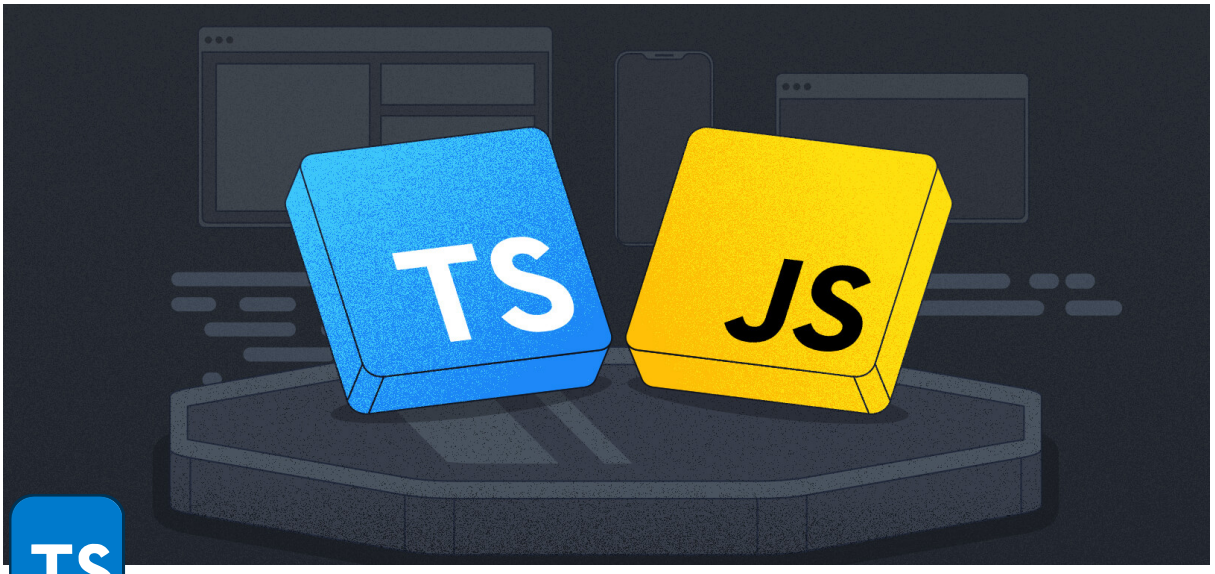



# Typescript Document

## What is TypeScript?

TypeScript is an open-source pure object-oriented programming language. It is a strongly typed superset of JavaScript which compiles to plain JavaScript. It contains all elements of JavaScript. It is a language designed for large-scale JavaScript application development, which can be executed on any browser, any Host, and any Operating System. TypeScript is a language as well as a set of tools. TypeScript is the ES6 version of JavaScript with some additional features.

TypeScript files use the `.ts` extension rather than the `.js` extension of JavaScript files.


# Why use TypeScript?

We use TypeScript because of the following benefits.

- TypeScript supports Static typing, Strong typing, Modules, Optional Parameters, etc.
- TypeScript supports object-oriented programming features such as classes, interfaces, inheritance, generics, etc.
- TypeScript is fast, simple, and most importantly, easy to learn.
- TypeScript provides the error-checking feature at compilation time. It will compile the code, and if any error is found, then it highlights the mistakes before the script is run.
- TypeScript supports all JavaScript libraries because it is the superset of JavaScript.
- TypeScript supports reusability because of the inheritance.
- TypeScript makes app development as quick and easy as possible, and the tooling support of TypeScript gives us autocompletion, type checking, and source documentation.
- TypeScript has a definition file with .d.ts extension to define external JavaScript libraries.

- TypeScript supports the latest JavaScript features, including ECMAScript 2015.
- TypeScript gives all the benefits of ES6 plus more productivity.
- Developers can save a lot of time with TypeScript.

## Advantage of TypeScript over JavaScript

- TypeScript always highlights errors at compilation time during development, whereas JavaScript points out errors at runtime.
- TypeScript supports strongly typed or static typing, whereas this is not in JavaScript.
- TypeScript runs on any browser or JavaScript engine.
- Great tooling supports with IntelliSense which provides active hints as the code is added.
- It has a namespace concept by defining a module.

