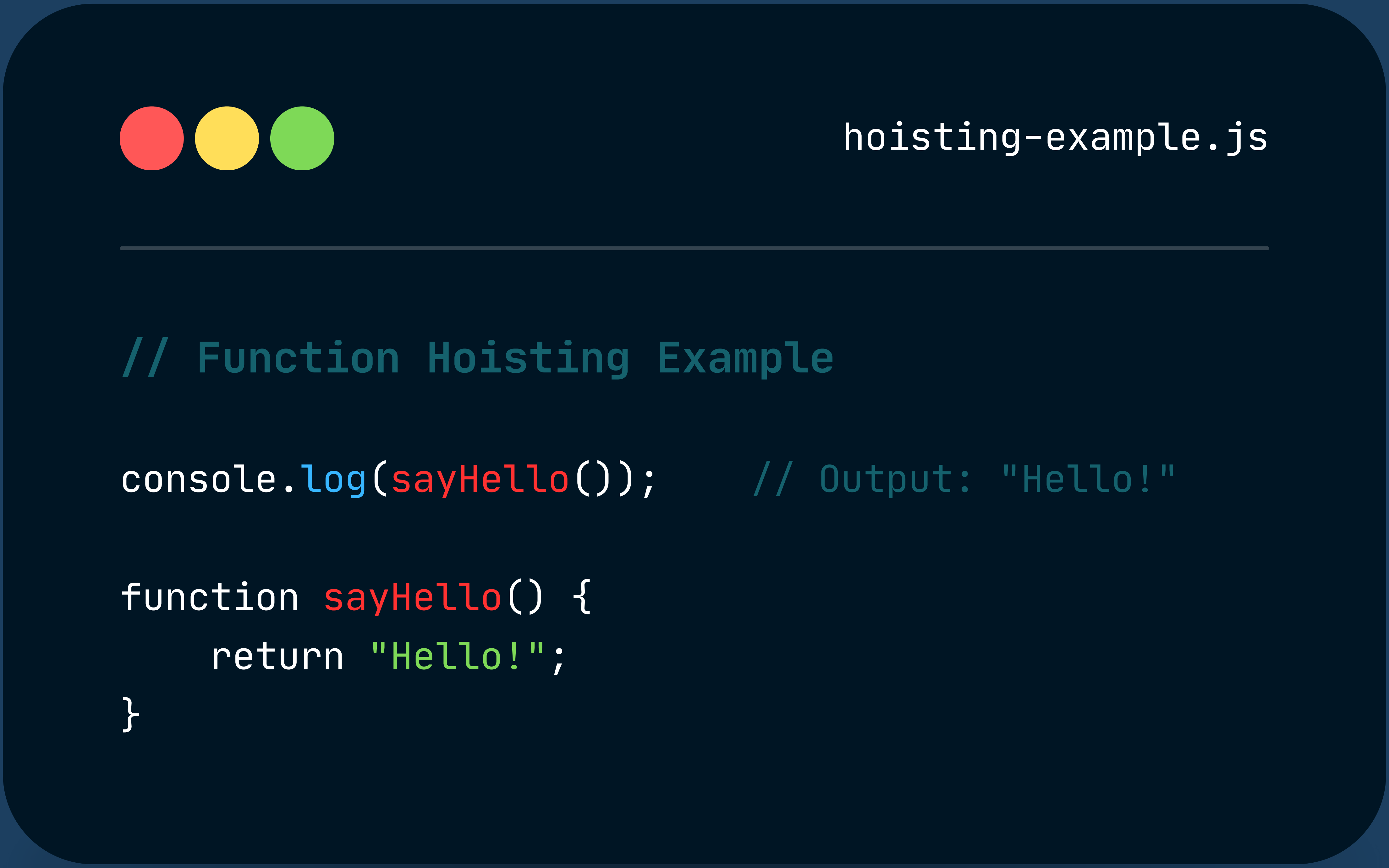
Javascript Hoisting

Hoisting in JavaScript:

During the **memory creation phase**, the **JavaScript engine** moves all **declarations (variables and functions) to the top** of their **scope**. This means that regardless of where functions and variables are declared in the code, they are **moved to the top** in the background before the code executes.

"**Put Simply:** JavaScript's default behavior of moving function and variable declarations to the top of the scope."

