Batch: D-2	**Roll No.: 16010123325**
Experiment / assignment / tutorial No._4___	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Demonstrate the Use of node.js and Express.js

AIM: To implement the node js Concepts based on the following topics.

Problem Definition:

Problem statement:

Consider the basic concepts, which are useful in the creation of an application.

Considering the following points, demonstrate the functionality of each with a simple script

1) Basic Routing:

1. Build First server application using http module
2. Basic routing: Demonstrate it using simple HTML/Json file
3. Demonstrate the callback in node.js

2) **File operation**

- Check Permissions of a File or Directory.
- Checking if a file or a directory exists.
- Determining the line count of a text file.
- Reading a file line by line.
- See the file content through browser.

3) Building your custom modules

-To demonstrate this use some mathematics function to create custom module.

4) Blocking and Non Blocking

Resources used:
https://www.w3schools.com/nodejs/
https://nodejs.org/api/index.html

Expected OUTCOME of Experiment:

CO 2:. Illustrate the concepts of various front-end, back-end web application development technologies & frameworks using different web development tools.

Books/ Journals/ Websites referred:

1. Shelly Powers Learning Node O' Reilly 2 nd Edition, 2016.

Pre Lab/ Prior Concepts:

Write details about the following content

- Node js.

  Node.js is a JavaScript runtime built on Chrome's V8 engine that allows you to run JavaScript outside the browser mainly for building server-side applications.
  It's event-driven, non-blocking, and great for real-time apps like chats or APIs.

- Basic Routing

  Routing means defining how the server responds to client requests at specific URLs (paths).
  Example:

```
const http = require('http');
const server = http.createServer((req, res) => {
  if (req.url === "/") res.end("Home Page");
  else if (req.url === "/about") res.end("About Page");
  else res.end("404 Not Found");
});
server.listen(3000);
```

  Each route gives a different response based on the path.

- Custom module

A custom module is a file you create yourself and export functions or variables from, to use elsewhere.
Example:

math.js

```
exports.add = (a, b) => a + b;
```

app.js

```
const math = require('./math');
console.log(math.add(5, 3));
```

Helps in organizing code and reusing logic.

- Asynchronous Programming

In Node.js, many operations (like reading files, making API calls) are asynchronous they don't block other code while waiting to finish.
Example:

```
const fs = require('fs');
fs.readFile('file.txt', 'utf8', (err, data) => {
  console.log(data);
});
console.log("Reading file...");
```

"Reading file…" prints first, showing non-blocking behaviour.

Basic Routing

Methodology:

