Node JS-1

Introduction

What is Node.js?

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

Why Node.js?

Node.js uses asynchronous programming!

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

Here is how Node.js handles a file request:

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

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

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

What Can Node.js Do?

- Node.js can generate dynamic page content.
- Node.js can create, open, read, write, delete, and close files on the server.

- Node.js can collect form data.
- Node.js can add, delete, modify data in your database.

What is a Node.js File?

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

Setup

Download Node.js

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

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

```
PS D:\NODE JS> npm -v

9.5.0

PS D:\NODE JS> node -v

v18.14.2
```

REPL

REPL stands for

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

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

It performs the following tasks −

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

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

REPL Commands

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

Starting REPL

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

```
$ node
```

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

```
PS C:\Users\priyen> node
Welcome to Node.js v18.15.0.
Type ".help" for more information.
>
```

> (> indicates that you are in REPL Node)

Simple Expression

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

```
$ node
> 1 + 3
4
> 1 + ( 2 * 3 ) - 4
3
>
```

Use Variables

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

```
$ node
```

```
> x = 10
10
> var y = 10
Undefined   =➔  ( repl.repl.ignoreUndefined = true) will ignore undefined error.
> x + y
20
> console.log("Hello World")
Hello World
undefined
```

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

Multiline Expression

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

```
$ node
> var x = 0
undefined
> do {
  ... x++;
  ... console.log("x: " + x);
  ... }
while ( x < 5 );
x: 1
x: 2
x: 3
x: 4
x: 5
undefined
>
```

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

Underscore Variable

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

```
$ node
> var x = 10
```

```
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

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

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

```
const fun=(a,b)=>
{console.log("Hello");
return a+b;
}
console.log("Addition is =",fun(10,20));
```

//Output:

Hello

Addition is = 30

Undefined

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');

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

Synchronous-Blocking

Example of File system with blocking(Synchronous) concept

```
var ps=require("fs");
ps.writeFileSync("Hello.txt","Hello World")
var data=ps.readFileSync("Hello.txt");
//console.log(data)
console.log(data.toString());
console.log("Program ended");
```

- **Output:**

```
Hello World
Program ended
```

Task-1

Write node Example with File system methods.

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

```
var ps=require("fs");
ps.mkdirSync("node");
ps.writeFileSync("node/write.txt","Hello");
ps.appendFileSync("node/write.txt","Hi");
data=ps.readFileSync("node/write.txt");
console.log(data);
console.log(data.toString());
ps.renameSync("node/write.txt"," node/readwrite.txt")
ps.unlinkSync("node/readwrite.txt");
```

Task-2

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

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

ps.writeFileSync("TextFiles/source.txt","Hello Mitesh \n");

ps.appendFileSync("TextFiles/dest.txt"," How Are You");

data=ps.readFileSync("TextFiles/dest.txt","utf-8");

ps.appendFileSync("TextFiles/source.txt",data);

data1=ps.readFileSync("TextFiles/source.txt","utf-8");

ps.writeFileSync("TextFiles/new.txt",data1);
```

Task-3

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");
console.log(data);
b=JSON.parse(data);
console.log(b);
```

Task-4

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

//string format

```
var ps=require("fs");
ps.writeFileSync("s1.txt","50 -1 99 20 0 56 78 59");
data=ps.readFileSync("S1.txt","utf-8");
data=data.split(" ");
data.sort();
console.log(data);
Output:
[
  '-1', '0',  '20',
  '50', '56', '59',
  '78', '99'
]
```

//integer format

```
var ps=require("fs");
ps.writeFileSync("task.txt","0 1 99 20 33 -44 50");
data=ps.readFileSync("task.txt","utf-8")
  console.log(data);
data=data.split(" ");
  console.log(data);
for(i=0;i<data.length;i++)
{
    data[i]=parseInt(data[i]);
}
d1=data.sort();
```

```
for(x=0;x<data.length;x++)
{
    for(y=0;y<data.length-1;y++)
    {
        if(data[y]>data[y+1])
        {
           temp=data[y];
           data[y]=data[y+1];
           data[y+1]=temp;

        }
    }
}
//d1=data.sort();
  console.log(data);
  ps.writeFileSync("sorted.txt",JSON.stringify(data));
```

```
  console.log(d1);
```

Task-5

- **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' }
```

Asynchronous Programming Using Callbacks

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

There are some cases that code runs (or must run) after something else happens and also not sequentially. This is called asynchronous programming.

