# TypeScript

## Basic Types

## Content

## 1. Type Annotation

- The following syntax shows how to specify type annotations for variables and constants

```
let variableName: type;

let variableName: type = value;

const constantName: type = value;
```

```
let counter: number;
```

```
counter = 1;
```

```
let name: string = 'John';

let age: number = 25;

let active: boolean = true;
```

## 2. Number

```
let counter: number = 0;
let x: number = 100,
    y: number = 200;
```

```
let price = 9.95;
```

- **Binary Number:** leading zero by letter b or B

```
let bin = 0b100;
let anotherBin: number = 0B010;
```

- **Hexadecimal number:** leading zero by letter x or X

```
let hexadecimal: number = 0XA;
```

- **Big Integer:** end with letter n

```
let big: bigint = 9007199254740991n;
```

## 3. String

- TypeScript uses double quotes (") or single quotes (') to surround string literals.

```
let firstName: string = 'John';

let title: string = "Web Developer";
```

- Multi-line string using the backtick (`)

```
let description = `This TypeScript string can

span multiple

lines

`;
```

- String interpolations allow you to embed the variables into the string

```
let firstName: string = `John`;
let title: string = `Web Developer`;
let profile: string = `I'm ${firstName}.
I'm a ${title}`;

console.log(profile);
```

- Output

```
I'm John.
I'm a Web Developer.
```

## 4. Boolean

- boolean type has two values: true and false. The boolean type is one of the primitive types in TypeScript.

```
let pending: boolean;
pending = true;
```

- **Boolean operator**

| Operator | Meaning |
|---|---|
| AND | && |
| OR | \|\| |
| NOT | ! |

```
const pending: boolean = true;
const notPending = !pending; // false
```

```
const hasError: boolean = false;
const completed: boolean = true;
```

```
let result = completed && hasError;
```

