NodeJS

NodeJS : Content

- What is NodeJS?
- Installing NodeJS?
- Verify NodeJS and npm
- What is REPL and How to use It?
- Node Modules
  - Events Module
  - OS Module
  - HTTP Module
  - File I/O Module
- Environment Variables dotenv
- Questionnaire

What is Node Js?

- Node.js is a popular open-source, cross-platform runtime environment that allows developers to run JavaScript code on the server side

Features:

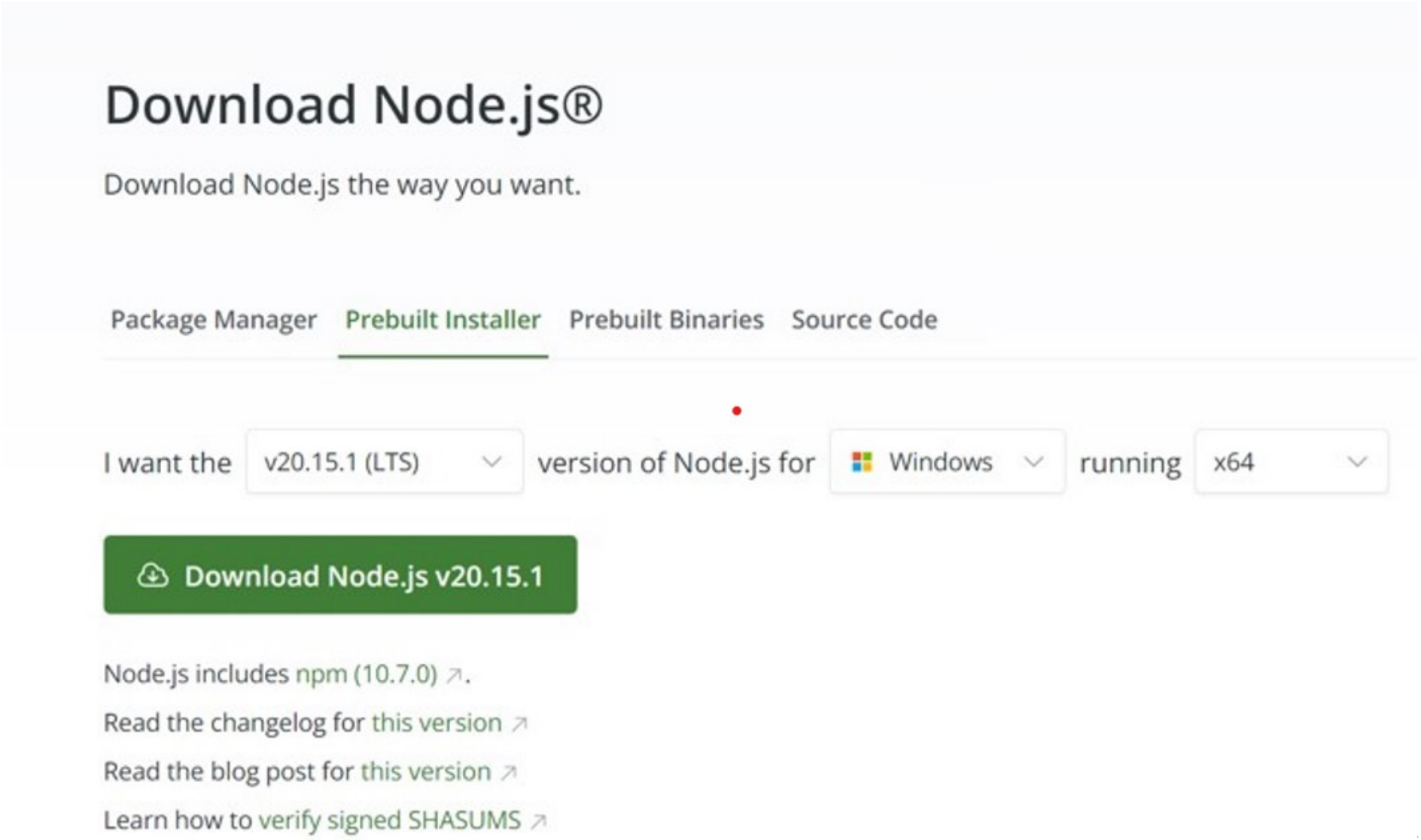
- Event-Driven Architecture: Node.js uses an event-driven, non-blocking I/O model. This means it can handle many connections simultaneously without waiting for any single process to complete before moving on to the next one, which improves performance and scalability.
- Single Programming Language: With Node.js, you can use JavaScript for both client-side and server-side programming, allowing for a more streamlined development process and codebase.
- npm (Node Package Manager): Node.js comes with npm, the largest ecosystem of open-source libraries and tools. This makes it easy to manage dependencies and use packages developed by the community.

What is Node Js?

- Asynchronous Programming: Node.js uses asynchronous programming techniques to handle tasks like reading from a file or making a network request. This helps in improving the performance of applications by not blocking the execution thread while waiting for these tasks to complete.
- Use Cases: Node.js is commonly used for building web servers, APIs, real-time applications (like chat applications or online gaming), and various other network applications.

