

NODE JS

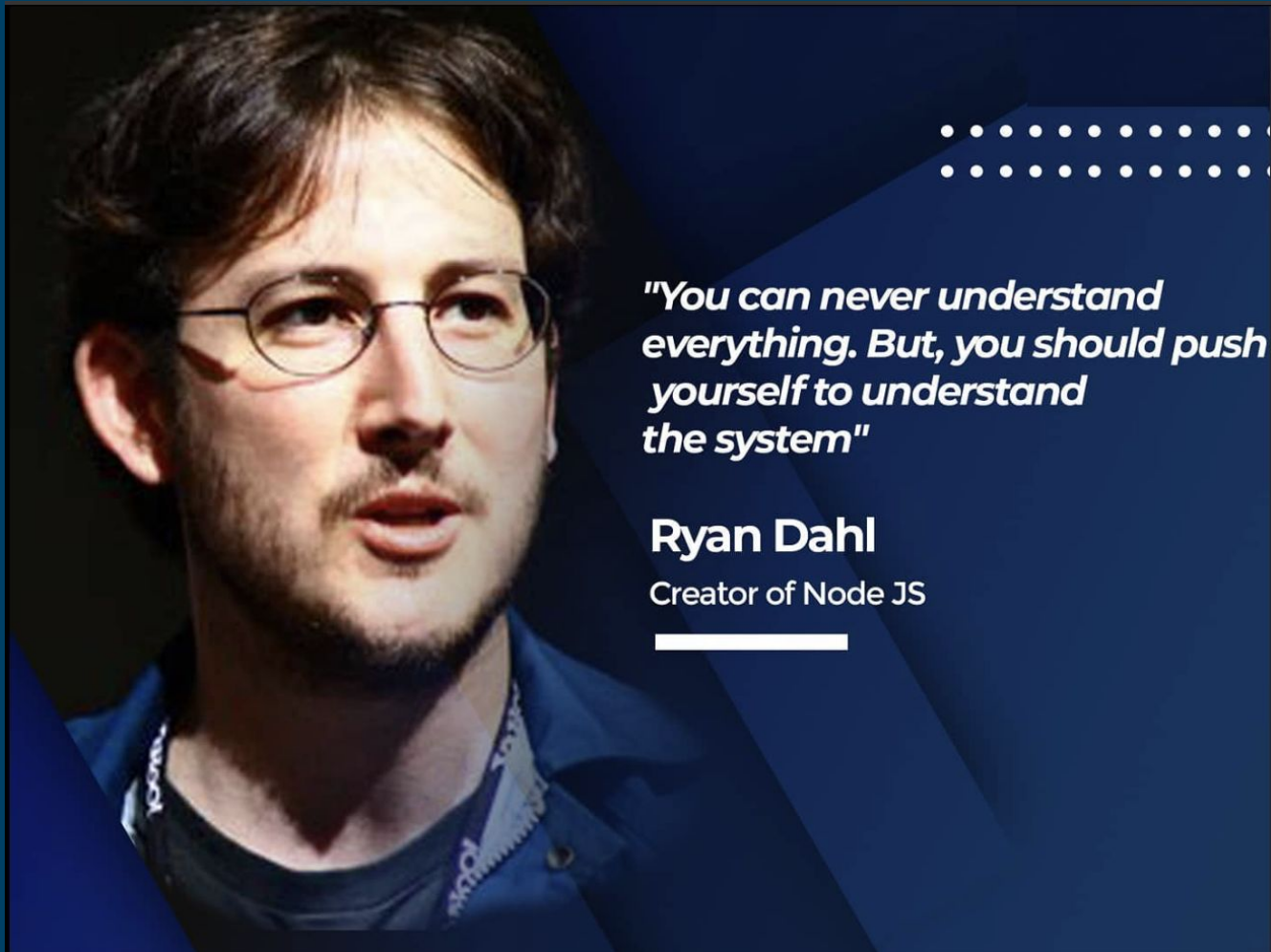
By Gajanan Kharat - CODEMIND Technology

Contact us [8600 2550 66](tel:8600255066)



Node

- MERN and MEAN
 - What is node and Why to learn it ?
 - Global Objects
 - Module Types
 - Core Modules
-



"You can never understand everything. But, you should push yourself to understand the system"

Ryan Dahl

Creator of Node JS

Node.js History

Node.js was developed by **Ryan Dahl**, a software engineer

Background:

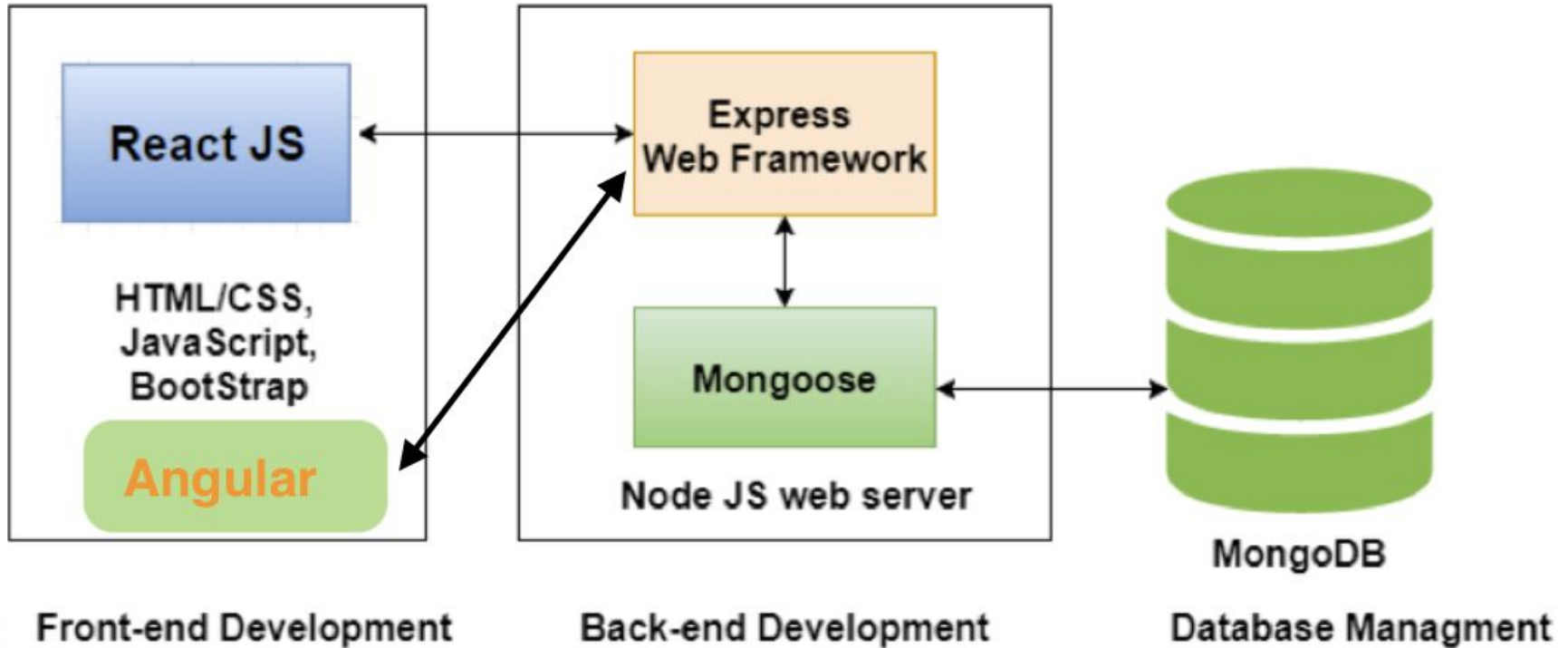
- Ryan Dahl was born in 1981 in the United States.