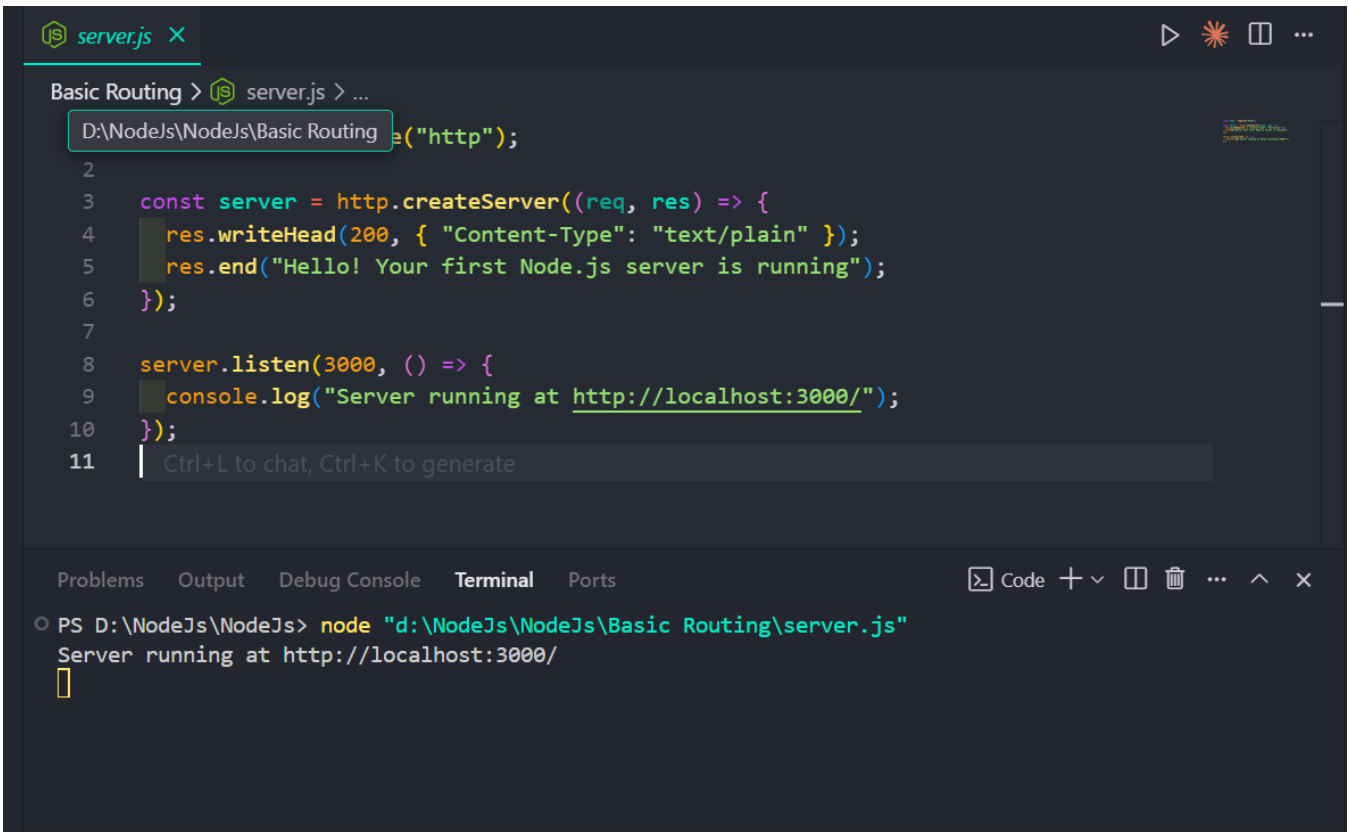
Server Setup- This program demonstrates how to **create a simple web server** using the **Node.js http module**.The server listens on a specific port (3000) and sends a plain text response "Hello! Your first Node.js server is running" to any client request.

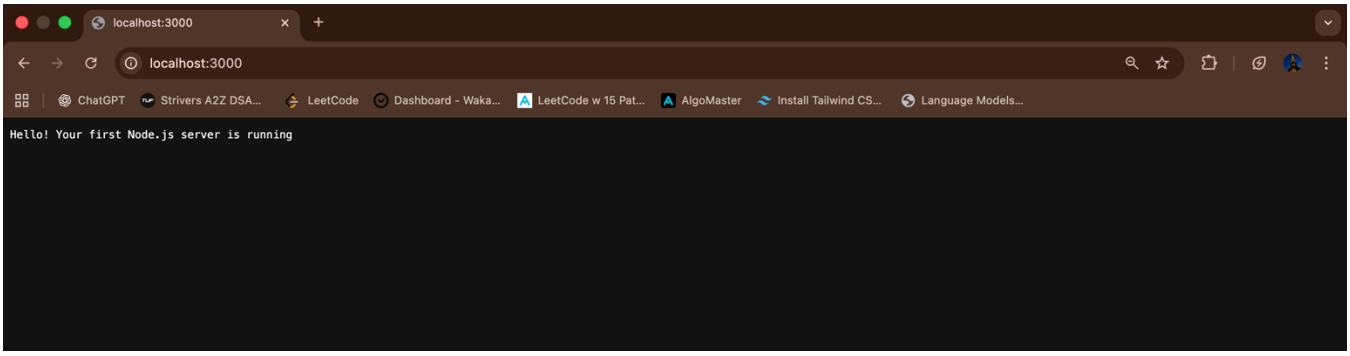
Callback- The program demonstrates asynchronous programming in Node.js using a callback function**.** A callback allows one function to be executed after another function finishes, without blocking the main thread.
Here, greet() takes a name and a callback function, prints a greeting, and then calls the callback to print a goodbye message.

Routing- This program demonstrates basic routing in Node.js using the HTTP module**.** Routing helps the server respond differently based on the requested URL path. Each route (/, /about, /data) returns a different type of response such as HTML or JSON, while invalid routes return a 404 error page.

