TypeScript Basics for Automation Testers – Day 3

Topic: Variables in TypeScript (Part 1 – Understanding Keywords)

What is a Variable?

A **variable** is a **container** that holds some data.

Example:

```
let x = 10;
let name = "Yogi";
```

Here:

- `x` and `name` are **variables (containers)**
- `10` and `"Yogi"` are **values** stored inside them

Note:
Always use meaningful variable names.
It's a good practice to start variable names in **lowercase** in TypeScript.

Keywords Used to Declare Variables

In **JavaScript** and **TypeScript**, we can declare variables using three keywords:

1. `var`
2. `let`
3. `const`

Before we go technical, let's understand the meaning.

Keyword	English Meaning	Description
var	"variable" → something that can change	Old way to declare variables, not preferred now
let	"let it change" → value can be updated	Used when variable value can change later
const	"constant" → fixed, does not change	Used when the value must stay the same

---

Declaring Variables

In Java, we must declare the data type when creating a variable.
In TypeScript and JavaScript, **data type is optional**.

Syntax:

```
keyword variableName: dataType(optional) = value;
```

Example:

```
let age: number = 28;
let name = "Yogi";
```

Here:

- `number` is the **data type**
- In TypeScript, the `number` type is used for both integers and decimals.

Keywords in Detail (var, let, const)

We can differentiate them in **five ways**:

1. **Scope of the variable**
2. **Declaration (value assignment)**
3. **Re-declaration**
4. **Re-initialization / Re-assignment**
5. **Hoisting**

Note:
`var` is **not recommended** in modern JavaScript or TypeScript because it has **function scope**, while `let` and `const` have **block scope**.

---

1. What is Functional Scope?

Meaning of "function":
A function is like a **small task** or **block of work** that performs a specific action.

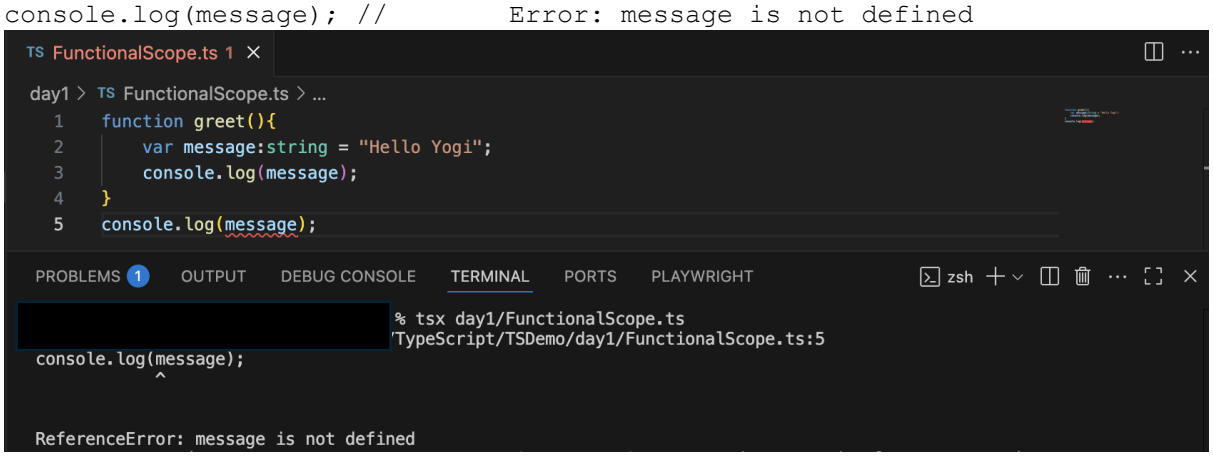
Example in real life:
Imagine a washing machine — it has a *wash function* that only works inside that machine. Anything outside can't access it.

Functional Scope (in programming):
If a variable is declared using `var` inside a function, it can be accessed **anywhere inside that function**, but **not outside** it.

Example:

```
function greet() {
  var message = "Hello Yogi";
  console.log(message); // Accessible
}
```

```
console.log(message); //        Error: message is not defined
```



Additional Note:
If you notice, the first console.log() is not printed in the console because it is written inside a function.
Anything declared inside a function will only execute when the function is called.

