

Experiment No. 6

Aim:

Node.js: Installation and Configuration, Callbacks, Event loops,
Creating express app.

A. Theory:

1. What is Node.js?

Node JS is open source server environment. It allows to run JavaScript on the server.

2. How to use Node.JS

Create a Node.js file named "myfirst.js", and add the following code:

myfirst.js

```
var http = require('http');
```

```
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!'); }).listen(8080);
```

Save the file on your computer: C:\Users\Your Name\myfirst.js

The code tells the computer to write "Hello World!" if anyone (e.g. a web browser) tries to access your computer on port 8080.

Command Line Interface

Node.js files must be initiated in the "Command Line Interface" program of your computer.

Navigate to the folder that contains the file "myfirst.js", the command line interface window should look something like this:

```
C:\Users\Your Name>_
```

Initiate the Node.js File

The file you have just created must be initiated by Node.js before any action can take place.

Start your command line interface, write `node myfirst.js` and hit enter:

Initiate "myfirst.js":

```
C:\Users\Your Name>node myfirst.js
```

Now, your computer works as a server!

If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!

Start your internet browser, and type in the address: <http://localhost:8080>

1. Event Loop

```
console.log("This is the first statement");  
  
setTimeout(function(){  
    console.log("This is the second statement");  
}, 1000);  
  
console.log("This is the third statement");
```

Output:

This is the first statement

This is the third statement

This is the second statement

Explanation: In the above example, the first console log statement is pushed to the call stack, and "This is the first statement" is logged on the console, and the task is popped from the stack. Next, the `setTimeout` is pushed to the queue and the task is sent to the Operating system and the timer is set for the task. This task is then popped from the stack. Next, the third console log statement is pushed to the call stack, and "This is the third statement" is logged on the console and the task is popped from the stack.

When the timer set by the `setTimeout` function (in this case 1000 ms) runs out, the callback is sent to the event queue. The event loop on finding the call stack empty takes the task at the top of the event queue and sends it to the call stack. The callback function for the `setTimeout` function runs the instruction and "This is the second statement" is logged on the console and the task is popped from the stack.

3. Callbacks

Node.js callbacks are a special type of function passed as an argument to another function. They're called when the function that contains the callback as an argument completes its execution, and allows the code in the callback to run in the meantime. Callbacks help us make asynchronous calls.

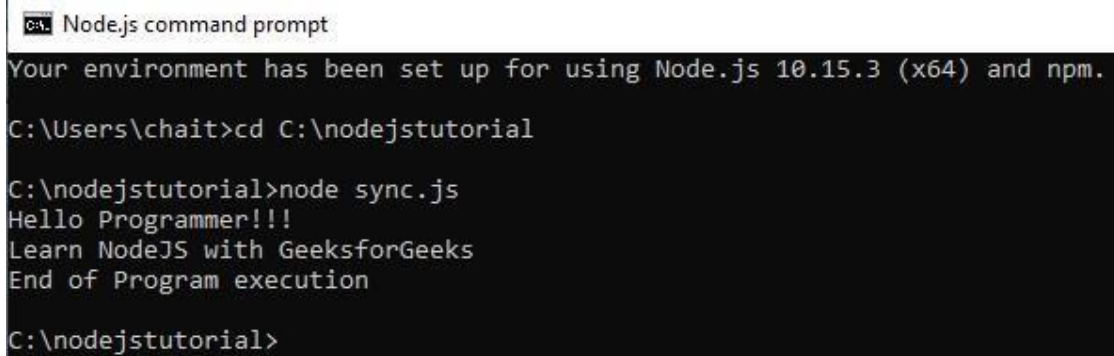
A **callback** is a function that is called when a task is completed, thus helping in preventing any kind of blocking and a callback function allows other code to run in the meantime.

Using the Callback concept, Node.js can process a large number of requests without waiting for any function to return the result which makes Node.js highly scalable. For example: In Node.js, when a function starts reading the file, it returns the control to the execution environment immediately so that the next instruction can be executed. Once file I/O gets completed, the callback function will get called to avoid blocking or waiting for File I/O.