Implementation Details:

Server-


server.js
Basic Routing > server.js > ...
D:\NodeJs\NodeJs\Basic Routing
("http");
const server = http.createServer((req, res) => {
res.writeHead(200, { "Content-Type": "text/plain" });
res.end("Hello! Your first Node.js server is running");
});
server.listen(3000, () => {
console.log("Server running at http://localhost:3000/");
});
Ctrl+L to chat, Ctrl+K to generate
Problems
Output
Debug Console
Terminal
Ports
Code
PS D:\NodeJs\NodeJs> node "d:\NodeJs\NodeJs\Basic Routing\server.js"
Server running at http://localhost:3000/


localhost:3000
localhost:3000
ChatGPT
Strivers A2Z DSA...
LeetCode
Dashboard - Waka...
LeetCode w 15 Pat...
AlgoMaster
Install Tailwind CS...
Language Models...
Hello! Your first Node.js server is running

Callback-

```
1  function greet(name, callback) {
2    console.log("Hello, " + name);
3    callback();
4  }
5
6  function sayGoodbye() {
7    console.log("Goodbye!");
8  }
9
10 greet("Shreyans", sayGoodbye);
11   Ctrl+L to chat, Ctrl+K to generate
```

```
Problems  Output  Debug Console  Terminal  Ports        Code
PS D:\NodeJs\NodeJs> node "d:\NodeJs\NodeJs\Basic Routing\callback.js"
Hello, Shreyans
Goodbye!
PS D:\NodeJs\NodeJs>
```

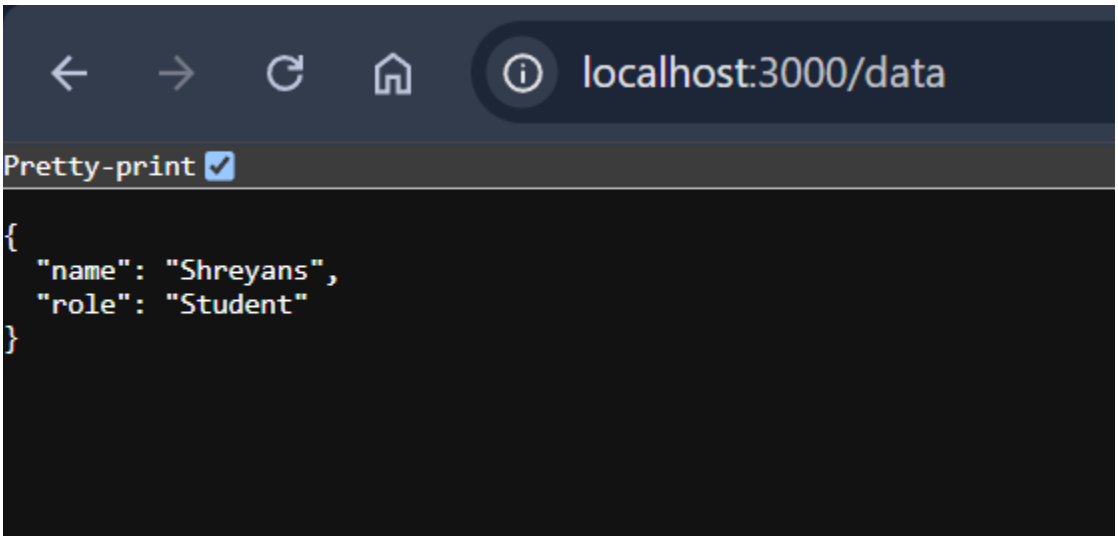
Routing-

Basic Routing > routing.js > ...

```
1   const http = require("http");
2   const fs = require("fs");
3
4   const server = http.createServer((req, res) => {
5     if (req.url === "/") {
6       res.writeHead(200, { "Content-Type": "text/html" });
7       res.end("<h1>Welcome to Home Page</h1>");
8     } else if (req.url === "/about") {
9       res.writeHead(200, { "Content-Type": "text/html" });
10      res.end("<h1>About Us Page</h1>");
11    } else if (req.url === "/data") {
12      res.writeHead(200, { "Content-Type": "application/json" });
13      res.end(JSON.stringify({ name: "Shreyans", role: "Student" }));
14    } else {
15      res.writeHead(404, { "Content-Type": "text/html" });
16      res.end("<h1>404 Page Not Found</h1>");
17    }
18  });
19
20  server.listen(3000, () => console.log("Routing server running at http://localhost:
21    Ctrl+L to chat, Ctrl+K to generate
```

Problems Output Debug Console **Terminal** Ports Code

```
PS D:\NodeJs\NodeJs> node "d:\NodeJs\NodeJs\Basic Routing\routing.js"
Routing server running at http://localhost:3000/
```



Steps for execution:
Server-

1. Import the http module

- The built-in http module allows Node.js to create web servers and handle HTTP requests/responses.