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

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

How to create a Callback?

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

```
const message = function() {
   console.log("This message is shown after 3 seconds");
}
setTimeout(message, 3000);
```

There is a built-in method in JavaScript called “setTimeout”, which calls a function or evaluates an expression after a given period of time (in milliseconds). So here, the “message” function is being called after 3 seconds have passed. (1 second = 1000 milliseconds)
In other words, the message function is being called after something happened (after 3 seconds passed for this example), but not before. So the message function is an example of a callback function.

JavaScript setInterval() Method: The setInterval() method repeats a given function at every given time interval.
JavaScript setTimeout() Method: This method executes a function, after waiting a specified number of milliseconds.

What is an Anonymous Function?
Alternatively, we can define a function directly inside another function, instead of calling it. It will look like this:

```
setTimeout(function() {
   console.log("This message is shown after 3 seconds");
}, 3000);
```

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

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

```
setTimeout(() => {
   console.log("This message is shown after 3 seconds");
}, 3000);
```

Example of File system with Non blocking(ASynchronous) concept
By using callbacks, we can write asynchronous code in a better way. The following example creates a new file called test.txt and writes "Hello World" into it asynchronously.

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

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

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

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

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

Output:

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

Some callback examples

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

Output: LJU after 5 sec.

```
<html>
<head>
</head>
<body>
   <p id="demo"></p>
   <script>
      function mydisplay(sum)
      {
         document.getElementById("demo").innerHTML="<b>"+ sum +"</b>";
      }
      function mycals(num1,num2,mycallback)
      {
         sum=num1+num2;
         mycallback(sum);
      }
      mycals(13,15,mydisplay);
   </script>
</body>
</html>
```

Output : 28

```
<html>
   <head>

   </head>
   <body>
      <p id="p1"></p>
      <script>
      function add(a,b)
      {
         obj=document.getElementById("p1");
         obj.innerHTML=(a+b);
      }
      a=2;
      b=5;
      setInterval(
         function()
         {
            add(++a,++b);
         },1000
      );
   </script>
   </body>
</html>


Output : Display 9 and then incremented
```

Task

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

```
<html>
   <body>
      <p id="demo" style="color:blue"></p>
      <script>
         size = 15;
         function add() {
                obj = document.getElementById("demo");
                obj.innerHTML = "hello";
                obj.style.color ="Blue";
                obj.style.fontSize = size;
                if (size <= 50) {
                    size++;
                }
            }
            setInterval(add, 1000);
        </script>
    </body>
</html>
```

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

```
<html>
    <head>
        <style>
            p{
                color:blue;
            }
        </style>
    </head>
    <body>
        <p id="p1"> Hello</p>
        <button onclick="fun2()">font-size</button>
        <script>
        font="2";
        function fun(font)
        {
           document.getElementById("p1").style.fontSize=font;
        }
        function fun2()
        {
            setInterval(
                function()
                {
                    if(font<=50)
                    {
                        fun(font++);
                    }
            },50
          );
        }
    </script>
    </body>
</html>
```

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

```
var fs=require("fs");
fs.writeFile("abc.txt","Today is a good day",(err)=> {if(err){console.log("completed")}}});
fs.appendFile("abc.txt","Today is a good day",function(err)
{
  if(err){console.log("completed")
}
});
fs.readFile("abc.txt",(err,data)=>{
if(err){
  console.error(err);
}
console.log(data.toString())
});
console.log("File Operations ended")
```