---

2. What is Block Scope?

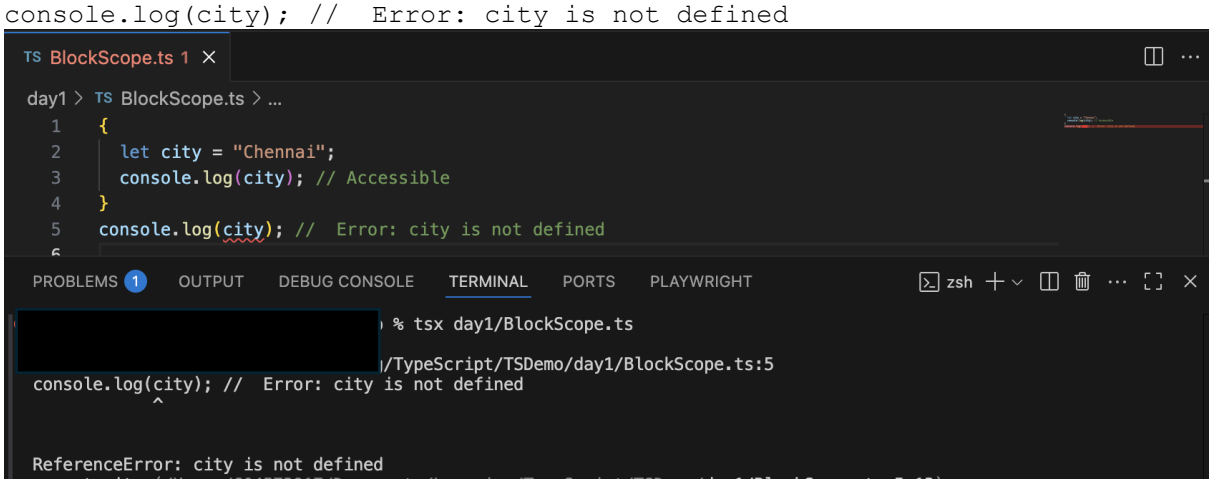
Meaning of "block":
A block means **a closed section**, something surrounded by `{ }`.

Example in real life:
Think of a **room** inside a house. Whatever is inside the room (block) is private to that room.

Block Scope (in programming):
Variables declared using `let` or `const` are only accessible **inside the block** where they are defined.

Example:

```
{
  let city = "Chennai";
  console.log(city); // Accessible
}
console.log(city); //  Error: city is not defined
```



Additional Note:
If you notice in the block scope example, the first console.log() inside the block is printed, but the one outside the block gives an error.
This is because a variable declared inside a block { } is accessible only within that block, not outside it.

Why Block Scope is Preferred

Block scope helps prevent accidental overwriting of variables.
In large projects (like automation frameworks), this avoids confusing variable conflicts.
That's why we recommend using **`let`** or **`const`** instead of **`var`**.

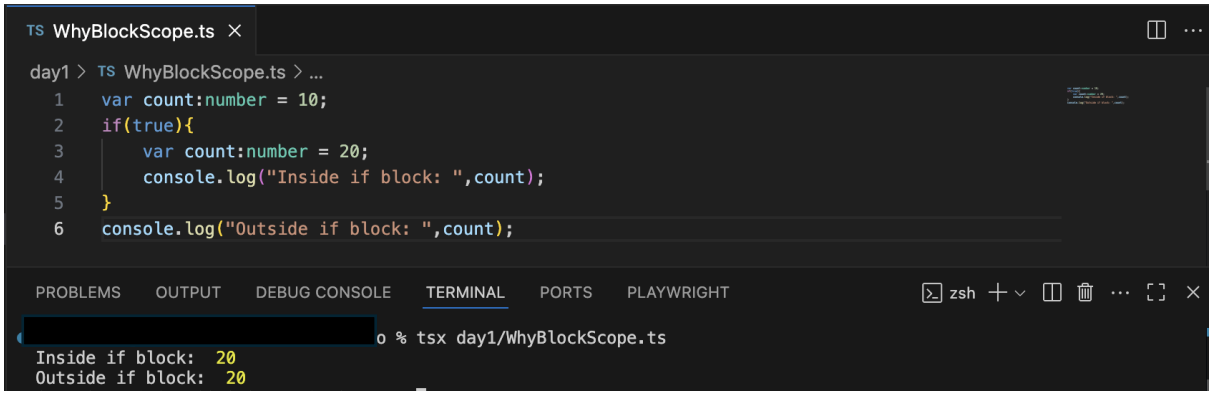
Example:

```
var count = 10;

if (true) {
  var count = 20; // Same variable (function scoped)
  console.log("Inside if block:", count);
}

console.log("Outside if block:", count);
```

Output:

```
Inside if block: 20
Outside if block: 20
```

Here, the value of `count` changed everywhere because `var` is **function scoped** — the variable is shared between blocks.