Example 1: without callback

```
const fs = require("fs");
const filedata = fs.readFileSync('inputfile1.txt');
console.log(filedata.toString()); console.log("End
of Program execution");
```

Explanation: the `fs` library is loaded to handle file-system-related operations. The `readFileSync()` function is synchronous and blocks execution until finished. The function blocks the program until it reads the file and then only it proceeds to end the program **Output:**



```
Node.js command prompt
Your environment has been set up for using Node.js 10.15.3 (x64) and npm.

C:\Users\chait>cd C:\nodejstutorial

C:\nodejstutorial>node sync.js
Hello Programmer!!!
Learn NodeJS with GeeksforGeeks
End of Program execution

C:\nodejstutorial>
```

Example 2: with callback

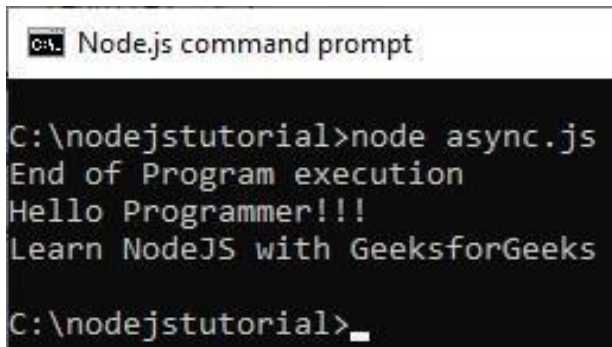
Code for reading a file asynchronously (non-blocking code) in Node.js. Create a text file **inputfile1.txt** with the following content. **Hello Programmer!!!**

Learn NodeJS with GeeksforGeeks

Create an **async.js** file with the following code:

```
// Write a JavaScript code
const fs = require("fs");
fs.readFile('inputfile1.txt', function (ferr, filedata) {
  if (ferr) return console.error(ferr);
  console.log(filedata.toString());
});
console.log("End of Program execution");
```

Explanation: the *fs* library is loaded to handle file-system related operations. The `readFile()` function is asynchronous and control return immediately to the next instruction in the program while the function keep running in the background. A callback function is passed which gets called when the task running in the background are finished.

Output:A screenshot of a Windows command prompt window titled "Node.js command prompt". The window shows the following text:

```
C:\nodejstutorial>node async.js  
End of Program execution  
Hello Programmer!!!  
Learn NodeJS with GeeksforGeeks  
C:\nodejstutorial>_
```

Express App: Express.js

Express is a popular and modern web app development framework that includes all of the latest features and capabilities. It makes it simple to create modern, powerful applications.

Express is built on top of Node. Express adds more features while building apps. Node is just a Javascript environment with libraries to make it easy to write software, whereas Express extends Node specifically to add middleware, routing, and much more.

Express provides methods to specify what function is called for a particular HTTP verb (GET , POST , SET , etc.) and URL pattern ("Route"), and methods to specify what template ("view") engine is used, where template files are located, and what template to use to render a response.

B.Program:**Event_Loop.js:**

```
console.log("Fetching data from server...");
setTimeout(() => {
    console.log("Fetched data from server");
}, 1000);
setTimeout(() => {
    console.log("Recieved data")
}, 2000);
```

