CREDO SYSTEMZ
simplifying IT

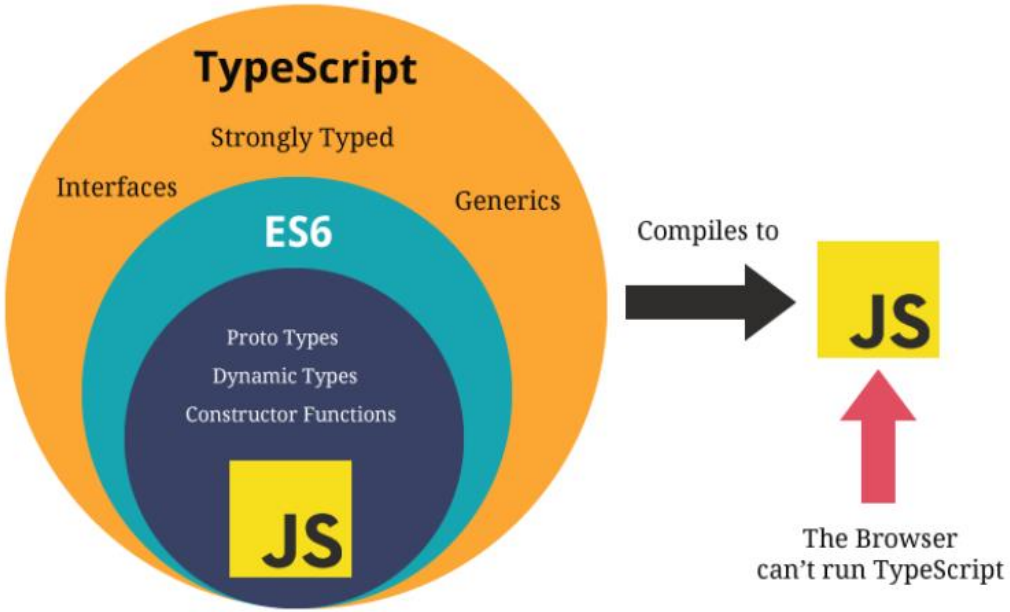
TypeScript - Training

Agenda

- Introduction to TypeScript
- Why TypeScript?
- TypeScript Compilation?
- Environment Setup - TypeScript
- Variables
- Functions
- Class
- Interfaces

What is TypeScript?

- TypeScript is wrapper around the JavaScript.
- TypeScript is Strongly Typed Language which means you have to be specific with Variable type.
- TypeScript doesn’t run in the browser. We need to compile it to JS to run.
- TypeScript is a typed superset of JavaScript and pure Object oriented.



Why TypeScript?

CREDO SYSTEMZ
simplifying IT

- Due to the static typing, code written in TypeScript is more predictable, and is generally easier to debug.
- Makes it easier to organize the code for very large and complicated apps.
- You can use TypeScript for other JS libraries, Because TS compiles plain JS.
- TS is platform Independent.

TypeScript Compilation

CREDO SYSTEMZ
simplifying IT

- The file extension of TypeScript is ".ts".
- The TSC (TypeScript Compiler) is a source-to-source compiler (transcompiler / transpiler).
- The TSC generates the equivalent JS code from the source TS code. This process is termed as transpilation.
- tsc first.ts This will create a new file named first.js in the same location. Keep in mind that if you already had a file named first.js, it would be overwritten.
- You can run multiple files at the time by using tsc first.ts second.ts third.ts

Environment Setup - TypeScript

CREDO SYSTEMZ
simplifying IT

- Install Node.js – Node.js is the environment on which you will run the TypeScript compiler. Note that you don't need to know node.js
- To verify if the installation was successful, enter the command ***node –v*** in the terminal window
- Type the following command in the terminal window to install TypeScript - **npm install -g typescript**
- Install Visual studio if not installed

Run TypeScript File

CREDO SYSTEMZ
simplifying IT

- Create a new file of your choice with file ext ".ts"
- Add a function in the typescript file like below

```
function add(a:number,b:number){
   return a+b;
}
let sum1: number = add(3,5)
console.log(sum1)
```

