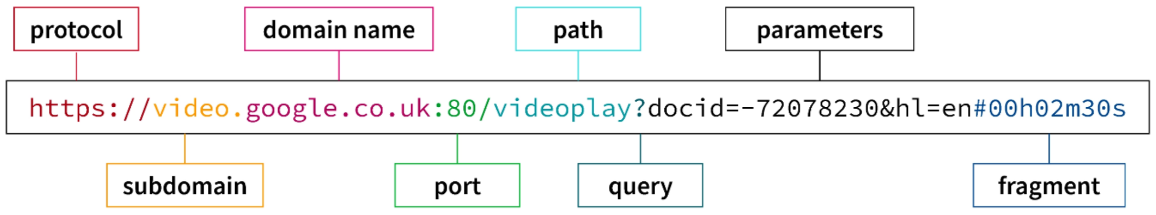
Node JS-2

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.

The returned object contains the following properties

- **protocol**: This specifies the protocol scheme of the URL (ex: https:).
- **slashes**: A Boolean value that specifies whether the protocol is followed by two ASCII forward slashes (//).
- **auth**: This specifies the authentication segment of the URL (ex: username: password).
- **username**: This specifies the username from the authentication segment.
- **password**: This specifies the password from the authentication segment.
- **host**: This specifies the host name with the port number.
- **hostname**: This specifies the host name excluding the port number.
- **port**: Indicates the port number.
- **pathname**: Indicates the URL path.
- **query**: This contains a parsed query string unless parsing is set to false.

- **hash**: It specifies the fragment portion of the URL including the "#".
- **href**: It specifies the complete URL.
- **origin**: It specifies the origin of the URL.
- **etc...**

<u>Syntax of url.parse()</u> : 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 (Static URL)**

```
var u=require("url");
var addr="http://localhost:8080/default.htm?Name=Prof.Khushbu&Initial=PKP";
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',
  hostname: 'localhost',
  hash: null,
  search: '?Name=Prof.Khushbu&Initial=PKP',
  query: [Object: null prototype] {Name: 'Prof.Khushbu', Initial: 'PKP'},
  pathname: '/default.htm',
  path: '/default.htm?Name=Prof.Khushbu&Initial=PKP',
  href: 'http://localhost:8080/default.htm?Name=Prof.Khushbu&Initial=PKP'
}
localhost:8080
/default.htm
[Object: null prototype] { Name: 'Prof.Khushbu', Initial: 'PKP' }
```

- **Google Search Link as URL (Dynamic 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'
}
```

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
```

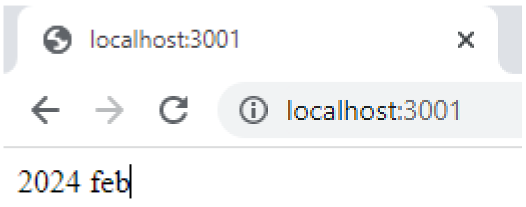
Query string

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);
```


localhost:3001

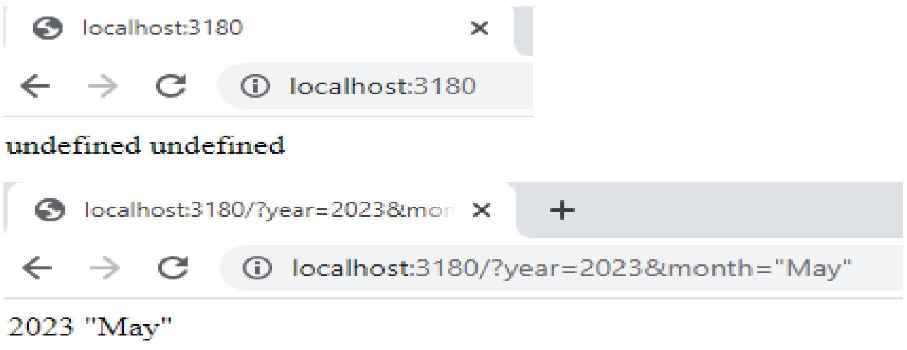
localhost:3001

```
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&mon

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

```
2023 "May"
```

- **Write a nodejs script to print query string of url on console as well as on file using ES6 callback.**
  Fetch query as an object. Need to convert query to string using stringify and then we can write it in file.

```
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");
            });
```

- **Fetch query as a string. We can directly write query string in a file.(Alternate approach)**

