Remember

JSON (Java Script Object Notation)

- ✓ JSON stands for **J**ava**S**cript **O**bject **N**otation
- ✓ It is a **text format** for storing and transporting data
- ✓ It is "self-describing" and easy to understand
- ✓ It is a lightweight data-interchange format
- ✓ It is language independent

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- *null*

JSON values **cannot** be one of the following data types:

- a function
- a date
- *undefined*

These values can not be parsed.

Example:

```
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()

- ✓ When receiving data from a web server, the data is always a string.
- ✓ Parse the data with **JSON.parse()**, and the data becomes a JavaScript object.

We received this text from a web server:

```
'{"var1":"LJ","var2":"University"}'
```

So, we need to parse it to convert it to OBJECT.

JSON.stringify()

- ✓ When sending data to a web server, the data has to be a string.
- ✓ Convert a JavaScript object into a string with **JSON.stringify().**

we have this object in JavaScript:

```
{"var1":"LJ","var2":"University"}
```

Use the JavaScript function **JSON.stringify()** to convert it into a string.

Example

```
console.log(JSON.parse("{'p':50}");
```

This is **not valid JSON**, because JSON requires double quotes for keys and strings. This will throw an error.

Valid JSON string

```
console.log(JSON.parse('{"p":50}'));
```

Write a function named "firstlast" that takes an array and returns an object with below conditions.

1) **The first element of an array must be an object's key**
2) **The last element of an array must be a key's value.**

Input = ['abc','def','ghi','jkl']

Output = { abc: 'jkl' }

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

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

Output:

{ "abc": "jkl" }

Output = { "0": "ghi" }

```
function firstlast(a) {
   var temp = {};
   temp[0] = a[2];
   return temp;
}

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

Chapter 2 Node JS

- ✓ Latest version of NODE JS is 21.7.2.
- ✓ 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**.
- ✓ **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.
- ✓ 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**

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)

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.
- **.editor** – enable editor mode to perform tasks

- **.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. (.save j1.js)
- **.load *filename*** − load file content in current Node REPL session. (.load j1.js)

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.

Example

```
> 7==="7"
false
> a=_+20
20
> 7==”7”
true
> console.log(_)
true
undefined
> _+30
NaN
> p=true
true
> d=_+29
30
```

Explanation

1	7 === "7"	false	**Strict comparison (different types)**
2	a = _ + 20	20	**_ is false (false is 0, so 0 + 20 = 20)**
3	7 == "7"	true	**Loose comparison**
4	console.log(_)	true,undefined	**_ becomes true but console.log() returns undefined**
5	_ + 30	NaN	**undefined + 30 is NaN**
6	p = true	true	**Assigns true to p**
7	d = _ + 29	30	**_ is true, true is 1, so 1 + 29 = 30**

Example

What will be the output of following code? Consider output Only after completion of loop. (undefined is ignored)

```
>t=2
>t = t+_
do{
  … ++t;
  … console.log(t);
…}while(t<10)
```

Output: 5 6 7 8 9 10

File System (fs) Module (Synchronous)

Write node Example with File system methods. (CRUD Operation)

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());  //Or add “utf-8”

//data=ps.readFileSync("node/write.txt",”utf-8”);

ps.renameSync("node/write.txt"," node/readwrite.txt")
ps.unlinkSync("node/readwrite.txt");
```

```
d = data.sort((a,b)=>a-b);
```

Asynchronous Programming Using Callbacks

A callback is just a function passed as an argument to another function, usually to be **executed later**.

```
function sayHello(name) {
  console.log("Hello, " + name);
}
function greetUser(callback) {
  const userName = "Alice";
  callback(userName); // invoke the callback
}

greetUser(sayHello);
```

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

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

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

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

Asynchronous File system (Non-blocking concept)

Example to understand writeFile, appendFile, readFile, rename, unlink methods.

