JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation.
- JSON is a lightweight text format for storing and transporting data.
- JSON is often used when data is sent from a server to a web page.
- JSON is "self-describing" and easy to understand.
- A JSON object is an unordered set of name/value pairs.

JavaScript has a built in functions

1. Converting JSON strings into JavaScript objects: **JSON.parse()**
2. converting an object into a JSON string: **JSON.stringify()**

```
var a= {
  "name": "Test",        // String
  "age": 25,             // Number

  "ispass": false,     // Boolean
  "address": {           // Object (Nested JSON)
    "city": "Ahmedabad",
    "zip": "380015"
  },
  "subjects": ["Math", "Science"],  // Array
  "year": null   // Null
}

console.log(a);
console.log(a.age)
console.log(a[‘address’])
console.log(a[‘address’][‘city’])
```

Output:

```
{
  name: 'Test',
  age: 25,
  ispass: false,
  address: { city: 'Ahmedabad', zip: '380015' },
  subjects: [ 'Math', 'Science' ],
  year: null
}
25
{ city: 'Ahmedabad', zip: '380015' }
Ahmedabad
```

JSON.parse()

Example:
```
<script>
let obj = JSON.parse('{"var1":"LJ","var2":"University"}');
console.log(obj);
console.log(obj.var1 + " " + obj.var2);
</script>
```
Output:
```
{var1: 'LJ', var2: 'University'}
LJ University
```

JSON.stringify()

Example:
```
<script>
let obj = JSON.stringify({var1: 'LJ', var2: 'University'});
console.log(obj);
</script>
```
Output:
```
'{"var1":"LJ","var2":"University"}'
```

Method to create JSON object

Syntax:

obj[key]=value

eg.
```
var obj2={}
obj2[“a”]=111
console.log(obj2)
```

output:
```
{“a”:111}
```

Node JS-1

Download node js from https://nodejs.org/en and install it. To check downloaded version type node -v in console.

The Node.js installer includes the NPM(Node Package Manager). For version check npm –v.

Node.js is an open source server environment.

Node.js allows you to run JavaScript on the server.

REPL

REPL stands for

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

REPL Commands

- **ctrl + c** − terminate the current command.
- **ctrl + c twice** − terminate the Node REPL.
- **ctrl + d** − terminate the Node REPL.
- **Up/Down Keys** − see command history and modify previous commands.
- **tab Keys** − list of current commands.
- **.help** − list of all commands.
- **.break** − exit from multiline expression.
- **.clear** − resets the REPL context to an empty object and clears any multi-line expression currently being input.
- **.save *filename*** − save the current Node REPL session to a file.
- **.load *filename*** − load file content in current Node REPL session.

Starting REPL

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

```
$ node
```

Underscore Variable

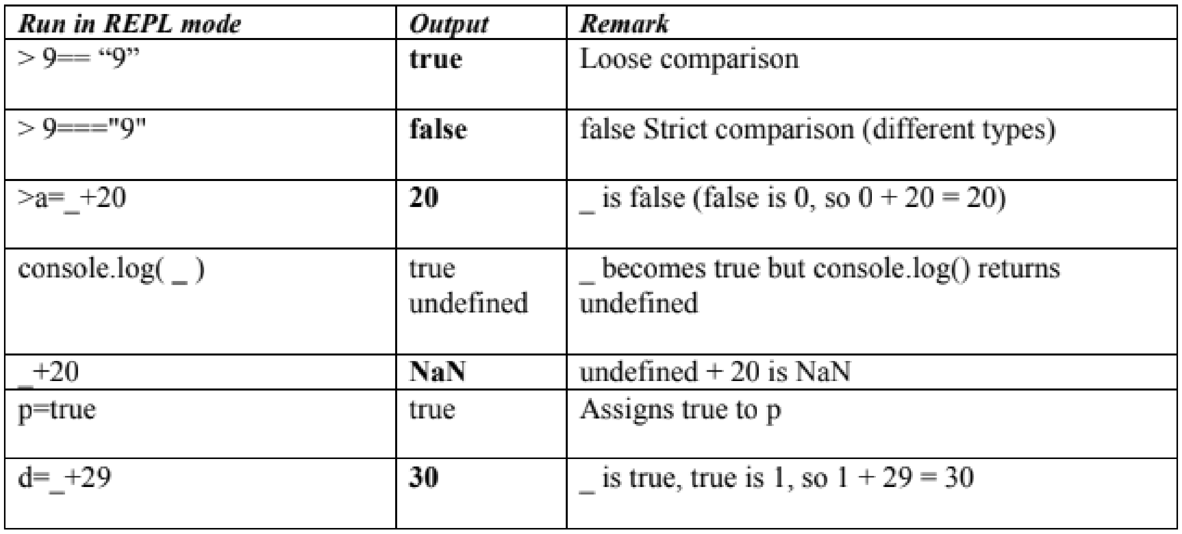
You can use underscore (_) to get the last result −

```
>.editor
```

Type .editor to enter in editor mode (Block wise execution only)