2. Create the server

- Use http.createServer() to create a server instance.
- The callback function takes two parameters:
  - req → represents the request object from the client.
  - res → represents the response object used to send data back to the client.

3. Set the response headers and message

- res.writeHead(200, { "Content-Type": "text/plain" }) sets the status code to 200 (OK) and defines that the response type is plain text.
- res.end() sends the response message and ends the request.

4. Start the server and listen on port 3000

Callback-

1. Define the greet() function

- It accepts two parameters: name and callback.
- It prints "Hello, " + name.
- After printing, it calls the callback() function.

2. Define the sayGoodbye() function

- It prints "Goodbye!" to the console.

3. Call the greet() function

- Pass "Shreyans" as the name and sayGoodbye as the callback function.
- The sequence of execution:
  1. Print “Hello, Shreyans”
  2. Then call sayGoodbye() → prints “Goodbye!”

Routing-

1. Import required modules

- http module to create a web server.
- fs module is imported (optional here, can be used for file operations later).

2. Create the server

- Use http.createServer() to handle incoming requests and send responses.
- The callback receives two parameters: req (request) and res (response).

3. Define different routes

- Check the URL using req.url and respond accordingly:
  - / → Home Page (HTML response)
  - /about → About Us Page (HTML response)
  - /data → JSON response
  - Other routes → 404 Page Not Found

4. Start the server

- Listen on port **3000** and display a message when the server is running.