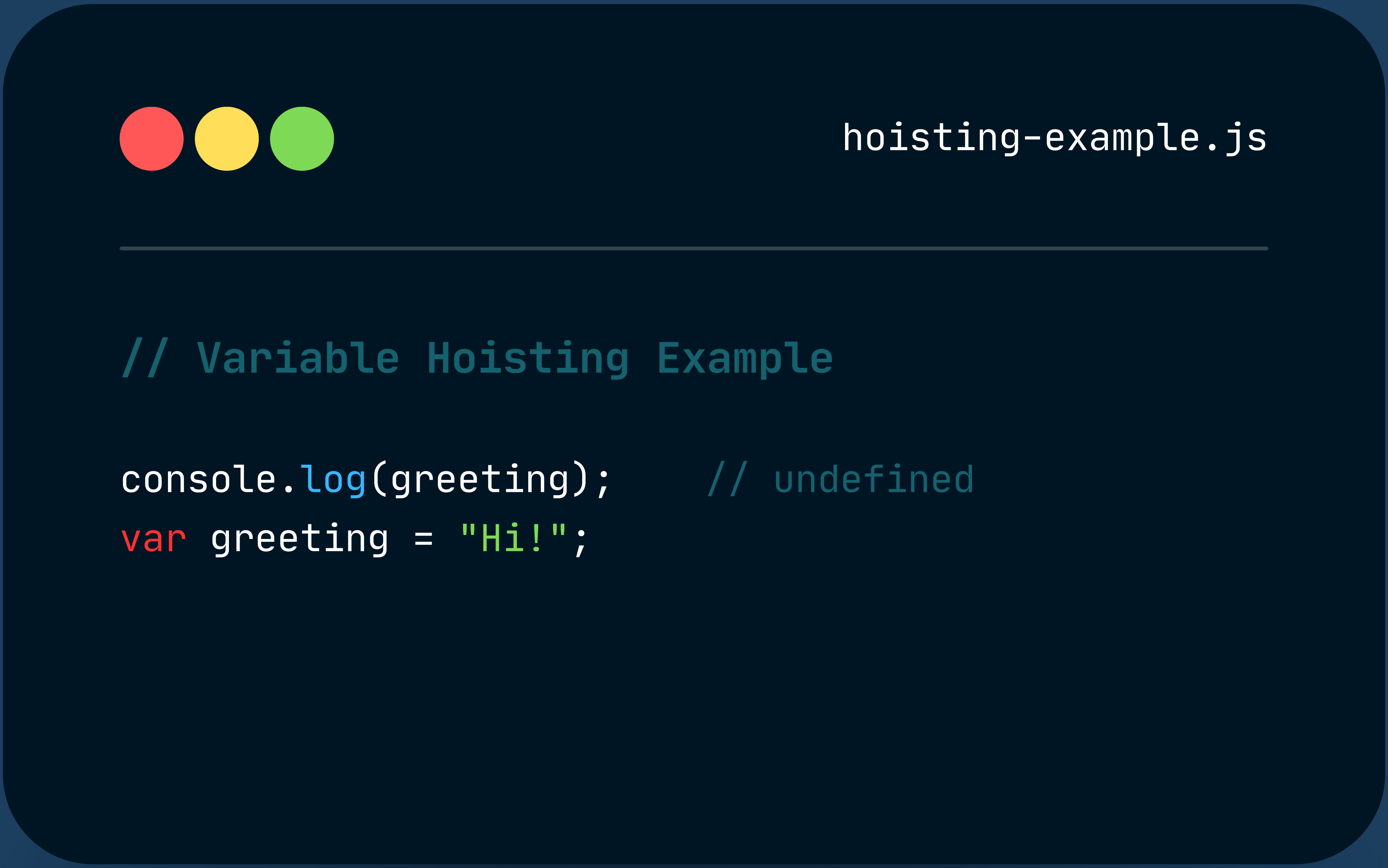


```javascript
// Function Hoisting Example

console.log(sayHello());    // Output: "Hello!"

function sayHello() {
    return "Hello!";
}
```

"**Function declarations** are **fully hoisted**. You can call them before their definition. **Except** functions declared as variables."