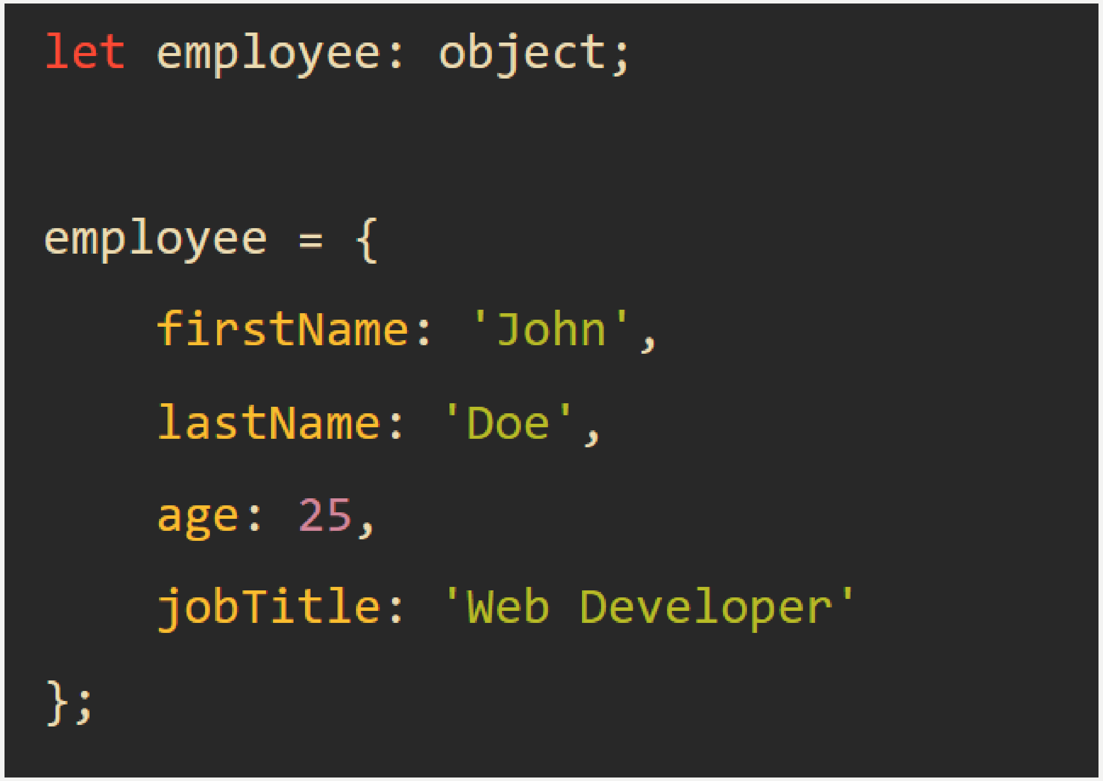
```
result = completed || hasError;
```

## 5. Object Type

- object type represents all values that are not in primitive types. The following are primitive types in TypeScript

  number bigint string boolean

  null undefined symbol



```
let employee: object;

employee = {
    firstName: 'John',
    lastName: 'Doe',
    age: 25,
    jobTitle: 'Web Developer'
};
```

## 6. Array

- array is an ordered list of data. To declare an array that holds values of a specific type.

```
let skills: string[];
```

```
skills[0] = "Problem Solving";
skills[1] = "Programming";
```

```
skills.push('Software Design');
```

```
let skills = ['Problem Sovling','Software Design','Programming'];
```

```
let skills: string[];
skills = ['Problem Sovling','Software Design','Programming'];
```

- Storing values of mixed types

```
let scores = ['Programming', 5, 'Software Design', 4];
```

## 7. Tuple

- A tuple works like an array with some additional considerations:
  - The number of elements in the tuple is fixed.
  - The types of elements are known, and need not be the same.

```
let skill: [string, number];
skill = ['Programming', 5];
```

- Optional Tuple Element

```
let bgColor, headerColor: [number, number, number, number?];
bgColor = [0, 255, 255, 0.5];
headerColor = [0, 255, 255];
```

## 8. Enum

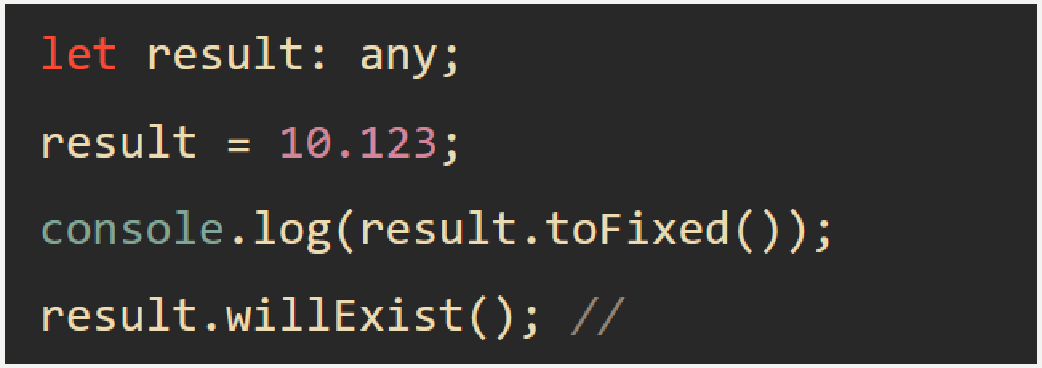
- An enum is a group of named constant values. Enum stands for enumerated type. To define an enum, you follow these steps:
  - First, use the enum keyword followed by the name of the enum.
  - Then, define constant values for the enum.

```
enum name {constant1, constant2, ...};
```

```
enum ApprovalStatus {
    draft,
    submitted,
    approved,
    rejected
};
```

