**Variable declaration in TypeScript**

TypeScript variables also follow JavaScript variable naming conventions. Similar to JavaScript, we use var, let, and const keywords to declare variables.

Since TypeScript is a **strongly typed** programing language, we specify the type when we declare a variable.

The Syntax for declaring a variable in TypeScript - let [identifier] : [type] = value; .

**Example:**

let myString: string;

myString = 'This is a string';

myString =5; // Error: Type '5' is not assignable to type 'string'.

**Datatypes in TypeScript**

The basic datatypes used in Typescript are:

**Boolean**

The boolean type is to represent a true or false value.

let isEmpty: boolean = true;

**Number**

TypeScript uses the number type to store decimal, hexademial, binary and octal literals in a variable.

let a: number = 6;

let b: number = 0xf00d; //hexadecimal

let c: number = 0b1010; //binary

let d: number = 0o744; //octal

**String**

TypeScript uses a string type that represents text characters enclosed within double quotes (") or single quotes (').

let color: string = "yellow";

color = 'blue';

We can also use template strings, which can span multiple lines and have embedded expressions.

let fullName: string = 'Bob Bobbington';

let age: number = 37;

let sentence: string = `Hello, my name is ${ fullName }.

I'm ${ age } years old.`;

**Undefined and Null**

In TypeScript, both undefined and null have their own types named undefined and null respectively. By default, null and undefined are subtypes of all other types. So we can assign null and undefined to number, string, etc.

let u: undefined = undefined;

let n: null = null;

**Any**

We may need to describe a type of a variable that we do not know when we are writing an application. In this case, we can label the variable with the any type.

let variable: any = 14;

variable = "maybe a string instead";

variable = false;

**Void**

In TypeScript, the void type is used as the return type of functions that do not return a value.

function sayHello(): void {

console.log("Hello!!!");

}

Declaring variables of type void is not useful, because we can then only assign null or undefined values to them.

**Arrays**

Like in JavaScript, TypeScript allows you to work with arrays of values. We can use the type of the elements followed by [] or use a generic array typeArray<elemType> for declaring the array type.

let list: number[] = [1, 2, 3];

let list: Array<number> = [1, 2, 3];

let list: any[] = [1, "two", 3];

**Tuples**

TypeScript introduced a new type called Tuple, which is an array with a fixed number of elements whose types are known.

var employee: [number, string] = [1, "John"];

//Accessing Tuple

employee[0]; // returns 1

employee[1]; // returns "John"

var person: [number, string, boolean] = [1, "Steve", true];

// tuple array

var employee: [number, string][];

employee = [[1, "John"], [2, "Adam"], [3, "Jeff"]];

**Enum**

TypeScript supports the enum type that allows us to declare a set of named constants. It is a collection of related values that can be numeric or string values.

enum Color {Red, Green, Blue}

let c: Color = Color.Green;

enum Color {Red = 1, Green, Blue}

let colorName: string = Color[2];

console.log(colorName); // Displays 'Green' as its value is 2 above

**Never**

The never type represents a type of values that never occur. For instance, never is the return type for a function that always throws an exception or one that never returns.

// Function returning never must have unreachable end point

function error(message: string): never {

throw new Error(message);

}

[TypeScript: Handbook - Basic Types (typescriptlang.org)](https://www.typescriptlang.org/docs/handbook/basic-types.html)