**Node JS**

* Latest version of NODE JS is 20.0.0. 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 stands for**

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

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

It performs the following tasks −

* **Read** − Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
* **Eval** − Takes and evaluates the data structure.
* **Print** − Prints the result.
* **Loop** − Loops the above command until the user presses **ctrl-c** twice.

### **Starting REPL**

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

$ node

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

$ node

>

### **Simple Expression**

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

$ node

> 1 + 3

4

> 1 + ( 2 \* 3 ) - 4

3

>

### **Use Variables**

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

$ node

> x = 10

10

> var y = 10

undefined

> x + y

20

> console.log("Hello World")

Hello World

undefined

### **Multiline Expression**

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

$ node

> var x = 0

undefined

> do {

... x++;

... console.log("x: " + x);

... }

while ( x < 5 );

x: 1

x: 2

x: 3

x: 4

x: 5

undefined

>

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

### **Underscore Variable**

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

$ node

> var x = 10

undefined

> var y = 20

undefined

> x + y

30

> var sum = \_

undefined

> console.log(sum)

30

undefined

>

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

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

const fun=(a,b)=&gt;

{console.log(“Hello”);

return a+b;

}

console.log(”Addition is =”,fun(10,20));

//Output:

Hello

Addition is = 30

Undefined

var t=55;

undefined

do{

... t++

... console.log(t);

... } while(t&lt;=60)

56

57

58

59

60

61

undefined

## REPL Commands

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

To remove undefined error

**“repl.repl.ignoreUndefined = true”**

**Node.js uses asynchronous programming!**

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

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

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

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

## File System module

The Node.js file system module allows you to work with the file system on your computer.

To include the File System module, use the require() method:

**var fs = require('fs');**

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

**Example of File system with blocking(Synchronous) concept**

* var ps=require("fs");
* var data=ps.readFileSync("Hello.txt");
* console.log(data)
* console.log(data.toString());
* console.log("Program ended");
* **Output:**
* Hello. welcome to LJU
* Program ended

## Asynchronous Programming Using Callbacks

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

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

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

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

**How to create a Callback?**

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

const message = function() {

console.log("This message is shown after 3 seconds");

}

setTimeout(message, 3000);

There is a built-in method in JavaScript called “setTimeout”, which calls a function or evaluates an expression after a given period of time (in milliseconds). So here, the “message” function is being called after 3 seconds have passed. (1 second = 1000 milliseconds)

In other words, the message function is being called after something happened (after 3 seconds passed for this example), but not before. So the message function is an example of a callback function.

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

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

**What is an Anonymous Function?**

Alternatively, we can define a function directly inside another function, instead of calling it. It will look like this:

setTimeout(function() {

console.log("This message is shown after 3 seconds");

}, 3000);

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

**Callback as an Arrow Function**

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

setTimeout(() => {

console.log("This message is shown after 3 seconds");

}, 3000);

===========================================================================

**Example of File system with Non blocking(ASynchronous) concept**

By using callbacks, we can write asynchronous code in a better way. The following example creates a new file called test.txt and writes "Hello World" into it asynchronously.

var fs = require('fs');

fs.writeFile('test.txt', 'Hello World!', function (err) {

if (err)

console.log(err);

else

console.log('Write operation complete.');

});

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

var ps=require("fs");

ps.readFile("Hello.txt",function(err,data)

{

    if(err)

    {

        return console.error(err);

    }

    console.log(data.toString());

    console.error("completed");

}

);

console.log("Program ended");

**Output:**

Program ended

Hello. welcome to LJU

Completed

**Some callback examples**

<html>

<head>

</head>

<body>

<p id="id"></p>

<script>

setTimeout(myfun,5000);

function myfun()

{

document.getElementById("id").innerHTML="LJU";

}

</script>

</body>

</html>

<html>

<head>

</head>

<body>

<p id="demo"></p>

<script>

function mydisplay(sum)

{

document.getElementById("demo").innerHTML="<b>"+ sum +"</b>";

}

function mycals(num1,num2,mycallback)

{

sum=num1+num2;

mycallback(sum);

}

mycals(13,15,mydisplay);

</script>

</body>

</html>

<html>

<head>

</head>

<body>

<p id="p1"></p>

<script>

function add(a,b)

{

obj=document.getElementById("p1");

obj.innerHTML=(a+b);

}

a=2;

b=5;

setInterval(

function()

{

add(++a,++b);

},1000

);

</script>

</body>

</html>

<html>

    <body>

        <p id="p1"> Hello</p>

        <button onclick="fun1()">font-size</button>

        <script>

        font="15";

        function fun(font)

        {

           document.getElementById("p1").style.fontSize=font;

           document.getElementById("p1").style.color= "blue";

        }

        function fun1()

        {

            setInterval(

                function()

                {

                    if(font<=50)

                    {

                        fun(font++);

                    }

                },50

            );

        }

    </script>

    </body>

</html>

<html>

    <body>

        <p id="demo" style="color:blue"></p>

        <script>

            size = 15;

            function add() {

                obj = document.getElementById("demo");

                obj.innerHTML = "hello";

                obj.style.color ="red";

                obj.style.fontSize = size;

                if (size <= 50) {

                    size++;

                }

            }

            setInterval(add, 1000);

        </script>

    </body>

</html>

**Write node Example with File system methods.**

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

var ps=require("fs");

ps.mkdirSync("node");

ps.writeFileSync("node/write.txt","Hello");

ps.appendFileSync("node/write.txt","Hi");

data=ps.readFileSync("node/write.txt");

console.log(data);

console.log(data.toString());

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

ps.unlinkSync("node/readwrite.txt");

**Read data from file and sort that data in ascending order.**

var ps=require("fs");

ps.writeFileSync("s1.txt","50 -1 99 20 0 56 78 59");

data=ps.readFileSync("S1.txt","utf-8");

data=data.split(" ");

data.sort();

console.log(data);

**Output:**

[

'-1', '0', '20',

'50', '56', '59',

'78', '99'

]