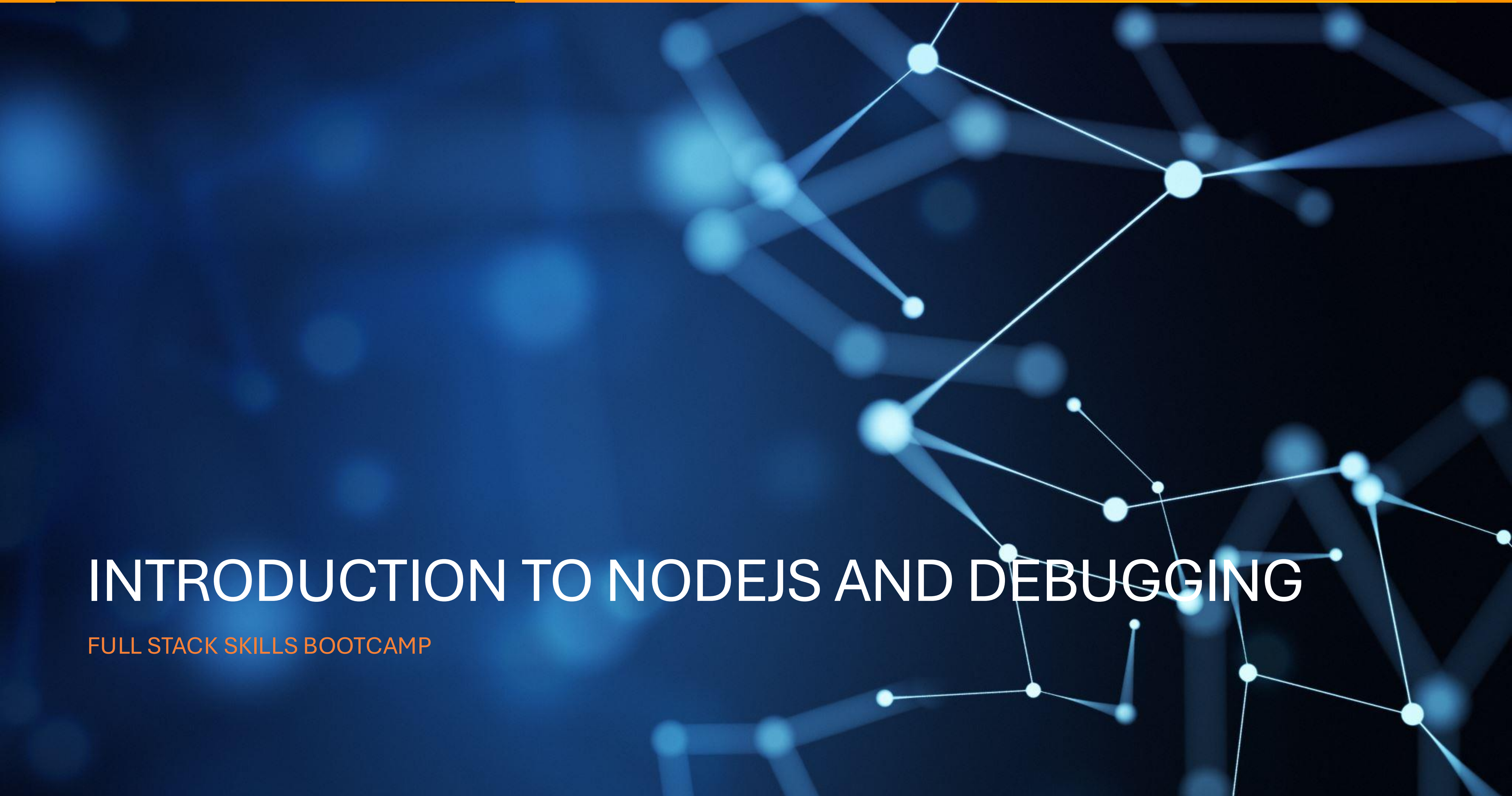


INTRODUCTION TO NODEJS AND DEBUGGING

FULL STACK SKILLS BOOTCAMP

INTRODUCTION TO NODEJS AND DEBUGGING

- **Lesson Overview:**
- In this lesson, we will be introduced to:

1. The NodeJS environment
2. Debugging techniques
3. Modern ES6/7 techniques
4. Using the file system

WHAT IS NODEJS?

