Type Script Manual

Is a super set of javaScript .

It is not dynamically typed as javaScript (should specify the type) .

Installing TypeScript globally on my machine using npm install -g typescript --save-dev.

Or installing it in my project only using npm install typescript --save-dev.

To invoke a TypeScript compiler run npx tsc file.ts .

To compile all typescript files one type add typescript Compiler Config File: (not needed In angular)

npx tsc - –init

open file :

“strict”: true 🡪 decomment it

Now you can use( npx tsc )only

Primitive DataType :

String , number , Boolean

let age:number;

age = 24 ;

let isTall : boolean = true ;

let userName : string ;

userName = "Mohamed" ;

Arrays and Objects:

// objects and arrays

let person : {

    name: string ,

    age : number

}

person = {

    name : "mohamed",

    age : 20

}

// array of person

let people : {

    name: string ,

    age : number

}[];

people =[{name : "mohamed"  , age :20} , person , {name : "ghada" , age : 33}] ;

console.log(people);

Type Inference :

When creating variable and don’t specify specific type for it , and assign a value for this variable it saves the variable’s type , if assigned any other value with another type you will get and error

let cousre = "Angular"

cousre = 12 ; // here get error because cousre like cousre:string

Union Types:

Adding more than one type for a variable

let subject : string | number  ;

subject = "Math" ;

subject = 1234 ;

Type Aliases :

Define my own complex type and keep using it as type reference for variables

type School ={

    name : string ,

    id : number

};

let school : School ;

school = {

    name : "Almanara Language School",

    id : 1

}

Functions :

function print ( a : any) { // get error because there is a built in function called print so change its name

    console.log(a);

}

Generics :

It is away to avoide Type inference Problems , guarantee that all variables of same type in the same function

// Generics

function insertFromBegining <T> (array :T[] , value:T){ // this guarantee that the type of array elements and type of value are the same

    return [value ,...array];

}

const arrayStrings = ["Mohamed" , "Amr"] ;

const value = "Ghada" ;

const arrayNumbers = [2,3,4,5] ;

const value2 = 1 ;

insertFromBegining(arrayNumbers , value2); // same types

insertFromBegining(arrayStrings , value) ; // same types

Classes :

class Student {

//    public firstName : string ;

//    public lastName :  string ;

//    private courses : string[] ;

//     constructor (fn : string , ln : string , c : string[]){

//         this.firstName = fn ;

//         this.lastName = ln ;

//         this.courses = c

//     }

    // OR

    constructor (public fistName : string , public lastName : string , private courses : string[]){ }

    getCourses (){ // we don’t write keyword “Function”

        return this.courses ;

    }

}

const student = new Student("Mohamed" , "Elsaeed" , ["Math" , "CS1"]);

console.log(student.getCourses()); // now to get all courses can not access courses variable directly because it is private

Interfaces :

It is like type keyword but in addition there is a feature that classes can implements this interface .

It forces a class to be implemented in certain way .

Class that implements this interface can add additional features that not exists in the interface.

It is just draw the structure of the class that will implement it but not the implementation.

interface Human {

    name : string ;

    age : number ;

    greet : ()=> void ;

    sleep : ()=> void ;

}

class Teacher implements Human {

    name: string;

    age: number;

    office : number ;

    constructor( name :  string , age : number , office : number){

        this.age = age ;

        this.name = name ;

        this.office = office ;

    }

    greet (){

        console.log("Hello") ;

    }

    sleep(){

        console.log("I wanna sleep")

    }

}