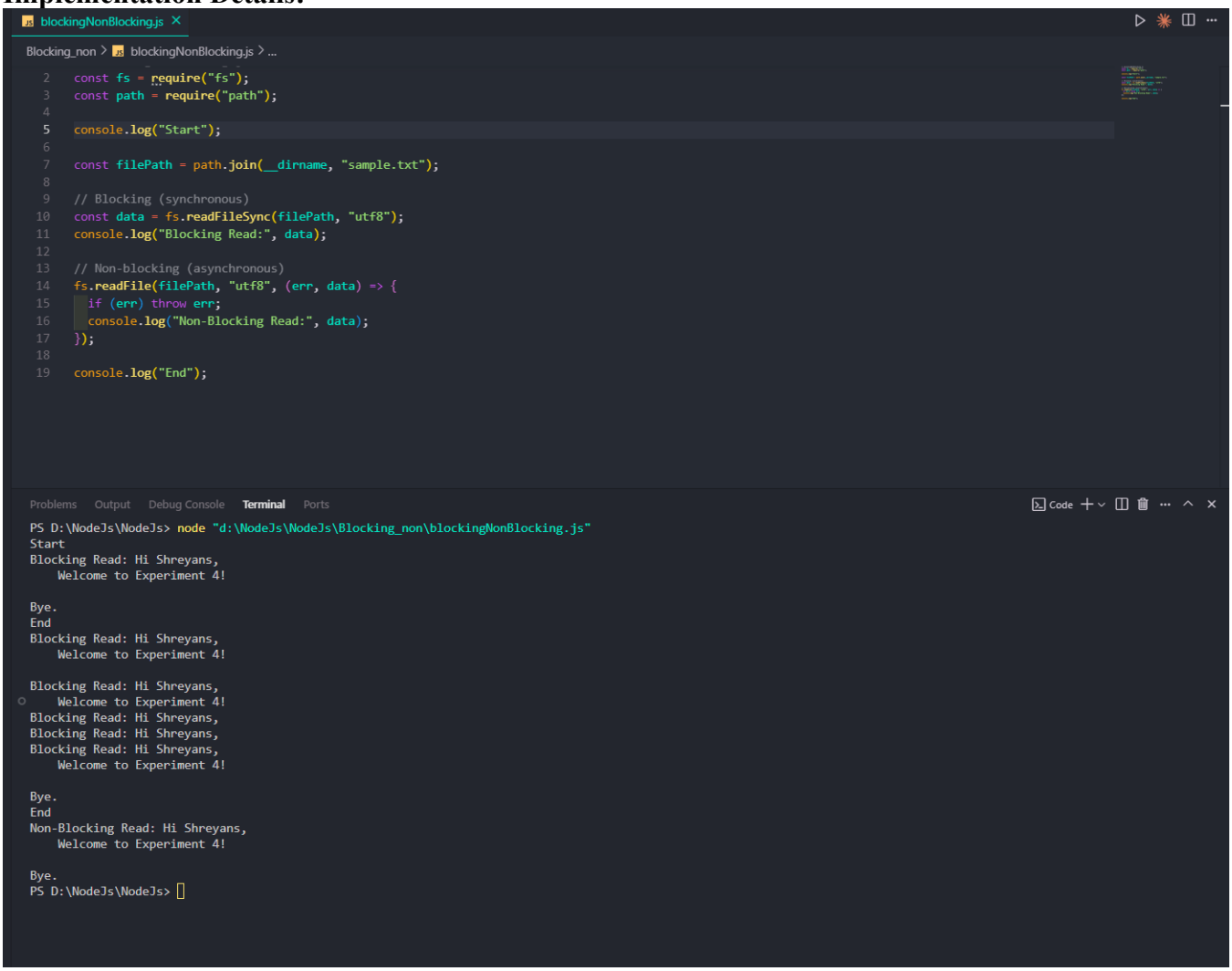
Blocking & Non-Blocking

Methodology:

This program demonstrates the difference between **blocking (synchronous)** and **non-blocking (asynchronous)** file operations in **Node.js** using the built-in fs (File System) module.

- **Blocking (Sync):** Code executes line-by-line next line waits until the current task finishes.
- **Non-blocking (Async):** Code doesn't wait other operations continue while the file is being read.

Implementation Details:



Steps for execution:

1. Import the fs module

- The fs (File System) module provides methods to work with files — both synchronously and asynchronously.

2. Start program execution

- Print "Start" to indicate the beginning of execution.

3. Blocking (Synchronous) File Read

- Use fs.readFileSync() to read the file.
- The program waits until the entire file content is read before moving to the next line.

Execution is **paused** until reading completes.

4. Non-blocking (Asynchronous) File Read

- Use fs.readFile() with a callback.
- The program doesn't wait — it continues executing the next lines.
- When reading finishes, the callback is executed.

5. End program execution

Custom Module

Methodology:

This program demonstrates the creation and use of a custom module in Node.js**.**
A custom module allows separating reusable logic (like mathematical operations) into a separate file and then importing it into another file using require().
This promotes modularity**,** code reusability, and clean structure in Node.js applications.

Implementation Details:

Custom Module > useMath.js > ...

```
1  const math = require("./mathOps");
2
3  console.log("Add:", math.add(10, 5));
4  console.log("Subtract:", math.subtract(10, 5));
5  console.log("Multiply:", math.multiply(10, 5));
6  console.log("Divide:", math.divide(10, 5));
```

Problems Output Debug Console Terminal Ports

```
PS D:\NodeJs\NodeJs> node "d:\NodeJs\NodeJs\Custom Module\useMath.js"
Add: 15
Subtract: 5
Multiply: 50
Divide: 2
PS D:\NodeJs\NodeJs>
```

Steps for execution:

1. Create the custom module — mathOps.js

- Define and export multiple functions using exports.
  Each function performs a basic arithmetic operation.

2. Create the main file — app.js

Import the custom module using require() and call its functions.

- require("./mathOps") loads the exported functions from the custom module. Each function is then executed with example inputs.

File Operations

Methodology:

This program demonstrates various **file system,** using the built-in fs, readline, and http modules. It performs the following tasks:

- Checks read/write permissions of a file.
- Checks if the given path is a file or directory.
- Counts the number of lines in a text file.
- Reads the file **line by line** using streams.
- Serves the file content in the **web browser** through an HTTP server.

