test.js file:

```
const sub=(a,b)=>

{return(a-b);

}

constmul=(a,b)=>

{return(a*b);

}

const name="Hello"

module.exports={sub,mul,name};
```

In another file:

```
var {sub,mul,name}=require("./test.js");

console.log(sub(100,20));

console.log(mul(10,2));

console.log(name)
```

Program:

Write a node.js script to create calculator using external module having a function add(),sub(), mul(), div(). This function returns result of calculation. Write all necessary .js files.

ABC.js

```
exports.add = function (x, y)

{return x + y;

};

exports.sub = function (x, y)

{return x - y;

};

exports.mult = function (x, y)

 {return x * y;

};

exports.div = function (x, y)

 {return x / y;

};
```

calculate.js

```
const calculator = require('./ABC.js);

let x = 50, y = 20;
```

```
console.log("Addition of 50 and 20 is "+ calculator.add(x, y));
console.log("Subtraction of 50 and 20 is "+ calculator.sub(x, y));
console.log("Multiplication of 50 and 20 is "+ calculator.mult(x, y));
console.log("Division of 50 and 20 is "+ calculator.div(x, y));
```

Output:

```
Addition of 50 and 20 is 70
Subtraction of 50 and 20 is 30
Multiplication of 50 and 20 is 1000
Division of 50 and 20 is 2.5
```

Program:

Write a node.js script to find all prime no.s between 1-50 using external module having a function checkPrime(). This function returns Boolean value on the basis of a no. is prime or not prime. Write all necessary .js files.

1.js

```
constcheckPrime = (num) =>
{ var temp = 0
for(let i=2;i<num;i++)
{   if(num%i==0)
{     temp++;
    }
}
if(temp==0)
{return true;
}
else
{return false;
}
}
module.exports=checkPrime;
```

2.js

```
var checkPrime = require("./1.js")
for(i=1;i<=50;i++)
{
let x=checkPrime(i);
if(x==true)
{
console.log(i+" Prime Number");
}
else{
console.log(i+" Not a Prime Number")
}
}
```

Program:

Write a node.js script to create my own module to calculate reverse of a given number. That module should be use to compute all numbers between 1 to 100 in which square of reverse & reverse of sqaure is same. This has call of reverse twice so call it from module. Also keep a function to compute average of any number of elements

1.js

```
const rev=(n)=>
{   var r=0;
   r2=n*n;
   while(n>0)
   {
        r=(r*10)+(n%10);
        n=parseInt(n/10);
    }
    r3=r*r;
   return [r,r2,r3];
}
constavg=(x,y)=>
{
   var average=(x+y)/2;
   return average;
}
module.exports={rev,avg};
```

In another file:

```
var {rev,avg} =require("./1.js");
arr=rev(12);
// console.log(arr[0]);
// console.log(arr[1]);
// console.log(arr[2]);
fno=arr[2];
arr2=rev(arr[1]);
sno=arr2[0];
if(fno==sno)
{
    console.log("Equal");
}
else
```

```
{
console.log("not equal");
}
console.log(avg(5,15));
```

Events

If you worked with JavaScript in the browser, you know how much of the interaction of the user is handled through events: mouse clicks, keyboard button presses, reacting to mouse movements, and so on.

On the backend side,Node.js offers us the option to build a similar system using the events module.

events module, offers the EventEmitterclass, which we'll use to handle our events. You

initialize that using

```
constEventEmitter = require('events'); consteventEmitter
= newEventEmitter();
```

This object exposes, among many others, the onand emitmethods.

- emitis used to trigger an event
- onis used to add a callback function that's going to be executed when the event is triggered

For example, let's create a startevent, and as a matter of providing a sample, we react to that by just logging to the console:

```
eventEmitter.on('start',()=>{ console.log('started');
});
```

When we run

```
eventEmitter.emit('start');
```

the event handler function is triggered, and we get the console log.

