TypeScript

Data Types

TypeScript: Overview

TypeScript is a strongly typed superset of JavaScript that adds optional static typing to the language.

It provides several built-in data types that you can use to define variables, function parameters, and return types.

Let’s dive into the basic data types in TypeScript with simple examples.

Fathima Samila
@samila99

1. number:

A number type can represent both integers and floating-point numbers.

```
let age: number = 25;
let height: number = 5.9;
console.log(age, height);  // Output: 25 5.9
```

2. string:

A string type represents a sequence of characters.

```
let name: string = "MissTechie";
console.log(name);  // Output: MissTechie
```

Fathima Samila
@samila99

3. boolean:

A boolean type represents true or false.

```
let isActive: boolean = true;
let isCompleted: boolean = false;
console.log(isActive, isCompleted);  // Output: true false
```

4. array:

An array in TypeScript can be declared using type[] or Array<type>.

```
let numbers: number[] = [1, 2, 3, 4, 5];
let fruits: Array<string> = ["apple", "banana", "cherry"];
console.log(numbers);  // Output: [1, 2, 3, 4, 5]
console.log(fruits);   // Output: ['apple', 'banana', 'cherry']
```

Fathima Samila
@samila99

5. tuple:

A tuple is an ordered collection of values of different types.

```
let person: [string, number] = ["Alice", 30];
console.log(person);  // Output: ["Alice", 30]
```

6. enum:

An enum is a way of defining a set of named constants.

```
enum Color { Red = "RED", Green = "GREEN"}
let myColor: Color = Color.Green;
console.log(myColor);  // Output: GREEN
```

Fathima Samila
@samila99

7. any:

The any type allows a variable to be of any type, essentially bypassing type checking.

```
let something: any = 42;
something = "Hello";  // Valid
something = true;     // Valid
console.log(something);  // Output: true
```

8. void:

The void type is used when a function doesn't return anything.

```
function greet(name: string): void {
    console.log(`Hello, ${name}`);
}
greet("Alice");  // Output: Hello, Alice
```

Fathima Samila
@samila99

9. null and undefined:

In TypeScript, both null and undefined are distinct types.

```
let nothing: null = null;
let unknown: undefined = undefined;
console.log(nothing, unknown);  // Output: null undefined
```

10. object:

Represents a non-primitive type (anything that is not number, string, boolean, etc.).

```
let person: object = { name: "Alice", age: 30 };
console.log(person);  // Output: { name: "Alice", age: 30 }
```

Fathima Samila
@samila99

11. unknown:

Represents a value that could be anything, but requires type checking before use.

```
let userInput: unknown;
userInput = "Hello";
if (typeof userInput === "string") {
  console.log(userInput.toUpperCase()); // Safe to use
}
```

12. never:

Represents a value that never occurs, often used for functions that throw errors or never return.

```
function throwError(message: string): never {
    throw new Error(message);
}
```

Fathima Samila
@samila99

13. Union Types:

Allows you to specify exact values a variable can hold.

```
let direction: "left" | "right" | "up" | "down";
direction = "left"; // Valid
direction = "diagonal"; // Error
```

14. Literal Types:

Allows you to create a custom name for a type.

```
type StringOrNumber = string | number;
let value: StringOrNumber;
value = "Hello"; // Valid
value = 42; // Valid
```

Fathima Samila
@samila99