Entering editor mode (Ctrl+D to generate output, Ctrl+C to cancel

Test Cases



Run in REPL mode	*Output*	*Remark*
> 9== “9”	**true**	Loose comparison
> 9==="9"	**false**	false Strict comparison (different types)
>a=_+20	**20**	_ is false (false is 0, so 0 + 20 = 20)
console.log(_)	true undefined	_ becomes true but console.log() returns undefined
_+20	**NaN**	undefined + 20 is NaN
p=true	true	Assigns true to p
d=_+29	**30**	_ is true, true is 1, so 1 + 29 = 30

setInterval(), setTimeout()

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

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

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

Display clock using setInterval Method

```
function updateTime() {
  // Get the current time in HH:MM:SS format for India timezone
  const timeString = new Date().toLocaleTimeString()
  console.log(timeString) }
updateTime();

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

callback examples

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>
```

Output : Display 9 and then incremented

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

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

                obj = document.getElementById("demo");
                obj.innerHTML = "hello";
                obj.style.color ="blue";
                obj.style.fontSize = size + “px”;
                if (size <= 50) {
                    size++;
                }
            }
            setInterval(add, 50);
        </script>
    </body>
</html>
```

❖ 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');
```

Synchronous mode

	Syntax	**example**
Create folder	fs.mkdirSync(folder_name)	fs.mkdirSync("Details")
Create file	fs.writeFileSync(file_name,data)	fs.writeFileSync("user.txt","Hello")
append the data	fs.appendFileSync(file_name,data)	fs.appendFileSync("user.txt","Hi")
Read data	fs.readFileSync(filr_name) **note:** it gives buffer data so convert it in string using “utf-8” or .toString() method.	var data=fs.readFileSync("user.txt","utf-8")
Rename file	fs.renameSync(file_name,file_new_name)	fs.renameSync("user.txt","user1.txt")
Delete file	fs.unlinkSync(file_name)	fs.unlinkSync("user.txt")
Delete Folder	fs.rmdirSync(folder_name)	fs.rmdirSync("Details")

Asynchronous mode

	Syntax	**example**
Create folder	fs.mkdir(folder_name,callback)	fs.mkdir("details",function (err) { if (err){ console.log(err)})
Create file	fs.writeFile(file_name,data,callback)	fs.writeFile('test.txt', 'Hello World!', function (err) { if (err){ console.log(err)})
append the data	fs.appendFile(file_name,data,callback)	fs.writeFile('test.txt', 'Hi', function (err) { if (err){ console.log(err)})
Read data	fs.readFile(filr_name,encoding(optional),callback) **note:** it gives buffer data so convert it in string using "utf-8" or .toString() method.	ps.readFile('test.txt', function(e,data){ if(e) { return console.error(e);} console.log(data.toString()); **// if you want buffer data then remove to string** console.error("complete")});
Delete file	fs.unlink(file_name,callback)	fs.unlink('test.txt',function (err) { if (err){ console.log(err)})
Delete Folder	fs.rmdir(folder_name,callback)	fs.rmdir("details",function (err) { if (err){ console.log(err)})

OS Module: Operating System

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

Example:

```
os=require("os");
  console.log(os.arch());
  console.log(os.hostname());
  console.log(os.platform());
  console.log(os.tmpdir());
  console.log(os.freemem());
// os.freemem(): This method returns an integer value that specifies the amount of free system
memory in bytes.
  a1=os.freemem();
  console.log(`${a1/1024/1024/1024}`);
```

Output:

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

Path Module

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

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
```

HTTP Module: Render Response, Read HTML File Server, Routing

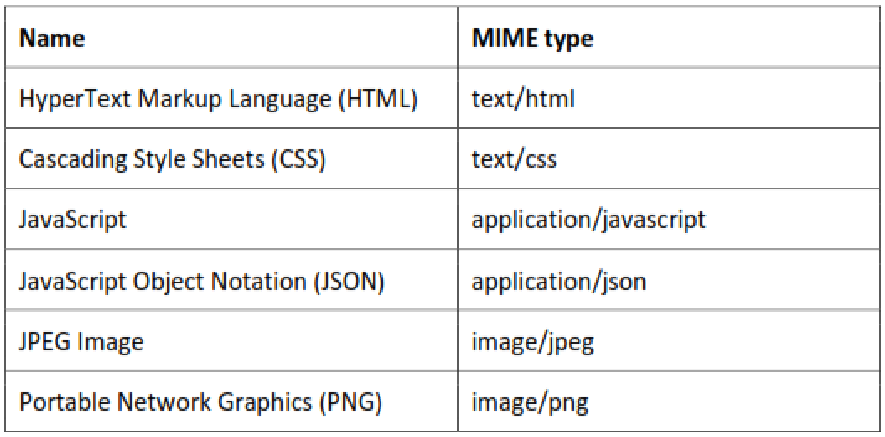
Example to create server and print “Hello world” message in **index,js** file

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

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

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

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



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

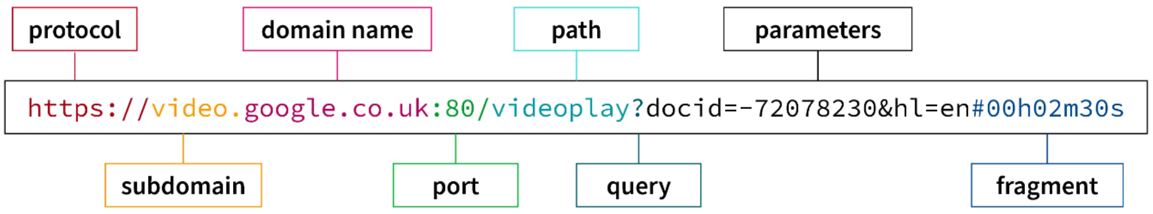
Create HTTP webpage on which home page display “Home page”, student page shows “Student page” and any other page shows “Page Not found”.
(Render Response & Routing)

