TypeScript – let & const

1. Problem Statement

Imagine you are developing a modern application where data must be handled safely and predictably.

- Some values should be able to change as your program runs (like a user's score or the current page).
- Other values should never change once set (like a mathematical constant or the name of your application).
- You want to avoid bugs caused by accidentally changing or reusing variables in the wrong place.

The challenge:
How do you declare variables in TypeScript so that you control where and how they can be changed, and prevent accidental mistakes in your code?

2. Learning Objectives

By the end of this lesson, you will be able to:

- Declare variables using `let` and `const` in TypeScript.
- Understand the difference between block scope and function/global scope.
- Recognize when to use `let` versus `const`.
- Avoid common mistakes with variable redeclaration and reassignment.

3. Concept Introduction with Analogy

Analogy: Lockers and Safes in a Secure Facility

- **Let variables** are like lockers: you can open them, put in new items, or swap items out, but only if you have access to that room (block).
- **Const variables** are like safes with a combination set once: you can put something in when you first use it, but after that, it's locked forever and can't be changed.
- Both lockers and safes are only accessible in the room (block) where they're placed.

This system keeps your valuables safe and prevents accidental mix-ups, just like `let` and `const` keep your data safe in TypeScript.

4. Technical Deep Dive

Declaring Variables with `let`

- The `let` keyword declares a variable with **block scope**.
- You must follow the rules for naming identifiers.
- You can assign a type and a value, or just a type.

Syntax:

```
let var_name: var_type = value;
```

Example:

```
let car_name: string = "Brezza";
let car_price: number = 1000000;
console.log(car_name);   // Output: Brezza
console.log(car_price);  // Output: 1000000
```

Variable Scope

- Variables declared with `let` are **block scoped**.
- You cannot access a `let` variable outside the block where it is declared.

Example:

```
let bool: boolean = true;
if (bool) {
    let result: number = 10;
    console.log(result); // Accessible here
}
// console.log(result); // Error: result is not defined here
```

Redeclaration Rules

- You **cannot re-declare** a variable with `let` in the same scope.

Example:

```
let animal: string = "cat";
// let animal: string = "dog"; // Error: Cannot redeclare block-scoped vari
console.log(animal); // Output: cat
```

Same Name in Different Blocks

- You **can** declare variables with the same name in different blocks.

Example:

```
let bool: boolean = false;
if (bool) {
    let num: number = 1;
    console.log(num); // Only in this block
} else {
    let num: number = 2;
    console.log(num); // Only in this block
}
// Both num variables are separate and do not conflict.
```

Declaring Variables with `const`

- The `const` keyword declares a **constant** (cannot be changed after assignment).
- Must be initialized at the time of declaration.
- Has block scope, just like `let`.

Syntax:

```
const var_name: var_type = value;
```

Example:

```
const lang: string = 'TypeScript';
const PI: number = 3.14;
console.log(`Language: ${lang}`);      // Output: Language: TypeScript
console.log(`Value of PI: ${PI}`);     // Output: Value of PI: 3.14
```

Const Rules: No Redeclaration or Reassignment

- You **cannot re-declare** or **reassign** a `const` variable in the same scope.

Example:

```
if (true) {
    const PI: number = 3.14;
    console.log(PI);
    // const PI: number = 3.14; // Error: Cannot redeclare block-scoped var
    // PI = 3.15; // Error: Cannot assign to 'PI' because it is a constant.
}
```

5. Step-by-Step Data Modeling & Code Walkthrough

1. **Declare a variable with `let`:**

```
let userName:  string  =  "Alex";`
```

2. **Declare a constant with `const`:**

```
const MAX_USERS: number = 100;
```

3. **Block scope demonstration:**

```
if (true) {
 let sessionId: string = "abc123";
 console.log(sessionId); // Works here
}
 // console.log(sessionId); // Error: sessionId is not defined here
```

4. **No redeclaration or reassignment with `const`:**

```
const appName: string = "MyApp";
// appName = "YourApp"; // Error: Cannot assign to 'appName'
```

6. Interactive Challenge

Your Turn!

- Declare a variable `score` with `let` and assign it a number.
- Inside a block (e.g., an `if` statement), declare another `score` variable with a different value and print it.
- Declare a constant `COUNTRY` and assign it your favorite country.
- Try to change the value of `COUNTRY` and observe what happens.
- Try to re-declare `score` in the same block and see the result.

8. Common Pitfalls & Best Practices

- **Always initialize `const` variables when declaring them.**
- **Use `let` for variables that change, `const` for variables that never change.**
- **Don't try to access block-scoped variables outside their block.**
- **Never redeclare a `let` or `const` variable in the same scope.**

- **Prefer `const` by default for safety; use `let` only when necessary.**

8. Quick Recap & Key Takeaways

- `let` and `const` provide block scope and prevent accidental redeclaration.
- Use `let` for variables that may change, `const` for values that should never change.
- Both `let` and `const` are safer and more predictable than traditional variable declarations.
- Always initialize `const` variables and avoid reassigning them.

9. Optional: Programmer's Workflow Checklist

- Use `let` for variables that may change.
- Use `const` for variables that should never change.
- Never redeclare a variable with `let` or `const` in the same scope.
- Always initialize `const` variables.
- Keep variable scope as small as possible (prefer block scope).
- Test variable access inside and outside blocks to understand scope.