You can pass arguments to the event handler by passing them as additional arguments to emit():

```
eventEmitter.on('start',number=>{ console.log(`started
${number}`);
});
```

```
eventEmitter.emit('start',23);
```

Multiple arguments:

```
eventEmitter.on('start',(start, end)=>{ console.log(`started from ${start} to
${end}`);
});
eventEmitter.emit('start',1,100);
```

Methods

Sr.No.	Method & Description
1	**addListener(event, listener)** Adds a listener at the end of the listeners array for the specified event.
2	**on(event, listener)** Adds a listener at the end of the listeners array for the specified event.
3	**removeListener(event, listener)** Removes a listener from the listener array for the specified event.
4	**emit(event, [arg1], [arg2], [...])** Execute each of the listeners in order with the supplied arguments. Returns true if the event had listeners, false otherwise.
5	**listenerCount(emitter, event)** Returns the number of listeners for a given event.

Example

Create a js file named main.js

```
var EventEmitter=require("events"); var ee=new
EventEmitter();
var connectHandler=function ()
  {
        console.log("connection succesfully"); ee.emit("data_received");
  }
  ee.on("connection",connectHandler); ee.on("data_received",function(){console.log("data received
  succesfully");}); ee.emit("connection");
  console.log("thanks");
  ee.listenerCount("connection");
```

Program 1

Write a node js script to create two listeners for a common event. Call their respective callbacks. Print number of events associated with an emitter. remove one of the listeners and call remaining listeners again. print number of remaining listeners also.

```
var e=require("events");
var ee=new e.EventEmitter(); var
connectHandler1=function ()
  {
        console.log("connection 1 succesfully");
          }
  var connectHandler2=function ()
  {
        console.log("connection 2 succesfully");


  }
  ee.on("connection",connectHandler1);
  ee.on("connection",connectHandler2); ee.emit("connection");
  console.log(ee.listenerCount("connection"));
  ee.removeListener("connection",connectHandler1);
  console.log(ee.listenerCount("connection"));
```

Program 2:

Write a node js script to write the text “This is data” to new.txt file. After that append the text “That is data” to same new.txt file. After that read the file and print file contents on console. After finishing read operation , print the line “Thanks for using my program” on console. write ,append,read sequence must be maintain. all read ,write operations are asynchronous. Use event handling.

```
var ps=require("fs"); var
e=require("events");
var ee=new e.EventEmitter();

var write_f=function ()
  {  ps.writeFile("new.txt","This is data",(err)=>
        {
                console.error();
        });

  }
  ee.on("write_file",write_f); var
  append_f=function ()
  { ps.appendFile("new.txt","That is data",(err)=>
        {
              console.error();
              ee.emit("read_file");
        });

ee.on("append_file",append_f); var
read_f=function ()
{  ps.readFile("new.txt","utf-8",(err,data)=>
        {
              console.log(data);
              ee.emit("thanks");
        });
}
ee.on("read_file",read_f); var
thank=function ()
  {
        console.log("Thanks for using my program");
  }

  ee.on("thanks",thank);
  ee.emit("write_file");
```

Program:
Write node js script to handle events as asked below.
1) Check the radius is negative or not. If negative then display message "Radius" must bepositive" else calculate the perimeter of circle.
2) Check side is negative or not. If negative then display message "Side must be positive"else calculate the perimeter of square.

```
var e = require("events"); var ee =
new e();
const radiushandler = () => { console.log("Radius must be
positive");
}

ee.on("negside", sidehandler = () => { console.log("Side
must be positive");
});

const findval = (r,s) =>
{
        {
        ee.emit("negradius");
        }
        else{
                var rperi = 2 * 3.14 * r;
                console.log(rperi);
                 }
            if (s < 0)
         {
                ee.emit("negside");
                }
        else{
                var speri = 4*s; console.log(speri);
        }
}

ee.on("negradius", radiushandler);
ee.on("findval", findval); ee.emit("findval",-2,-
3);
```

Program:
Write node js script to handle event of write a data in file, append data in file and then readthe file and display data in console.

```
var fs=require("fs"); var
ee=new e();
ee.on("connection",function()
{
fs.writeFile("b.txt","This is data",(err)=> {console.log()}); console.log("File Written");
ee.emit("data-append");
ee.emit("data-read");
});
ee.on("data-append",function()
{
fs.appendFile("b.txt","That is data",(err)=> {console.log()}); console.log("File Appended");
});
ee.on("data-read",function()
{
fs.readFile("b.txt",(err,data)=>
{
if(err){ console.error(err);
}
console.log(data.toString());
});
});
ee.emit("connection");
```

Program

Write a node.js script using event handling to consider an errorneous triangle to find area. Take fix values of all three sides.