- **Definition:** Node.js is a JavaScript runtime built on Chrome's V8 engine.
- **Purpose**: Allows JavaScript to run server-side, outside the browser.
- **Key Features**: Non-blocking I/O, event-driven, fast and lightweight.

Step8Up Academy

INSTALLING NODEJS

- **Step-by-Step Installation:**

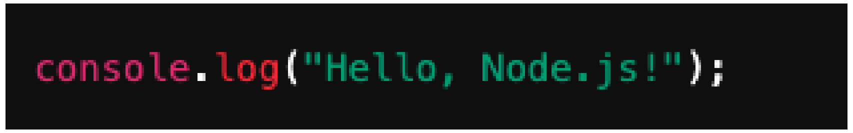
  Go to NodeJS Official Website.

  - Download the latest stable version for your OS.
  - Run the installer and verify installation with node -v in the terminal.
- **NPM (Node Package Manager):** Automatically installed with Node.js. Allows for easy installation of libraries.

RUNNING A SIMPLE JAVASCRIPT FILE

- **File Example**: index.js
- **How to Run**:

  Open terminal, navigate to the directory containing index.js.
- Run the command: node index.js.



```
console.log("Hello, Node.js!");
```

DEBUGGING SIMPLE SCRIPTS AND FUNCTIONS

- **How to Start Debugging**:
  Use the Run | Start Debugging command in VSCode to start debugging.
- **Key Debugging Actions**:

  step over: executes the function then stops
- step into: steps into the function
- step out: runs to the end of the function then stops
- continue: Continue until breakpoint.
- stop: stops the application and debugging

  demo...

ES6/7 VS TRADITIONAL JAVASCRIPT

- **Differences**:

  Syntax improvements (arrow functions, modules).

  More readable and concise code.

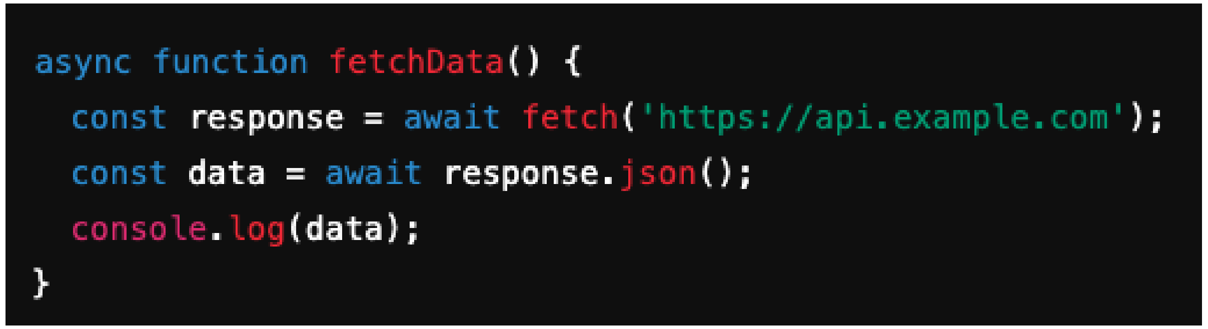
  Focus on better handling of asynchronous operations (e.g., async/await).

ASYNC/AWAIT

- **What is Async/Await?**:

  An easier way to work with asynchronous code.

- **Example**:



```
async function fetchData() {
  const response = await fetch('https://api.example.com');
  const data = await response.json();
  console.log(data);
}
```

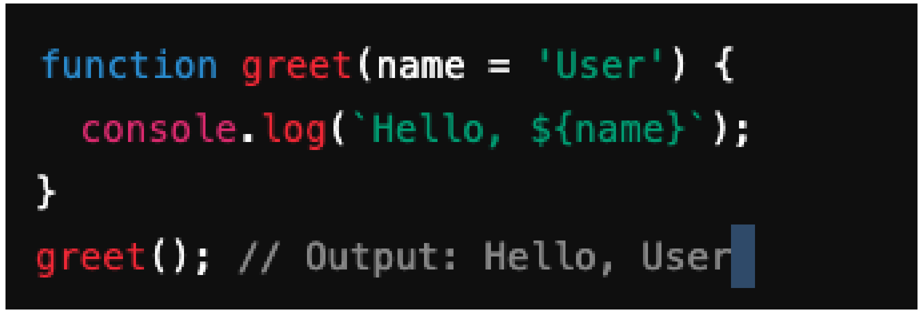
Step8Up Academy

DEFAULT FUNCTION PARAMETERS

- **Definition:**

  Set default values for parameters in functions.

