TypeScript Data Types

Dynamically Typed Vs Statically Typed Languages:

JavaScript is a Dynamically Typed Language

- In JavaScript, variable types are checked **at runtime**, and you can change the type of a variable later.
- **Example:**

```
let age = 25;       // age is a number
age = "twenty-five";   // Now age is a string (no error)
console.log(age);     // Output: "twenty-five"
```

✅ No errors because JavaScript allows type changes dynamically.

TypeScript is a Statically Typed Language

- In TypeScript, variable types are checked **at compile time**, and you **cannot** change the type later.
- **Example:**

```
let data:number = 10;  // data is a number
data = "ten";    // Error: Type 'string' is not assignable to type 'number'
```

❌ TypeScript catches this error before the code runs.

Type-Safety

JavaScript is Not Type-Safe

- JavaScript allows operations between incompatible types, leading to unexpected behaviour.
- **Example:**

```
const result = "5" + 3; // JavaScript converts 3 to string
console.log(result);    // Output: "53" (not 8)
```

❌ No error, but the result may not be what you intended.

TypeScript is Type-Safe

- TypeScript prevents operations between incompatible types.
- **Example:**

```
const result: number = "5" + 3;  //Error: Type 'string' is not assignable to type 'number'
```

✅ TypeScript warns you about potential bugs early.

Key Takeaways:

- **Dynamic Typing (JS):** Types are flexible (checked at runtime).
- **Static Typing (TS):** Types are fixed (checked at compile time).
- **Type Safety (TS):** Prevents operations with wrong types, reducing bugs.

TypeScript Types, Annotations & Type Inference

1. TypeScript Types (Data Types)

- **Definition:** Built-in or custom categories for variables (e.g., number, string, boolean).
- **Example:**

```
let isDone: boolean = true;  // `boolean` is a TypeScript type
let score: number = 100;     // `number` is a TypeScript type
```

2. Type Annotations

- **Definition:** Explicitly telling TypeScript the type of a variable using : type.
- **Example:**

```
let name: string = "Alice";  // `: string` is a type annotation
let age: number = 30;        // `: number` is a type annotation
```

3. Type Inference

- **Definition:** TypeScript *automatically* guesses the type if you don't annotate it.
- **Example:**

```
let message = "Hello";      // TypeScript infers `message` as `string`
let count = 42;             // TypeScript infers `count` as `number`
// message = 123;     ❌ Error (TypeScript knows `message` must stay a string)
```

Key Differences:

- **Type Annotation:** You manually define the type.
- **Type Inference:** TypeScript intelligently figures it out.

TypeScript Data Types

Primitive Types (Built-in Types)

1. Number

- Represents **both integers and decimals** (e.g., 42, 3.14).

2. String

- Represents **text data**.
- Can be written in:
    - Single quotes ('Hello')
    - Double quotes ("Hello")
    - Backticks (`Hello ${name}`) for **template literals**.

3. Boolean

- Represents **true or false** values.
- Used in **conditions** (e.g., if (isLoggedIn) {...}).

4. Null

- Represents an **intentional empty value** (let x = null).

5. Undefined

- Represents a **variable declared but not assigned** (let y; → y is undefined).

6. Any

- A **flexible type** that allows **any value** (disables TypeScript checks).
- Avoid using unless necessary (let z: any = "Hello"; z = 10;).

7. Union Type

- Allows **multiple types** for a variable (let id: string | number = "123";).

8. Void

- Used for **functions that don't return anything** (function log(): void { console.log("Hi"); }).

Non-Primitive Types (Objects & Custom Types)

1. Array
2. Tuple
3. Class
4. Functions
5. Interface

Key Takeaways:

✓ **Primitive types** (number, string, boolean, etc.) are **basic** and hold single values.

✓ **Non-primitive types** (Array, Class, Interface, etc.) are **complex** and hold structured data.

✓ **Avoid any** when possible—use **proper types** for better code safety.

✓ **Union types (|)** allow flexibility when a variable can be multiple types.