

What is a Variable?

In programming, a variable is a **named location** in a computer's memory that stores a value.

This value can be a **number**, **text** (string), **boolean** value (true or false), reference to an **object**, and more.

Swipe through to explore the different types and their roles in coding magic



1. **var**: The Classic Variable

The '**var**' was the **OG** in JavaScript for variable declaration, but it's scoped to functions and sometimes led to unexpected behaviors.

It's considered **outdated** with the arrival of '**let**' and '**const**.'

Basic Example:

```
var name = "John Doe";  
console.log(name); // Output: "John Doe"
```

Drawback: One of the major drawbacks of using var in JavaScript is that it is function-scoped, not block-scoped. This can lead to unexpected results.

In the example code, even though `var x = 5;` is inside the if block, x is accessible outside of this block within the same function. This is because var is function-scoped, not block-scoped. This can lead to unexpected behavior if you're not careful, because you might think x would only exist within the if block.

```
function example() {  
  if (true) {  
    var x = 5;  
  }  
  console.log(x); // Output: 5  
}  
  
example();
```

2. **let**: The Block-Scope Champion

The '**let**' brought **block-scoping** into JavaScript, limiting the variable's scope to the block it's declared in.

It's great for reassigning values within the same scope!

```
Variables.js

let age = 25;
console.log(age); // Output: 25

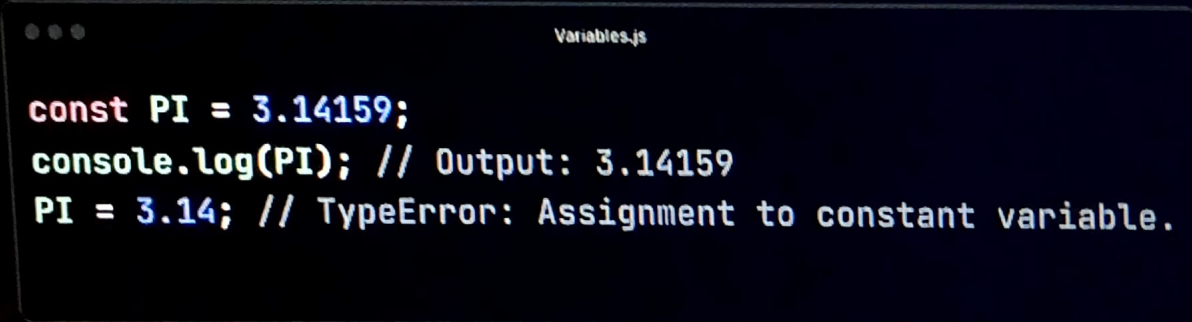
if (true) {
  let age = 30;
  console.log(age); // Output: 30
}

console.log(age); // Output: 25
```


3. **const**: The Immutable Hero

The '**const**' creates **immutable** variables. Once assigned, their values **can't be changed**.

It's perfect for values that shouldn't be reassigned, providing stability in code.



```
Variables.js

const PI = 3.14159;
console.log(PI); // Output: 3.14159
PI = 3.14; // TypeError: Assignment to constant variable.
```

Comparison

	var	let	const
Scope	Function scope	Block scope	Block scope
Hoisting	Yes, to top of function scope, initialized as undefined	Yes, to top of block scope, not initialized	Yes, to top of block scope, not initialized
Re-declaration	Allowed in the same scope	Not allowed in the same scope	Not allowed in the same scope
Re-assignment	Allowed	Allowed	Not allowed
Initialization	Optional. If not done, variable is undefined	Optional. If not done, variable is in "temporal dead zone" until declaration	Must be initialized at declaration