Education:

- He studied computer science at Utah Valley University.

MEAN → Mongo, Express, Angular and Node

MERN → Mongo, Express, React and Node



What is node.js ?

- Node.js is a runtime environment and it is not a framework and not a programming language
- Node.js is an open source and cross platform javascript runtime environment
- Node.js runs the V8 JavaScript engine
- A Node.js app runs in a single process, without creating a new thread for every request
- Node.js uses event driven, non blocking I/O model to handle concurrent request with single thread

Node.js vs Browser

- Both browser and Node uses JavaScript as their programming language
- With browser we have access to DOM, Web API, where as with node.js we have modules provide to access to file system, OS
- Node.js supports both the commandJS `require()` and ES module systems `import()`.

Why to learn Node

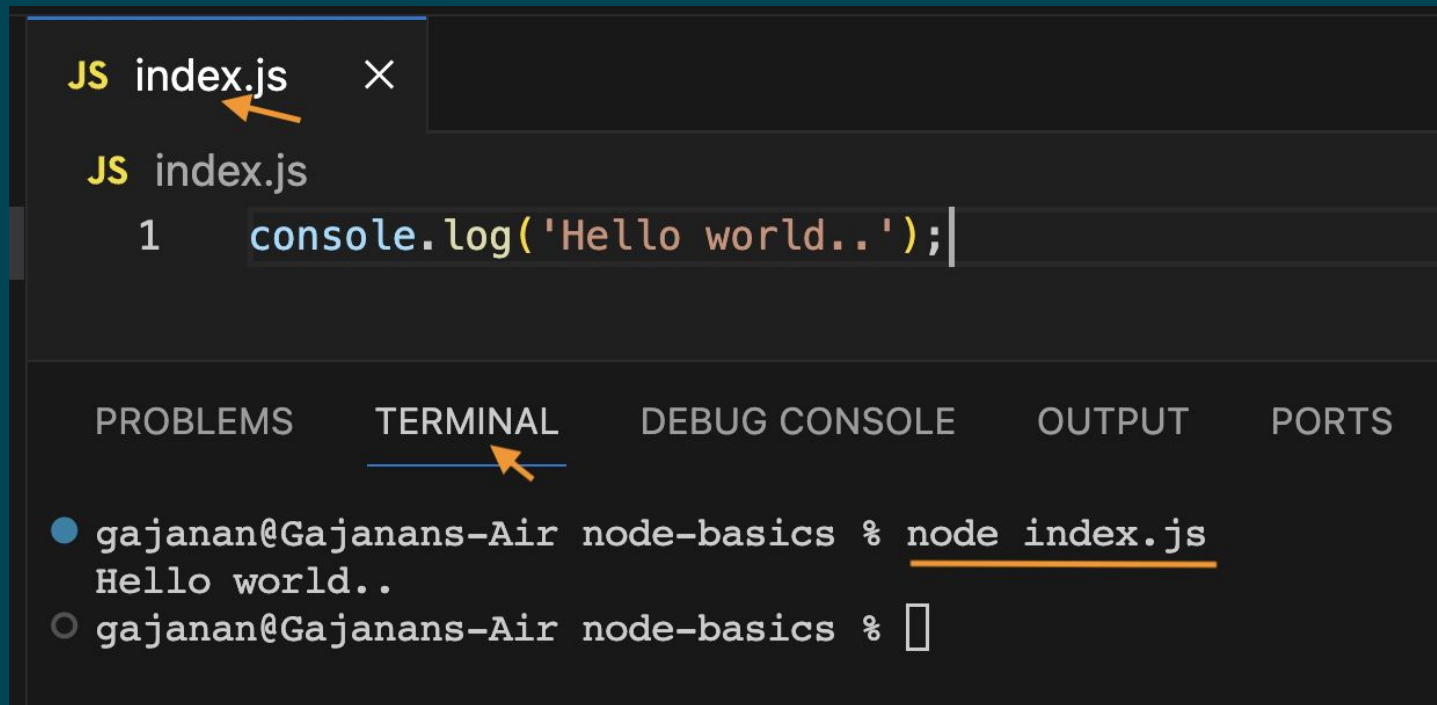
- Read the article → <https://dev.to/anthonydelgado/javascript-is-eating-the-world>
- Only prerequisite is JS and many companies like Netflix, Uber are using nodejs in their production environment

To run JavaScript file using node

In current working directory use the command on terminal or command prompt or VS code terminal

→ node index.js.