- **Example:**



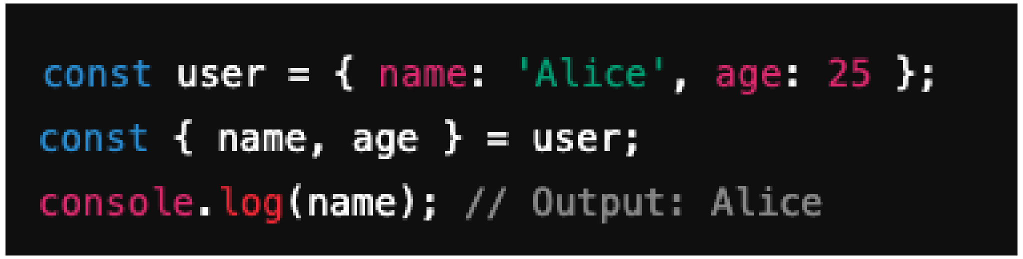
```
function greet(name = 'User') {
  console.log(`Hello, ${name}`);
}
greet(); // Output: Hello, User
```

DESTRUCTURING

- **Purpose:**

  Extract values from objects and arrays.

- **Example**



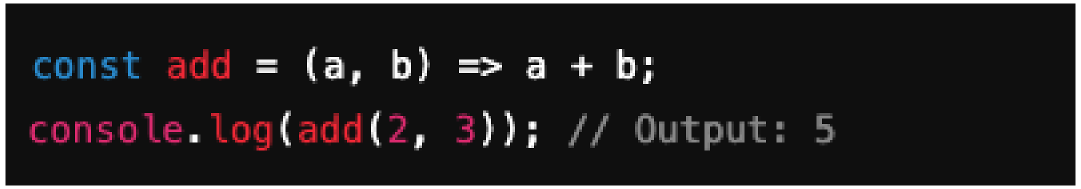
```javascript
const user = { name: 'Alice', age: 25 };
const { name, age } = user;
console.log(name); // Output: Alice
```

ARROW FUNCTIONS

- **Definition:**

  A shorter syntax for writing functions.

- **Example**



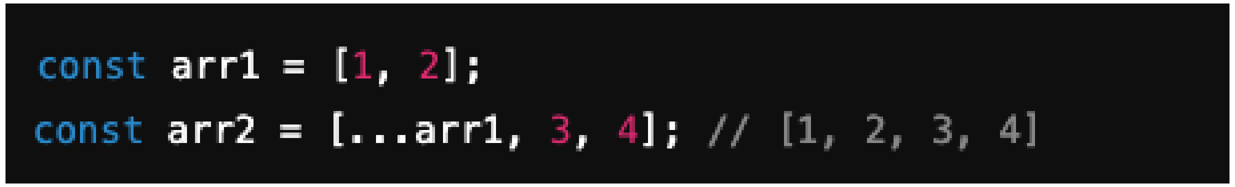
```
const add = (a, b) => a + b;
console.log(add(2, 3)); // Output: 5
```

SPREAD OPERATOR

- **Definition:**

  Expands iterable elements (arrays, objects) into individual elements.

- **Example:**



```
const arr1 = [1, 2];
const arr2 = [...arr1, 3, 4]; // [1, 2, 3, 4]
```

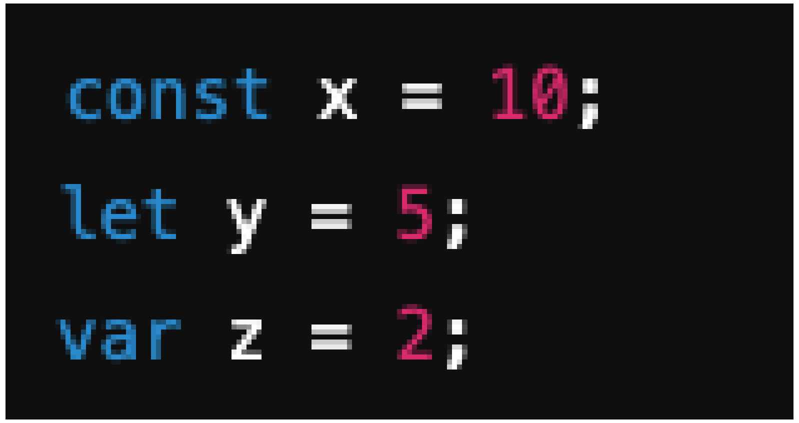
VAR VS LET VS CONST

- **Differences**:

  var: Function-scoped, can be re-declared.