Output:

```
File Operations ended
Today is a good dayToday is a good day
```

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

```
const shape =
  [
    {
      name: "circle",
      diameter: 8
    },
    {
      name: "square",
      side: 10
    }
    ]
var ps=require("fs");

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

b=JSON.parse(data);

if( b[0].name == 'circle'){
    var perimeter = (b[0].diameter/2) * 3.14 * 2 ;
    console.log(perimeter);
}
if ( b[1].name == 'square'){
    var peri = (b[1].side) *4  ;
    console.log(peri);
}
ps.appendFileSync("shape.txt","\nPerimeter of circle = "+ JSON.stringify(perimeter)+ "\nPerimeter of square
= "+JSON.stringify(peri));
```

Output:

```
[{"name":"circle","diameter":8},{"name":"square","side":10}]
Perimeter of circle = 25.12
Perimeter of square = 40
```

OS Module : Operating System

Get information about the computer's operating system:

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

Example:

```
os=require("os");
console.log(os.arch());
console.log(os.hostname());
console.log(os.platform());
console.log(os.tmpdir());
console.log(os.freemem());
a1=os.freemem();
console.log(`${a1/1024/1024/1024}`);
```

Output:

```
x64
SYCEIT309A-115
win32
C:\Users\foram\AppData\Local\Temp
298242048
0.2777595520019531
```

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

```
var ps=require("fs");
var os=require("os");
console.log(os.arch());
console.log(os.hostname());
console.log(os.platform());
console.log(os.tmpdir());
f = os.tmpdir();
freemem=os.freemem()/1024/1024/1024;
ps.mkdirSync(f+"/AA");
```

```
if(freemem > 1){
ps.writeFileSync(f+"/AA/temp.txt","Sufficient memory")
}
else{
   ps.writeFileSync(f+"/AA/temp.txt","Low memory")
}
```

Output:

```
x64
ITICT406-182
win32
C:\Users\LJIET\AppData\Local\Temp
```

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

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

console.log(os.platform());
f = os.tmpdir();
p = os.platform();

if(p ==  "win32"){
ps.writeFileSync(f+"/AAAA/temp1.txt","You are working on windows 32 bit")
}
else{
   ps.writeFileSync(f+"/AAAA/temp.txt","You are working on windows 64 bit")
}
```

Path Module

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

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

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

Example:

```
var pm=require("path");
path1=pm.dirname("D:/FSD-2/node/addon.txt");
  console.log("Path: " + path1);
path2=pm.extname("D:/FSD-2/node/addon.txt");
  console.log("Extension: "+path2);
path2=pm.basename("D:/FSD-2/node/addon.txt");
  console.log("Basename: "+ path2);
path2=pm.parse("D:/FSD-2/node/addon.txt");
  console.log(path2);
  console.log(path2.root);
  console.log(path2.dir);
  console.log(path2.base);
  console.log(path2.ext);
  console.log(path2.name);
```

Output:

```
Path: D:/FSD-2/node
Extension: .txt
Basename: addon.txt
{
  root: 'D:/',
  dir: 'D:/FSD-2/node',
  base: 'addon.txt',
  ext: '.txt',
  name: 'addon'
}
D:/
D:/FSD-2/node
addon.txt
.txt
addon
// PS D:\Priyen_Patel>
```

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

```
var pm=require("path");
path=pm.dirname("D:/LJ/abc.html");
console.log(path);
path=pm.basename("D:/LJ/abc.txt");
console.log(path);
ext = pm.extname("D:/LJ/abc.txt")
console.log(ext);
path=pm.parse("D:/LJ/abc.html");
console.log(path);

if(path.ext == ".txt"){
   console.log("Text Document");
}else{
   console.log("Not a text Document");
}
```

Output:

```
D:/LJ
abc.txt
.txt
{
  root: 'D:/',
  dir: 'D:/LJ',
  base: 'abc.html',
  ext: '.html',
  name: 'abc'
}
Not a text Document
```

HTTP Module

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

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

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

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

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

Example1:

```
var http = require('http');
var server = http.createServer(                    //create a server object
function (req, res) {
  res.write('Hello World!');           //write a response to the client
  res.end();                           //end the response can be empty or include string
}).listen(8080);                       //the server object listens on port 8080
//(or server.listen(5051) instead of listen());
```

Output on http://localhost:8080/
Hello World!

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

```
var h=require("http");
var server=h.createServer(
   function(req,res)
   {
if(req.url=="/")
{
   res.writeHead(200,{"content-type":"text/html"});
   res.write("<b> Home page </b>");
   res.end();
}
else if(req.url=="/student")
{
   res.writeHead(200,{"content-type":"text/plain"}); //plain shows code as it is
   res.write("<i> Home page1 </i>");
   res.end();
}
else
{
   res.writeHead(404,{"content-type":"text/html"});
   res.write("<h1> Page Not found </h1>");
   res.end("Thanks");
  res.write("Bye");

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

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

```
var http=require("http");
var server=http.createServer(
   function(req,res)
```

```
    {
    if(req.url=="/")
    {
const a={"Name":"ABC", "Age":35};
        res.writeHead(200,
        {"content-type":"application/json"
        });
        res.write("Thank you..!");
        res.write(JSON.stringify(a));
        res.end();
    }
});
server.listen(6008);
```

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

After URL Module

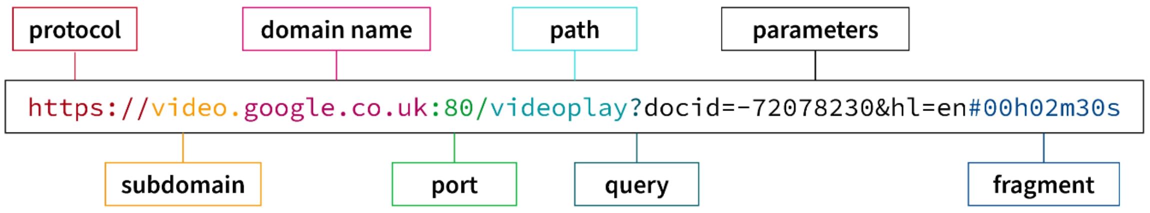
Write a nodejs program load a simple html file on nodejs web server and print its content as html content.

```
var h=require("http");
var ps=require("fs");
var u=require("url");
var addr="http://localhost:8080/16.html";
var q=u.parse(addr,true);
data=ps.readFileSync("."+q.pathname);
var server=h.createServer(
    function(req,res)
    {
    res.writeHead(200,{"content-type":"text/html"});
    //res.writeHead(200,{"content-type":"text/plain"}); gives content of file(Whole program
will display in port)
    res.write(data);
    res.end();

});
server.listen(6051);
```

URL module

The URL module contains functions that help in parsing a URL. In other words, we can split the different parts of a URL easily with the help of utilities provided by the URL module.


protocol
domain name
path
parameters
https://video.google.co.uk:80/videoplay?docid=-72078230&hl=en#00h02m30s
subdomain
port
query
fragment

The syntax for including the url module in your application:

var url=require("url");

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

url.parse() – This method takes the url string as a parameter and parses it. The url module returns an object with each part of the url as property of the object.

Syntax of url.parse() :
url.parse(url_string, parse_query_string, slashes_host)

Description of the parameters :

- **url_string :** <string> It is the URL string.
- **parse_query_string :** <boolean> It is a boolean value. By default, its value is false. If it is set to true, then the query string is also parsed into an object. Otherwise, the query string is returned as an unparsed string.
- **slashes_host :** <boolean> It is a boolean value. By default, its value is false. If it is set to true, then the token in between // and first / is considered host.