```
var h=require("http");
var server=h.createServer(
    function(req,res)
    {
if(req.url=="/")
{
   res.writeHead(200,{"content-type":"text/html"});
   res.write("<b> Home page </b>");
   res.end();
}
else if(req.url=="/student")
{
   res.writeHead(200,{"content-type":"text/plain"}); //plain shows code as it is
   res.write("<i> Home page1 </i>");
   res.end();
}
else {
   res.writeHead(404,{"content-type":"text/html"});
   res.write("<h1> Page Not found </h1>");
   res.end("Thanks");
   }   });
server.listen(5001);
console.log("Thanks for run");
```

Nodemon

- To install: **npm install -g nodemon**
- To check version : **Nodemon -v**

URL module


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

var url=require("url");

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...**

Example

```
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'
}
```

How to create, export and use our own modules

<u>Syntax:</u>

```
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.

<u>Method 1 (Export module with full name using ES6)* simple JS function also work</u>

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));
```

<u>Method 2 (Export more than one function)</u>

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));
```

<u>Method 3(obj. destructing)</u>

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));
```

<u>Method 4 (Export in single line including variable name)</u>

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)
```

NPMjs (Chalk, Validator)

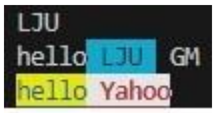
npm install chalk

Category	Methods
Text Colors	red(), green(), blue(), yellow(), cyan(), magenta(), white(), gray(), black()
Bright Text Colors	redBright(), greenBright(), blueBright(), yellowBright(), cyanBright(), magentaBright(), whiteBright()
Background Colors	bgRed(), bgGreen(), bgBlue(), bgYellow(), bgCyan(), bgMagenta(), bgWhite(), bgBlack()
Bright Backgrounds	bgRedBright(), bgGreenBright(), bgBlueBright(), bgYellowBright(), bgCyanBright(), bgMagentaBright(), bgWhiteBright()
Text Styles	bold(), italic(), underline(), strikethrough(), inverse()

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
```

Example:

```
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!'));
```

Validator module

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
console.log(validator.isEmail('test@'))             // false
```

Example2 : Check whether string is in lowercase or not

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

Example3: Check whether string is empty or not

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

Example4: Check JSON

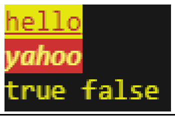
```
import v from "validator"
console.log(v.isJSON('{"name1":"ABC","age":30}'))
```

cv.js

```
import ch from "chalk";
import validator from "validator"

var test = ch.red.underline.bgYellow("hello")+ch.bold.bgRed.italic.yellow("\nyahoo")
console.log(test)

console.log(validator.isLowercase(test), validator.isEmail(test))
```



```
hello
yahoo
true false
```

Module Wrapper Function

```
(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.

Events

1) eventEmitter.addListener(event, listener): It adds the listener at the end of the listener's array for the specified event.

2) eventEmmitter.on("event-name", callback): It is similar to eventEmmitter.addListner (event, listener) .

3) eventEmitter.once(event, listener): It fires at most once for a particular event and will be removed from listeners array after it has listened once. Returns emitter, so calls can be chained.

Emitting events: Every event is named event in nodejs. We can trigger an event by emit(event, [arg1], [arg2], […]) function. We can pass an arbitrary set of arguments to the listener functions.

Syntax:

```
eventEmitter.emit(event, [arg1], [arg2], [...])
```

Removing Listener:

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.

ListenerCount: The **eventEmitter.listenerCount()** returns the number of listeners listening to the specified event.

Syntax:

```
eventEmitter.listenerCount( event)
```

Steps for event handling script

```
// 1) Import “events” module
var e=require("events");
// 2) Create EventEmitter object
var ee=new e.EventEmitter();
// 3) Bind connection event with the handler/function
ee.on("sayName",()=>{
   console.log("your name is xyz")
});
// 4) Emit / Fire connection event
ee.emit("sayName");
```

Position matters

```
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: 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");
```

Output:

```
status code id 200 and page is ok
```

Example: 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.

```
ee.removeAllListeners('conn', listener1);
count = ee.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 after removing all listeners: 0