(1) If any of the side is negative, then print the message “Sides must be positive” using event handler.
(2) If perimeter of triangle is negative then print the message “Perimeter must be positive” using event handler.
(3) Both above messages must be printed in sequence

```
var e = require("events"); var ee =
new e();
const findval = (s1,s2,s3) =>
{
        if (s1< 0||s2<0||s3<0)
        {
              ee.emit("negside");
        }

                var peri = s1+s2+s3; if(peri<0)
                {
                        ee.emit("negperi");
                }
                else
              { console.log("perimeter="+peri);
                }

}

ee.on("negside",()=>{console.log("side must be positive")}); ee.on("negperi",
()=>{console.log("perimeter must be positive")}); ee.on("findval",findval);
ee.emit("findval",2,-3,8);
```

JSON Processing in Node JS

If you have JSON data as part of a string, the best way to parse it is by using the JSON.parse method that’s part of the JavaScript standard since ECMAScript 5, and it’s provided by V8, the JavaScript engine that powers Node.js.

Program :Defining an array of objects 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);
```

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

```
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));
```

Program: Write a Node.js script to create a class student by assigning name & result in form of members. Create one member function named as topper of X which returns topper student object. Details of this topper student should be printed on file as well as on console

```
var ps=require("fs"); class
person
{
        constructor(name,result)
        {
                this.result=result; this.name=name;
        }

        topperofx(p1,p2)
{
        if(p1.result>p2.result)
        {
                console.log("P1.result="+p1.result); return p1;
        }

        else
        {
                console.log("P2.result="+p2.result); return p2;
        }

}
}
var   p1=new   person("xyz",90);   var
p2=new      person("pqr",80);      var
p3=new person();
var a=p3.topperofx(p1,p2);
const jsonstr=JSON.stringify(a); console.log(jsonstr);
ps.writeFileSync("topper.txt",jsonstr);
```

Program
Write a Node.js script to create a class person by assigning name & age in form of members. Create one member function named as elder of X which returns elder person object. Details of this elder person should be printed on file as well as on console.

```
var ps=require("fs");
class person
{
        constructor(name,age)
        {
                this.age=age; this.name=name;
        }

elderofx(p1,p2)
{
        if(p1.age>p2.age)
        {
                console.log("P1.age="+p1.age); return p1;
        }

        else
        {
                console.log("P2.age="+p2.age); return p2;
        }

}
}
Var p1=new person("xyz",3); var
p2=new person("pqr",33);
```

```
var p3=new person();
var a=p3.elderofx(p1,p2);
const jsonstr=JSON.stringify(a);
console.log(jsonstr);
ps.writeFileSync("elder.txt",jsonstr);
```

Program

Write a node.js script to create a class time & assign members hour, minute & second. Create two objects of time class & add both the time objects so that it should return is third time object. Third time object should have hour, minute & second such that after addition if second exceeds 60 then minute value should be incremented. If minute exceeds 60 then hour value should be incremented.

```
Class time
{
        constructor(hour,min,sec)
        {
                this.hour=hour; this.min=min;
                this.sec=sec;
        }

addition(p2)
{ var p3=new time();
      p3.hour=p2.hour+p1.hour;
      p3.min=p2.min+p1.min;
      p3.sec=p2.sec+p1.sec; return p3;
}
}
var p1=new time(5,5,20); var
p2=new time(2,58,55); var
a=p1.addition(p2); if(a.sec>60)
{ a.sec=a.sec%60;
a.min=a.min+1;
}
if(a.min>60)
{
                a.min=a.min%60;
                a.hour=a.hour+1;
}

console.log(a);
```

Q-137: Writing data to file, appending data to file and then reading the file data using using ES6 callback.