Example

- **LocalHost as URL**

var u=require("url");

```
var addr="http://localhost:8080/default.htm?year=2017&month=february";
var q1=u.parse(addr,true);
    console.log(q1);
    console.log(q1.host);
    console.log(q1.pathname);
    console.log(q1.query);
```

Output:

```
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'localhost:8080',
  port: '8080',
  hash: null,
  search: '?year=2023&month=may',
  query: [Object: null prototype] { year: '2023', month: 'may' },
  pathname: '/default.htm',
  path: '/default.htm?year=2023&month=may',
  href: 'http://localhost:8080/default.htm?year=2023&month=may'
}
localhost:8080
/default.htm
[Object: null prototype] { year: '2023', month: 'may' }
```

- **Google Search Link as URL**

```
var u=require("url");
var adr1="https://www.google.com/search?q=good+morning";
var q=u.parse(adr1,true); //query will be given as JSON Object
 console.log(q);
```

Output:

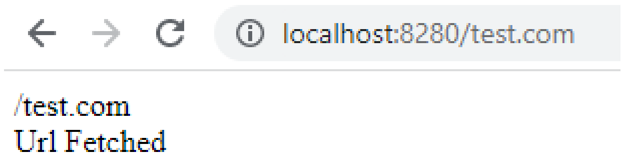
```
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'www.google.com',
  port: null,
  hostname: 'www.google.com',
  hash: null,
  search: '?q=good+morning',
  query: [Object: null prototype] { q: 'good morning' },
  pathname: '/search',
  path: '/search?q=good+morning',
  href: 'https://www.google.com/search?q=good+morning'
}
```

Read the URL

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

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

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



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

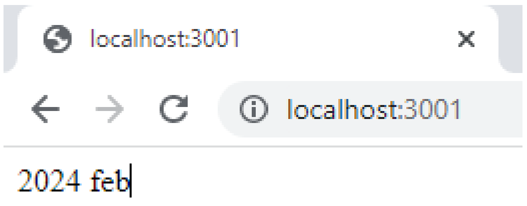
Query string

Get the details from query string

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

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

```
var http = require('http');
var url = require('url');
var addr="http://localhost:8080/default.html?year=2024&month=feb";
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  /*Use the url module to get the querystring*/
  var q = url.parse(addr, true).query;
  /*Return the year and month from the query object:*/
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(3001);
```

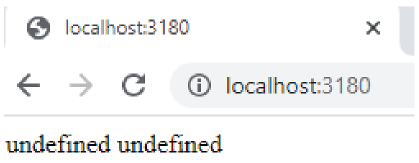


```
2024 feb
```

2) Make changes in url at browser and request url to display data.

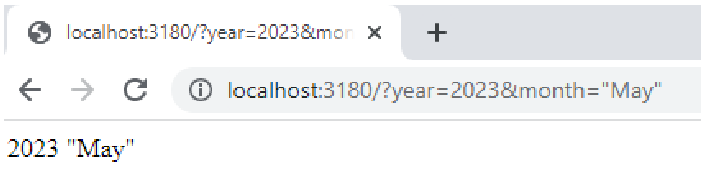
```
var http = require('http');
var url = require('url');

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);
```


localhost:3180

localhost:3180

undefined undefined


localhost:3180/?year=2023&mor

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

2023 "May"

Task: Find a leap year from static url

```
var u=require("url");
var addr="http://localhost:8080/default.html?year=2025&month=feb";
var q=u.parse(addr,true);
console.log(q);
// console.log(q.host);
// console.log(q.pathname);
// console.log(q.search);
var qdata=q.query;
console.log(qdata.year);
if(qdata.year%4==0)
{
  console.log("Its a leap year")
}
else{
  console.log("Its not a leap year")
}
================================================================================
```

Output:

```
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'localhost:8080',
  port: '8080',
  hostname: 'localhost',
  hash: null,
  search: '?year=2025&month=feb',
  query: [Object: null prototype] { year: '2025', month: 'feb' },
  pathname: '/default.html',
  path: '/default.html?year=2025&month=feb',
  href: 'http://localhost:8080/default.html?year=2025&month=feb'
}
2025
Its not a leap year
```