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

- **Task: Write a nodejs program <u>which fetch filename from requested url</u> 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);
```

<u>Note:</u> # data=ps.readFileSync(**"."+q.pathname)**; - The dot (`.`) refers to the current directory. We can also use without "." By writing entire path.

- **Write a nodejs program <u>load a simple html file on nodejs web server</u> 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. 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.

Syntax:

```
exports.function_name = function(arg1, arg2, ....argN) {
// function body
};
```

Different Way to Export Own Module

Example: Create two files 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 (Export module with full name using ES6)* simple JS function also work

In calc.js file:

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

In another file:

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

Method 2 (Export more than one function)

In calc.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("./calc.js ");
console.log(d1.s(10,5));
console.log(d1.m(10,15));
```

Method 3(obj. destructing)

In calc.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("./calc.js ");
console.log(d2(10,7));
console.log(e2(10,12));
```

Method 4 (Export in single line including variable name)

In calc.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("./calc.js ");
console.log(sub(100,20));
console.log(mul(10,2));
console.log(name)
```

- **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.**

calc1.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;
};
```

calc2.js

```
const calculator = require('./calc1.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
```

NPMjs (Chalk, Validator)

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

- **Chalk module**

In Node.js is the third-party module that is used for styling the format of text and allows us to create our own themes in the node.js project.

Advantages of Chalk Module:

1. It helps to customize the color of the output of the command-line output
2. It helps to improve the quality of the output by providing several color options like for warning message red color and many more

To install chalk: npm install chalk

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

In package.json file:

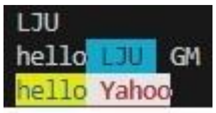
```
{"type": "module", // Add this line manually in package.json
  "dependencies": {
    "chalk": "^5.2.0"
  }
}
```

Note: If chalk installed successfully and package.json and package-lock.json files are not created, then try **npm init** command inside created folder. And then try to install chalk inside folder.

Example:

```
import ch from "chalk";
const log=console.log;
log("LJU");
log("hello"+ch.bgCyan(" LJU ")+" GM ")
log(ch.blue.underline.bgYellow("hello")+ch.red.bold.underline.bgWhite(" Yahoo"));
```

Output:



```
LJU
hello LJU  GM
hello Yahoo
```

Chalk comes with an easy to use composable API where you just chain and nest the styles you want.

```
import chalk from 'chalk';

const log = console.log;

// Combine styled and normal strings

log(chalk.blue('Hello') + ' World' + chalk.red('!'));

// Compose multiple styles using the chainable API

log(chalk.blue.bgRed.bold('Hello world!'));

// Pass in multiple arguments

log(chalk.blue('Hello', 'World!', 'Foo', 'bar', 'biz', 'baz'));

// Nest styles

log(chalk.red('Hello', chalk.underline.bgBlue('world') + '!'));

// Nest styles of the same type even (color, underline, background)

log(chalk.green('I am a green line ' +chalk.blue.underline.bold('with a blue substring') +

        ' that becomes green again!'));

// Use RGB colors in terminal emulators that support it.

log(chalk.rgb(123, 45, 67).underline('Underlined reddish color'));

log(chalk.hex('#DEADED').bold('Bold gray!'));
```

Easily define your own themes

```
import chalk from 'chalk';

const error = chalk.bold.red;

const warning = chalk.hex('#FFA500'); // Orange color

console.log(error('Error!'));

console.log(warning('Warning!'));
```

Take advantage of console.log string substitution

```
import chalk from 'chalk';

const name = 'Mam';

console.log(chalk.green('Hello %s'), name);

//=> 'Hello Mam'
```

Validator module

The Validator module is popular for validation. Validation is necessary to check whether the data is in format or not, so this module is easy to use and validates data quickly and easily.

Feature of validator module:

- It is easy to get started and easy to use.
- It is a widely used and popular module for validation.

Functions are available in this module like

isEmail(), isEmpty(), isLowercase(), isBoolean(), isCurrency(), isDecimal(), isJSON(), isFloat(), isCreditCard(), isHexadecimal (), isCurrency, isDecimal (), etc.

To install validator: npm install validator

Example1 : Check whether given email is valid or not

```
import validator from "validator"
let email = 'test@gmail.com'
console.log(validator.isEmail(email))               // true
email = 'test@'
console.log(validator.isEmail(email))              // false
```

Example2 : Check whether string is in lowercase or not

```
import validator from "validator"
let  name = 'hellolju'
console.log(validator.isLowercase(name))       // true
name = 'HELLOLJU'
console.log(validator.isLowercase(name))     // false
```

Example3: Check whether string is empty or not

