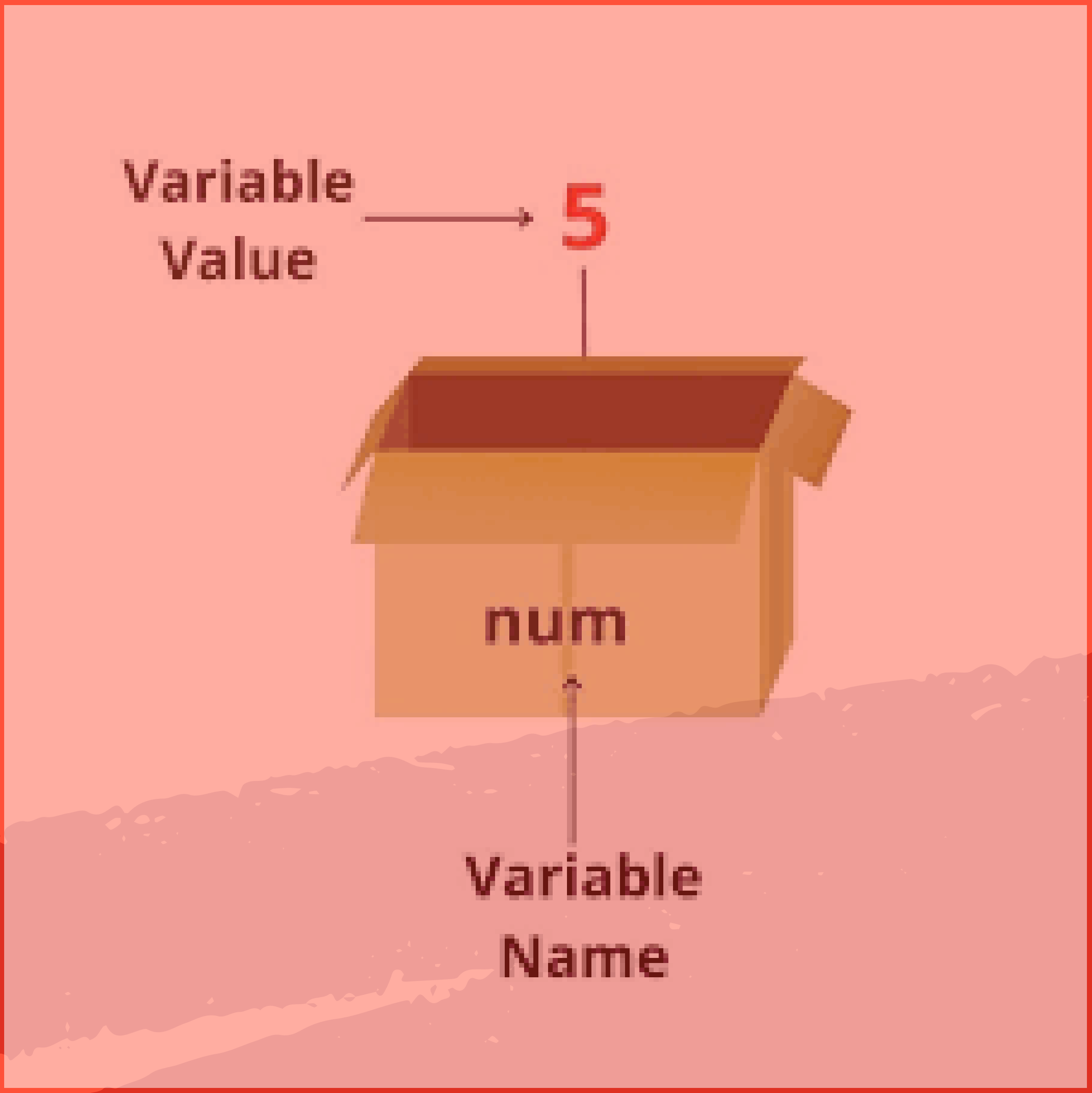
Javascript Variables



A Clear Guide

What are JavaScript Variables?