Write a nodejs script to print query string of url on console as well as on file using ES6 callback.

```
var u=require("url");
var ps=require("fs");
var adr1=" http://localhost:8080/default.html?year=2025&month=feb";
        var q1=u.parse(adr1,true);
        var qdata=q1.query;
          console.log(qdata);
        ps.writeFile("fsd2.txt",JSON.stringify(qdata),(err)=>
        {
                console.log("completed");
        });
```

Task: Write a nodejs program which fetch filename from requested url and print that file's data on http web server.

```
var h=require("http");
var ps=require("fs");
var u=require("url");
var server=h.createServer(
    function(req,res)
    {
        var q=u.parse(req.url,true);
        data=ps.readFileSync("."+q.pathname);
    res.writeHead(200,{"content-type":"text/html"}); //text/plain gives program
    res.write(data);
    res.end();

});
server.listen(6052);
```

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

```
var h=require("http");
var ps=require("fs");
var u=require("url");
var addr="http://localhost:8080/16.html"; (html file name)
var q=u.parse(addr,true);
data=ps.readFileSync("."+q.pathname);
var server=h.createServer(
   function(req,res)
   {
   res.writeHead(200,{"content-type":"text/html"});
   //res.writeHead(200,{"content-type":"text/plain"}); gives content of file(Whole program
will display in port)
   res.write(data);
   res.end();

});
server.listen(6051);
```

How to create, export and use our own modules

❖ **Own Module**

The node.js modules are a kind of package that contains certain functions or methods to be used by those who imports them. Some modules are present on the web to be used by developers such as fs, path,http,url etc. You can also make a package of your own and use it in your code.

Example -

Create two file with name – calc.js and index.js and copy the below code snippet. The calc.js is the custom node module which will hold the node functions. The index.js will import calc.js and use it in the node process.

Method 1

In 29.js file:

```
const add=(a,b)=>
{
   return(a+b);
}
module.exports=add;
```

In another file:

```
var d=require("./29.js");
console.log(d(10,15));
```

Method 2

In 29.js file:

```
const sub=(a,b)=>
{
   return(a-b);
}
const mul=(a,b)=>
{
   return(a*b);
}
module.exports.s=sub;
module.exports.m=mul;
```

In another file:

```
var d1=require("./29.js");
console.log(d1.s(10,5));
console.log(d1.m(10,15));
```

Method 3

In 29.js file:

```
const sub=(a,b)=>
{
   return(a-b);
}
const mul=(a,b)=>
{
   return(a*b);
}
module.exports.d2=sub;
module.exports.e2=mul;
```

In another file:

```
var {d2,e2}=require("./29.js");
console.log(d2(10,7));
console.log(e2(10,12));
```

Method 4

In 29.js file:

```
const sub=(a,b)=>
{
   return(a-b);
}
const mul=(a,b)=>
{
   return(a*b);
}
const name="Hello"
module.exports={sub,mul,name};
```

In another file:

```
var {sub,mul,name}=require("./29.js");
console.log(sub(100,20));
console.log(mul(10,2));
console.log(name)
```

- **Write a nodejs script to create my own module to calculate reverse of a given . That module should be used to compute all numbers between 1 to 100 in which square of reverse and reverse of square is same. These has call of reverse twice so call it from module.**

In 31.js file

```
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 [r2,r3];
}
module.exports=rev;
```

In another file:

```
var rev =require("./31.js");
arr=rev(49);
   console.log("Square of reverse is: " + arr[0]);
   console.log("Reverse of square is: " + arr[1]);
fno=arr[0];
sno=arr[1];
if(fno==sno)
{
    console.log("Equal");
}
else
{
    console.log("not equal");
}
```

Output:

```
Square of reverse is: 2401
Reverse of square is: 8836
not equal
```