```
import validator from "validator"
let name = ''
console.log(validator.isEmpty(name))  // true
name = 'helloLJU'
console.log(validator.isEmpty(name))  // false
```

Try:

```
console.log(v.isEmail("hello@gmail.com"));
console.log(v.isHexadecimal("ABC"));
console.log(v.isLowercase("xyz"));
console.log(v.isEmpty(""));
console.log(v.isCurrency("123"));
console.log(v.isCurrency(“pkp"));
console.log(v.isDecimal("1.22"));
console.log(v.isJSON('{"name1":"ABC","age":30}'));
```

Module Wrapper Function

NodeJS does not run our code directly, it wraps the entire code inside a function before execution. This function is termed as Module Wrapper Function. Before a module's code is executed, 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 inside each module in Node. Though these parameters are global to the code within a module yet they are local to the module (because of the function wrapper as explained above). These parameters provide valuable information related to a module.

The variables like **__filename** and **__dirname**, that tells us the module's absolute filename and its directory path.

The meaning of the word 'anonymous' defines something that is unknown or has no identity. In JavaScript, an anonymous function is that type of function that has no name or we can say which is without any name. When we create an anonymous function, it is declared without any identifier. It is the difference between a normal function and an anonymous function.

Use of Module Wrapper Function in NodeJS:

- The top-level variables declared with var, const, or let are scoped to the module rather than to the global object.
- It provides some global-looking variables that are specific to the module, such as:
  - The module and exports object that can be used to export values from the module.
  - The variables like __filename and __dirname, that tells us the module's absolute filename and its directory path.

Example -1:

```
( function()
    {
       const name="LJU";
       console.log(name);
       console.log(__filename);
       console.log(__dirname);
    }
  )();
//console.log(name);   // //it will throw error because name is in wrapper function which cannot
be                                             used outside
```

Output:

```
LJU
D:\Trynode\ex1.js
D:\Trynode
```

Events

Every action on a computer is like an event. For, e.g., when a connection is made, or you open any file.

Node.js is an asynchronous event-driven JavaScript runtime. Its event-driven architecture allows it to perform asynchronous tasks. It has a built-in module called the 'events' module, where you can create, fire, and listen for your events. When a function (Callback) listens to a particular event to occur, all the callbacks subscribing to that event are fired one by one in the sequence in which they were registered..

Functions(Callbacks) listen or subscribe to a particular event to occur and when that event triggers, all the callbacks subscribed to that event are fired one by one in order to which they were registered.
1) eventEmmitter.on("event-name", callback):
This functions an event with its handler(listener) and calls the callback function once event is emitted.

2) EventEmitterObj.emit("event-name"):
Triggers an event with a specific string name.

All the objects that emit events are instances of the EventEmitter class.

- An EventEmitter object has two functions:
    1) Emit a named event.
    2) Attach or detach listeners to the named event.

You can assign event handlers to the events with the EventEmitter object.

- **emit** is used to trigger an event

- **on/addListener** is used to add a callback function that's going to be executed when the event is triggered.
- **Syntax:**
  eventEmitter.emit(event, [arg1], [arg2], [...])

  eventEmitter.on(event, listener)

  eventEmitter.addListener(event, listener)

Steps: -
1) Import "events" module
Var e=require("events");

2) Create EventEmitter object
Var ee=new e.EventEmmiter();

3) Define event listeners using the **on** method.
ee.on('eventName', () => {

```
  // Event handler code
});
```

4) Emit /fire an event to trigger the listeners.

```
ee.emit('eventName');
```

Steps for event handling script:

- Import “events” module

  ```
  var event = require('events');
  ```

- Create EventEmitter object

  ```
  var ee = new event.EventEmitter();
  ```

- Generate Event Handler as follows:

  ```
  var connectHandler = function connect()
  {
    console.log(“connection successful”);
    ee.emit("data-received")      //fire data-received event
  }
  ```

- Bind connection event with the handler

  ```
  ee.on("connection",connectHandler);
  ```

- Bind data-received event with an anonymous function

  ```
  ee.on("data-received",function(){
    console.log("data received successfully")
  })
  ```

- Fire connection event

  ```
  ee.emit("connection")
  console.log("Thanks")
  ```

Output:

```
connection successfully

data received successfully

Thanks
```

Example.