Imagine you have a storage box with different compartments, each labeled for specific items like sweaters, socks, handkerchiefs, nightwear, and scarves. Whenever you need an item, you simply go to the compartment with the correct label and retrieve it.

This is exactly how JavaScript variables work. Each variable is like a labeled compartment in your storage box, where you store specific pieces of data. When you need to use that data, you reference the variable by its name, just like how you go directly to the labeled compartment to get the item you need. This organization makes it easy to access and manage your data efficiently.

A pictoral representation of the scenerio

JavaScript variables store data that can be used and manipulated throughout your code. They are fundamental to programming and come in three main types: **var**, **let**, and **const**.

Declaring Variables

JavaScript variables store data that can be used and manipulated throughout your code.

Variables can be declared using 3 different keywords in javascript: **var**, **let**, and **const**.

These keywords have similarities and differences based on their **Scope**, **Hoisting**, **Assignment.**

Let’s look at what each of these mean:

Scope: There are two types of scopes in Javascript

- **Function Scope:** Variables declared with var are function-scoped, meaning they are accessible throughout the function in which they are declared.
- **Block Scope:** Variables declared with let and const are block-scoped, meaning they are only accessible within the block (denoted by {}) in which they are declared.

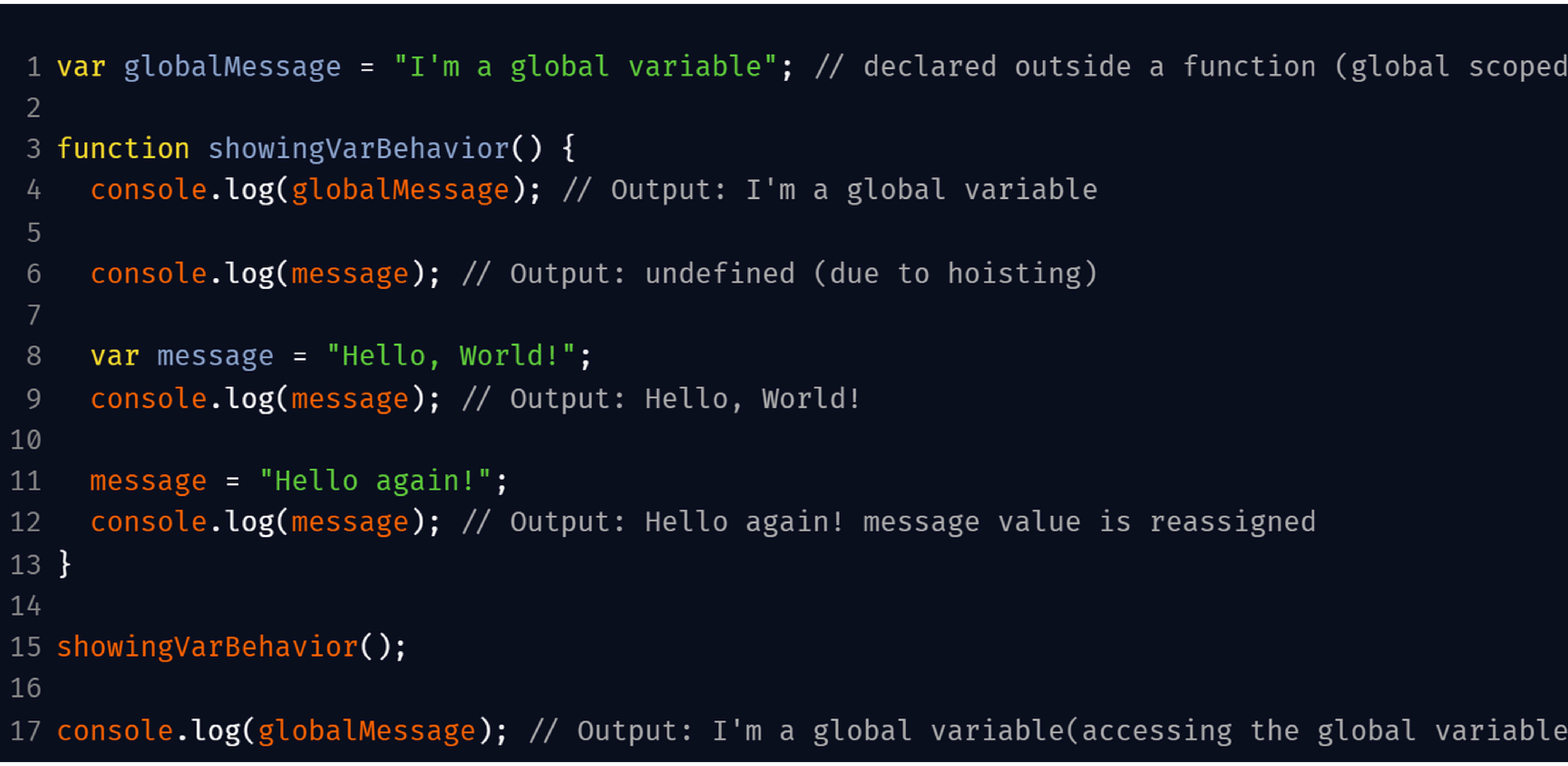
Hoisting: Hoisting is JavaScript's default behavior of moving variable and function declarations to the top of their containing scope before code execution. Only the declarations are hoisted, not the initializations.