```
fs = require("fs")
fs.writeFile('test1.txt', 'Hello World!', (err) => {
   if (err) { console.log("Error Generated"+err); }
   else { console.log("Written"); }
});

fs.appendFile('test1.txt', '\nGood Morning!', (err) => {
  if (err) { console.log("Error Generated"+err); }
  else { console.log("Appended"); }
});

fs.readFile('test1.txt',"utf-8", (readErr, data) => {
   if (readErr) { console.error("Error Generated: "+readErr) }
   else { console.log(data); }
})

fs.rename('test1.txt','test2.txt',() => {console.log("Renamed")})

fs.unlink('test2.txt', unlinkErr => {
  if (unlinkErr) { throw unlinkErr; }
  else { console.log("Deleted")}
});

console.log("Last sentence")
```

Output is dependent on time to be taken to complete the particular task.

Write a Node.js script that asynchronously writes data to a file named 'test1.js'.
If no error occurs during the writing process, the script should then append additional data to the same file.
Finally, it should read the content of the file, including the newly written and appended data, and display it in the console.

Or

Writing data to file, appending data to file and then reading the file data using using ES6 callback.

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

OS Module : Operating System

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

Example:

```
var os=require("os");
console.log(os.arch());
console.log(os.hostname());
console.log(os.platform());
console.log(os.tmpdir());
```

Output:

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

Note:
os.freemem() = bytes,
os.freemem()/1024 = KB,
os.freemem()/1024/1024 = MB,
os.freemem()/1024/1024/1024 = GB

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");
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);
```

Output:

```
D:/LJ
abc.txt
.txt
{
  root: 'D:/',
  dir: 'D:/LJ',
  base: 'abc.html',
  ext: '.html',
  name: 'abc'
}
```

What will be the root and name values for the path "/LJ/abc.html"?

```
var pm=require("path");
path=pm.parse("/LJ/abc.html");
console.log(path);
console.log("root: "+path.root+"\nname: "+path.name);
```

Output:

```
{ root: '/', dir: '/LJ', base: 'abc.html', ext: '.html', name: 'abc' }
root: /
name: abc
```

HTTP Module

Add an HTTP Header

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

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

```
const http = require('http');
const server = http.createServer((req, res) => {
  if (req.url === "/") {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.end(`
      <h1>Welcome to Home Page</h1>
      <a href="/about">About Us</a><br><a href="/contact">Contact Us</a>
    `);
  } else if (req.url === "/about") {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.end(`<h1 style="color: blue; font-weight: bold;">Welcome to LJ University</h1>`);
  } else if (req.url === "/contact") {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.end(`<h2 style="color: red; font-style: italic;">Email: abc@ljinstitutes.edu.in</h2>`);
  } else {
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    res.end("Page not found");
  }
```

```
});

server.listen(3000, () => {console.log("Server started");});
```

Write a Node.js program to create a web server that serves a simple HTML file. When a user requests the URL in a browser, the server should read the HTML file using URL module and send its content as the HTTP response.

```
var h=require("http");
var fs=require("fs");
var url=require("url");
var server=h.createServer(
   function(req,res)
   {
     var q=url.parse(req.url,true);
     data=fs.readFileSync("."+q.pathname);
         res.writeHead(200,{"content-type":"text/html"});
         res.write(data);
         console.log(data);
         res.write("<h1> hello</h1>")
         res.end();
});
server.listen(6055);
```

NOTE:

You need "." when constructing a **dynamic file path** from req.url (or similar sources), because:

1. **req.url includes / at the beginning**
2. If req.url = "/1.html", then "1.html" works, but "/1.html" (absolute path) does not.
3. "." + req.url ensures the path remains relative: "./1.html".

In below example res.write() is used **after** res.end(), which is **not allowed**.

```
const http = require('http');
const server = http.createServer((req, res) => {
res.writeHead(200, { 'Content-Type': 'text/html' });
res.end(`<h1>Welcome to Home Page</h1>`);
res.write("hello")
});
server.listen(3000);
```

res.end() sends the response and **closes the connection**.
res.write("hello") **after res.end() causes an error**:
Error [ERR_STREAM_WRITE_AFTER_END]: write after end

Nodemon

npm install -g nodemon

To check version

Nodemon -v

Once nodemon is installed it might throw an error. We need delete the file as mention in error. Below is the file path.

