

# LET & CONST

In JavaScript, **let** and **const** are two ways to declare variables, each with specific characteristics.

## let:

**Block Scope:** Variables declared with **let** are only accessible within the block (like a set of curly braces { }) where they are defined. This is called block scope.

**Reassignable:** You can change the value of a variable declared with **let** after it's been assigned.

**No Hoisting:** While technically **let** is hoisted (moved to the top of the block), you cannot use the variable before declaring it because it's in a **"temporal dead zone."**

### Example:

```
javascript Copy code  
  
let age = 25;  
age = 26; // This is allowed with 'let'
```

## const:

**Block Scope:** Like **let**, variables declared with **const** are block-scoped.

**Not Reassignable:** Once you assign a value to a **const** variable, you cannot change it. It must be assigned a value when it's declared and cannot be reassigned later.

**Immutable Reference, Not Value:** While you can't reassign a **const** variable, if it's an object or array, you can still modify the contents of that object or array.

### Example:-

```
javascript Copy code  
  
const name = "Alice";  
// name = "Bob"; // This would cause an error because 'name' is a constant  
  
const person = { age: 30 };  
person.age = 31; // This is allowed, you can change properties of the object
```

## Key Differences:

**let:** Use when you expect the variable value to change later.

**const:** Use when the variable value should not change after it's set.

**In summary,** **let** allows you to reassign a variable, while **const** locks the

variable to its initial value, though the contents of objects or arrays declared with `const` can still be modified. Both are block-scoped, making them safer and more predictable compared to the older `var` keyword.

### Are `let` and `const` declarations hoisted? What is temporal dead zone?

**Yes**, both `let` and `const` declarations are hoisted in JavaScript. However, they behave differently from variables declared with `var`.

#### Hoisting:

Hoisting means that JavaScript moves declarations to the top of their containing scope during the compilation phase before the code is executed.

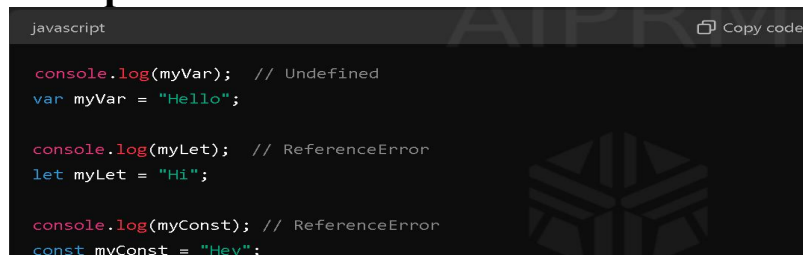
For `let` and `const`, the declaration is hoisted to the top of the block, but they are not initialized. This means that you cannot use them until the code reaches the line where they are actually declared.

#### Temporal Dead Zone (TDZ):

The **Temporal Dead Zone (TDZ)** is the time between entering the scope where a **let** or **const** variable is declared and when the variable is actually declared in the code.

During the TDZ, if you try to access the variable, you'll get a **"Reference Error"**. This happens because, even though the variable is hoisted, it hasn't been assigned a value yet.

### Example:-



```
javascript Copy code  
  
console.log(myVar); // Undefined  
var myVar = "Hello";  
  
console.log(myLet); // ReferenceError  
let myLet = "Hi";  
  
console.log(myConst); // ReferenceError  
const myConst = "Hey";
```

#### Explanation:

**var myVar:** The declaration is hoisted, and **myVar** is initialized with `undefined`. That's why you don't get an error, and it prints `undefined`.

**let myLet** and **const myConst:** The declarations are also hoisted, but they aren't initialized yet. Trying to access them before their declaration results in a **ReferenceError** because they are in the Temporal Dead Zone.

#### Summary:

Hoisting: `let` and `const` are hoisted but not initialized.

Temporal Dead Zone: The period where the variable exists but cannot be accessed until the declaration line is reached.

This helps prevent errors by ensuring you don't accidentally use variables before they are properly declared and initialized.