Assignment: This means that when a value is assigned to a variable, depending on the keyword used on that variable, the value can either be reassigned or remain constant.

Using **“var”**

It has both **global-scoped** and **function-scoped** but not block-scoped, and **it is hoisted**, meaning it's available throughout the function it's declared in, even before its declaration.

It **can be reassigned** to another value, i.e. the value can change from time to time.

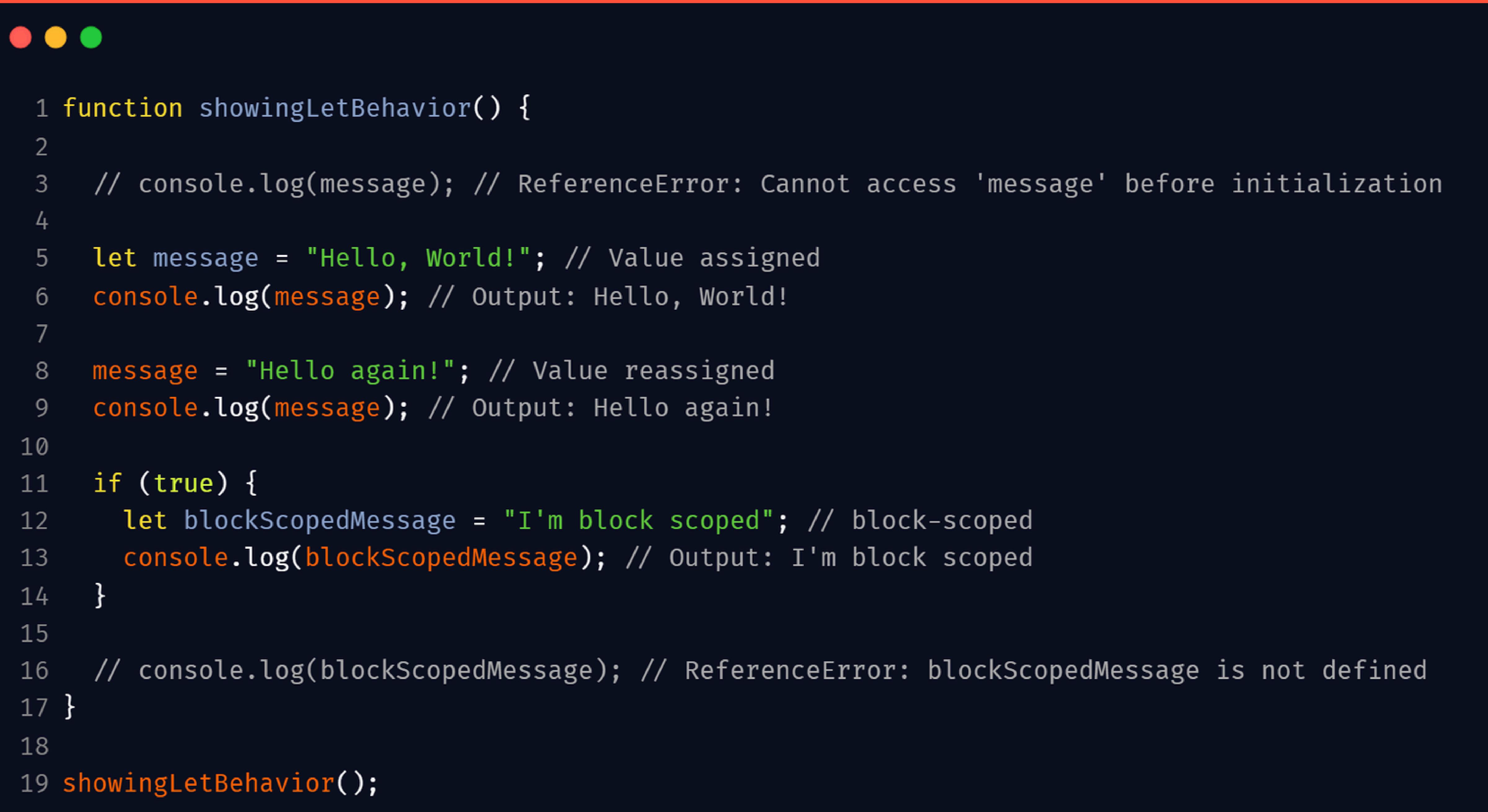


```
var globalMessage = "I'm a global variable"; // declared outside a function (global scoped

function showingVarBehavior() {
  console.log(globalMessage); // Output: I'm a global variable

  console.log(message); // Output: undefined (due to hoisting)

  var message = "Hello, World!";
  console.log(message); // Output: Hello, World!

  message = "Hello again!";
  console.log(message); // Output: Hello again! message value is reassigned
}

showingVarBehavior();

console.log(globalMessage); // Output: I'm a global variable(accessing the global variable
```