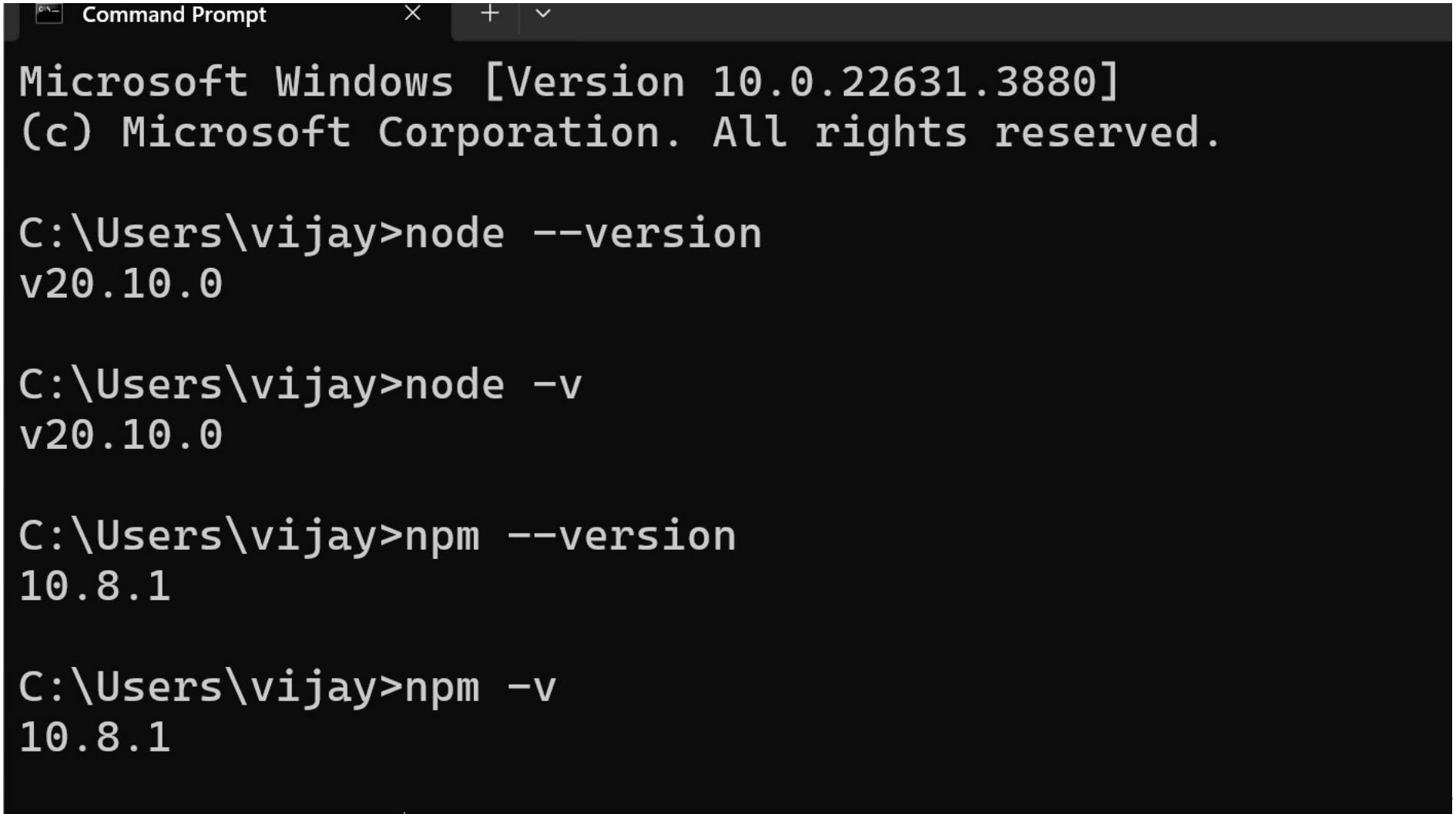
Installing Nodejs

Download Link: https://nodejs.org/en/download/prebuilt-installer



Verify Node Installation

- Node: Go to the command prompt and type node –v or node –version
- npm: Go to the command prompt and type npm –v or npm –version