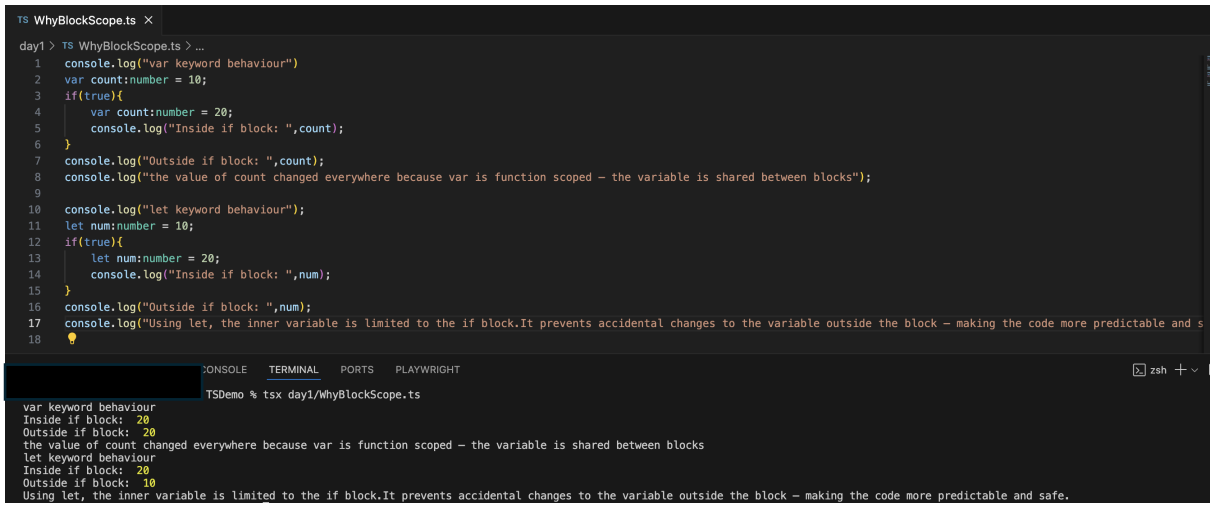
Now see the same example using `let`:

```
let count = 10;

if (true) {
  let count = 20; // Different variable (block scoped)
  console.log("Inside if block:", count);
}
```

```
console.log("Outside if block:", count);
```