Ex. node index.js



The screenshot shows the Visual Studio Code interface. At the top, a file explorer shows a file named `index.js` with a yellow arrow pointing to it. Below the file explorer, the code editor displays the content of `index.js`, which is a single line: `console.log('Hello world..');`. At the bottom, the terminal panel is active, showing the command `node index.js` being executed. The output of the command is `Hello world..`. The terminal panel has tabs for `PROBLEMS`, `TERMINAL` (which is selected), `DEBUG CONSOLE`, `OUTPUT`, and `PORTS`. A yellow arrow points to the `TERMINAL` tab.

```
JS index.js ×
JS index.js
1 console.log('Hello world..');
```

PROBLEMS TERMINAL DEBUG CONSOLE OUTPUT PORTS

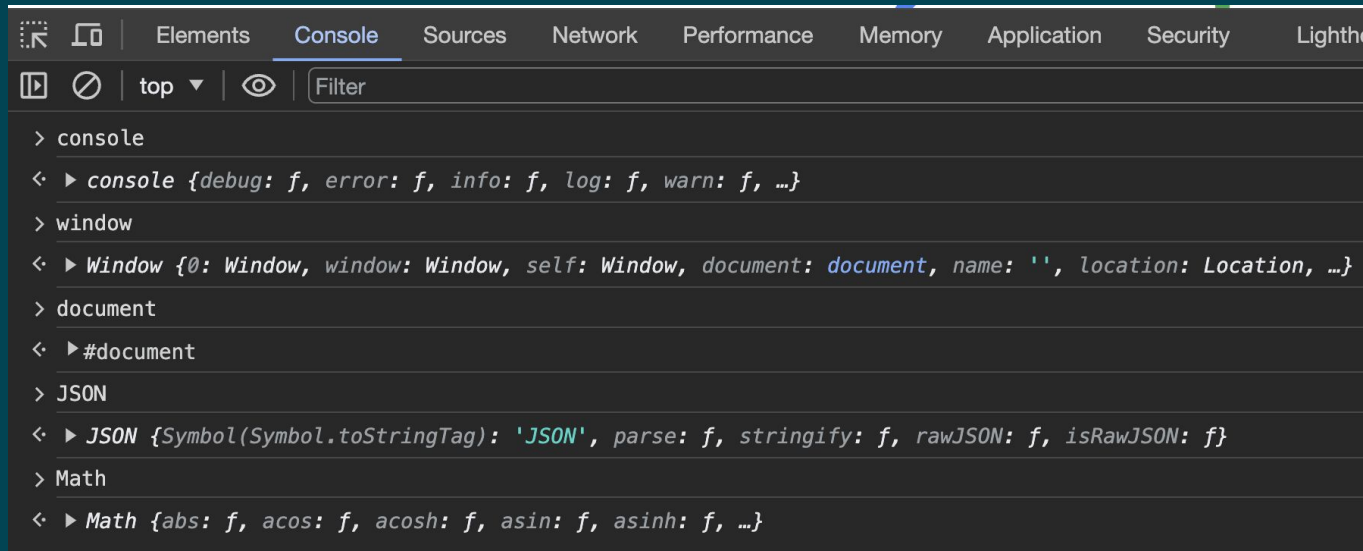
```
● gajanan@Gajanan's-Air node-basics % node index.js
Hello world..
○ gajanan@Gajanan's-Air node-basics %
```

Global Object

In both JavaScript (browser) and Node.js, a global object is an object that is always available and can be accessed anywhere in your code without needing to import or create it.

Global Objects in JS: No need to import, these are available in web browser environment.

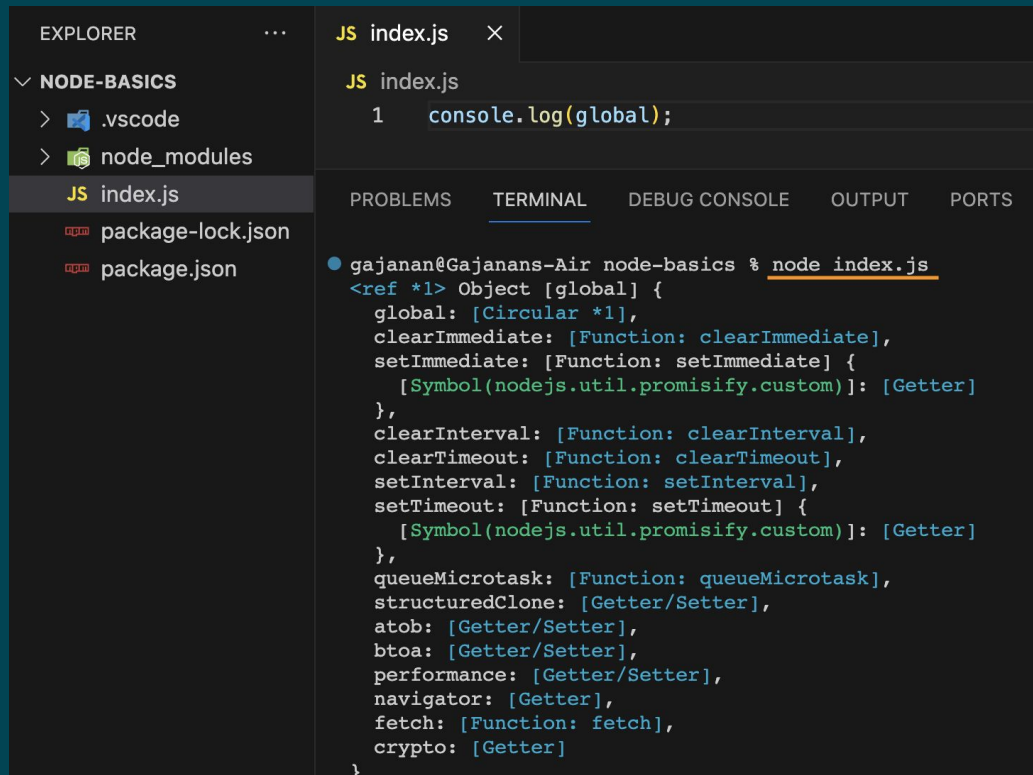
- **window**: The window object represents the global scope
- **console**: is part of the browser's environment, and it provides methods for logging information to the browser console
- **document**: The document object represents the HTML document loaded in the browser.
- **JSON**: that provides methods for working with JSON → `JSON.stringify()`, `JSON.parse()`



Node js 'global' object: global object is similar to window object in JS

In Node.js, the global object is simply referred to as **global**. It provides a set of global properties and functions that are available throughout our entire Node.js application.