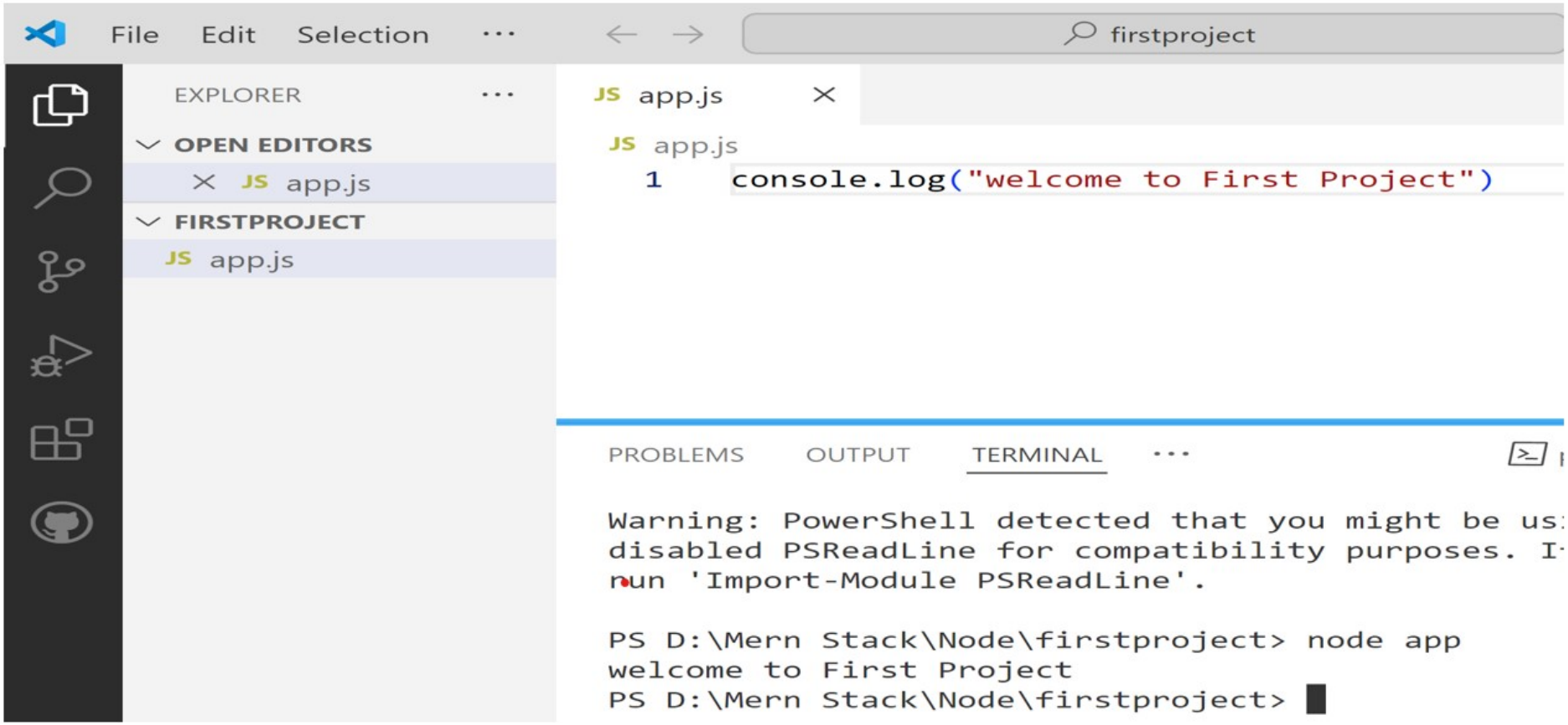


Create Project

- Create a project folder in the machine.
- Open project folder using Visual studio Code.
- Create a file app.js with sample code below:

Create Project

How to run: Open the terminal and type "node app" to run the project, Check below:



Create Project

`Every time If we make any changes in the project we have to stop the application and re-start the application`

Stop: `Ctrl+C`

Start: `node app`

Note: To overcome this problem a tool called nodemon is introduced

Nodemon:

- Nodemon is a command-line tool that helps with the speedy development of Node.js applications.
- It monitors your project directory and automatically restarts your node application when it detects any changes.
- It Means that we do not have to stop and restart your applications in order for your changes to take effect.
- Simply change the code and save the changes and see the output few minutes later.

Create Project

Install nodemon and use it:

- Open terminal and type the following command.
  npm install -g nodemon
- Run the project using the command.
  nodemon app

Note: If any error occurs do the following and run the project

- Click on the Start menu
- Type PowerShell.
- Right-click on Windows PowerShell and select Run as administrator.
- Run the following command
  `Set-ExecutionPolicy RemoteSigned`
- Now again run the project.

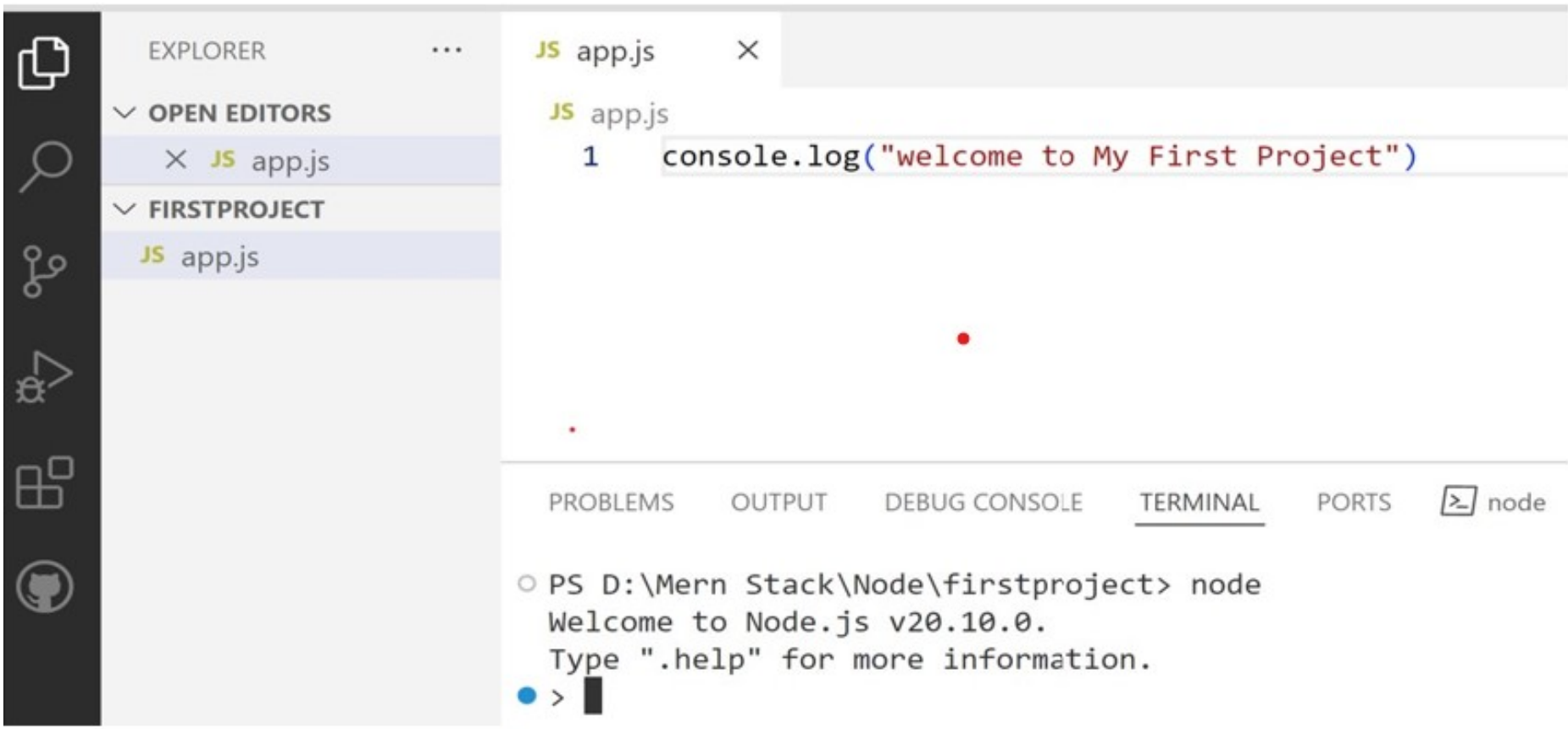
Note: If nodemon is installed successfully and able run the project using nodemon, we do not need to stop and start the application if we make any changes in the project.

REPL

- REPL Stands for Read-Eval-Print-Loop
- It is an interface where we can try for various nodejs commands or expressions and execute.
- REPL is nothing but functionality that facilitates programmers to execute codes at the command prompt.
- Nodejs comes with the REPL environment when it is installed by default.