Using **“let”**

Introduced in ES6 (ECMAScript 2015). **let** has **block scope**, meaning it's only available within the block it is defined.

Variables declared with **let are not hoisted** to the top of their containing scope. This means that you cannot reference its variable before its declaration line; doing so will result in a **ReferenceError**. They also **can be reassigned**.

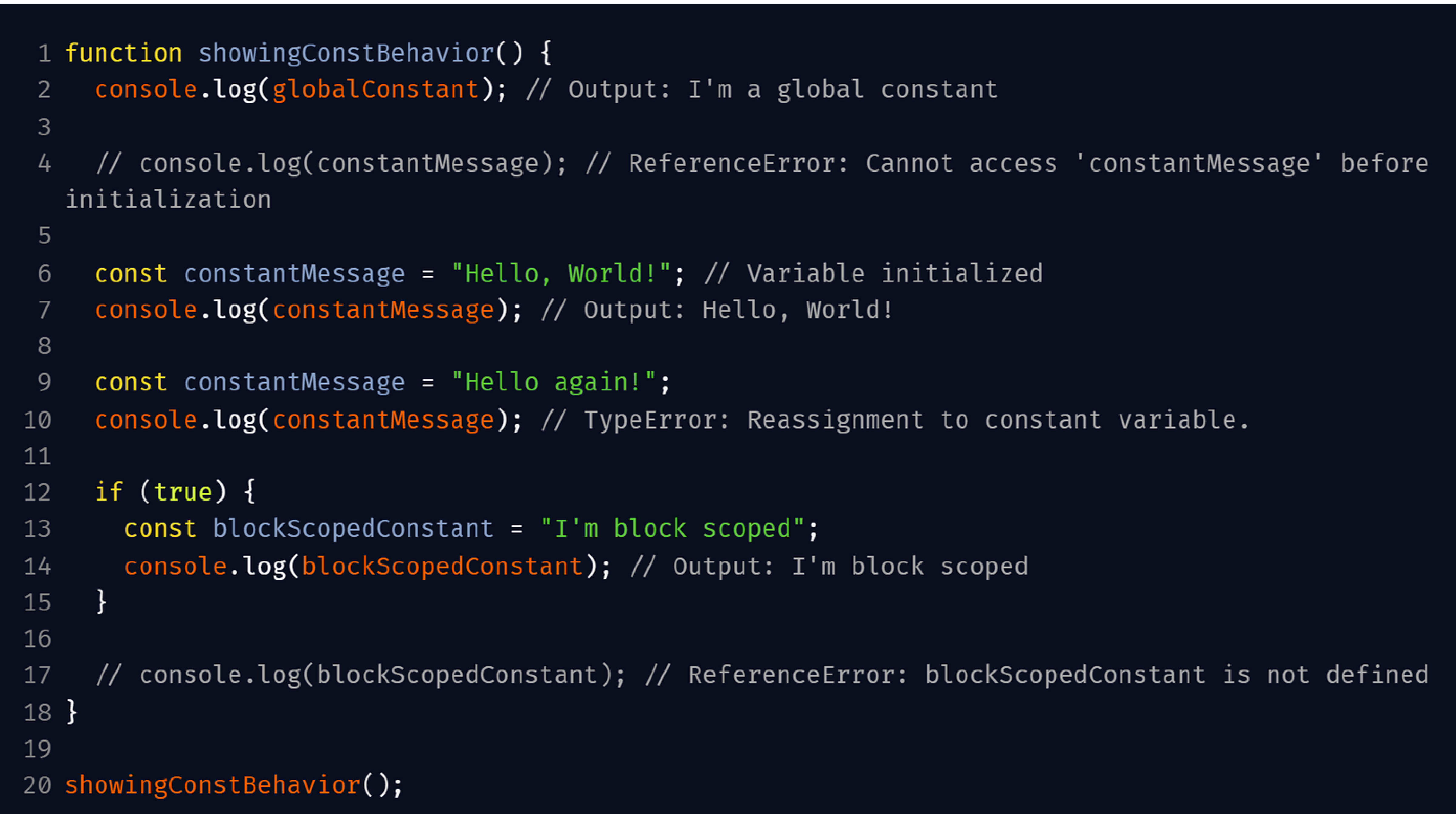


```javascript
function showingLetBehavior() {

  // console.log(message); // ReferenceError: Cannot access 'message' before initialization

  let message = "Hello, World!"; // Value assigned
  console.log(message); // Output: Hello, World!

  message = "Hello again!"; // Value reassigned
  console.log(message); // Output: Hello again!

  if (true) {
    let blockScopedMessage = "I'm block scoped"; // block-scoped
    console.log(blockScopedMessage); // Output: I'm block scoped
  }

  // console.log(blockScopedMessage); // ReferenceError: blockScopedMessage is not defined
}

showingLetBehavior();
```

Using “const”

