JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation.
- Curly braces hold objects and each name is followed by **':'**(colon), the name/value pairs are separated by , (comma). **{Key:Value,..}**
- **Datatypes**: String, Number, Boolean, Object, Array, Null

```
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
```

JavaScript has a built in functions

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

Note: For valid JSON Key must be assigned in double quote "key" followed by Single quote'{"key":value}'

Valid JSON string: JSON.parse(**'{"age":**10}**'**)
Invalid JSON string : JSON.parse("{'age':10}")

JSON.parse() – Output is in object

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() – Output is in string

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**

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

output:
```
{“a”:111}
```

```
function firstlast(a) {
   var temp = {};
   temp[0] = a[2];
   return temp;
}
 var data = [“abc”, “def”, “ghi”, “jkl”];
console.log(firstlast(data));
```

Output = **{ “0”: “ghi” } //temp[0] is key returns “0”**

- **Write a function 'transformFirstAndLast' that takes in an array, and returns an object with:**
  1) the first element of the array as the object's key, and
  2) the last element of the array as that key's value.

Hint:

Example input:

['Queen', 'Elizabeth', 'Of Hearts', 'Beyonce']

Function's return value (output):

{Queen : 'Beyonce'}

Solution:

```
function transformFirstAndLast(array) {
var object = {};
object[array[0]] = array[array.length-1];
return object;
}
var arrayList = ['Queen', 'Elizabeth', 'Of Hearts', 'Beyonce'];
console.log(transformFirstAndLast(arrayList));
```

Output :

```
{Queen: ‘Beyonce’}
```

Explaination:

array**[0]** → Accesses the **first element** of the array and treats it as a **key**.
array**[array.length - 1]** → Accesses the **last element** of the array and assigns it as a **value**.
object**[array[0]] = array[array.length - 1];** → Dynamically assigns the key-value pair in the object.

Node JS

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)

To remove undefined error: “repl.repl.ignoreUndefined = true”

ignoreUndefined - if set to true, then the repl will not output the return value of command if it's undefined. Defaults to false.

Try bellow TEST cases in REPL mode in sequence

Run in REPL mode ***> node (enter)***	***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);
```

CORE MODULE

❖ 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”, (err) =>{
		if (err){
		console.log(err)}})

Create file	fs.writeFile(file_name,data,callback)	fs.writeFile('test.txt', 'Hello World!', (err) => {
		if (err){
		console.log(err)}})

append the data	fs.appendFile(file_name,data,callback)	fs.appendFile('test.txt', 'Hi', (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.
		fs.readFile('test.txt', (e,data)=>{
		if(e) {
		return console.error(e);}
		console.log(data.toString());
		console.log ("complete")});

Rename file	fs.rename(file_name,file_new_name,callback)	fs.rename('test1.txt','test2.txt',() => {console.log("Renamed")})

Delete file	fs.unlink(file_name,callback)	fs.unlink('test.txt', (err) =>{
		if (err){
		console.log(err)}})

Delete Folder	fs.rmdir(folder_name,callback)	fs.rmdir(“details”, (err)=> {
		if (err){
		console.log(err)}})

- **Writing data to file, appending data to file and then reading the file data using ES6 Concept. (Async Nested operation)**

```
var fs=require("fs");
fs.writeFile("abc.txt","Today is a good day .\n",(err)=>{
          if(err){
                    console.log("Error in write")
                    }
         fs.appendFile("abc.txt"," Is it???",(err)=>{
                    if(err)
                    {
                       console.log("Error in append")
                    };
                    fs.readFile("abc.txt",(err,data)=>{
                              if(err){
                              console.log(err);
                              }
                              console.log(data.toString())
                              });
          });
})
console.log("File Operations ended")
```

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());
  a1=os.freemem();
  console.log(`${a1/1024/1024/1024}`);
```

Output:

```
x64
SYCEIT309A-115
win32
C:\Users\foram\AppData\Local\Temp
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"); // observe keys in object created by parse
  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

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

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

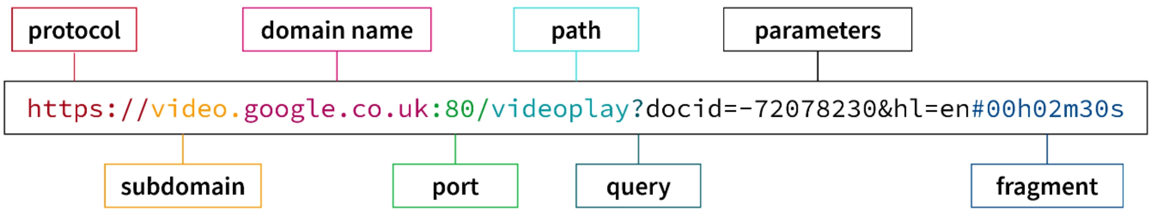
Output on browser → http://localhost:8080/

```
Hello World!
```

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);
```

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 keys in URL object

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'
}
```

```
var q=u.parse(adr1, fasle); //default value is false in query string
or
var q=u.parse(adr1);
will return string  value in query key
query: { q: 'good morning' } is in string
```

- **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);
```

On browser :http://localhost:6052/form.html will load form.html if it is at mention folder.

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

How to create, export and use our own modules

1. **Build Logic in one file**
2. **Export that file**
3. **Use File in other files whenever require**

Different Way to Export Own Module

<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 index.js file:

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

Run → node index.js

<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 index.js file:

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

Run → node index.js

<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.s=sub;
module.exports.m=mul;
```

In index.js file:

```
var {s,m}=require("./calc.js ");
console.log(s(10,7));
console.log(m(10,12));
```

Run → node index.js

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 index.js file:

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

Run → node index.js

Method 5 (Direct export)

In calc.js file:

```
exports.add = function (x, y) {
    return x + y;
};
```

In index.js file:

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

Run → node index.js

NPMjs (Nodemon,Chalk, Validator)

Nodemon module

To install nodemon: npm install nodemon

Run nodemon filename \

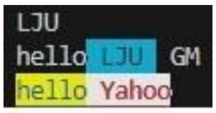
nodemon provides auto-run functionality for Node.js applications.

Chalk Module

To install validator: npm install chalk

```
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
```

Validator module

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

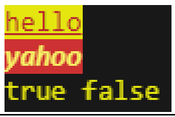
```
import v from "validator"
console.log(v.isJSON('{"name1":"ABC","age":30}'))
```

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.
These parameters provide valuable information related to a module.

Example:

```
console.log(__filename);
console.log(__dirname);
Output:
D:\node\e1.js  //returned path of current file
D:\node     //returned path till current file (folder)
```

Events

Initialize Event using core module named “events”

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

const ee = new EventEmitter();
```

Different methods:

Syntax	
eventEmitter.emit(event, [arg1], [arg2], [...])	Emits (triggers) an event. Any listeners for that event get called.
eventEmitter.on(event, listener)	Registers a listener for the specified event.
eventEmitter.addListener(event, listener)	This is an alias for .on(), works exactly the same.
eventEmitter.removeListener(event, listener)	Removes a specific listener for the event.
eventEmitter.removeAllListeners([event])	Removes all listeners for the given event. If no event is passed, removes all listeners for all events.
eventEmitter.listenerCount(event)	It returns the number of listeners listening to the specified 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");
```

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")
```

Output:

```
2 for conn event

listener1 executed

listener2 executed

1 for conn event

listener2 executed
```

Note: Must Do practice on Extra Tasks..