REPL

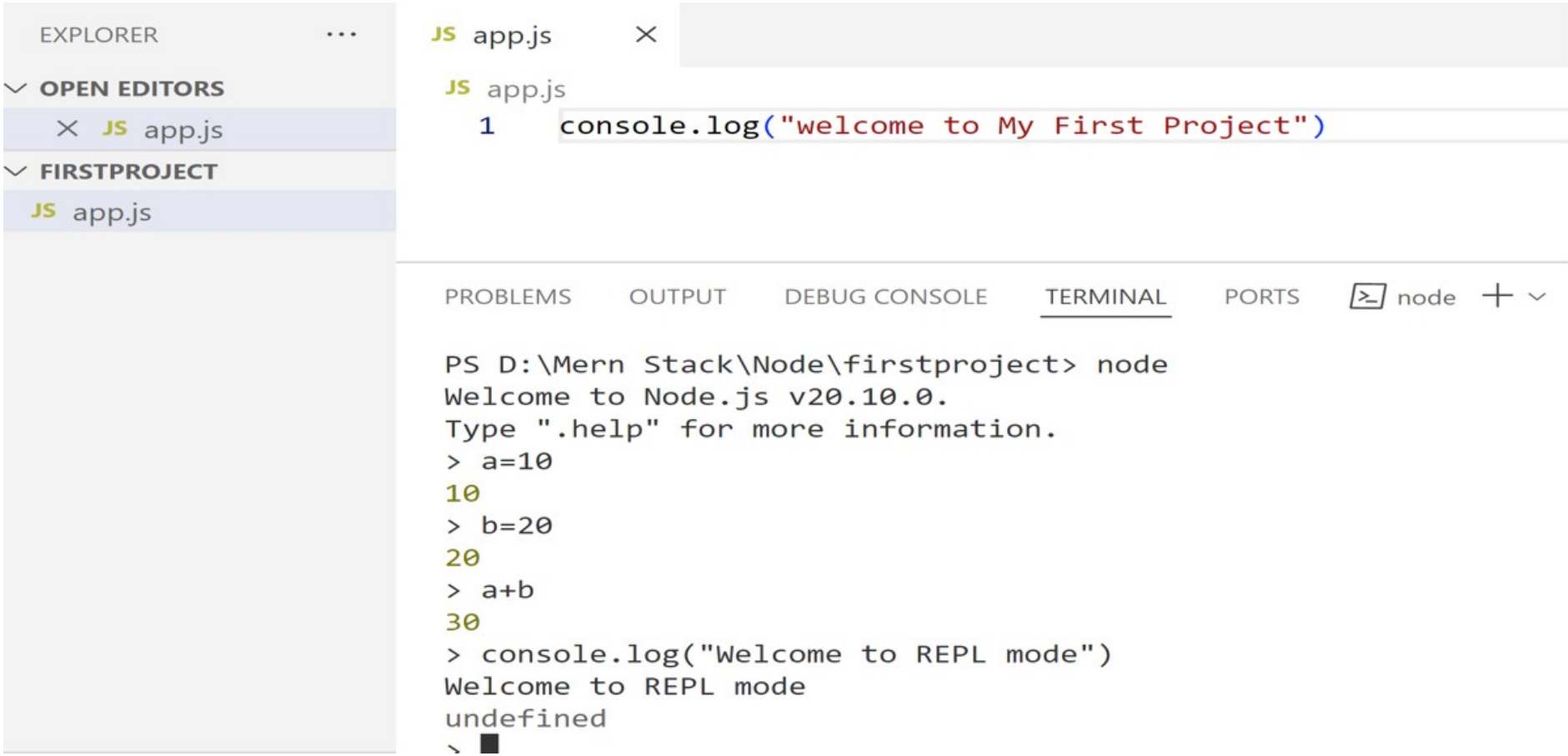
How to use it: Open the terminal window and type the command "node", We will move to REPL mode as shown below:



Important: .help command gives us Support, to know more about REPL

REPL

Let us evaluate some expressions:



Note: In JavaScript, If any function is not having a return statement, then by default the returned value is undefined.

REPL

Using REPL, We can also write and execute fuctions

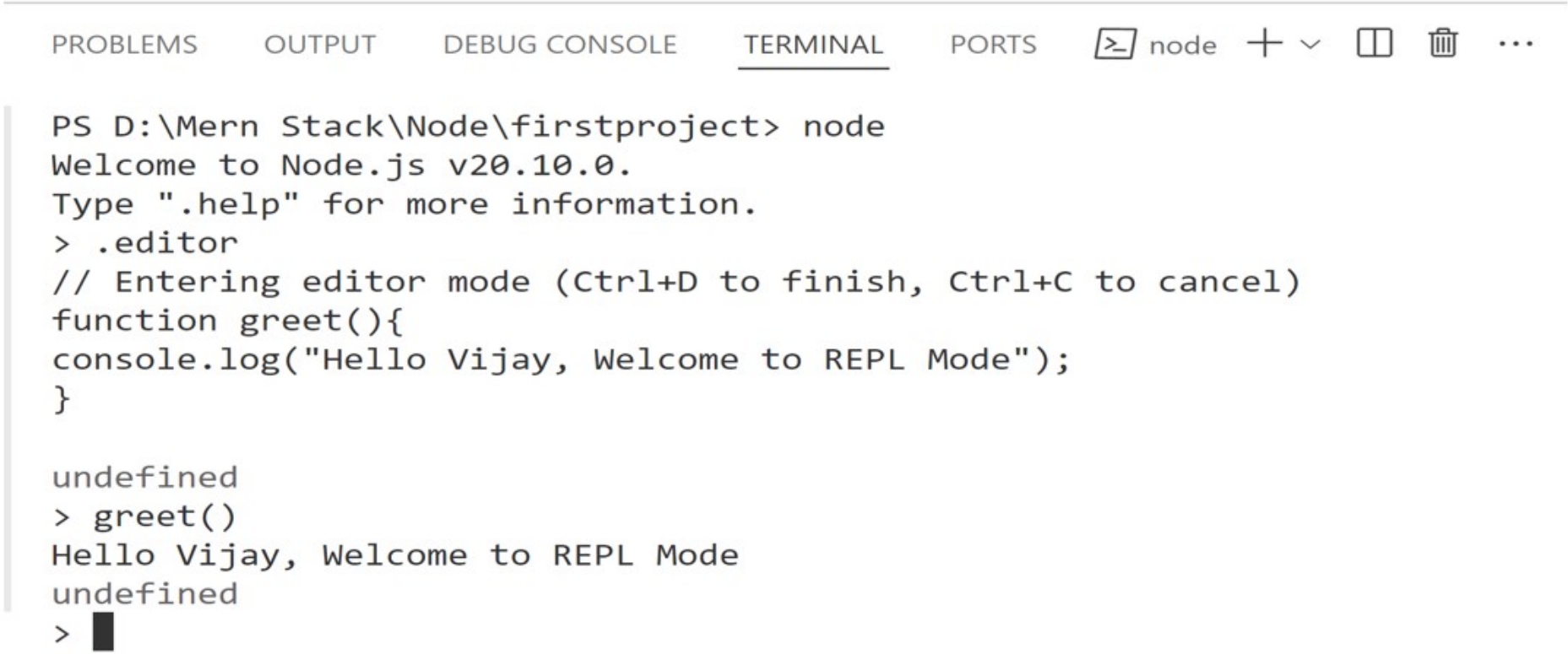
- Open terminal
- Type node to get into REPL mode
- Type .editor, that gives a provison to write a function and save
- write the function as shown below:

```
function greet(){
console.log("Hello Vijay, Welcome to REPL Mode");
}
```