Output:

```
Inside if block: 20
Outside if block: 10
```

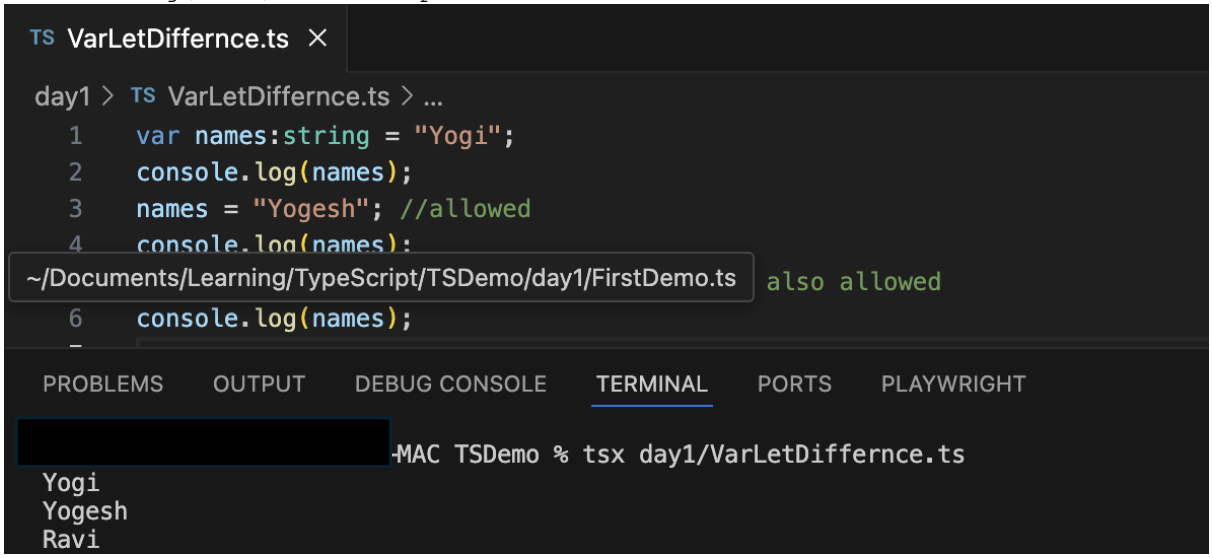


Explanation:
Using `let`, the inner variable is limited to the `if` block.
It prevents accidental changes to the variable outside the block — making the code more predictable and safe.

VAR

- Not recommended in modern TypeScript/JavaScript.
- Has **functional scope**, which can lead to confusion.
  Example:

```
var name = "Yogi";
name = "Yogesh"; //Allowed
console.log(name); //output: Yogesh
var name = "Ravi";  // Re-declaration also allowed
console.log(name);  // Output: Ravi
```

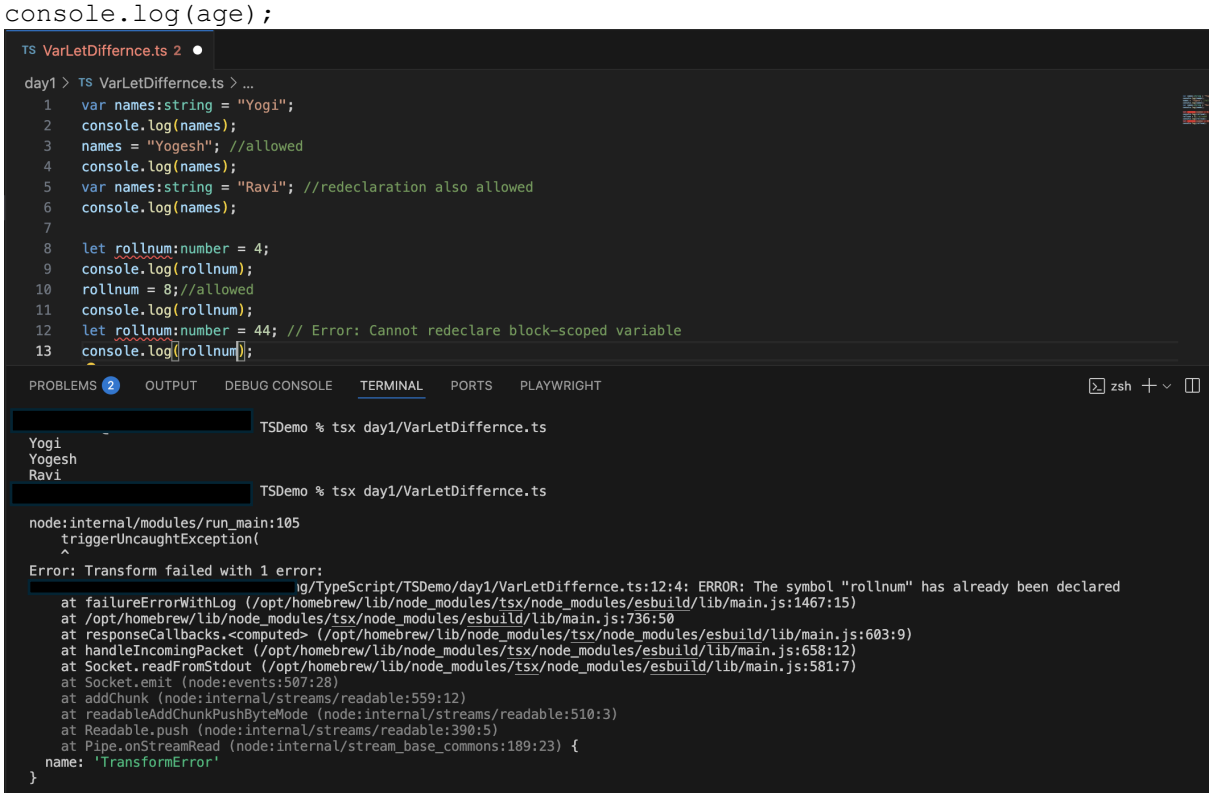


LET

- Use `let` when the variable value may **change**.
- Does **not** allow re-declaration, but allows **re-assignment**.