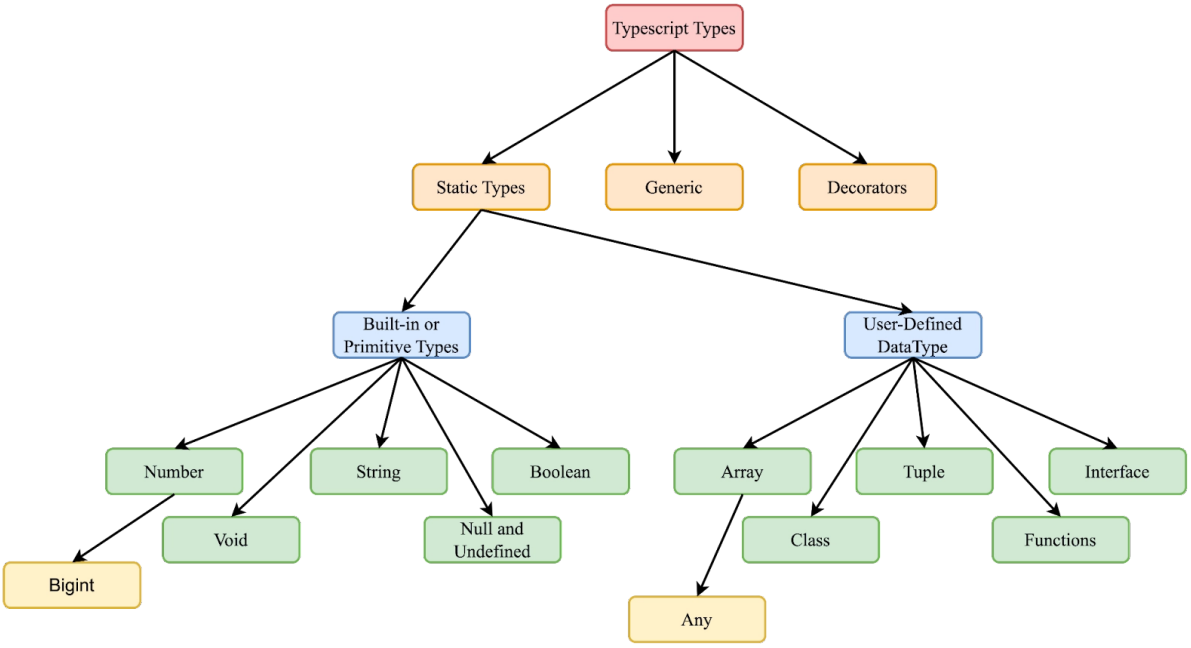
## Disadvantage of TypeScript over JavaScript

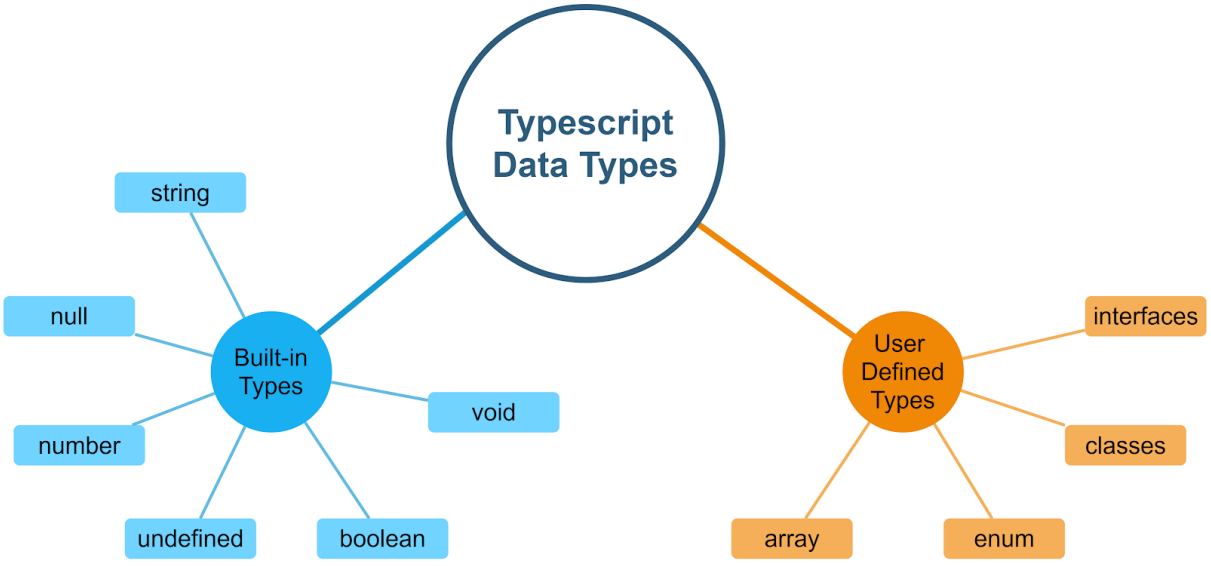
- TypeScript takes a long time to compile the code.
- TypeScript does not support abstract classes.
- If we run the TypeScript application in the browser, a compilation step is required to transform TypeScript into JavaScript.