- Press Ctrl+D to finish, Ctrl+C to cancel

REPL

Now execute the function at REPL terminal as shown below:



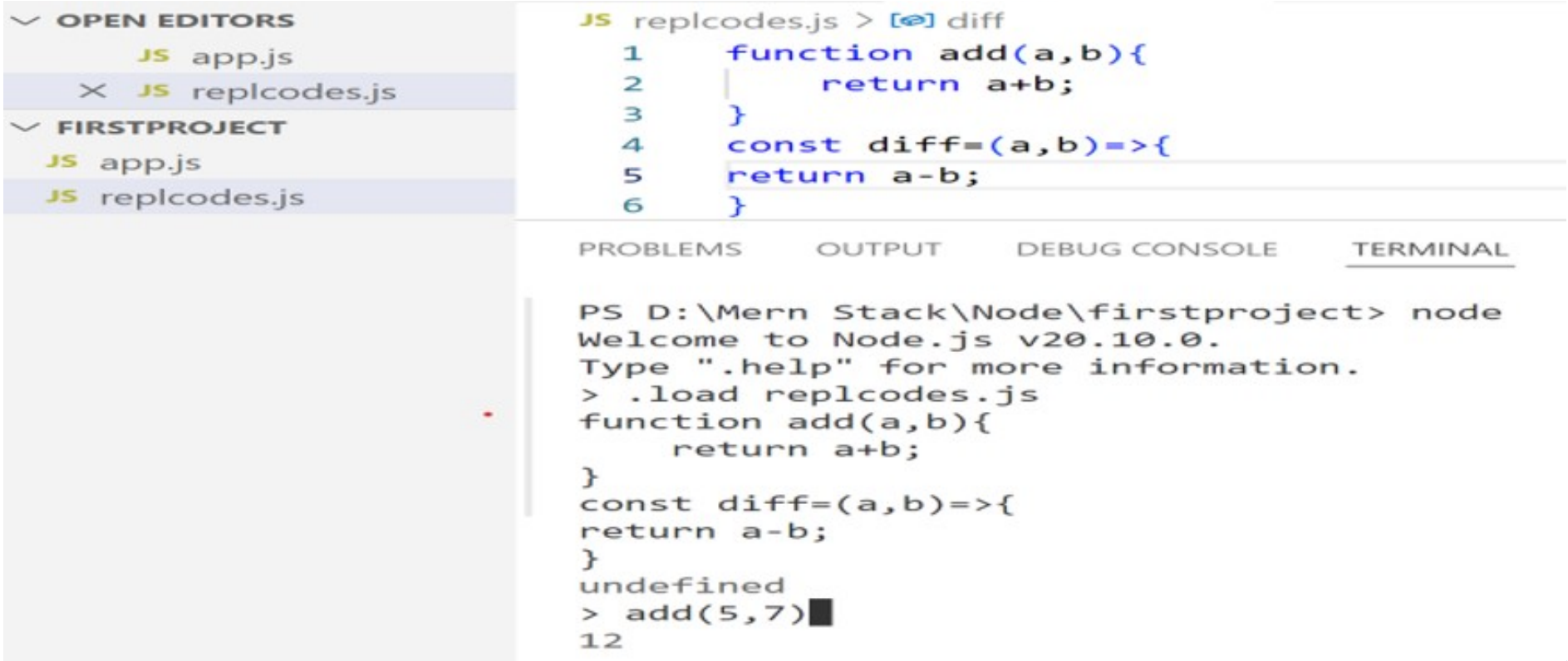
To save and come out of REPL mode, Press Ctrl+C twice

REPL

- We can load an existing js file and execute the code/functions in it.
- To load the file use the following command

  `.load replcode.js`
- Execute the functions in it to test as shown below:

See below screenshot for reference

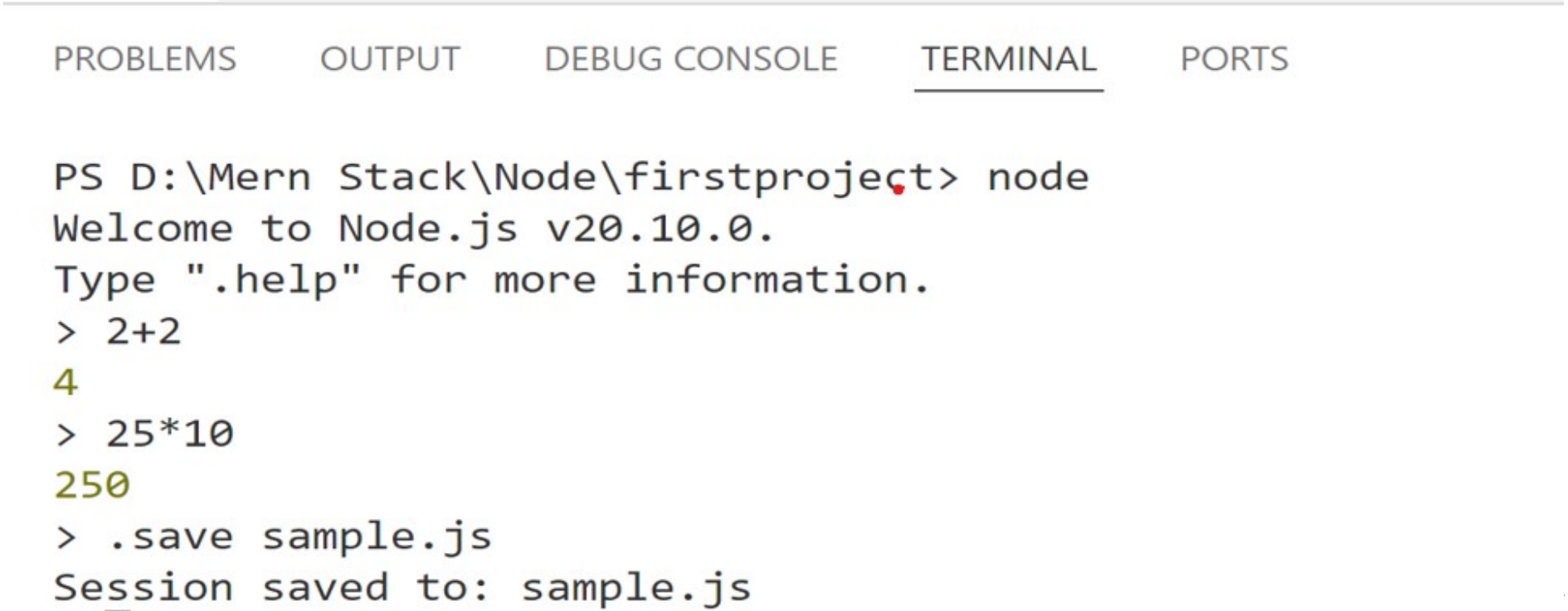


REPL

- we can save all evaluated commands in this REPL session to a file
- To save the file use the following command

  `.save savecode.js`
- A new file with the name savecode.js will be created in the current directory.

`See below screenshot for reference`



Node Modules

- A module is a piece of reusable JavaScript code
- Module is block of code that provide a simple or complex functionality encapsulated together.
- It could be a .js file or a directory containing .js files.
- We can export the content of these files and use them in other files.

Types of Node Modules:

`There are 3 types of node modules`

- Built-in modules
- Local modules
- Third-party modules

Node Modules

Built-in modules: Node.js has many built-in modules that are part of the platform and come with Node.js installation.
Syntax How we use them:

```
const module = require('module_name');
```

Example:

```
const http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('Welcome to Sample page returned from server
    !');
    res.end();
}).listen(3000);
```

The require() function returns an object because the Http module returns its functionality as an object.

Node Modules

Local modules: Local modules are created locally by the user in Node.js application.

There are two ways of creating a module:

One way:

Create a numberop.js file that has the following code:

```
exports.add = function (num) {
 let reversed = 0;
    while (num > 0) {
        // Get the last digit
        let digit = num % 10;
        reversed = reversed * 10 + digit;
        num = Math.floor(num / 10);
    }
    return reversed;
};
```

Node Modules

Since this file provides attributes outside of the module via exports, another file can use its exported functionality using the require() function. As below

```
const reverse = require('./numberop');
num=1234;
let ret=reverse.rev(num);
console.log(The reverse of given number is "+ret);
```

Output:

The reverse of given number is 4321

Node Modules

Another way:

Create a numberop.js file that has the following code:

```
function rev(num) {
    let reversed = 0;
        while (num > 0) {
            // Get the last digit
            let digit = num % 10;
            reversed = reversed * 10 + digit;
            num = Math.floor(num / 10);
        }
        return reversed;
   };

module.exports=rev;
```

Node Modules

Since this file provides attributes outside of the module via exports, another file can use its exported functionality using the require() function. As below

```
const reverse = require('./numberop');
num=2345;
let ret=reverse(num);
console.log("The reverse of given number is"+ret);
```

Output:

The reverse of given number is 5432

Node Modules

Third-party modules:

- Third-party modules are modules that are available online.
- Using the Node Package Manager(NPM) we install from the terminal into the project folder or globally.

Some of them are: mongoose, express, angular, and react.

Example: Installation

```
npm install express
```

Example: Create a Simple Express Server:

```
const express = require('express');
const app = express();
const port = 3000;
app.get('/', (req, res) => {
  res.send('Sample To demonstrate express module!');
});
app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${
    port}`);
});
```

Events

- Event is an action that happened in our application.
- Node.js has an event-driven architecture which can perform asynchronous tasks.
- Events is a module that allows us to handle events in nodejs.
- Using the events module we can dispatch/emit the custom events and respond/handle those events in non-blocking manner.

Syntax:

```
const EventEmitter=require('events');
var eventEmitter=new EventEmitter();
```

Listening events: Before emits any event, it must register functions(callbacks) to listen to the events.

Events

Syntax: These are the 3 ways that we can register the function

```
eventEmitter.addListener(event, listener)
eventEmitter.on(event, listener)
eventEmitter.once(event, listener)
```

- eventEmmitter.on(event, listener) and eventEmitter.addListener(event, listener) are identically same.
- eventEmitter.addListener(event, listener) is introduced in older version of nodejs and still exists for historical reasons and to maintain compatibility with older versions of Node.js.
- eventEmmitter.on(event, listener) is preferred and more conventional way to register event listeners in Node.js code.