Also introduced in ES6. It has **block scope** like let, but the value of a const variable **cannot be reassigned** after it's initialized.

Variables declared with **Const are not hoisted** to the top of their containing scope. This means that you cannot reference its variable before its declaration line; doing so will result in a **ReferenceError**.

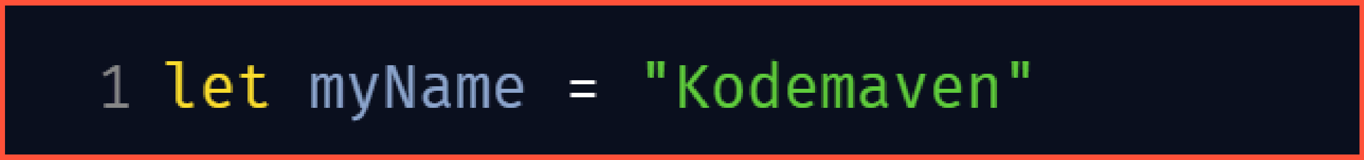


```javascript
function showingConstBehavior() {
  console.log(globalConstant); // Output: I'm a global constant

  // console.log(constantMessage); // ReferenceError: Cannot access 'constantMessage' before initialization

  const constantMessage = "Hello, World!"; // Variable initialized
  console.log(constantMessage); // Output: Hello, World!

  const constantMessage = "Hello again!";
  console.log(constantMessage); // TypeError: Reassignment to constant variable.

  if (true) {
    const blockScopedConstant = "I'm block scoped";
    console.log(blockScopedConstant); // Output: I'm block scoped
  }

  // console.log(blockScopedConstant); // ReferenceError: blockScopedConstant is not defined
}

showingConstBehavior();
```