Callback.js:

```
const fs = require('fs');
const filePath = 'example.txt';

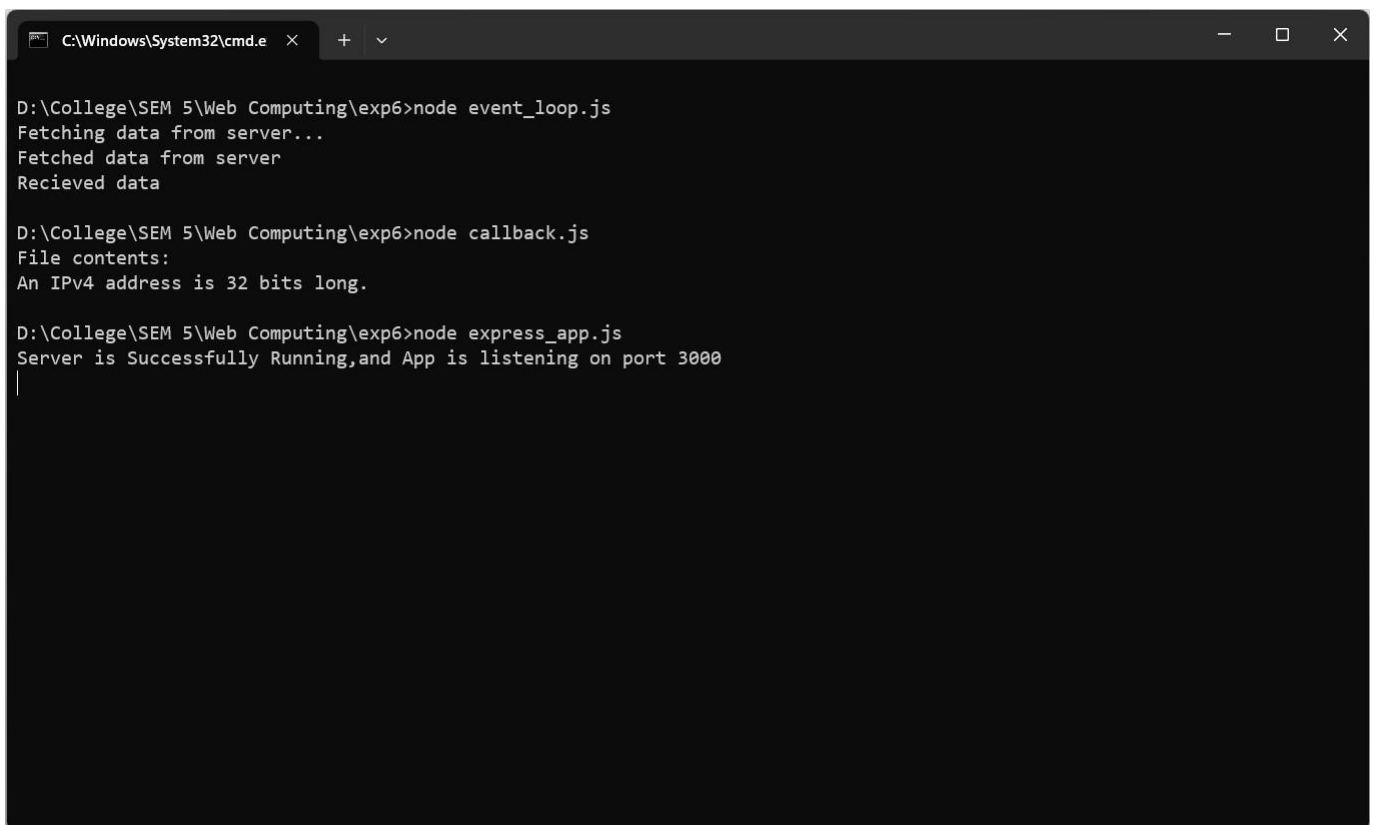
fs.readFile(filePath, 'utf8', (err, data) => {
    if (err) {
        console.error('Error reading the file:', err);
        return;
    }
    console.log('File contents:');
    console.log(data);
});
```

Express_app.js:

```
const express = require('express');

const app = express();
const PORT = 3000;

app.get('/', (req, res)=>{
    res.status(200);
    res.send("Welcome to root URL of Server");
});
app.listen(PORT, (error) =>{
    if(!error)
        console.log("Server is Successfully Running,and App is listening on port "+ PORT)
    else
        console.log("Error occurred, server can't start", error);
    }
});
```