Events

How it works:

- Nodejs adds the listener at the end of the listener's array for the specified event.
- Multiple calls to the same event and listener will add the listener multiple times and correspondingly fire multiple times.

Where as eventEmitter.once(event, listener) is bit different, registering the event process is same but, fires at most once for a particular event and will be removed from listeners array after it has listened once.

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], [...])
```

Events

Example:

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

// Initializing event emitter instances
var eventEmitter = new EventEmitter();

// Registering to myEvent
eventEmitter.on('myEvent', (msg) => {
    console.log(msg);
});

// Triggering myEvent
eventEmitter.emit('myEvent', "First event");
```

Output:

```
    First event
```

Events

Removing Listener: It is the concept of removing listener or listeners from listener's array for the specified event.

There are two ways of removing the listeners:

One way:

Syntax:

```
eventEmitter.removeListener(event, listener)
```

Example:

```
// Importing events
const EventEmitter = require('events');
// Initializing event emitter instances
var eventEmitter = new EventEmitter();
var fun1 = (msg) => {
    console.log("Message from fun1: " + msg);
};
var fun2 = (msg) => {
    console.log("Message from fun2: " + msg);
};
```

Events

Example Continued....

```
// Registering fun1 and fun2
eventEmitter.on('myEvent', fun1);
eventEmitter.on('myEvent', fun1);
eventEmitter.on('myEvent', fun2);
// Triggering myEvent
eventEmitter.emit('myEvent', "Event occurred"); // trigger
    fun1 twice
// Removes listener fun1 that was first registered
eventEmitter.removeListener('myEvent', fun1);
  // Triggering myEvent
eventEmitter.emit('myEvent', "Event occurred"); // trigger
    fun1 only one
```

Events

Output:

```
Message from fun1: Event occurred
Message from fun1: Event occurred
Message from fun2: Event occurred
Message from fun1: Event occurred
Message from fun2: Event occurred
```

Observations:

- In the above code, line 6 emits all the listeners registered under myevent.
- Line 8 removes at most one instance of the myevent listener which is in front of the queue.
  i.e., line 2 listener will be removed

Another way:

Syntax:

```
eventEmitter.removeAllListeners(event);
```

Example:

OS

- The os module in Node.js provides utilities for interacting with the operating system.
- It's built into Node.js and doesn't require any additional installation.
- This module provides methods and properties that allow you to retrieve information about the operating system and perform certain operations.

Common Methods and Properties of the Modules:

- os.arch(): Returns the operating system CPU architecture, such as 'x64', 'arm', etc.
- os.platform(): Returns the operating system platform, like 'linux', 'darwin' (macOS), 'win32', etc.
- os.cpus(): Returns information about each CPU/core installed, including model, speed (in MHz), and times.
- os.freemem(): Returns the amount of free system memory in bytes.
- os.totalmem(): Returns the total amount of system memory in bytes.

OS

- os.homedir(): Returns the home directory of the current user.
- os.hostname(): Returns the hostname of the operating system.
- os.uptime(): Returns the system uptime in seconds.
- os.networkInterfaces(): Returns an object containing network interfaces that have been assigned a network address.
- os.tmpdir(): Returns the operating system's default directory for temporary files.
- os.endianness(): Returns the endianness of the CPU.
- os.type(): Returns the operating system name.
- os.release(): Returns the operating system version.

OS

Example:

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

console.log('Operating System Information:');
console.log('----------------------------');
console.log(`OS Type: ${os.type()}`);
console.log(`OS Platform: ${os.platform()}`);
console.log(`OS Release: ${os.release()}`);
console.log(`CPU Architecture: ${os.arch()}`);
console.log(`CPU Endianness: ${os.endianness()}`);
console.log(`System Uptime: ${os.uptime()} seconds`);
console.log(`Load Average: ${os.loadavg()}`);
console.log(`Total Memory: ${(os.totalmem() / (1024 * 1024 *
    1024)).toFixed(2)} GB`);
console.log(`Free Memory: ${(os.freemem() / (1024 * 1024 *
   1024)).toFixed(2)} GB`);
console.log(`Home Directory: ${os.homedir()}`);
console.log(`Temporary Directory: ${os.tmpdir()}`);
console.log(`Host Name: ${os.hostname()}`);
```

OS

Output:

```
PS D:\Mern Stack\samples\NodeApp> node app
Operating System Information:
----------------------------
OS Type: Windows_NT
OS Platform: win32
OS Release: 10.0.22631
CPU Architecture: x64
CPU Endianness: LE
System Uptime: 471359.796 seconds
Load Average: 0,0,0
Total Memory: 15.79 GB
Free Memory: 6.83 GB
Home Directory: C:\Users\vijay
Temporary Directory: C:\Users\vijay\AppData\Local\Temp
Host Name: Vijay_Bollu
```

HTTP

The http module in Node.js is used to create an HTTP server and handle HTTP requests and responses.

Basic Setup of an HTTP Server:

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

// Create an HTTP server
const server = http.createServer((req, res) => {
  res.statusCode = 200; // Set the response status code to
    200 (OK)
  res.setHeader('Content-Type', 'text/plain'); // Set the
    content type to plain text
  res.end('Hello, World!\n'); // End the response and send
    back a message
});

// Listen on port 3000
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

HTTP

Output: Server running at http://localhost:3000/

Key Concepts in the http Module:

- http.createServer()
  - This method creates an HTTP server object that listens to HTTP requests. It takes a callback function that gets executed when a request is received.
  - The callback function has two parameters:
    req (request object): Contains the details of the incoming HTTP request.
    res (response object): Allows you to send a response back to the client.

HTTP

Key Concepts in the http Module:(Contd...)

- Request Object (req): The req object contains all the information about the HTTP request that the client has made, such as:
  - req.url: The requested URL.
  - req.method: The HTTP method (GET, POST, etc.).
  - req.headers: An object containing the headers of the request.
  - req.on('data', callback): Listens for the incoming data (useful for POST requests).
- Response Object (res): The res object is used to send a response back to the client. You can:
  - Set the status code of the response using res.statusCode.
  - Set headers using res.setHeader(name, value).
  - End the response using res.end(data).

HTTP

Handling Routes: We can handle different routes by checking the URL of the incoming request and sending different responses.

Handling Get Requests:

```
const http = require('http');
const server = http.createServer((req, res) => {
  if (req.url === '/' && req.method === 'GET') {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Welcome to the homepage!\n');
  } else if (req.url === '/about' && req.method === 'GET') {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('This is the About page.\n');
  } else {
    res.statusCode = 404;
    res.setHeader('Content-Type', 'text/plain');
```

HTTP

Handling GET Requests:(Contd...)