  let: Block-scoped, cannot be re-declared.

  const: Block-scoped, cannot be reassigned or re-declared.
- **Example**:



```
const x = 10;
let y = 5;
var z = 2;
```

ES6 MODULES

- **Importing and Exporting Modules:**

  Export functions or variables from one file.

```
export function add(a, b) { return a + b; }
```

  Import them in another file.

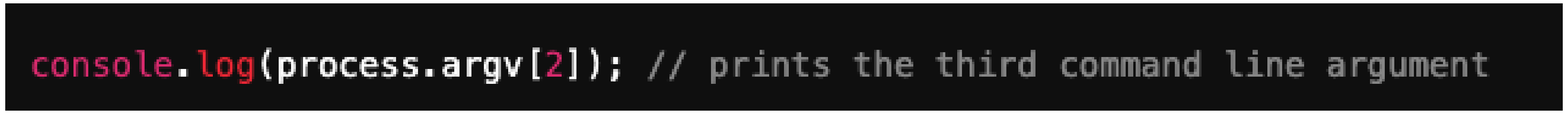
```
import { add } from './math.js';
console.log(add(2, 3));
```

PASSING COMMAND LINE PARAMETERS

- **How to Access Command Line Arguments in Node.js:**

  Use process.argv.

- **Example:**



```
console.log(process.argv[2]); // prints the third command line argument
```

THE FILESYSTEM IN NODE.JS

- **Core Functions:**
  - readFile: Reads the contents of a file.
  - writeFile: Writes data to a file, overwriting existing content.
  - appendFile: Adds data to an existing file without overwriting.

READING AND WRITING FILES

- **Reading a File Example:**

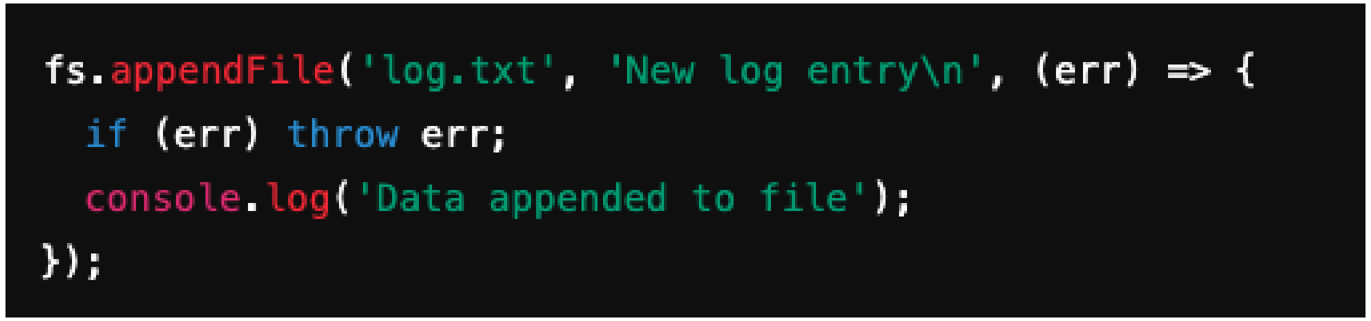
```
const fs = require('fs');
fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

- **Writing to a File Example:**

```
fs.writeFile('example.txt', 'Hello, world!', (err) => {
  if (err) throw err;
  console.log('File written successfully!');
});
```

USING APPENDFILE

- **Appending Data to a File:**



```javascript
fs.appendFile('log.txt', 'New log entry\n', (err) => {
  if (err) throw err;
  console.log('Data appended to file');
});
```

CONCLUSION

- Installed and ran Node.js.
- Debugged simple JavaScript scripts.
- Explored key features of modern JavaScript (ES6/ES7).
- Worked with the filesystem using readFile, writeFile, and appendFile.

QUESTIONS?