- variables declared without the var, let, or const keyword become properties of the global object.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure with 'NODE-BASICS' containing '.vscode', 'node_modules', 'index.js', 'package-lock.json', and 'package.json'. The 'index.js' file is selected. The main editor area shows the content of 'index.js', which contains a single line: `1 console.log(global);`. On the right, the 'TERMINAL' tab is active, showing the command `node index.js` and its output. The output is a detailed representation of the global object, including properties like `global`, `clearImmediate`, `setImmediate`, `clearInterval`, `clearTimeout`, `setInterval`, `setTimeout`, `queueMicrotask`, `structuredClone`, `atob`, `btoa`, `performance`, `navigator`, `fetch`, and `crypto`.

```
EXPLORER  ...
  ▼ NODE-BASICS
    > .vscode
    > node_modules
    JS index.js
    package-lock.json
    package.json

JS index.js
1  console.log(global);

PROBLEMS  TERMINAL  DEBUG CONSOLE  OUTPUT  PORTS

● gajanan@Gajanan-Air node-basics % node index.js
<ref *1> Object [global] {
  global: [Circular *1],
  clearImmediate: [Function: clearImmediate],
  setImmediate: [Function: setImmediate] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  clearInterval: [Function: clearInterval],
  clearTimeout: [Function: clearTimeout],
  setInterval: [Function: setInterval],
  setTimeout: [Function: setTimeout] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  queueMicrotask: [Function: queueMicrotask],
  structuredClone: [Getter/Setter],
  atob: [Getter/Setter],
  btoa: [Getter/Setter],
  performance: [Getter/Setter],
  navigator: [Getter],
  fetch: [Function: fetch],
  crypto: [Getter]
}
```

Global objects in node js

console: The console object, used for logging messages, is available globally.

process: Provides info. about the current Node.js process and allows interaction with the process.

__filename and __dirname: __filename and __dirname represent the current file and directory, respectively

require Function: The require function for importing modules is available globally.

Module: The module object is specific to each module (JavaScript file)

JS globalObjects.js ×

JS globalObjects.js

```
1 console.log(global);
2 console.log(console);
3 console.log(process);
4 console.log(__filename);
5 console.log(__dirname);
6 console.log(module);
```

Module in Node: A module is a file containing JavaScript code that defines variables, functions, or objects that can be used in other parts of our Node.js application

Module Types:

1. **Local Modules:** These are the modules that we create as seen in ES6 features.
2. **Core Modules or Built in Modules:** Node.js provides a set of built-in core modules that we can use without needing to install them separately. These modules offer essential functionality, such as file system operations (fs), HTTP server creation (http), and more.

We can use these modules in our code using 'require' function

```
const fs = require('fs')
```

3. Third Party Modules: We can also use third-party modules created by other developers. These modules are often available on the Node Package Manager (NPM). We can install them using `npm install` and then use `require` to include them in our code.

Command to install third party module

```
npm install thirdPartyModuleName
```

To use the module in our code we can use 'require' keyword

```
const externalModule = require('thirdPartyModuleName');
```

Core or Built in Modules

Official website → <https://nodejs.org/> Click the 'DOCS' in navigation bar and select the version. It will give us the list of available core modules.

OS: os (operating system) module is a built-in module that provides a way to interact with the operating system's information. It exposes various methods and properties to retrieve information about the underlying operating system.

JS os.js



JS os.js > ...

```
1  const os = require('os');
2  console.log(os.arch());
3  console.log(os.cpus());
4  console.log(os.freemem());
5  console.log(os.totalmem());
6  console.log(os.hostname());
7  console.log(os.platform());
8  console.log(os.release());
9  console.log(os.type());
10 console.log(os.uptime());
11 console.log(os.userInfo());
```


Assignment:

JS os.js



JS os.js > ...

```
1  const os = require('os');
2  console.log(os.arch());
3  console.log(os.cpus());
4  console.log(os.freemem());
5  console.log(os.totalmem());
6  console.log(os.hostname());
7  console.log(os.platform());
8  console.log(os.release());
9  console.log(os.type());
10 console.log(os.uptime());
11 console.log(os.userInfo());
```

JS globalObjects.js



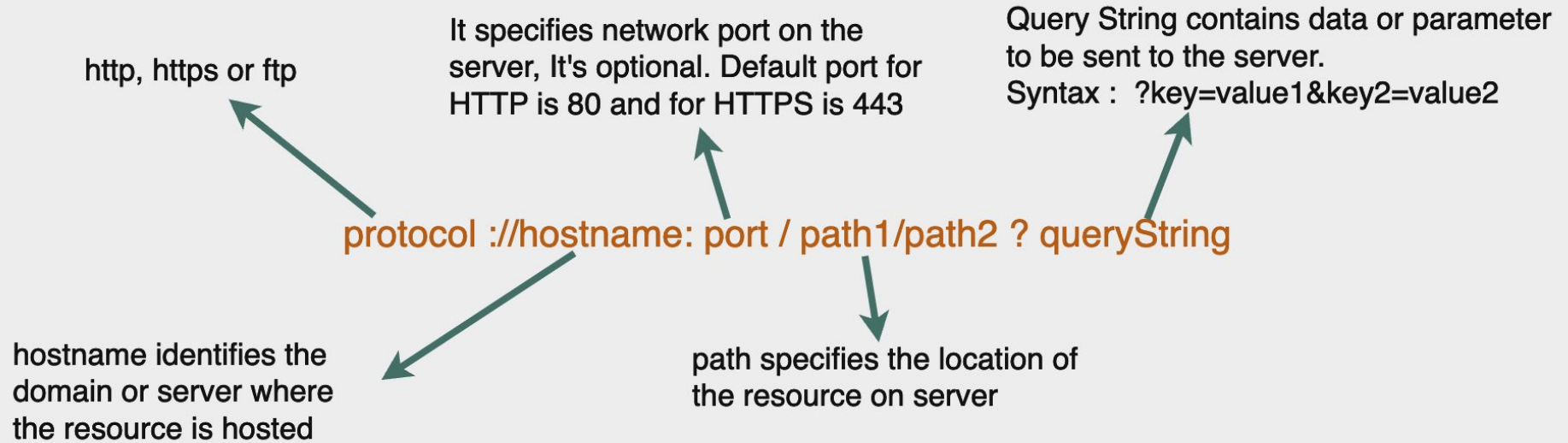
JS globalObjects.js

```
1  console.log(global);
2  console.log(console);
3  console.log(process);
4  console.log(__filename);
5  console.log(__dirname);
6  console.log(module);
```