```
    res.end('404 Not Found\n');
  }
});

server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

HTTP

Handling POST Requests: To handle POST requests, We can listen for the 'data' and 'end' events on the req object to capture the body of the request.

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

const server = http.createServer((req, res) => {
  if (req.method === 'POST' && req.url === '/submit') {
    let body = '';

    // Collect the data chunks
    req.on('data', chunk => {
      body += chunk.toString(); // Convert Buffer to string
    });

    // When all data has been received
    req.on('end', () => {
      console.log('Received body:', body);
      res.end('Data received successfully!');
    });
```

HTTP

Handling POST Requests:(Contd...)

```
  }
  else {
    res.statusCode = 404;
    res.end('404 Not Found');
  }
});
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

Request the following URL http://localhost:3000/submit using post man software in POST method and give the following as test data in the body:

```
Received body: {
    "name":"VIJAY"
}
```

Output: We see the test data as output at console

HTTP

Handling Query Parameters: We can parse query parameters from the URL using the built-in url module or a third-party module like querystring.

```
const http = require('http');
const url = require('url');

const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true); // Parse the
    URL with query parameters
  const query = parsedUrl.query; // Get the query parameters
     as an object
  res.statusCode = 200;
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(query)); // Send query parameters
    as JSON
});
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

HTTP

Handling Query Parameters:(Contd...) `Request the following URL http://localhost:3000/submit?name=vijay using post man software in POST method`

Output: We see the output in the postman as response:

```
{
    "name": "vijay",
}
```

Files

In Node.js, the fs (File System) module allows you to interact with the file system. It provides methods to read, write, delete, and manipulate files and directories.

Following are the Operations we do using fs module:

- Reading Files
- Writing to Files
- Appending to Files
- Deleting Files
- Renaming Files
- Checking if a File Exists
- Creating Directories
- Reading Directory Contents
- Watching Files for Changes
- Copying Files (Node.js 10.0.0 and later)

Files

The fs module has both synchronous and asynchronous methods. Asynchronous methods are preferred because they don't block the execution of your application while waiting for the file operation to complete.

Reading Files:

```
const fs = require('fs');
fs.readFile('sample.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading the file:', err);
  } else {
    console.log('File contents:', data);
  }
});
```

Note:create a file sample.txt in current directory and place some text in it.

Output: File contents: Welcome to SNIST

Files

Writing to Files:

```
const fs = require('fs');
const content = 'Welcome to WT Training ';

fs.writeFile('output.txt', content, (err) => {
  if (err) {
    console.error('Error writing to the file:', err);
  } else {
    console.log('File written successfully!');
  }
});
```

Note: If the output.txt file does not exists, It create a new file or Override the existing file contents with new content

Output: File written successfully!

Check the output.txt file we can see the content written in it.‘

Files

Appending to Files:

```
const fs = require('fs');
const additionalContent = 'This content will be appended to
    the file.';

fs.appendFile('output.txt', additionalContent, (err) => {
  if (err) {
    console.error('Error appending to the file:', err);
  } else {
    console.log('Content appended successfully!');
  }
});
```

Note: If the output.txt file does not exists, It create a new file or Override the existing file contents with new content

Output: Content appended successfully!

Check the output.txt file we can see the content updated in it.‘

Files

Deleting Files:

```
const fs = require('fs');
fs.unlink('output.txt', (err) => {
  if (err) {
    console.error('Error deleting the file:', err);
  } else {
    console.log('File deleted successfully!');
  }
});
```

Note: If the output.txt file does not exists, It throws an error

Output: File deleted successfully!

Files

Renaming Files:

```
const fs = require('fs');
fs.rename('oldname.txt', 'newname.txt', (err) => {
  if (err) {
    console.error('Error renaming the file:', err);
  } else {
    console.log('File renamed successfully!');
  }
});
```

Note: If the oldname.txt file does not exists, It throws an error

Output: File renamed successfully!

Files

Checking if a file exists:

```
const fs = require('fs');
fs.access('sample1.txt', fs.constants.F_OK, (err) => {
  if (err) {
    console.error('File does not exist.');
  } else {
    console.log('File exists.');
  }
});
```

Output: File exists

Files

Creating Directories:

```
const fs = require('fs');
fs.mkdir('newDir', (err) => {
  if (err) {
    console.error('Error creating directory:', err);
  } else {
    console.log('Directory created successfully!');
  }
});
```

Note: If the newDir directory exists, It throws an error Error Creating directory

Output: Directory created successfully

Environment Variables and dotenv

In Node.js, environment variables are often used to store configuration settings, secrets, and other sensitive information that you don't want to hard-code into your application.
We use .env file to manage your environment variables.
Prequisite: Install dotenv using the following command at the terminal
npm install dotenv
Steps:

- Create a new file .env in the application
- Write the necessary configuration in the above file Example show below:

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=password
```

Environment Variables and dotenv

- import the .env file and use the configuration settings as shown below:

```
      require('dotenv').config();
const dbHost = process.env.DB_HOST;
const dbUser = process.env.DB_USER;
const dbPassword = process.env.DB_PASSWORD;

console.log(`Connecting to database at ${dbHost} with
    user ${dbUser}`);
```

Questionnaire

Questions

Thank you