Best Practices for Declaring a Variable name

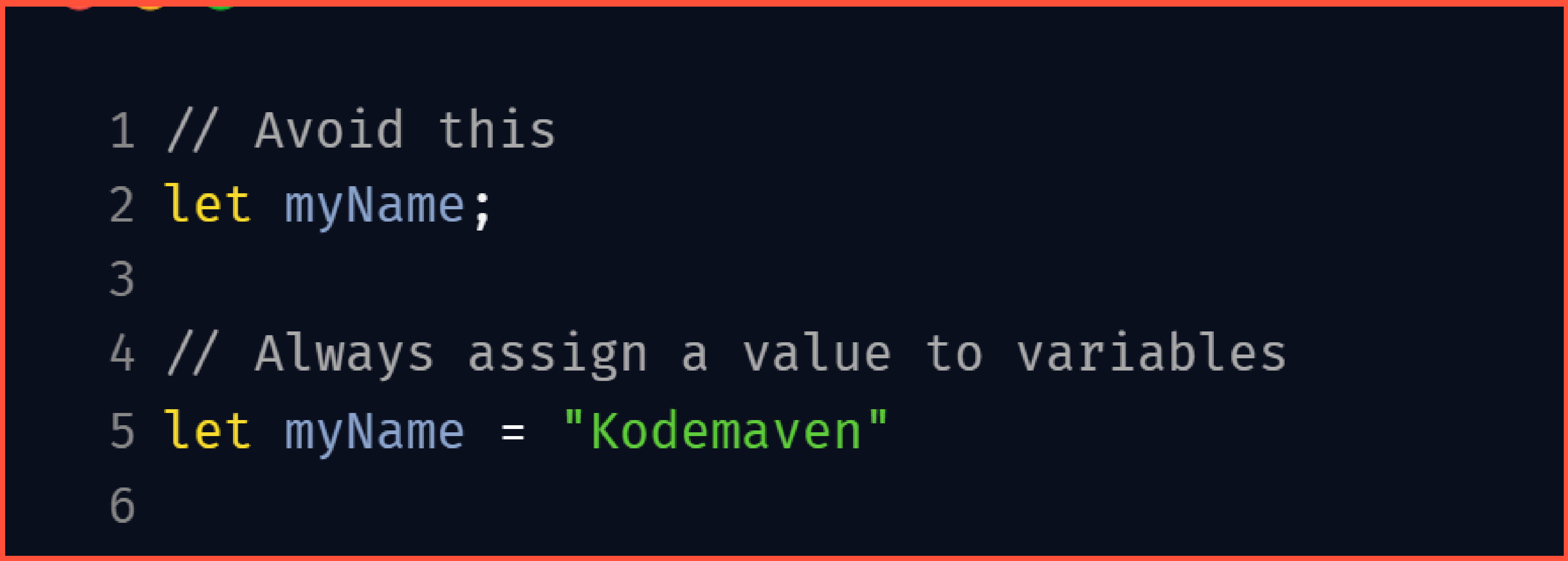
- **Use let and const:** Prefer let and const over var to avoid issues with function scope and hoisting.

- **Use Descriptive Names:** Variable names should be descriptive to make your code more readable.



```
let myName = "Kodemaven"
```

- **Initialize Variables:** Always initialize your variables to avoid unexpected undefined values.



```
// Avoid this
let myName;

// Always assign a value to variables
let myName = "Kodemaven"

```

- **Scope Awareness:** Be aware of the scope in which your variables are declared to prevent scope-related bugs.

I hope you found this material useful.

Remember to:

Like

Save for future reference

Share with your network, be helpful to someone 👌

Hi There! 👋🏼

Thank you for reading through

Did you enjoy this knowledge?

💼 Follow my LinkedIn page for more work-life balancing and Coding tips.

🌐 LinkedIn: Oluwakemi Oluwadahunsi