Path: The path module is a built-in module that provides utilities for working with file and directory paths. It helps in creating platform-independent file paths and performing path-related operations.

```
path.js  ×
; path.js > ...
1  const path = require('path');
2  // Join the paths
3  const fullPath = path.join(__dirname, '/path', 'to', 'directory', 'file.txt');
4  console.log(fullPath);
5  // To find file extension
6  const fileName = "example.txt";
7  const fileExt = path.extname(fileName);
8  console.log(`File extension is: ${fileExt}`);
9  // File base name
10 const file = "/home/user/documents/reports.pdf";
11 const baseName = path.basename(file);
12 console.log(`Base name is: ${baseName}`);|
13 // File Directory name
14 const dirName = path.dirname(file);
15 console.log(`Directory name is: ${dirName}`);
```

URL - Uniform Resource Locator: This module provides utilities for working with URLs, allowing you to parse, format, and manipulate URL strings



1. url module methods for parsing URL

`parse(urlString, parseQueryString)`: Parses a URL string and returns an object with its various components.

- `urlString`: The URL string to be parsed.
- `parseQueryString`: If true, the query property will always be set to an object.

JS path.js

JS url.js

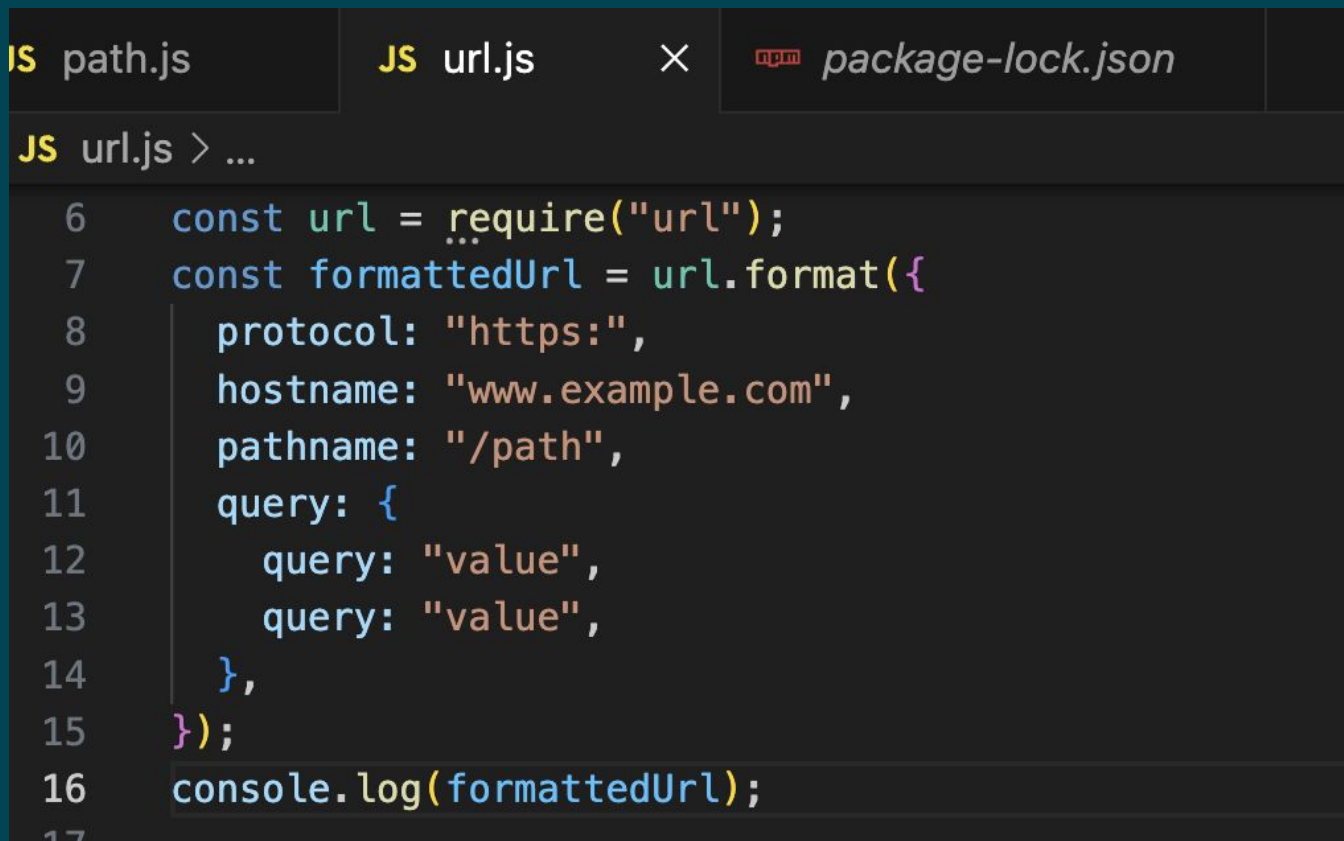
×

JS url.js > ...

```
1  const url = require('url');
2  const urlString = 'https://www.example.com/path?query=value';
3  const parsedUrl = url.parse(urlString, true);
4  console.log(parsedUrl);
```

2. url module methods for formatting URLs

`format(urlObject)`: Takes an object with URL components and returns a formatted URL string.—



```
JS path.js JS url.js × npm package-lock.json
JS url.js > ...
6   const url = require("url");
7   const formattedUrl = url.format({
8     protocol: "https:",
9     hostname: "www.example.com",
10    pathname: "/path",
11    query: {
12      query: "value",
13      query: "value",
14    },
15  });
16  console.log(formattedUrl);
17
```

3. url module methods for Resolving URL

`resolve(from, to)`: Resolves a relative URL (to) against a base URL (from).