Example:

```
let age = 28;
age = 29;  //  Allowed
console.log(age) //29
let age = 30; //  Error: Cannot redeclare block-scoped variable
console.log(age);
```

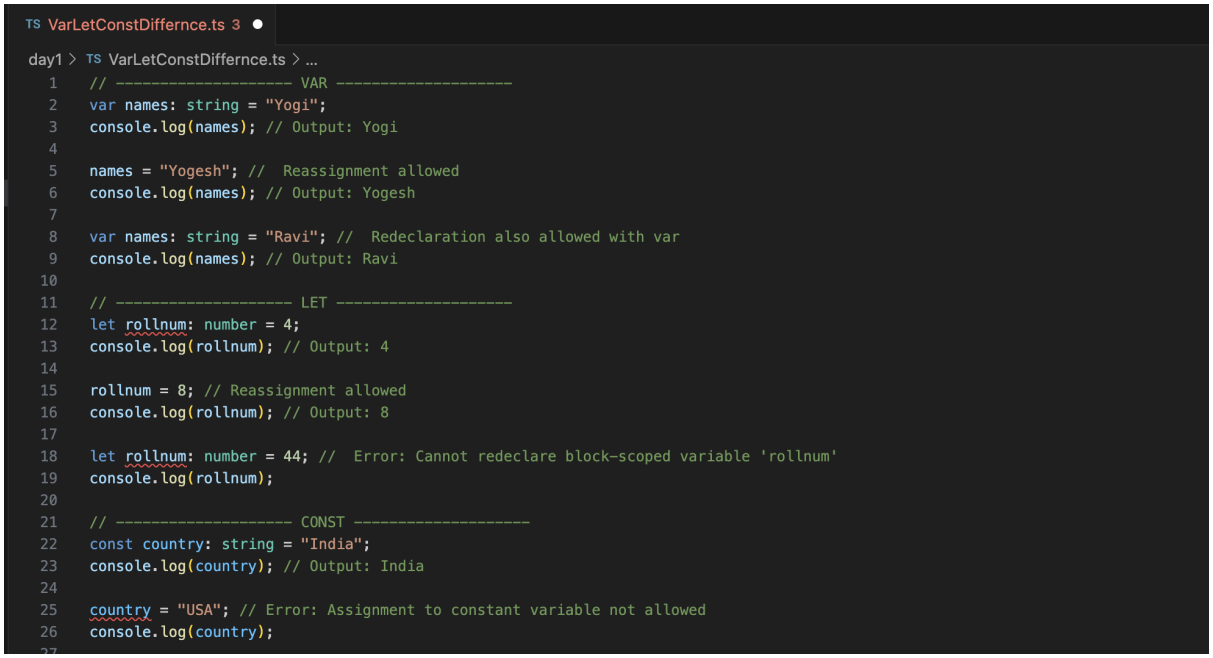


CONST

- Use `const` when the value **must not change**.
- Does **not** allow re-declaration or re-assignment.

Example:

```
const country = "India";
country = "USA";  //  Error: Assignment to constant variable
console.log(country);
```



Practice

1. Create a new file named `VariableDemo.ts`.
2. Try examples using `var`, `let`, and `const`.
3. Compile using:

```
tsc VariableDemo.ts
node VariableDemo.js
```

4. Observe which ones throw errors and note the reason.

Questions

1. What are the three keywords used to declare variables in TypeScript?
2. What is the difference between `var`, `let`, and `const`?
3. What is meant by functional scope?
4. What is meant by block scope?
5. Why is `var` not recommended in modern TypeScript?
6. Which keyword allows re-assignment but not re-declaration?
7. Which keyword is used when the variable value should not change?
8. Why is block scope preferred over function scope?

Answers

1. `var`, `let`, and `const`
2. `var` – function-scoped, `let` – block-scoped (can change), `const` – block-scoped (cannot change)
3. Functional scope means the variable is available only inside the function where it is declared.
4. Block scope means the variable is available only inside the `{ }` block where it is declared.
5. Because `var` can cause conflicts by allowing re-declaration and has no block control.
6. `let`
7. `const`
8. Block scope keeps variables safer by limiting access and preventing accidental overwrites.