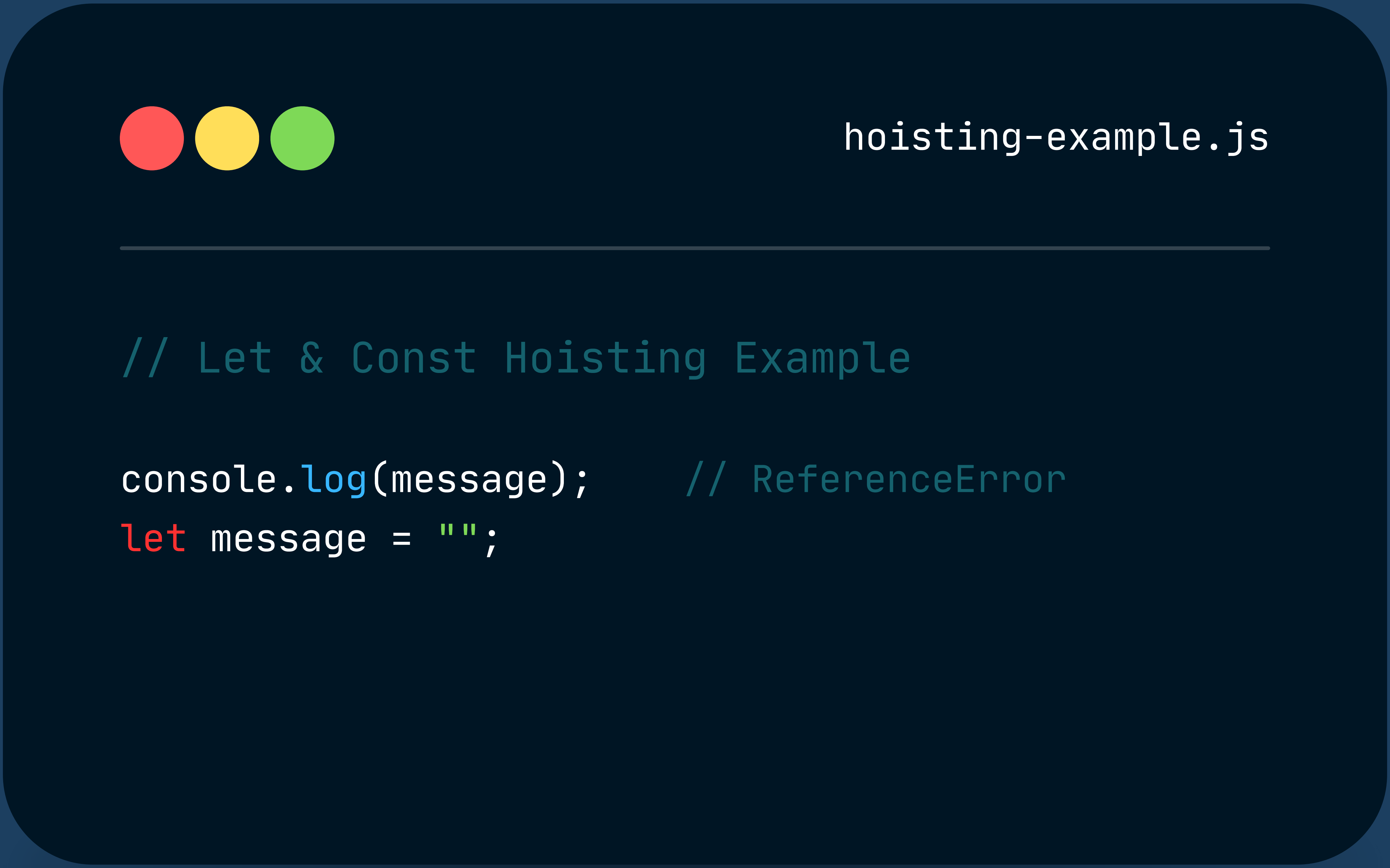


hoisting-example.js

```js
// Variable Hoisting Example

console.log(greeting);    // undefined
var greeting = "Hi!";
```

"Variable declarations are **hoisted**, but their **initializations are not**. This is why you get **'undefined'** when accessing them **before the declaration**."