JS path.js

JS url.js

×

JS url.js > ...

14

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

16 `const resolvedUrl = url.resolve('https://www.example.com/path', 'subpath');`

17 `console.log(resolvedUrl);`

'nodemon' Dependency → This will monitors the changes in our source code files and automatically restarts our server when changes are detected.

Installation → **npm install nodemon**

To run our JS file use → **nodemon start file_name.js**

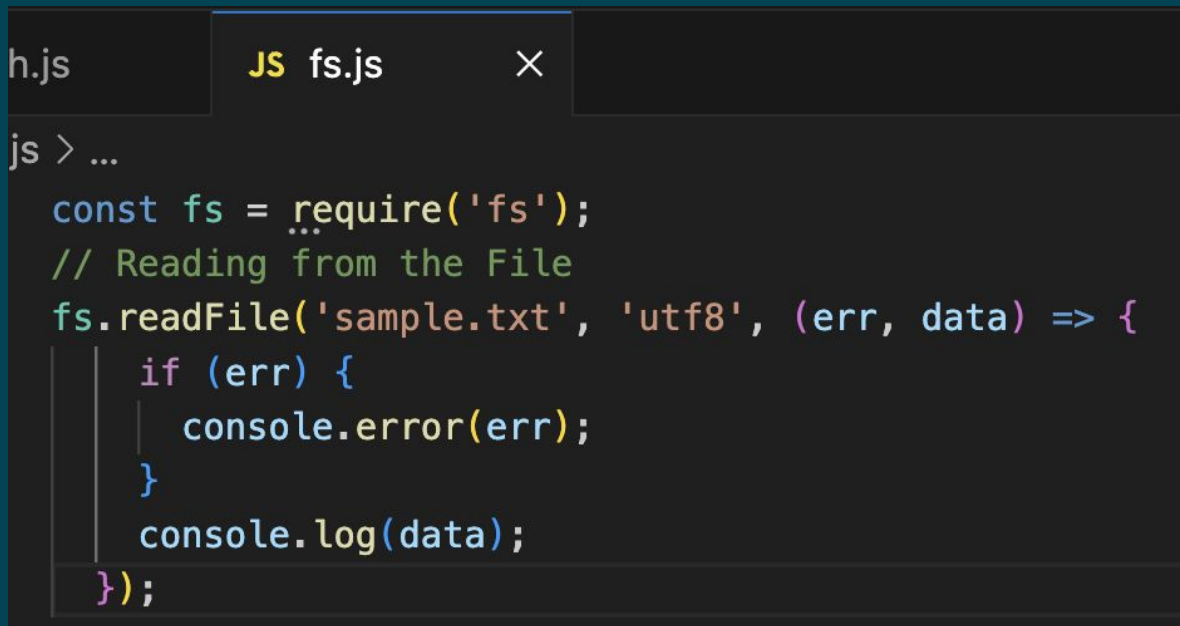
To run from script → **nodemon start**

Note: In case nodemon command is not working install the same dependency globally using -g option.

fs: The fs (File System) module provides a way to interact with the file system, allowing you to perform various operations such as reading, writing, and manipulating files and directories

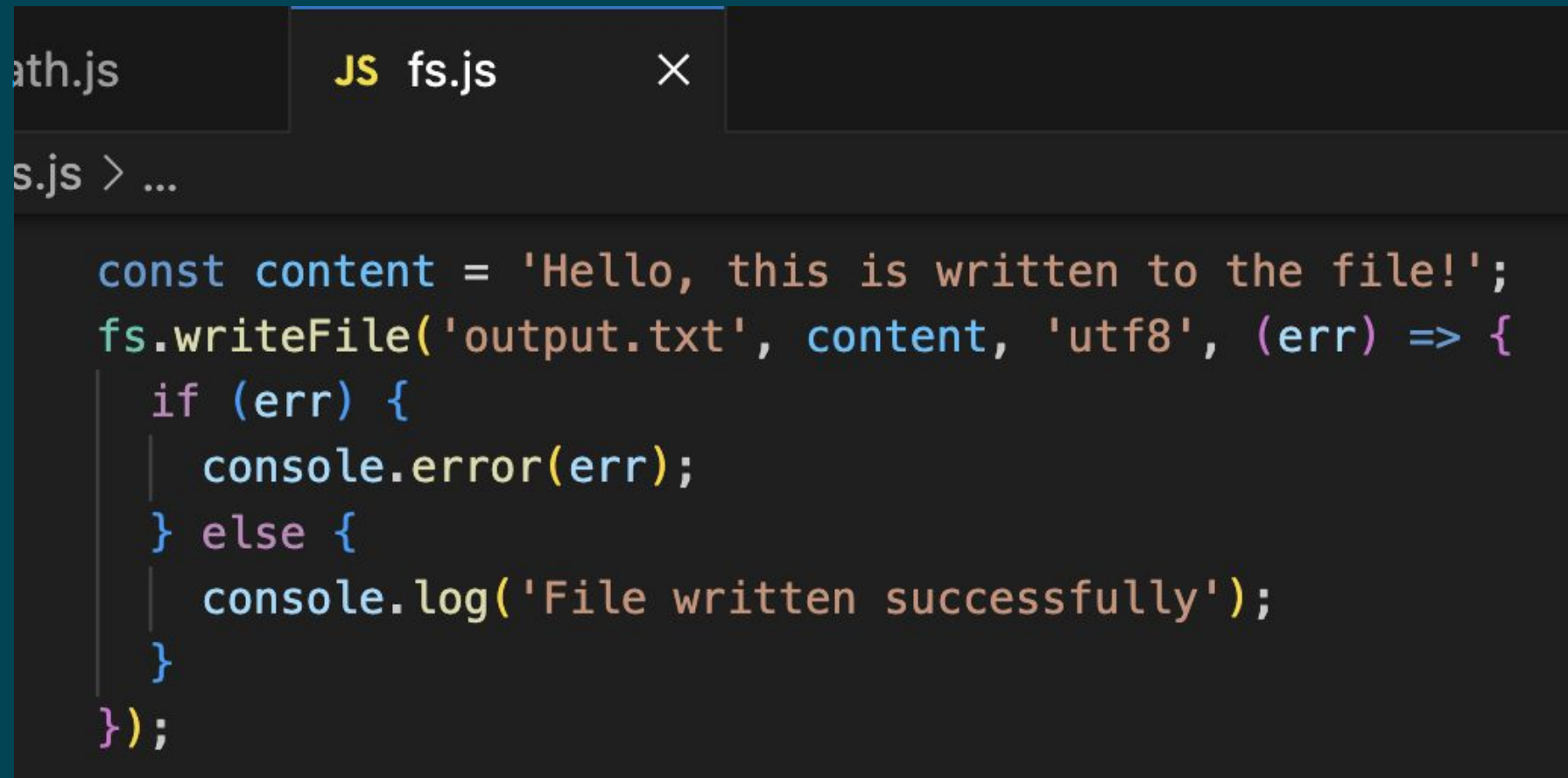
Reading the content from the file → sample.txt

