Variables in TypeScript

1. Problem Statement

Imagine you're developing a software system for a large company.

- You need to store data like names, scores, and settings in a way that is clear, safe, and organized.
- Every piece of information in your program must have a name and a place in memory.
- You want to avoid mistakes caused by misnaming, reusing, or misusing data.

The challenge:
How do you declare, use, and manage variables in TypeScript so your code is robust, readable, and error-free?

2. Learning Objectives

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

- Understand how to declare and use variables in TypeScript.
- Apply naming rules and best practices for variable names.
- Use type annotations and type inference.
- Understand variable scope (global, class, local).
- Use type assertions and recognize strong typing.

3. Concept Introduction with Analogy

Analogy: Labeled Drawers in an Organized Office

Think of variables as labeled drawers in an office:

- Each drawer has a unique label (name) and holds one kind of item (value).
- You must follow rules for labeling drawers so everyone can find what they need.
- Some drawers are accessible everywhere (global), some only in certain rooms (local), and some belong to specific team members (class scope).

Organizing your office this way prevents confusion and lost items-just like using variables correctly in TypeScript prevents bugs and errors.

4. Technical Deep Dive

Naming Rules for Variables

- Names can include letters, numbers, `_`, and `$`.
- Cannot start with a number.
- Cannot include spaces or most special characters.
- Cannot be a reserved keyword.
- Are case-sensitive.

Examples:

Valid	Invalid
firstName	first name
first_name	1number
num1	first-name
$result	Var

Declaring Variables in TypeScript

You can declare variables in several ways:

1. **Declare type and value:**

```
var name:  string  =  "mary";
```

The variable stores a value of type string.

2. **Declare type, no value:**

```
var name:  string;
```

The variable is a string, set to `undefined` by default.

3. **Declare value, no type:**

```
var name =  "mary";
```

The type is inferred from the value (`string`).

4. **Declare neither value nor type:**

```
var name;
```

The type is `any` and the value is `undefined` by default.

Example: Variables in TypeScript

```
var name: string = "John";
var score1: number = 50;
var score2: number = 42.50;
var sum = score1 + score2;
console.log("name: " + name);
console.log("first score: " + score1);
console.log("second score: " + score2);
console.log("sum of the scores: " + sum);
```

Output:

```
name: John
first score: 50
second score: 42.5
sum of the scores: 92.5
```

Strong Typing

TypeScript enforces strong typing.
If you try to assign a value of the wrong type, you get a compilation error:

```
var num:  number  =  "hello";  // Compilation error!
```

Type Assertion

TypeScript allows you to tell the compiler to treat a variable as a different type (type assertion).

Syntax:

```
var str = '1';
var str2: number = <number><any>str;
console.log(typeof(str2));
```

- This tells TypeScript to treat `str` as a `number` (even though it's actually a string).
- Type assertions are compile-time only and do not change the runtime type.

Inferred Typing

If you declare a variable without a type, TypeScript infers the type from the first value you assign:

```
var num = 2; // inferred as number
console.log("value of num " + num);
num = "12"; // Error: Cannot assign string to number
```

Variable Scope

- **Global Scope:** Declared outside any class or function; accessible everywhere.
- **Class Scope (Field):** Declared in a class, outside methods; accessible via the object.
- **Local Scope:** Declared inside a method or block; accessible only there.

Example:

```
var global_num = 12; // global variable
class Numbers {
   num_val = 13; // class variable
   static sval = 10; // static field

   storeNum(): void {
      var local_num = 14; // local variable
   }
}
console.log("Global num: " + global_num);
console.log(Numbers.sval); // static variable
var obj = new Numbers();
console.log("Class num: " + obj.num_val);
```

Output:

```
Global num: 12
10
Class num: 13
```

- Accessing `local_num` outside its method will cause an error.

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

1. **Declare variables with type and value:**

   ```
   var productName: string = "Bananas";
   var quantity: number = 25;
   ```

2. **Declare a variable with type only:**

```
var city:  string; city =  "Mumbai";`
```

3. **Declare a variable with value only (type inferred):**

```
var inStock =  true;  // inferred as boolean`
```

4. **Declare a variable with no type or value:**

```
var discount;
discount = 10; // type is any
discount = "none"; // still allowed
```

5. **Type assertion:**

```
var code = "123";
var codeNumber: number = <number><any>code;
```

6. **Variable scope:**

```
var globalVar = "I am global";
class Store {
  storeName = "Main Branch";
  static storeType = "Grocery";
  show() {
    var localVar = "I am local";
    console.log(localVar);
  }
}
```

6. Interactive Challenge

Your Turn!

- Declare a variable called `city` and assign it your favorite city as a string.
- Declare a variable called `temperature` with type `number` and assign it a value.
- Create a variable called `isRaining` and let TypeScript infer its type from the value you assign.

- Write a function called `weatherReport` that takes `city`, `temperature`, and `isRaining` as parameters and prints a message like:
  `"In <city>, it is <temperature>°C. Is it raining? <true/false>"`
- Try calling the function with your variables.

7. Common Pitfalls & Best Practices

- **Never start variable names with a number or use spaces.**
- **Use type annotations for clarity and safety.**
- **Don’t use reserved keywords as variable names.**
- **Keep variable scope as small as possible to avoid bugs.**
- **Use type assertions only when you are sure of the type.**

8. Quick Recap & Key Takeaways

- Variables are named storage locations in memory.
- TypeScript enforces naming rules and strong typing.
- You can declare variables with type, value, both, or neither.
- Variable scope controls where a variable can be accessed.
- Type assertion lets you tell the compiler how to treat a variable’s type.

9. Optional: Programmer’s Workflow Checklist

- Use valid, descriptive variable names.
- Declare types for variables when possible.
- Assign values that match the declared type.

- Understand and control variable scope.
- Use type assertions responsibly.
- Avoid using `any` unless absolutely necessary.
- Test variable assignments and observe compiler errors.