**JavaScript Refresher**

**JavaScript Fundamentals Refresher**

Before diving into the complexities of web3 development, it's crucial to have a solid grasp of JavaScript fundamentals. This lesson provides a quick refresher on core concepts, assuming you have NodeJS and npm installed. We'll use Visual Studio Code (VS Code) as our editor and run JavaScript directly from the terminal. If you're already comfortable with basic JavaScript, feel free to skip this section.

**Setting Up and Running Your First JavaScript File**

First, ensure you are working within your project directory (e.g., full-stack-web3-cu).

1. **Create the File:** In VS Code, right-click within the file explorer pane in your project folder and select "New File". Name this file mycode.js.
2. **Run the File:** Open your terminal within VS Code (or a separate terminal navigated to your project directory). To execute your JavaScript file using NodeJS, use the following command:

node mycode.js

* + **Important:** You must be in the same directory as mycode.js when running this command. You can use commands like ls (Linux/macOS) or dir (Windows) to list the files in the current directory and confirm mycode.js is present.

**Displaying Output with console.log**

The console.log() function is essential for viewing output and debugging your code. It prints messages or the values of variables to your terminal.

// Prints the string "hi" to the terminal

console.log("hi");

​

// Prints the string "bye" to the terminal

console.log("bye");

Run node mycode.js after adding this code (and saving the file!) to see the output.

**The Importance of Saving Your Code**

VS Code indicates an unsaved file with a white dot next to the filename in its tab. **Crucially, any changes you make to your code will***not***be reflected when you run the file unless you save it first.**

* Use Cmd+S (Mac) or Ctrl+S (Windows/Linux), or go to File -> Save.
* If you run node mycode.js with unsaved changes, you'll see the output from the *last saved version*. Save the file, then run the command again to see the updated output.

**Understanding Basic JavaScript Data Types**

JavaScript has several fundamental data types. Here are the key ones we'll cover now:

1. **String:** Represents textual data. Always enclose strings in single (') or double (") quotes.
   * Examples: "hello world", 'JavaScript', "123" (this is a string, not a number).
2. **Number:** Represents numerical data (integers or decimals). Do not use quotes.
   * Examples: 25, 16, 3.14, 0.
3. **Boolean:** Represents a logical value, either true or false. Do not use quotes.
   * Examples: true, false.

You can check the data type of a value or variable using the typeof operator:

// Examples of checking data types

const message = "hello";

console.log(typeof(message)); // Output: string

​

const isLoggedIn = false;

console.log(typeof(isLoggedIn)); // Output: boolean

​

const userAge = 25;

console.log(typeof(userAge)); // Output: number

​

const isFalseString = "false"; // Note the quotes

console.log(typeof(isFalseString)); // Output: string

​

const isOneNumber = 1; // Note no quotes

console.log(typeof(isOneNumber)); // Output: number

* **Note:** JavaScript is case-sensitive. The operator is typeof, not typeOf. Typos like this will cause errors.

**Storing Data with Variables: let vs. const**

Variables are used to store data values under a specific name, allowing you to reuse and manipulate data easily. JavaScript provides two primary ways to declare variables: let and const.

1. let**:** Declares a variable whose value *can* be changed (reassigned) later in the code.

let myVariable = "initial value";

console.log(myVariable); // Output: initial value

​

myVariable = "new value"; // Reassigning the variable

console.log(myVariable); // Output: new value

1. const**:** Declares a variable whose value *cannot* be reassigned after it's initially set. It represents a constant value. Attempting to reassign a const variable will result in an error.

// Working example

const fixedValue = "this cannot change";

console.log(fixedValue); // Output: this cannot change

​

// Example causing an error

const myConstant = "hello";

console.log(myConstant); // Output: hello

// The next line will cause an error if uncommented and run:

// myConstant = "goodbye";

If you attempt to reassign a const variable, NodeJS will throw an error similar to this:  
TypeError: Assignment to constant variable.

Choose const by default unless you specifically know you'll need to reassign the variable, in which case use let. This helps prevent accidental changes to values that should remain constant.

**Adding Notes with Comments**

Comments are lines in your code that the JavaScript interpreter ignores. They are used to add explanations, leave notes for yourself or others, or temporarily disable lines of code without deleting them.

* **Syntax:** Single-line comments start with //. Everything after // on that line is ignored.

// This is a single-line comment explaining the code below.

console.log("This line will run.");

​

// The following line is commented out, so it won't execute:

// console.log("This line will NOT run.");

​

// Comments can contain anything:

// asdfkljhasdflkjh23498hasdlkjf

​

// Use comments to clarify complex logic

const taxRate = 0.08; // 8% sales tax rate

​

// Tip: In VS Code, select lines and use Cmd+/ (Mac) or Ctrl+/ (Win/Linux)

// to quickly toggle comments on/off.

**Creating Reusable Code Blocks with Functions**

Functions allow you to group a block of code that performs a specific task. You define the function once and can then call (or invoke) it multiple times whenever you need to perform that task, making your code more organized and reusable.

* **Definition Syntax:**

function functionName(parameter1, parameter2) {

// Code to be executed goes here

// It can use parameter1 and parameter2

}

* **Calling Syntax:**

functionName(argument1, argument2);

Here's a simple example:

// Define a function named greet

function greet() {

console.log("Hello there!");

}

​

// Call the function multiple times

greet();

greet();

greet();

​

// Output:

// Hello there!

// Hello there!

// Hello there!

* **Note on Hoisting:** JavaScript has a feature called "hoisting" which sometimes allows you to call a function *before* its definition appears in the code. While this works for standard function declarations, it's generally considered good practice to define functions before you call them for better readability.

**Leveraging AI for Code Generation and Debugging**

AI tools like DeepSeek (or others like ChatGPT, Copilot) can be helpful assistants during development.

1. **Code Generation:** You can ask the AI to generate boilerplate code or simple functions. For example, you might ask for a JavaScript function to calculate the square root of a number.
2. **Debugging:** When you encounter an error message in your terminal (like the TypeError from reassigning a const), copy the entire error message. Paste it into the AI chat tool and ask questions like:
   * "What does this error mean?"
   * "How can I fix this error in my code?"
   * "[Paste the error message here] What is causing this?" (Using backticks helps the AI identify it as code/output).

AI can often quickly explain common errors and suggest fixes, accelerating your learning and debugging process.

**Key Workflow Tips**

* **Prerequisites:** Ensure NodeJS and npm are installed.
* **Run Location:** Always execute node yourfile.js from the *same directory* where yourfile.js is located.
* **SAVE YOUR FILES:** Remember to save (Cmd+S/Ctrl+S) before running node. The white dot in the VS Code tab means unsaved changes.
* **Terminal History:** Use the up and down arrow keys in your terminal to cycle through previous commands, saving you typing.
* **JavaScript Flexibility:** Be aware that JavaScript can sometimes be flexible (e.g., optional semicolons, hoisting), which can occasionally lead to confusion. Sticking to consistent practices helps.

This refresher covers the essential JavaScript concepts needed to get started. As we progress, we'll build upon these fundamentals.