fs.readFile() works asynchronously, so the program doesn't wait for the file read to complete before moving on to the next instruction.



```
h.js    JS fs.js    ×  
js > ...  
const fs = require('fs');  
// Reading from the File  
fs.readFile('sample.txt', 'utf8', (err, data) => {  
  if (err) {  
    console.error(err);  
  }  
  console.log(data);  
});
```

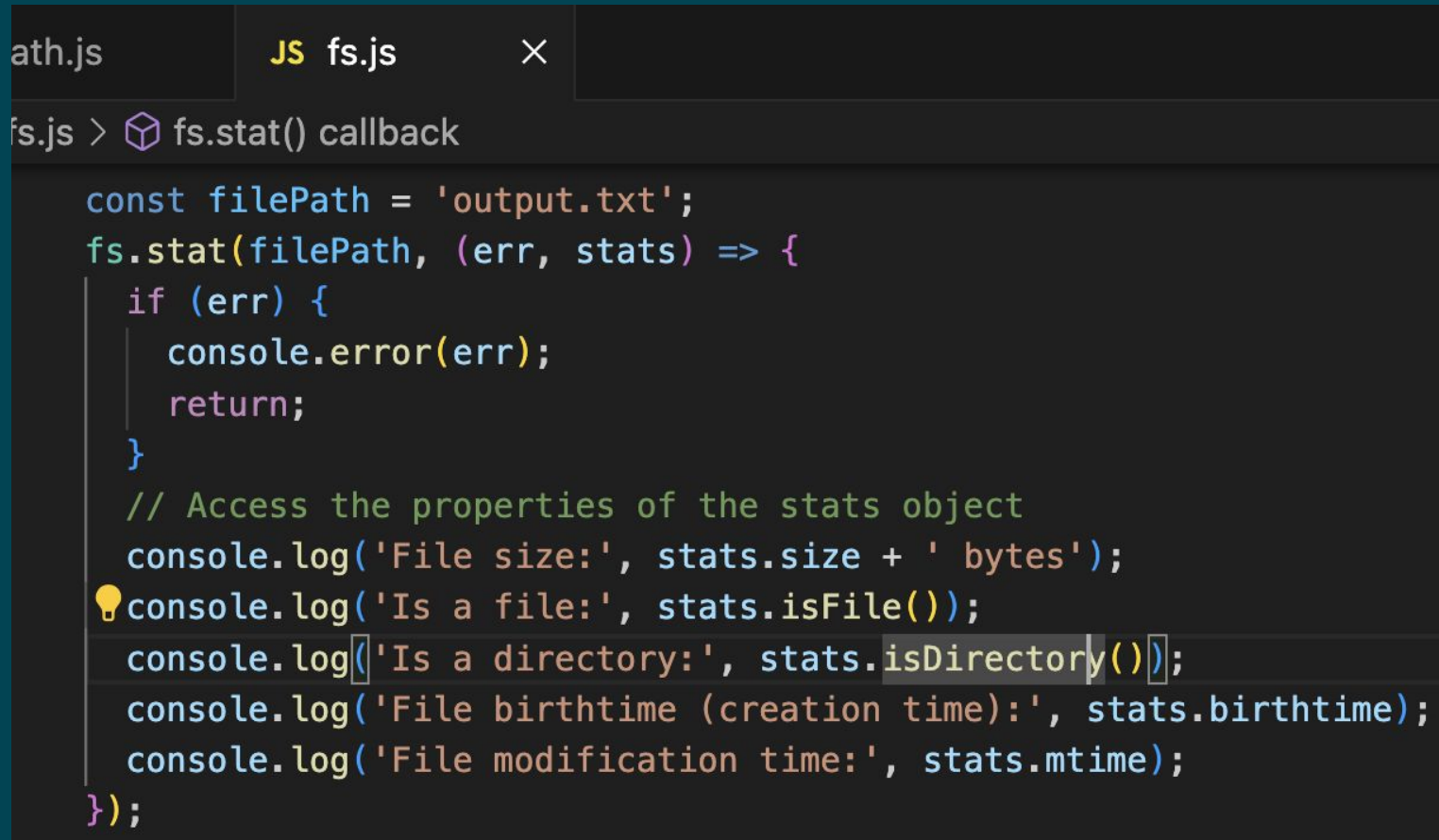

Writing content to the file: `fs.writeFile()` to write data to a file asynchronously.