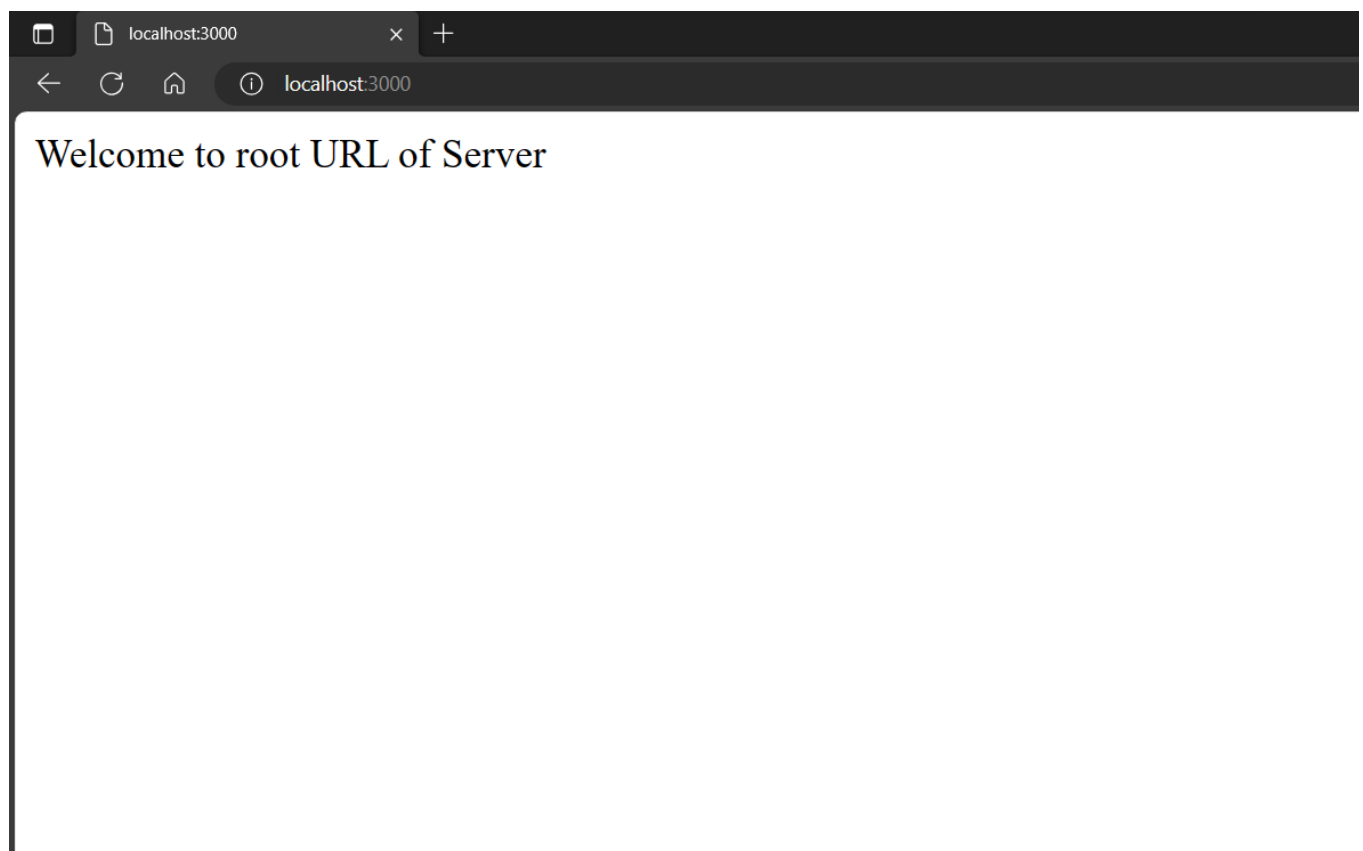
C.Output and findings:

```
C:\Windows\System32\cmd.e  X + v

D:\College\SEM 5\Web Computing\exp6>node event_loop.js
Fetching data from server...
Fetched data from server
Recieved data

D:\College\SEM 5\Web Computing\exp6>node callback.js
File contents:
An IPV4 address is 32 bits long.

D:\College\SEM 5\Web Computing\exp6>node express_app.js
Server is Successfully Running, and App is listening on port 3000
|
```



D. Conclusion

Thus, we have successfully learnt how to install and configure Node JS and performed the experiment to demonstrate various concepts like event loops and callbacks; along with creating a simple Express app.