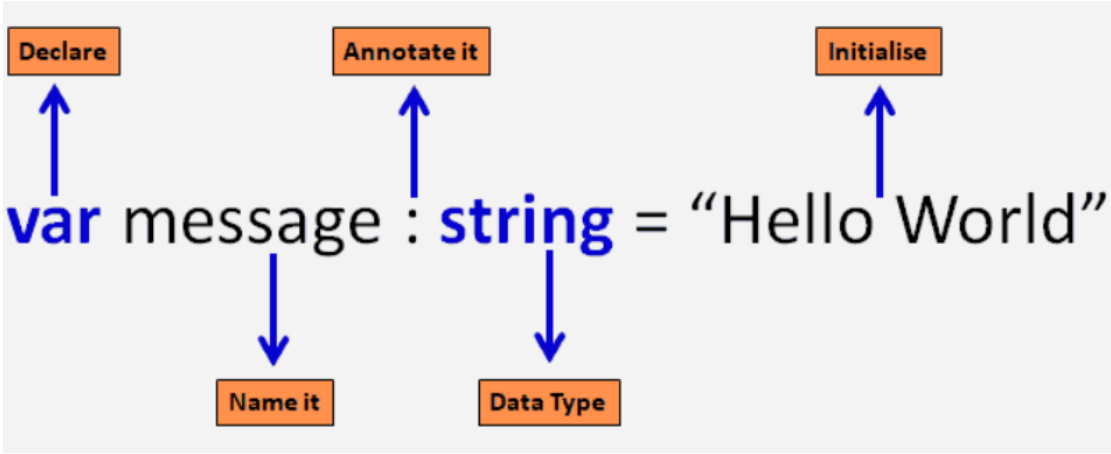
- Open command prompt on Windows and run this command - **tsc add.ts**
- The above command will compile the TypeScript file add.ts and create the Javascript file named add.js at the same location
- Refer add.js file in a web page using a script tag and see the result in the browser's developer console

Type Annotations

CREDO SYSTEMZ
simplifying IT

TypeScript is a typed language, we can specify the type

- Variables
- Function parameters
- Object properties



We cannot change the value using a different data type other than the declared data type of a variable.
If you try to do so, TypeScript compiler will show an error

- Type annotations are used to enforce type checking
- Type annotations helps compiler in checking types and helps avoid errors dealing with data types
- It is also a good way of writing code for easier readability and maintenance

Type Annotations

CREDO SYSTEMZ
simplifying IT

Variables:

```
var age: number = 32; // number variable
var name: string = "John";// string variable
var isUpdated: boolean = true;// Boolean variable
```

Type Annotation of Parameters:

```
function display(id:number, name:string)
{
   console.log("Id = " + id + ", Name = " + name);
}
```

Type Annotation in Object:

```
var employee : {
   id: number;
   name: string;
};

employee = {
  id: 100,
  name : "John"
}
```

If we try to assign a string value to id then the TypeScript compiler will give the following error.

Datatype

CREDO SYSTEMZ
simplifying IT

Number:

```
let first:number = 123;

let myNumber: number = 123;
myNumber.toString(); // returns '123'
```

```
let myNumber: number = 10.8788;

myNumber.toFixed(); // returns 11
myNumber.toFixed(1); //returns 10.9
myNumber.toFixed(2); //returns 10.88
```

String:

```
let employeeName:string = 'John Smith';
//OR
let employeeName:string = "John Smith";
```

Template String:

```
let firstName:string = “Mohit";
let lastName:string = “Kumar";

let fullName1: string = “My First Name is ” +firstName+” and Last Name is “ +
lastName

let fullName2 : string = ` My First Name is ${firstName} and Last Name is
${lastName}`;

console.log(fullName1);
console.log(fullName2);
```

Boolean:

```
let isPresent:boolean = true;
```

Datatype

CREDO SYSTEMZ
simplifying IT

Array:

```
let fruits: string[] = ['Apple', 'Orange', 'Banana'];
let fruits: Array<string> = ['Apple', 'Orange', 'Banana']; // Generic array
type

//Arrays can be declared and initialized separately
let fruits: Array<string>;
fruits = ['Apple', 'Orange', 'Banana'];

let ids: Array<number>;
ids = [23, 34, 100, 124, 44];
```

Any: - Dynamic Type

```
let something: any = "Hello World!";
something = 23;
something = true;

let arr: any[] = ["John", 212, true];
arr.push("Smith");
```

Void - void is used where there is no data

```
function sayHi(): void {
   console.log('Hi!')
}

let speech= sayHi();
console.log(speech); //Output:
undefined
```

Function

CREDO SYSTEMZ
simplifying IT

Functions are the primary blocks of any program, In TypeScript, functions can be of two types: **named and anonymous**

Named functions:

```
function display() {
    console.log("Hello TypeScript!");
}

display(); //Output: Hello TypeScript
```

```
function Sum(x: number, y: number) : number {
    return x + y;
}

Sum(2,3); // returns 5
```

Function Parameters

```
function Greet(greeting: string, name: string ) : string {
    return greeting + ' ' + name + '!';
}

Greet('Hello','Steve');//OK, returns "Hello Steve!"
Greet('Hi'); // Compiler Error: Expected 2 arguments, but got 1.
Greet('Hi','Bill','Gates'); //Compiler Error: Expected 2 arguments, but got 3.
```