The image shows a code editor with a dark theme. At the top, there are two tabs: 'ath.js' and 'JS fs.js'. The 'JS fs.js' tab is active. Below the tabs, the file path 's.js > ...' is visible. The main area of the editor contains the following JavaScript code:

```
const content = 'Hello, this is written to the file!';
fs.writeFile('output.txt', content, 'utf8', (err) => {
  if (err) {
    console.error(err);
  } else {
    console.log('File written successfully');
  }
});
```

stat(): The `fs.stat()` method in Node.js is used to retrieve information about a file or directory. It provides details such as file size, permissions, and timestamps.



```
ath.js  JS fs.js  ×  
fs.js >  fs.stat() callback  
  
const filePath = 'output.txt';  
fs.stat(filePath, (err, stats) => {  
  if (err) {  
    console.error(err);  
    return;  
  }  
  // Access the properties of the stats object  
  console.log('File size:', stats.size + ' bytes');  
   console.log('Is a file:', stats.isFile());  
  console.log('Is a directory:', stats.isDirectory());  
  console.log('File birthtime (creation time):', stats.birthtime);  
  console.log('File modification time:', stats.mtime);  
});
```

appendFile(): The `fs.appendFile()` method in Node.js is used to asynchronously append data to a file. If the file does not exist, it will be created. If the file already exists, the new data will be appended to the end of the file.

JS path.js

JS fs.js

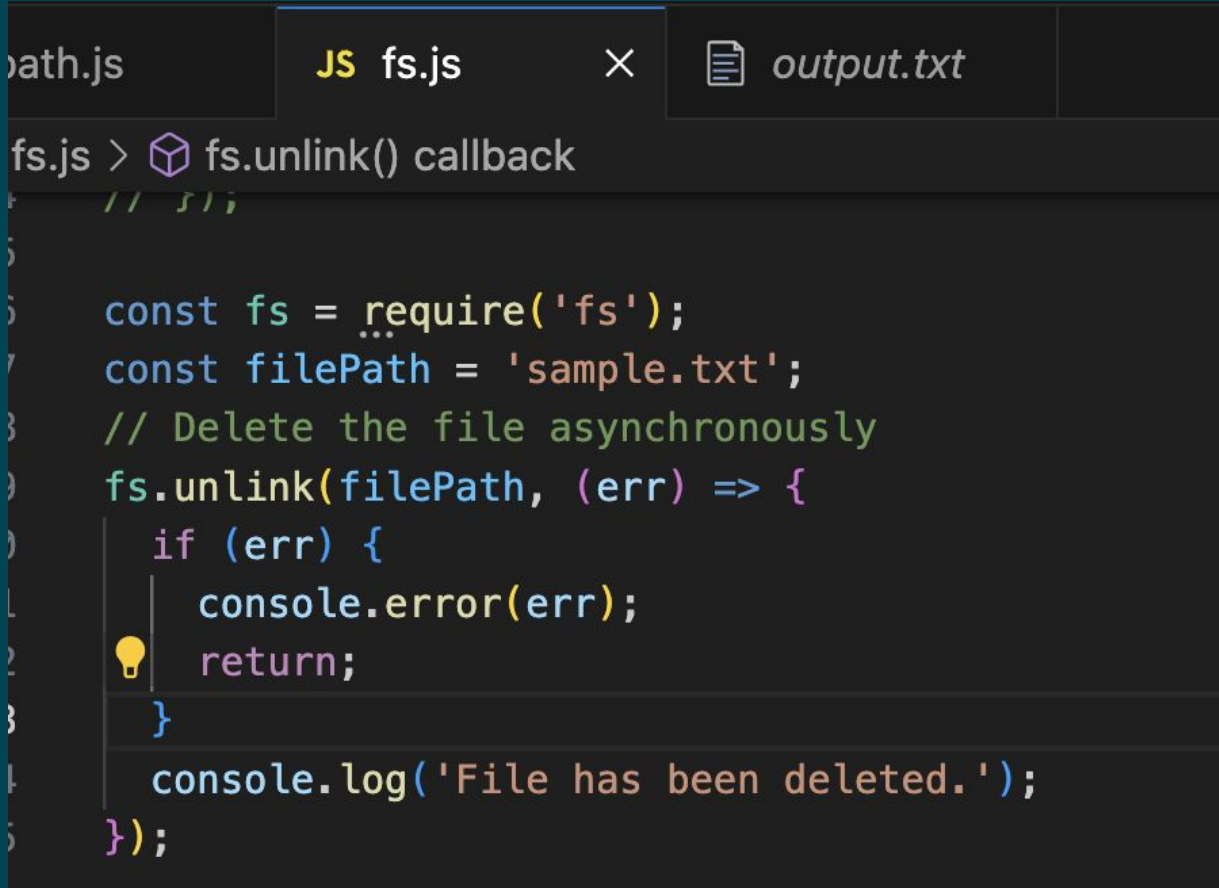
×

output.txt


JS fs.js > ...


```
34
35   const filePath = "output.txt";
36   const dataToAppend = '\n This data will be appended to the file.';
37   // Append data to the file asynchronously
38   fs.appendFile(filePath, dataToAppend, (err) => {
39       if (err) {
40           console.error(err);
41           return;
42       }
43       console.log('Data has been appended to the file.');
```

fs.unlink(): fs.unlink() method to delete (remove) a file asynchronously.



The screenshot shows a code editor with three tabs: 'path.js', 'JS fs.js', and 'output.txt'. The 'JS fs.js' tab is active. The code in the editor is as follows:

```
fs.js >  fs.unlink() callback
// ...

const fs = require('fs');
const filePath = 'sample.txt';
// Delete the file asynchronously
fs.unlink(filePath, (err) => {
  if (err) {
    console.error(err);
     return;
  }
  console.log('File has been deleted.');
```

`fs.mkdir()`: File System) module provides a method to create a directory (folder) asynchronously.

JS path.js

JS fs.js



output.txt

JS fs.js > ...

50

```
57 const fs = require('fs');
```

```
58 const directoryPath = 'new_directory';
```

```
59 // Create a directory asynchronously
```

```
60 fs.mkdir(directoryPath, (err) => {
```

```
61   if (err) {
```

```
62     console.error(err);
```

```
63     return;
```

```
64   }
```

```
65     console.log('Directory created successfully.');
```

```
66   });
```

Thank You



Success is not a milestone, it's a journey. And we have vowed to help you in yours.



www.codemindtechnology.com