C:/user/LJENG (This will be different) /Appdata/Roaming/npm/nodemon.ps1 (Follow the path and delete nodemon.ps1 file)

npm list -g command is used to check the path.

To run the file use below command

nodemon first.js

Chapter 3 Node JS

URL module

```
var u=require("url");
var addr="http://localhost:8080/default.html?year=2025&month=feb";
var q=u.parse(addr,true);
console.log(q);
```

==========

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'
}
```

Own Module

Method 1

In test.js file:

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

In another file test1.js where you want to use that module:

```
var a1=require("./test.js");
console.log(a1(10,15));
```

to run -> **node test1.js**

Method 2

In test.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 test1.js:

```
var a1=require("./test.js");
console.log(a1.s(10,5));
console.log(a1.m(10,15));
```

to run -> **node test1.js**

Method 3

In test.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 test1.js:

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

to run -> **node test1.js**

Method 4

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

to run -> **node test1.js**

Method 5

In test.js file:

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

In another file:

```
const a1 = require('./test.js');
console.log("Addition is: "
  + a1.add(50, 20));
```

to run -> **node test1.js**

Create **own Node.js module** (t1.js).

1. **Parses a given URL** to extract query parameters.
2. **Validates** that all parameters (a, b, c, d) are non-negative integers.
3. **Evaluates the mathematical expression**: a*c - a/d + b
4. **Returns the computed result** or an error message if any parameter is negative.

The main script (t2.js) should require the module. Pass a sample URL (**"http://example.com/calculate?a=20&b=30&c=40&d=-1"**) with query parameters (a=20, b=30, c=40, d=-1). Display the evaluation result.

t1.js

```
const url = require('url');
function evaluate(input) {
  const parsedUrl = url.parse(input, true);
  const query = parsedUrl.query; // Extract query parameters
  let a = parseInt(query.a);
```

```
    let b = parseInt(query.b);
    let c = parseInt(query.c);
    let d = parseInt(query.d);

    // Check if all parameters are >0
    if (a<0 || b<0 || c<0 || d<0 ) {
        return "Invalid input. Please provide valid numbers";
    }
    const result = a * c - a / d + b;
   console.log(`Result of expression (a*c - a/d + b) = ${result}`);
}
module.exports = evaluate;
```

t2.js

```
const eval = require('./t1');
const sampleUrl = "http://example.com/calculate?a=20&b=30&c=40&d=-1";
// Get and display the result
console.log(eval(sampleUrl));
```

Chalk module

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:

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
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
```

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),
ch.red.underline.bold.bgWhite(validator.isEmail(test)))
```

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
});
```

Example:

```
console.log(__filename);
console.log(__dirname);
Output:
D:\Trynode\ex1.js  //returned path of current file
D:\Trynode     //returned path till current file (folder)
```

Event Module

You initialize that using

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

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(): It returns the number of listeners listening to the specified event.

Example:

```
const EventEmitter = require('events');
const emitter = new EventEmitter();
function onLogin(user) {
  console.log(`${user} logged in.`);
}
emitter.on('login', onLogin);
emitter.emit('login', 'Test');   // Output: Test logged in.
console.log(emitter.listenerCount('login')); // 1
emitter.removeListener('login', onLogin);
emitter.emit('login', 'Test1');     // No output
console.log(emitter.listenerCount('login')); // 0
```

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("data-write",function()
{
   fs.writeFile("b.txt","Hello, ",(err)=> {console.log()});
   console.log("Data Written");
   ee.emit("data-append");
   ee.emit("data-read");
});
ee.on("data-append",function()
{
   fs.appendFile("b.txt","Good Morning!",(err)=> {console.log()});
   console.log("Data Appended");
});
ee.on("data-read",function()
{
   fs.readFile("b.txt","utf-8",(err,data)=>
   {
      if(err){
      console.error(err);
      }
      console.log(data);
   });
});
ee.emit("data-write");
```