## 9. Any Type

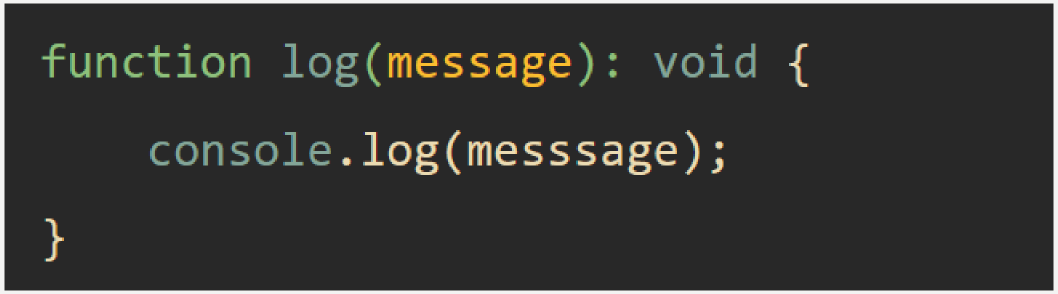
- Sometimes, you may need to store a value in a variable. But you don’t know its type at the time of writing the program. And the unknown value may come from a third-party API or user input



```
let result: any;
result = 10.123;
console.log(result.toFixed());
result.willExist(); //
```

## 10. Void Type

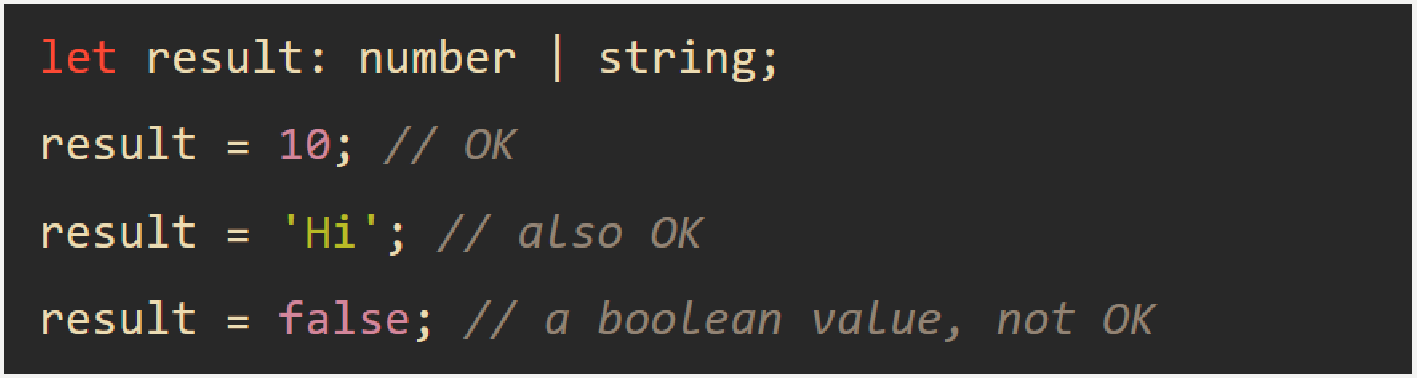
- The void type denotes the absence of having any type at all. It is a little like the opposite of the any type



```
function log(message): void {
    console.log(messsage);
}
```

## 11. Union Type

- union type that allows you to store a value of one or several types in a variable.



```typescript
let result: number | string;
result = 10; // OK
result = 'Hi'; // also OK
result = false; // a boolean value, not OK
```

## 12. Type Alias

- Type aliases allow you to create a new name for an existing type.

```
type chars = string;
let messsage: chars; // same as string type
```

```
type alphanumeric = string | number;
let input: alphanumeric;
input = 100; // valid
input = 'Hi'; // valid
input = false; // Compiler error
```

## 13. String Literal Type

- String literal types that define a type that accepts a specified string literal.

```
let mouseEvent: 'click' | 'dblclick' | 'mouseup' | 'mousedown';
mouseEvent = 'click'; // valid
mouseEvent = 'dblclick'; // valid
mouseEvent = 'mouseup'; // valid
mouseEvent = 'mousedown'; // valid
mouseEvent = 'mouseover'; // compiler error
```

# THE END