```
fs = require("fs")
fs.writeFile('test1.txt', 'Hello World!', function (err) {
if (err) { console.log("Error Generated"+err); }
else{
fs.appendFile('test1.txt', '\nGood Morning!', function (err) {
if (err){ console.log("Error Generated"+err); }
else{
fs.readFile('test1.txt',"utf-8", (readErr, data) => {
if (readErr) console.log("Error Generated: "+readErr)
console.log(data);
});
}
});
}
});
```

Q-138:Write a nodejs program load a simple html file from static url on nodejs s erver and print its content as html content.

```
var h=require("http");
var fs=require("fs");
var url=require("url");
var addr="http://localhost:6051/7.html";
var server=h.createServer(
function(req,res)
{
var q=url.parse(addr,true);
data=fs.readFileSync("."+q.pathname);
res.writeHead(200,{"content-type":"text/html"});
res.write(data);
res.end();
});
server.listen(5000);
```

Program: Write a nodejs program load a simple html file on nodejs web server and print its content as html content. (By requesting url from browser)

```
var h=require("http");
var fs=require("fs");
var url=require("url");
var server=h.createServer(
function(req,res)
{
var q=url.parse(req.url,true);
data=fs.readFileSync("."+q.pathname);
res.writeHead(200,{"content-type":"text/html"});
res.write(data);
console.log(data);
res.write("<h1> hello</h1>")
res.end();
});
server.listen(5000);
```

Q-156 Calculate following series for given values of x and n. x and n are any positive integers statically. Ans=1-(x/1!)+(x2 /2!)-(x3 /3!)+…..+(xn /n!) Create separate module function to compute factorial used in denominator.

// factorial.js

```
module.exports = function factorial(n) {
  if (n === 0 || n === 1) {
    return 1;
  }
  return n * factorial(n - 1);
};
```

// series.js

```
const factorial = require('./factorial');
```

```
});

function calculateSeries(x, n) {
  let result = 1;
  for (let i = 1; i <= n; i++) {
    const term = Math.pow(x, i) / factorial(i);
    result += i % 2 === 0 ? term : -term;
  }
  return result;
}
const x = 5;
const n = 5;
const result = calculateSeries(x, n);
console.log(`The result of the series for x=${x} and n=${n} is: ${result}`);
```

Q-157 Write a nodeJS script to fire an event named calculate which calculates the total marks of 5 subjects about of 25 marks and displays the total marks on console as an output.The calculate event fires another event name percentage which takes total marks as argument and percentage should get displayed in console.

```
const EventEmitter = require('events');
const marksEmitter = new EventEmitter();
marksEmitter.on('calculate', (a,b,c,d,e) => {
  const totalMarks = a+b+c+d+e
  console.log(`Total marks: ${totalMarks}`);


  marksEmitter.emit('percentage', totalMarks);
});


marksEmitter.on('percentage', (totalMarks) => {
  const maxMarks = 5 * 25;
  const percentage = (totalMarks / maxMarks) * 100;
  console.log(`Percentage: ${percentage.toFixed(2)}%`);
});
marksEmitter.emit('calculate', 23,20,22,21,20);
```

Q-159 Write a function that takes an array of numbers as input and returns the sum of all the numbers in the array after 3 seconds.

```
function sumArray(numbers, callback) {
  setTimeout(() => {
    let sum = 0;
    numbers.map(Number).forEach(num => {
      sum += num;
    });
    callback(sum);
  }, 3000);
}
const numbers = [10, 20, 30, 40, 50];
sumArray(numbers, (result) => {
  console.log(`The sum of the array is: ${result}`);
});
```

Q-160 Write a node.js script using Event handling to perform following tasks in sequence: a) Create file in it named abc.txt and enter data into it. b) Append data to that file abc.txt and print message "Data Appended Successfully". c) Read the content of the file abc.txt and print the content on http web server. d) Do all the operations of File using asynchronous file system module. And Lastly print the message "All operations performed successfully" on console.