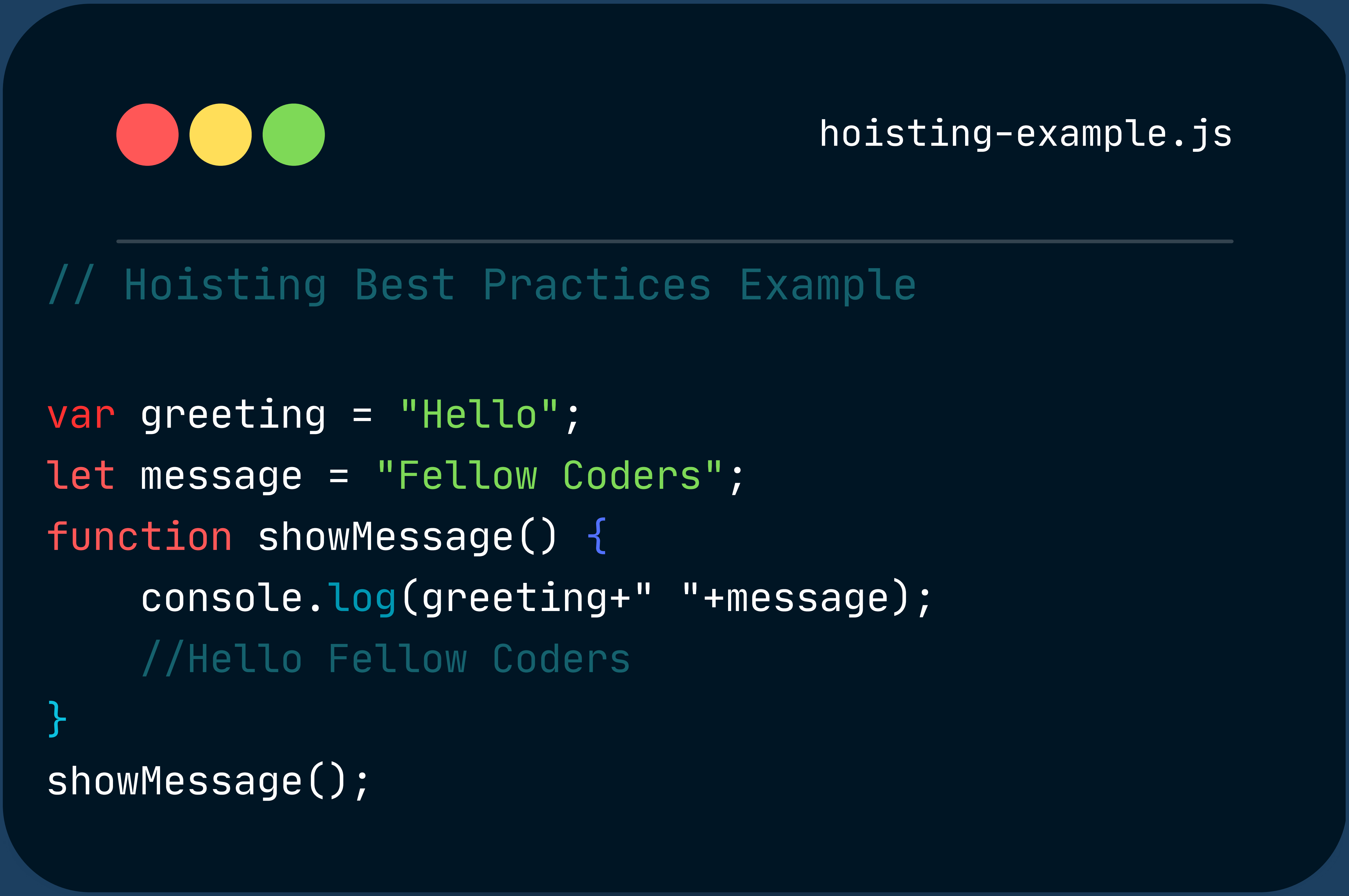


hoisting-example.js

```javascript
// Let & Const Hoisting Example

console.log(message);    // ReferenceError
let message = "";
```

"Variables declared with **let** and **const** are **hoisted** but **not initialized**, leading to a **ReferenceError** if accessed before declaration."


hoisting-example.js

```javascript
// Hoisting Best Practices Example

var greeting = "Hello";
let message = "Fellow Coders";
function showMessage() {
    console.log(greeting+" "+message);
    //Hello Fellow Coders
}
showMessage();
```

Best Practices

"Always declare your variables and functions at the top of their scope to avoid confusion and potential bugs."

FOLLOW FOR MORE JAVASCRIPT INSIGHTS!