```
var e=require("events");
var ee=new e.EventEmitter();
ee.on("sayName",()=>{
   console.log("your name is xyz")
});
ee.emit("sayName");
```

The code snippet you provided is an example of using the **events** module in Node.js to create and emit events.

In this code, you first require the **events** module by assigning it to the variable **e**. Then, you create a new instance of the **EventEmitter** class and assign it to the variable **ee**.
Next, you define an event listener for the event named **"sayName"**. When this event is emitted, the listener will be triggered, and it will log the message **"your name is xyz"** to the console.
Finally, you emit the event **"sayName"** by calling **ee.emit("sayName")**. This will trigger the event listener, and you will see the corresponding message printed in the console: **"your name is xyz"**.

```
var e=require("events");
var ee=new e.EventEmitter();
ee.emit("sayName");
ee.on("sayName",()=>{
   console.log("your name is xyz")
});
```

In this case, when you emit the event before defining the listener, there are no registered listeners yet, so nothing will happen.

Example-1:

```
const E = require('events');
const ee = new E();
ee.on('start', () => {
   console.log('started');  });
  ee.emit('start');
```

Example-2:

```
const E = require('events');

const ee = new E();
ee.on('start', (start, end) => {
   console.log(`started from ${start} to ${end}`);
  });

  ee.emit('start', 1, 100);
```

Output: started from 1 to 100

Example: Create a function which executes when a "scream" event is fired. To fire an event, use the emit() method.

```
var event = require('events');
var eventEmitter = new event.EventEmitter();

//Create an event handler
var handler = function () {
  console.log('Hello World!');
}

//Assign the event handler to an event
eventEmitter.on('scream', handler);

//Fire the event:
eventEmitter.emit('scream');
```

Output:

```
Hello World!
```

Example: create an event emitter instance and register a couple of call-back.

```
var e=require("events");
var ee=new e.EventEmitter();
ee.on("sayName",()=>{
    console.log("your name is Khushbu")
});
ee.on("sayName",()=>{
    console.log("your name is Mam")
});
ee.on("sayName",()=>{
    console.log("your name is Patel")
});
ee.emit("sayName");
```

o/p

```
PS D:\SEM-4\fsd-2\node> node eventhandle.js
your name is Khushbu
your name is Mam
your name is Patel
```

1. Require the **events** module and create a new instance of **EventEmitter**.
2. Commented out the line **ee.emit("sayName");**, so it won't be executed.
3. Define an event listener for the event named **"sayName"**. When this event is emitted, it will log the message **"your name is Khushbu"** to the console and emit the event **"xyz"**.
4. Define an event listener for the event named **"xyz"**. When this event is emitted, it will log the message **"your name is M"** to the console and emit the event **"abc"**.
5. Define an event listener for the event named **"abc"**. When this event is emitted, it will log the message **"your name is Patel"** to the console.
6. Emit the event **"sayName"** by calling **ee.emit("sayName")**.
7. The **"sayName"** event listener is triggered, logging **"your name is Khushbu"** to the console, and emitting the **"xyz"** event.
8. The **"xyz"** event listener is triggered, logging **"your name is M"** to the console, and emitting the **"abc"** event.
9. The **"abc"** event listener is triggered, logging **"your name is Patel"** to the console.

Example: registering for the event with call-back parameter.

```
var e=require("events");
var ee=new e.EventEmitter();
ee.on("sayName",(statusCode,msg)=>{
   console.log(`status code id ${statusCode} and page is ${msg}`);
});
ee.emit("sayName",200,"ok");
```

o/p

```
PS D:\SEM-4\fsd-2\node> node eventhandle2.js
status code id 200 and page is ok
```

Add / Remove Listener:

eventEmmitter.on(event, listener) and **eventEmitter.addListener(event, listener)** are pretty much similar. It adds the listener at the end of the listener's array for the specified event. Multiple calls to the same event and listener will add the listener multiple times and correspondingly fire multiple times. Both functions return emitter, so calls can be chained.

The **eventEmitter.removeListener()** takes two argument event and listener, and removes that listener from the listeners array that is subscribed to that event.

While **eventEmitter.removeAllListeners()** removes all the listener from the array which are subscribed to the mentioned event.

Syntax:
eventEmitter.removeListener(event, listener)

eventEmitter.removeAllListeners([event])

Note:
 Removing the listener from the array will change the sequence of the listener's array, hence it must be carefully used.
 The **eventEmitter.removeListener()** will remove at most one instance of the listener which is in front of the queue.

Task: Write a NodeJs script to create two listeners for a common event. Call their respective callbacks. Print no. of events associated with an emitter. Remove one of the listener and print no of remaining listeners.