```
const fs = require('fs');
const http = require('http');
const EventEmitter = require('events');
const eventEmitter = new EventEmitter();


// Task 1: Create a file named abc.txt and enter data into it
eventEmitter.on('createFile', () => {
  fs.writeFile('abc.txt', 'Initial data\n', () => {
        eventEmitter.emit('appendFile');
  });
});
```

```
// Task 2: Append data to abc.txt and print message
eventEmitter.on('appendFile', () => {
  fs.appendFile('abc.txt', 'Appended data\n', () => {
      console.log('Data Appended Successfully');
       eventEmitter.emit('readFile');
  });
});


// Task 3: Read content of abc.txt and print on HTTP server
eventEmitter.on('readFile', () => {
  fs.readFile('abc.txt', 'utf8', (err, data) => {
    http.createServer((req, res) => {
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end(data);
    }).listen(8080);
      console.log('All operations performed successfully');
  });
});
eventEmitter.emit('createFile');
```

QB-161 Write node js script to fetch values from url given below and display output as asked. "https://www.google.com/exam.txt?c1=Hello&c2=FSD2+T1+Test&c3= Welcome+to+LJU#AllTheBest" 1) Data must be written as below in file named "exam.txt". File name must be fetched from the url given above. Output: Hello! Welcome to LJU FSD2 T1 Test #AllTheBest 2) Read content from file "exam.txt" and send response to server and display data in "/" page in same format as above but in H1 tag and in red color. 3) If any other page is requested it shows "Page not found" message in plain text.

```
const http = require('http');
const fs = require('fs');
const url = require('url');
const urlPath =
"https://www.google.com/exam.txt?c1=Hello!&c2=FSD2+T1+Test&c3=Welcome+to+LJU#AllTheBest"
;
const reqUrl = url.parse(urlPath, true);
```

```
//console.log(reqUrl);

fs.writeFileSync("."+reqUrl.pathname,reqUrl.query.c1+"\n"+reqUrl.query.c3+" "+
reqUrl.query.c2+"\n"+reqUrl.hash)

var data=fs.readFileSync("."+reqUrl.pathname,"utf-8");

var array=data.split("\n");

var data1='';

for(var i=0;i<array.length;i++)

  {

    data1=data1+array[i]+"<br>";

  }

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

            if(req.url=='/')

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

             res.end(`<h1 style="color: red;">${data1}</h1>`);

             }

             else{

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

              res.end("Page not found")

             }


}).listen(3000);
```

Q-162 Create HTTP webpage on which Home page display “Welcome to Log in page” in blue color and font size must be 32px, Login page shows one HTML file from static URL having Form with detail for Username, Password, submit and reset button, Gallery page reflect one Image “hello.jpg” and any other page shows “Page Not found”. Write all necessary files to perform task. (Image already exist in same folder)

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

const fs = require('fs');

const url = require('url');

var addr="http://localhost:3000/7.html";

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

            if(req.url=='/')
            {
            res.end(`<p style="color:blue;font-size:32px">Welcome to log in page</p>`);
            }
            else if(req.url=='/Login'){
             var q=url.parse(addr,true);
               data=fs.readFileSync("."+q.pathname);
               res.writeHead(200,{"content-type":"text/html"});
               res.write(data);
               res.end();
               }
             else if(req.url=='/Gallery')
               { data=fs.readFileSync("."+"/hello.jpg");
             res.writeHead(200,{"content-type":"image/jpeg"});
             res.write(data);
             res.end();
               }
 }).listen(3000);
```

7.html file

```
<html>
<head>
   </head>
<body>
   <form>
      Username:<input type="text"/><br/>
      password:<input type="text"/><br/>
      <input type="submit"/>
      <input type="reset"/>
   </form>
</body>
</html>
```