

TypeScript

by

Anand Kulkarni

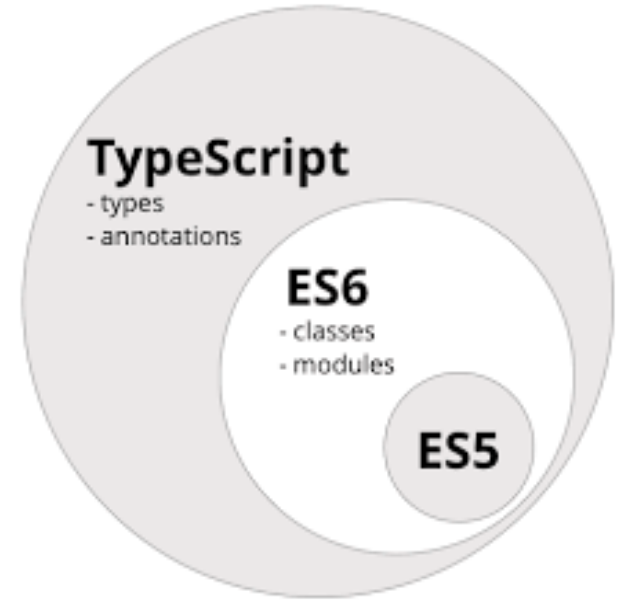
anand.kulkarni1@zensar.com

Table of Content

Module	Topic
Module 1:	Introduction to TypeScript
Module 2:	First TypeScript application
Module 3:	Built-in Types
Module 4:	Classes & Interfaces
Module 5:	Generics

TypeScript

- TypeScript is a free and open-source programming language developed and maintained by Microsoft.
- It is a superset of JavaScript, and adds optional static typing and class-based object-oriented programming to the language.
- The major feature supported in TypeScript which is not available in ES6 is 'types' & 'annotations'.
- You can try TypeScript code online at <https://www.typescriptlang.org/play/>



Writing TypeScript App

- Create a new directory, for example 'typescript_demo'.
- Change directory: `cd typescript_demo`
- Install typescript using npm: `npm install -g typescript`
- Write a .ts file, may be greeter.ts.
- Compile greeter.ts using tsc compiler: `tsc greeter.ts`
- Now write index.html & include greeter.js file into it.
- Finally open index.html inside browser.

Built-in Types in TypeScript

1. Boolean
2. Number
3. String
4. Array
5. Tuple
6. Enum
7. Any
8. Void
9. Null
10. Undefined

Using Built-in data types

TypeScript supports variables having specific type. Thus, when you create a variable, you may not only say 'var' or 'let' but you can specify its type like integer, string etc.

```
let isDone: boolean = false;
```

```
let decimal: number = 6;
```

```
let color: string = "blue";
```

```
let x: [string, number]; //tuple
```

```
x = ["hello", 10]; // OK
```

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

```
let c: Color = Color.Green;
```

Using Built-in data types

Any:

```
let notSure: any = 4;
```

```
notSure = "maybe a string instead";
```

```
notSure = false; // okay, definitely a Boolean
```

Void:

```
function warnUser(): void { alert("This is my warning message"); }
```

Null and Undefined:

```
let u: undefined = undefined;
```

```
let n: null = null;
```

Classes in TypeScript continue..

- TypeScript class contains fields, constructors & methods.
- You can have inheritance among classes using 'extends' keyword.
- However, TypeScript does not support multiple inheritance.

Classes in TypeScript

TypeScript class is similar to ES6 class, except we can specify the type of the fields the class is going to have.

```
class Car {  
  
    engine:string; //field  
  
    constructor(engine:string) {  
  
        this.engine = engine  
  
    }  
  
    disp():void {  
  
        console.log("Function displays Engine is : "+this.engine)  
  
    }  
}
```

Interfaces

An interface is a syntactical contract that an entity should conform to.

```
interface IPrintable {  
    max_elements: number;  
    print: ()=>void  
}  
  
let circle: IPrintable = {  
    max_elements: 100,  
    print:()=>console.log("Circle printed ", circle.max_elements)  
}  
  
circle.print();  
console.log("max elements = ", circle.max_elements);
```

Arrays with Interfaces

Interface can define both the kind of key an array uses and the type of entry it contains. Index can be of type string or type number.

```
interface namelist { [index:number]:string }
```

```
var list2:namelist = ["John",1,"Bran"] //Error. 1 is not type string
```

```
interface ages { [index:string]:number }
```

```
var agelist:ages;
```

```
agelist["John"] = 15 // Ok
```

```
agelist[2] = "nine" // Error
```

Multiple inheritance with Interfaces

```
interface IParent1 {
```

```
    v1:number
```

```
}
```

```
interface IParent2 {
```

```
    v2:number
```

```
}
```

```
interface Child extends IParent1, IParent2 { }
```

```
var lobj:Child = { v1:12, v2:23}
```

```
console.log("value 1: "+this.v1+" value 2: "+this.v2); //Output: value 1: 12 value 2: 23
```

Generics

Generics help us to create reusable components. It means, a component with generics are capable of working over variety of types rather than a single one.

Suppose there is a function that returns the same value as its argument:

```
function echo(arg: number): number { return arg; }
```

The 'echo' function will work only for number as input. If we want to use it for other types like string, array, boolean etc. then we should write argument type as 'Any'.

```
function echo(arg: any): any { return arg; }
```

However, function with 'any' will not really make it generic because we actually are losing the information about what that type was when the function returns.

Generics continue...

Let us write the 'echo()' function with generics support:

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

```
let output = echo<string>("myString"); // type of output will be 'string'.
```

Thank you!!