```
var event = require('events');
var ee = new event.EventEmitter();
var listener1 = function listener1() {
  console.log("listener1 executed")
}
var listener2 = function listener2() {
  console.log("listener2 executed")
}
ee.addListener("conn",listener1)
ee.on("conn",listener2)
var count=ee.listenerCount("conn")          //counts listeners for conn event
console.log(count+" for conn event")
ee.emit("conn")
ee.removeListener("conn",listener1)         //remove listener1 form conn event
var count=ee.listenerCount("conn")
console.log(count+" for conn event")
ee.emit("conn")
// Above program ends here. Below is additional task of remove all listeners. and count the
listener.

eventEmitter.removeAllListeners('conn', listener1);
count = eventEmitter.listenerCount("conn");
console.log("Again Count afetr removing all listeners: " + count );
eventEmitter.emit("conn");
```

Output:

```
2 for conn event

listener1 executed

listener2 executed

1 for conn event

listener2 executed
```

```
//After removing all listeners
Again Count afetr removing all listeners: 0
```

Task-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 ne .txt file. After that read the file and print file concept on console. After finishing read operation, print the line “Thanks for using my program” on console. All read/write operations are asynchronous.

```
const fs = require('fs');
const e = require('events');
const ee = new e.EventEmitter();
ee.on('write', () => {
  fs.writeFile("new.txt", 'this is data', () => {
    console.log('Successfully wrote to file.');
    ee.emit('append');
  });
});
ee.on('append', () => {
  fs.appendFile("new.txt", '\nthat is data', () => {
    console.log('Successfully appended to file.');
    ee.emit('read');
  });
});
ee.on('read', () => {
  fs.readFile("new.txt", 'utf8', (err, data) => {
   console.log(data);
    ee.emit('finish');
  });
});
ee.on('finish', () => {
  console.log('Thanks for using my program.');
});
ee.emit('write');
```

JSON Processing

When we create a simple object in a JS with key & value pair, then it is called as JSON object. Even if we provide each key pair as string, then also it treats them without string keys like an object. It is not possible to write JSON object directly to the file. We can write after calling stringify() method. Only because string can be easily written to the file; we can write individual properties of JSON object directly.

- **Process below JSON objects in abc.txt with statement : John has BMW 320 at 30** .

```
Obj = {
        "name" : "John ",
        "age" : 30,
        "cars" : [
                { "name" : "Ford",  "models":[ "Fiesta", "Focus", "Mustang" ] },
                { "name" : "BMW", "models" : [ "320", "X3", "X5" ] },
                { "name" : "Fiat", "models" : [ "500", "Panda" ] }
       ] }.
```

Program:

```
var fs=require("fs")
obj = {
  "name" : "John ",
  "age" : 30,
  "cars" : [
        { "name" : "Ford",  "models":[ "Fiesta ", "Focus ", "Mustang " ] },
        { "name" : "BMW", "models" : [ "320 ", "X3", "X5" ] },
        { "name" : "Fiat", "models" : [ "500", "Panda" ] }
] }
var data= obj.name+"has " +obj.cars[1].name+obj.cars[1].models[0]+"at "+obj.age+"."
console.log(data)
data1=JSON.stringify(data)
fs.writeFileSync("abc.txt",data1)
```

Miscellaneous Programs

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

```
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);
       Filename ="."+q.pathname;
       ps.readFile(Filename,function(err,data)
       {
        if(err){
            res.writeHead(404,{"content-type":"text/hplain"})
            res.end("Error che")
        }
        else{
            res.writeHead(200,{"content-type":"text/html"}); //text/plain gives program
            console.log(data)
            console.log("Hello")
            res.write(data);
            res.end()

           // return(res.end())
        }
       });
       console.log("Hello")
});
server.listen(6050);
console.log("Hi") //
```

Task: 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="https://www.google.com/search?q=url+module+in+node+js&rlz=1C1YTUH_enIN1039
```

```
IN1039&oq=url+modu&aqs=chrome.0.0i512j69i57j0i512l4j0i390l4.2814j0j7&sourceid=chro
me&ie=UTF-8";

var q1=u.parse(adr1,true);
var qdata=q1.query;
   console.log(qdata);
ps.writeFile("fsd2.txt",qdata.q,(err)=>
{
   console.log("completed");
});
```

Output:

```
[Object: null prototype] {

  q: 'url module in node js',

  rlz: '1C1YTUH_enIN1039IN1039',

  oq: 'url modu',

  aqs: 'chrome.0.0i512j69i57j0i512l4j0i390l4.2814j0j7',

  sourceid: 'chrome',

  ie: 'UTF-8'

}

completed
```

