

# TypeScript Variables

- Variable is a container which can hold data.
- Variables in TypeScript (and JavaScript) can be declared using **var**, **let**, or **const**.
- Each has different behaviours in terms of **scope**, **value assignment**, **re-declaration**, **re-assignment**, and **hoisting**.

## 1. Scope

### var → Function Scope

- Variables declared with var are accessible anywhere inside the function.
- Can lead to unexpected behaviour because they are not limited to blocks (if, for, etc.).

#### Example:

```
function exampleVar() {  
    if (true) {  
        var message = "Hello, World!";  
    }  
    console.log(message); // Works! (function-scoped)  
}  
exampleVar(); // Output: "Hello, World!"
```

### let & const → Block Scope

- Variables are **only** accessible inside the block {} where they are declared.
- Safer and more predictable than var.

#### Example:

```
function exampleLetConst() {  
    if (true) {  
        let message = "Hello, let!";  
        const greeting = "Hello, const!";  
    }  
    // console.log(message); // Error: Not accessible outside block  
    // console.log(greeting); // Error: Not accessible outside block  
}  
exampleLetConst();
```

## 2. Value Assignment at Declaration

**var** and **let** - Value assignment is **not mandatory**.

**const** - Value assignment is **mandatory**.

**Example:**

```
var b
```

```
console.log(b); // Output: undefined
```

```
let d;
```

```
console.log(d); // Output: undefined
```

```
const f; // ❌ Error: Missing initializer in `const` declaration
```

```
const g = 60; // ✅ Works because value is assigned at declaration
```

## 3. Re-declaration

Keyword	Allows Re-declaration?
<b>var</b>	✅ Yes
<b>let</b>	❌ No
<b>const</b>	❌ No

**Examples:**

```
var city = "New York";
```

```
var city = "Los Angeles"; // ✅ Allowed (Problem: Can cause bugs!)
```

```
let country = "USA";
```

```
// let country = "Canada"; // ❌ Error (Safer!)
```

```
const planet = "Earth";
```

```
// const planet = "Mars"; // ❌ Error (Safer!)
```

## 4. Re-assignment

Keyword	Allows Re-assignment?
var	✓ Yes
let	✓ Yes
const	✗ No

### Examples:

```
var age = 25;
```

```
age = 30; // ✓ Allowed
```

```
let score = 50;
```

```
score = 60; // ✓ Allowed
```

```
const pi = 3.14;
```

```
// pi = 3.14159; // ✗ Error (Cannot change a constant)
```

## 5. Hoisting (Variable Access Before Declaration)

- **var**: Hoisted but initialized as undefined.
- **let & const**: Hoisted but **not initialized** (cannot be used before declaration).

### Example:

```
console.log(a); // undefined (var is hoisted)
```

```
var a = 10;
```

```
console.log(b); // ✗ Error (Cannot access before initialization)
```

```
let b = 20;
```

```
console.log(c); // ✗ Error (Cannot access before initialization)
```

```
const c = 30;
```

## Summary:

Feature	var	let	const
Scope	Function	Block	Block
Value Assignment at Declaration	✗ Not Mandatory	✗ Not Mandatory	✓ Mandatory
Re-declare	✓ Allowed	✗ Not Allowed	✗ Not Allowed
Re-assign/ Reinitialization	✓ Allowed	✓ Allowed	✗ Not Allowed
Hoisting	✓ (undefined)	✗ (Not initialized)	✗ (Not initialized)
Best Use	✗ Avoid	✓ Changing values	✓ Constants

### var

1. **Scope** – Function-scoped (limited to the function where it is declared).
2. Can be **redeclared** and **reinitialized** within the same scope.
3. Assigning a value at the time of declaration is **optional**.

### let

1. **Scope** – Block-scoped (limited to the enclosing {...} block).
2. Cannot be **redeclared** within the same scope but can be **reinitialized**.
3. Assigning a value at the time of declaration is **optional**.

### const

1. **Scope** – Block-scoped (limited to the enclosing {...} block).
2. Cannot be **redeclared** or **reinitialized** within the same scope.
3. Assigning a value at the time of declaration is **mandatory**.



## Best Practices

- ✓ **Avoid var** – It can cause unexpected bugs due to function scope.
- ✓ **Use let** – When a variable needs to change later.
- ✓ **Use const** – For values that should never change (constants).

## Comments

### Single-line comment

- **Shortcut (Windows/Linux):** Ctrl + /
- **Shortcut (Mac):** Cmd + /

// This is a single-line comment

### Multi-line (Block) comment

- **Shortcut (Windows/Linux):** Shift + Alt + A
- **Shortcut (Mac):** Shift + Option + A

/\*

This is a multi-line comment

Spanning multiple lines

\*/