Implementation Details:

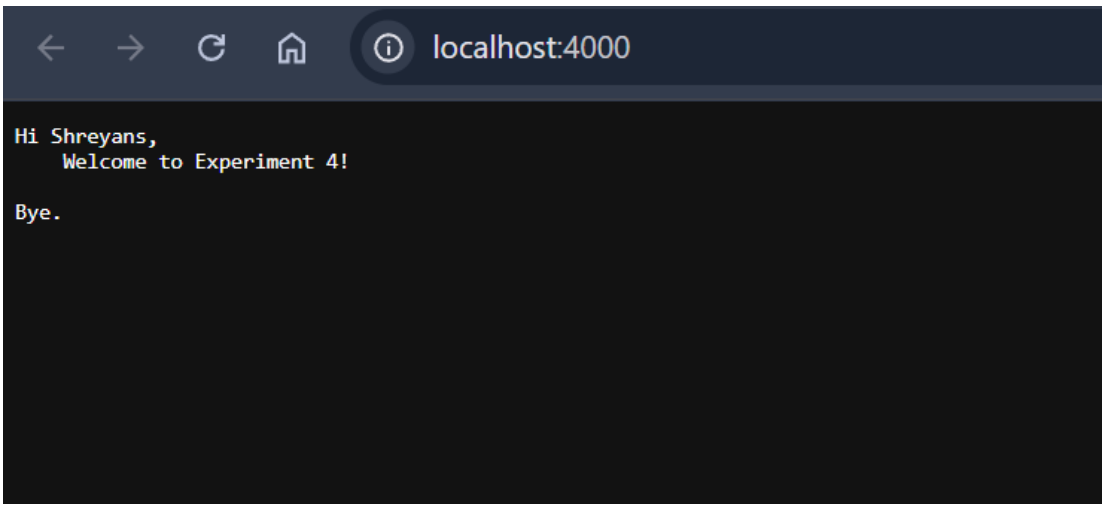
blockingNonBlocking.js fileOps.js

File Operations > fileOps.js > ...

```
const fs = require("fs");
const readline = require("readline");
const http = require("http");
const path = require("path");

const filePath = path.join(__dirname, "sample.txt");

// 1) Check Permissions
fs.access(filePath, fs.constants.R_OK | fs.constants.W_OK, (err) => {
  console.log(err ? "No Read/Write access" : "Read/Write access granted");
});

// 2) Check if File or Directory Exists
fs.stat(filePath, (err, stats) => {
  if (err) return console.log("File/Directory does not exist");
  console.log(stats.isFile() ? "It is a file" : "It is a directory");
});

// 3) Line Count
fs.readFile(filePath, "utf8", (err, data) => {
  if (err) throw err;
  console.log("Line count:", data.split("\n").length);
});

// 4) Read File Line by Line
const rl = readline.createInterface({
  input: fs.createReadStream(filePath),
  output: process.stdout,
  terminal: false,
});
rl.on("line", (line) => {
  console.log("Line:", line);
});

// 5) Serve File Content in Browser
http.createServer((req, res) => {
  fs.readFile(filePath, "utf8", (err, data) => {
    if (err) {
      res.writeHead(500);
      res.end("Error reading file");
    } else {
      res.writeHead(200, { "Content-Type": "text/plain" });
      res.end(data);
    }
  });
}).listen(4000, () => console.log("Check file content at http://localhost:4000/"));
```

Problems Output Debug Console Terminal Ports

```
PS D:\NodeJs\NodeJs> node "d:\NodeJs\NodeJs\File Operations\fileOps.js"
Check file content at http://localhost:4000/
Read/Write access granted
It is a file
Line: Hi Shreyans,
Line:     Welcome to Experiment 4!
Line:
Line: Bye.
Line count: 4
```



Steps for execution:

1. Import required modules

- fs → For file operations (reading, permissions, stats).
- readline → For reading files line by line.
- http → To create a basic web server.

2. Check file permissions

- fs.access() checks if sample.txt has read and write permissions.
- If access is allowed → prints "Read/Write access granted".
- Otherwise → "No Read/Write access".

3. Check if file or directory exists

- fs.stat() retrieves information about the path.
- stats.isFile() checks whether it's a file or directory.
- Displays the result accordingly.

4. Count total lines in the file

- fs.readFile() reads the entire file content asynchronously.
- data.split("\n").length counts total number of lines.

5. Read file line by line

- Uses readline.createInterface() with a readable stream.
- Emits a "line" event for every line in the file.
- Prints each line to the console.

6. Serve file content via HTTP

- Creates a server using http.createServer().
- Reads sample.txt and sends its contents as response.

Conclusion:

This experiment successfully demonstrates the use of Node.js for creating servers, handling routing, using custom modules, managing asynchronous operations, and performing file system tasks. It shows how Node.js enables efficient, non-blocking, and modular server-side development using its built-in modules and event-driven architecture.