# TypeScript Type

The TypeScript language supports different types of values. It provides data types for JavaScript to transform it into a strongly typed programming language. JavaScript doesn't support data types, but with the help of TypeScript, we can use the data types feature in JavaScript. TypeScript plays an

important role when the object-oriented programmer wants to use the type feature in any scripting language or object-oriented programming language. The Type System checks the validity of the given values before the program uses them. It ensures that the code behaves as expected.




# Built-In Type:

| Name | Description |
| --- | --- |
| string | Represents text data |
| number | Represents numeric values |
| boolean | Has true and false values |
| null | Has one value: null |
| undefined | has one value: undefined. It is the default value of an uninitialized variable |
| symbol | represents a unique constant value |

# User-Defined Type:

### Array:-

An array is a collection of elements of the same data type. Like JavaScript, TypeScript also allows us to work with arrays of values.

```
var list : number[] = [1, 3, 5];
var list : Array<number> = [1, 3, 5];
```

### Touple:-

The Touple is a data type that includes two sets of values of different data types. It allows us to express an array where the type of a fixed number of elements is known, but they are not the same. For example, if we want to represent a value as a pair of a number and a string.

```
let a: [string, number]; // Declare a touple
a = ["hi", 8, "how", 5]; // Initialize it
```

### Functions:-

A function is the logical block of code to organize the program. Like JavaScript, TypeScript can also be used to create functions either as a named function or as an anonymous function. Functions ensure that our program is readable, maintainable, and reusable. A function declaration has a function's name, return type, and parameters.

```
function add(a: number, b: number): number {
                return a + b;
}
```

## Interface:-

An Interface is a structure that acts as a contract in our application. It defines the syntax for classes to follow, meaning a class that implements an interface is bound to implement all its members. It cannot be instantiated but can be referenced by the class which implements it. The TypeScript compiler uses an interface for type-checking that is also known as "duck typing" or "structural subtyping."

```
interface Calc {
    subtract (first: number, second: number): any;
}

let Calculator: Calc = {
        subtract(first: number, second: number) {
                        return first - second;
        }
}
```

## Enums:-

Enums define a set of named constants. TypeScript provides both string-based and numeric-based enums. By default, enums begin numbering their elements starting from 0, but we can

also change this by manually setting the value to one of its elements. TypeScript gets support for enums from ES6.

```
enum color {
        Red, Green, Blue
};

let result = color.Green;
```

## Class:-

Classes are used to create reusable components and act as a template for creating objects. It is a logical entity that stores variables and functions to perform operations. TypeScript gets support for classes from ES6. It is different from the interface which has an implementation inside it, whereas an interface does not have any implementation inside it.

```
class Student{
            RollNo: number;
            Name: string;
            constructor(_rollNo: number, _name: string){
                        this.RollNo = _rollNo;
                        this.Name = _name;
                }
            showDetails(){
                        console.log(this.RollNo + " : " +
                }
}
```

## Generic:-

Generic is used to create a component that can work with a variety of data types rather than a single one. It allows a way to create reusable components. It ensures that the program

is flexible and scalable in the long term. TypeScript uses generics with the type variable **<T>** that denotes types. The type of generic functions is just like non-generic functions, with the type parameters listed first, similar to function declarations.

```
function identity<T>(arg: T): T {
                return arg;
        }

let output1 = identity<string>("myString");
let output2 = identity<number>( 100 );
```

## Decorators:-

A decorator is a special data type that can be attached to a class declaration, method, property, accessor, and parameter. It provides a way to add both annotations and a meta-programing syntax for classes and functions. It is used with the "@" symbol.

A decorator is an experimental feature that may change in future releases. To enable support for the decorator, we must enable the experimental Decorators compiler option either on the command line or in our **tsconfig.json**.

```
function f() {
        console.log("f(): evaluated");
        return function (target, propertyKey: string, descrip
                                console.log("f(): called");
            }
}

class C {
            @f()
            method() {}
}
```

**Prepared By**: Girish Gondaliya | Follow Me on LinkedIn