

## **Node JS Assignments**

### **1. Basic Node.js**

- **Task:** Create a Node.js script that reads a text file and prints its contents to the console.
    - Use the `fs` module to read files.
    - Handle errors for non-existent files.
- 

### **2. Read and Display File Contents**

- **Task:** Create a script that:
    - Reads the contents of a file (`example.txt`) and prints it to the console.
    - Handles errors if the file doesn't exist.
  - **Key Methods:** `fs.readFile` or `fs.readFileSync`.
- 

### **3. Create and Write to a File**

- **Task:** Create a program that:
    - Prompts the user to input a string (use `readline` or `process.stdin`).
    - Writes the string to a new file (`output.txt`).
    - Confirms that the file has been created successfully.
  - **Key Methods:** `fs.writeFile` or `fs.writeFileSync`.
- 

### **4. Append Data to a File**

- **Task:** Create a script that:
    - Appends a timestamp to a file (`log.txt`) each time the script is run.
    - If the file does not exist, create it first.
  - **Key Methods:** `fs.appendFile`.
- 

### **5. File and Directory Operations**

- **Task:** Build a script to:
  - Check if a directory (`data`) exists. If not, create it.
  - Create a file (`data/info.txt`) inside the directory and write "Hello, Node.js!" into it.
  - Read and display the content of the file.

- **Key Methods:** `fs.existsSync`, `fs.mkdir`, `fs.writeFile`, `fs.readFile`.
- 

## 6. Delete a File

- **Task:** Write a script that:
    - Checks if a file (`temp.txt`) exists.
    - If it exists, delete the file and log a success message.
    - If it doesn't exist, log an error message.
  - **Key Methods:** `fs.unlink`.
- 

## 7. System Information Script

- **Task:** Create a script that displays the following system information:
    - Operating system name and version.
    - System architecture (e.g., `x64`).
    - Total memory and free memory in MB.
    - System uptime in hours.
  - **Key Methods:** `os.type`, `os.release`, `os.arch`, `os.totalmem`, `os.freemem`, `os.uptime`.
- 

## 8. User Info and Home Directory

- **Task:** Build a script that:
    - Retrieves the current user's information (username, home directory, and shell/command interpreter).
    - Logs the information in a human-readable format.
  - **Key Methods:** `os.userInfo`.
- 

## 9. Create System Log

- **Task:** Write a program that:
    - Creates a `system_log.txt` file.
    - Writes the system's hostname, platform, and network interfaces into the file.
    - Adds a timestamp of when the log was created.
  - **Key Methods:** `os.hostname`, `os.platform`, `os.networkInterfaces`.
- 

## 10. File Path Details

- **Task:** Create a script that:

- Accepts a file path as input (use `process.argv`).
  - Extracts and displays the following details:
    - Directory name.
    - Base file name.
    - File extension.
  - Handles invalid paths gracefully.
  - **Key Methods:** `path.dirname`, `path.basename`, `path.extname`.
- 

## 11. Join and Normalize Paths

- **Task:** Build a program that:
    - Joins multiple segments to create a valid file path (e.g., `folder`, `subfolder`, `file.txt`).
    - Normalizes the generated path to remove redundant segments like `..` or `..`.
    - Prints the resulting valid path.
  - **Key Methods:** `path.join`, `path.normalize`.
- 

## 12. Absolute vs Relative Path

- **Task:** Create a script that:
    - Checks if a given path (provided as a command-line argument) is absolute or relative.
    - Converts a relative path into an absolute path based on the current working directory.
    - Displays both the input path and the resulting absolute path.
  - **Key Methods:** `path.isAbsolute`, `path.resolve`.
- 

## 13. Promisify a Callback Function

- **Task:** Create a script that:
    - Uses the `util.promisify` method to convert the `fs.readFile` function into a promise-based function.
    - Reads the content of a file (`example.txt`) asynchronously using the promisified version.
    - Logs the content to the console or logs an error message if the file cannot be read.
  - **Key Methods:** `util.promisify`, `fs.readFile`.
- 

## 14. Inspect an Object

- **Task:** Write a script that:

- Creates a complex JavaScript object with nested properties and arrays.
  - Uses the `util.inspect` method to display the object in a human-readable format with custom options (e.g., depth, colors).
  - Prints the formatted output to the console.
  - **Key Methods:** `util.inspect`.
- 

## 15. Custom Deprecation Warning

- **Task:** Build a script that:
    - Creates a deprecated function using `util.deprecate`.
    - Logs a warning message when the deprecated function is called.
    - Includes a replacement suggestion in the warning message.
  - **Key Methods:** `util.deprecate`
- 

## 16. Create and Manipulate Buffers

- **Task:** Write a script that:
    - Creates a buffer from a string (e.g., "Hello, Buffer!").
    - Converts the buffer to JSON format and logs it to the console.
    - Converts the buffer back to a string and prints the result.
  - **Key Methods:** `Buffer.from`, `Buffer.toJSON`, `buffer.toString`.
- 

## 17. Buffer Concatenation

- **Task:** Create a script that:
    - Creates two buffers with different strings (e.g., "Node.js " and "Buffer").
    - Concatenates these buffers into a single buffer.
    - Logs the resulting buffer and its string representation.
  - **Key Methods:** `Buffer.concat`.
- 

## 18. Buffer Encoding and Decoding

- **Task:** Develop a script that:
    - Creates a buffer from a string using `utf8` encoding.
    - Converts the buffer to `base64` encoding and logs the result.
    - Decodes the `base64` encoded string back to its original form and prints it.
  - **Key Methods:** `Buffer.from`, `buffer.toString` (with encoding options).
- 

## 19. HTTP Server

- **Task:** Build a basic HTTP server that responds with "Hello, World!" on the root URL.
    - Use the `http` module.
    - Respond with a 404 status for other routes.
- 

## 20. Url Basics

- **Task:** Create a simple Node server with three routes:
    - Home `(/)` returning "Welcome to my site".
    - About `(/about)` returning some information about you.
    - A 404 handler for all other routes.
- 

## 21. REST API

- **Task:** Create a RESTful API to manage a list of tasks (CRUD operations).
    - Use Express.js.
    - Implement endpoints:
      - `GET /tasks` to list all tasks.
      - `POST /tasks` to create a new task.
      - `PUT /tasks/:id` to update a task.
      - `DELETE /tasks/:id` to delete a task.
    - Store tasks in an in-memory array.
- 

## 22. File System Operations

- **Task:** Create a program that:
    - Writes user input from the console into a file.
    - Reads and displays the content of the file.
- 

## 23. Using Environment Variables

- **Task:** Create a server that uses environment variables for configuration (e.g., port and database URL).
    - Use the `dotenv` package to manage `.env` files.
- 

## 24. Node.js with Mysql

- **Task:** Create a simple application to store user information in a MySQL database.

- Implement CRUD operations for users.
- 

## 25. Task Scheduler

- **Task:** Schedule tasks to execute at specific intervals or times.
    - Use the `node-schedule` or `cron` package.
    - Create a script to send a console log every minute.
- 

## 26. Upload Files

- **Task:** Create an API to upload files to the server.
    - Use the `multer` package.
    - Store the uploaded files in a designated folder.
- 

## 27. Testing with Mocha & Chai

- **Task:** Write unit tests for a simple function (e.g., add two numbers).
    - Use `mocha` and `chai` to write and execute the tests.
- 

## 28. Build a CLI Tool

- **Task:** Create a command-line tool that accepts arguments and performs a task, like converting JSON to CSV or vice versa.
  - Use the `commander` or `yargs` package.