Function

CREDO SYSTEMZ
simplifying IT

Optional Parameters:

- The parameters that may or may not receive a value can be appended with a '?' to mark them as optional
- All optional parameters must follow required parameters and should be at the end.

```
function Greet(greeting: string, name?: string ) : string {
   return greeting + ' ' + name + '!';
}

Greet('Hello','Steve');//OK, returns "Hello Steve!"
Greet('Hi'); // OK, returns "Hi undefined!".
```

Default Parameters:

- TypeScript provides the option to add default values to parameters
- if the user does not provide a value to an argument, TypeScript will initialize the parameter with the default value

```
function Greet(name: string, greeting: string = "Hello") : string {
   return greeting + ' ' + name + '!';
}

Greet('Steve');//OK, returns "Hello Steve!"
Greet('Steve', 'Hi'); // OK, returns "Hi Steve!".
Greet('Bill'); //OK, returns "Hello Bill!"
```

Function

CREDO SYSTEMZ
simplifying IT

Anonymous functions:

- An anonymous function is one which is defined as an expression
- Expression is stored in a variable. So, the function does not have a name
- These functions are invoked using the variable name that the function is stored in

```
let greeting = function() {
   console.log("Hello TypeScript!");
};

greeting(); //Output: Hello TypeScript!
```

```
let Sum = function(x: number, y: number) : number
{
   return x + y;
}
Sum(2,3); // returns 5
```

Arrow Functions

- Fat arrow notations are used for anonymous functions
- Using fat arrow =>, we dropped the need to use the function keyword
- Parameters are passed in the parenthesis ()
- function expression is enclosed within the curly brackets { }

```
(param1, param2, ..., paramN) => expression
```

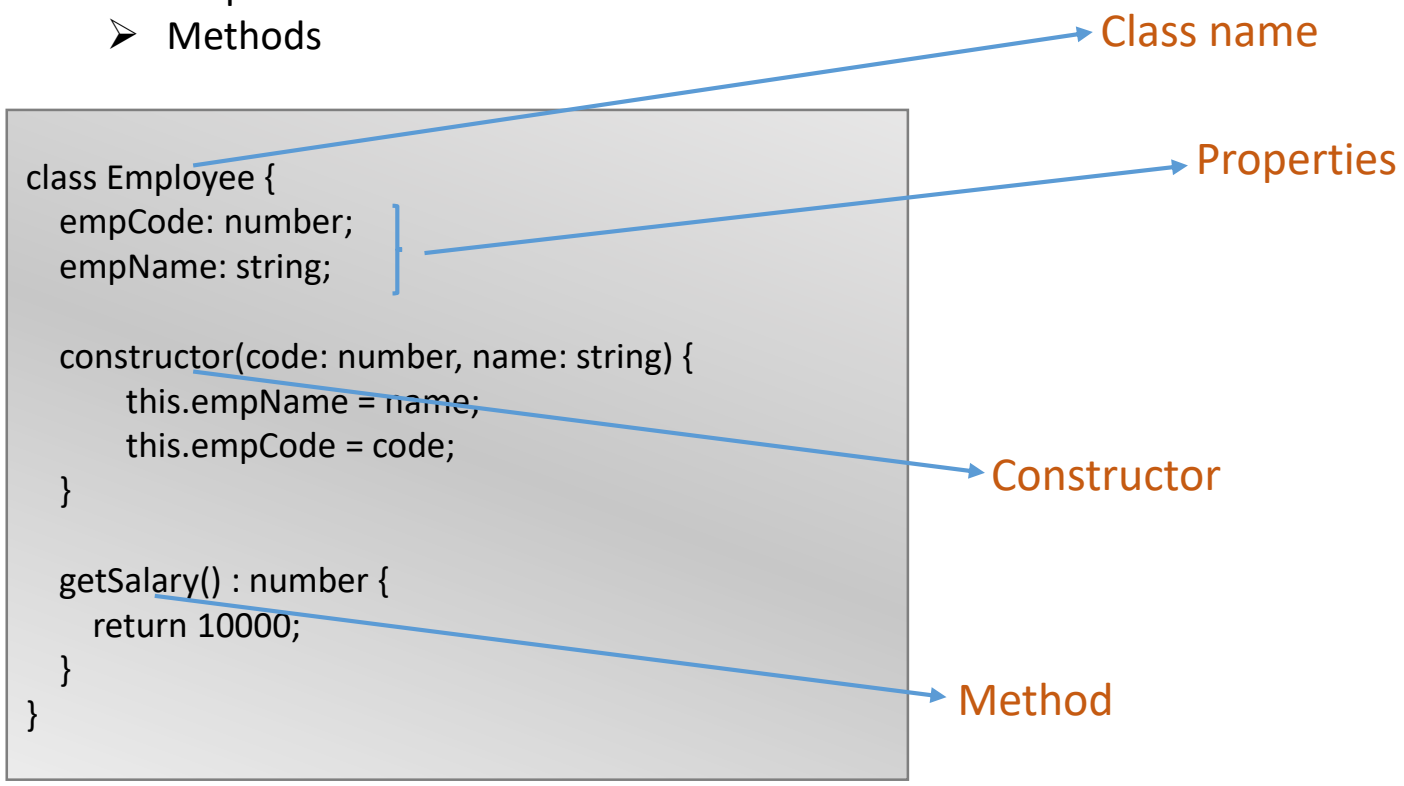
```
let sum = (x: number, y: number): number => {
   return x + y;
}

sum(10, 20); //returns 30
```