In file: url module in node js

- **Write a nodejs script to create own module to calculate reverse of a given number. That module should be used to check given number of which square of reverse and reverse of square is same.**

In rev.js file

```
        function reversenum(num)
        {
```

```
    let rev=0;
    while(num>0)
    {
        rev=rev*10+(num%10);
        num=parseInt(num/10);
    }

    return rev;
}
function square(num1)
{

    return num1*num1;
}
function checknum(num2)
{
    a=square(num2);
    b=square(reversenum(num2));
    if(a==reversenum(b))
    {
        console.log("Number is equal")
    }
    else
    {
        console.log("Number is not equal")
    }}

module.exports.reversenum=reversenum;
module.exports.square=square;
module.exports.checknum=checknum;
```

In another file:

```
var d=require("./rev.js");
d.checknum(12);
```

Output:

```
Number is equal
```

Prime number

Write all necessary .js files to create module having a function to check numbers from 2 to 50 are prime number or not.

1.js

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

2.js

```
var PrimeNumber = require("./31.js")
for(i=2;i<=50;i++)
{
let x=PrimeNumber(i);
if(x==true)
{
    console.log(i+" Prime Number");
}
else{
    console.log(i+" Not a Prime Number")
}
}
```

Output:

```
2 Prime Number
3 Prime Number
4 Not a Prime Number
5 Prime Number ………. upto 50
```

- **Write node js script to handle event of write a data in file, append data in file and then read the file and display data in console.**

```
var e=require("events");
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");
```

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 be positive" 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) => {

   if (r < 0) {
      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("negside", sidehandler);
ee.on("findval", findval);
ee.emit("findval",-2,-3);
```

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

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

const ee = new e.EventEmitter();
const fileName = 'new.txt';

ee.on('write', () => {
  fs.writeFile(fileName, 'this is data', () => {
    console.log('Successfully wrote to file.');
    ee.emit('append');
  });
});

ee.on('append', () => {
  fs.appendFile(fileName, '\nthat is data', () => {

    console.log('Successfully appended to file.');
    ee.emit('read');
  });
});

ee.on('read', () => {
  fs.readFile(fileName, 'utf8', (err, data) => {
   console.log(data);
    ee.emit('finish');
  });
});

ee.on('finish', () => {
  console.log('Thanks for using my program.');
});

ee.emit('write');
```

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.EventEmitter()
const negativePerimeterHandler=()=>{
console.log("Perimeter must be positive ")
}
const negativeSideHandler=()=>{
console.log("Side must be positive")
}
const findError=(a,b,c)=>{
if(b<0||c<0||a<0){
ee.emit("Negativeside")
}
if((a+b+c)<0){
ee.emit("NegativePerimeter")
}
}
ee.on("findError",findError)
ee.on("NegativePerimeter",negativePerimeterHandler)
ee.on("Negativeside",negativeSideHandler)
ee.emit("findError",-10,-15,20)
```

Stock profite/loss color based on server

```
// index.js

// Importing the local module
const { calculateProfitOrLoss } = require('./calc');
const http = require('http');

// Creating an HTTP server
const server = http.createServer((req, res) => {
  // Example usage
  const purchasePrice = 50; // $50 per share
  const sellingPrice = 60; // $40 per share
  const quantity = 50; // 50 shares

  // Calculate profit or loss
  const { type, amount } = calculateProfitOrLoss(purchasePrice, sellingPrice, quantity);

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

  // Set the CSS style based on profit or loss
  let colorStyle = '';
  if (type === 'Profit') {
    colorStyle = 'green';
  } else if (type === 'Loss') {
    colorStyle = 'red';
  }

  // Send the HTML response
  res.end(`<p style="color: ${colorStyle};">Type: ${type}, Amount: ${amount}</p>`);
});

// Listening on port 3000
const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

```
// calc.js

// Function to calculate profit or loss
function calculateProfitOrLoss(purchasePrice, sellingPrice, quantity) {
    const costPrice = purchasePrice * quantity;
    const sellingValue = sellingPrice * quantity;
    const profit = sellingValue - costPrice;
    const loss = costPrice - sellingValue;

    if (profit > 0) {
      return { type: 'Profit', amount: profit };
    } else if (loss > 0) {
```

```
    return { type: 'Loss', amount: loss };
  } else {
    return { type: 'No Profit No Loss', amount: 0 };
  }
}



// Exporting the functions to make them accessible in other files
module.exports = { calculateProfitOrLoss, };
```