Classes

- Classes are the fundamental entities used to create reusable components
- Functionalities are passed down to classes and objects are created from classes

A class can include the following:

- Constructor
- Properties
- Methods



Classes - Constructor

- The constructor is a special type of method which is called when creating an object
- In TypeScript, the constructor method is always defined with the name "constructor“
- It is not mandatory for a class to have a constructor

Class without Constructor

```
class Employee {
   empCode: number;
   empName: string;
}
```

```
class Employee {

   empCode: number;
   empName: string;

   constructor(empcode: number, name: string ) {
      this.empCode = empcode;
      this.name = name;
   }
}
```

Creating an Object of Class

An object of the class can be created using the new keyword

```
class Employee {
   empCode: number;
   empName: string;
}

let emp = new Employee();
```

Classes - Inheritance

CREDO SYSTEMZ
simplifying IT

- One of the most fundamental patterns in class-based programming is being able to extend existing classes to create new ones using inheritance
- TypeScript classes can be extended to create new classes with inheritance, using the keyword **extends**

```
class Animal {
  move(distanceInMeters: number = 0) {
    console.log(`Animal moved ${distanceInMeters}m.`);
  }
}

class Dog extends Animal {
  bark() {
    console.log("Woof! Woof!");
  }
}

let dog = new Dog();
dog.bark();
dog.move(10);
dog.bark();
```

Data Modifiers

CREDO SYSTEMZ
simplifying IT

Controlling the visibility of class data members is a concept of **Encapsulation**

There are three types of access modifiers in TypeScript:

- public
- private
- protected

Public

- By default, all members of a class in TypeScript are public.
- All the public members can be accessed anywhere without any restrictions

```
class Employee {
   public empCode: string;
   empName: string;
}

let emp = new Employee();
emp.empCode = 123;
emp.empName = "Swati";
```

Data Modifiers

CREDO SYSTEMZ
simplifying IT

private

- The private access modifier ensures that class members are visible only to that class and are not accessible outside the containing class

```
class Employee {
   private empCode: number;
   empName: string;
}
let emp = new Employee();
emp.empCode = 123; // Compiler Error
emp.empName = "Swati";//OK
```

protected

- The protected access modifier is similar to the private access modifier,
- except that protected members can be accessed using their deriving classes

```
class Student {
   public studCode: number;
   protected studName: string;
}
class Person extends Student {

   showStudentDetails():void{
   console.log(this. studCode)
   console.log(this. studName)
 }
}
```

Data Modifiers

CREDO SYSTEMZ
simplifying IT

readonly

- TypeScript includes the *readonly* keyword that makes a property as read-only in the class
- Prefix readonly is used to make a property as read-only. Read-only members can be accessed outside the class, but their value cannot be changed

```
class Employee {
    readonly empCode: number;
    empName: string;
}
let emp = new Employee();
emp.empCode = 20; //Compiler Error
emp.empName = 'Bill';
```

Static

- The static members of a class are accessed using the class name and dot notation, without creating an object **e.g. <ClassName>.<StaticMember>**

```
class Circle {
    static pi: number = 3.14;

    static calculateArea(radius:number) {
        return this.pi * radius * radius;
    }
}
Circle.pi; // returns 3.14
Circle.calculateArea(5); // returns 78.5
```

Interface

- An interface defines the syntax that any entity must adhere to
- Interfaces define properties, methods, and events, which are the members of the interface
- It often helps in providing a standard structure that the deriving classes would follow.

```
interface interface_name {
}
```

```
interface IPerson {
   firstName:string,
   lastName:string,
   sayHi: ()=>string
}

var customer:IPerson = {
   firstName:"Tom",
   lastName:"Hanks",
   sayHi: ():string =>{return "Hi there"}
}

console.log("Customer Object ")
console.log(customer.firstName)
console.log(customer.lastName)
console.log(customer.sayHi())
```