TypeScript is a javascript superset, it means that typescript is in the end a language , a programming language building upon javascript. It takes the javascript language and adds new features and advantages to it.

Typescript of can’t be executed by javascript environments like the browser.

! will tells in typescript that will never yield null

// npm init

// npm install

// npm install –save-dev lite-server

// npm start

**Core types:-**

//number – it can holds both int and float

//string – “hi”, ‘hi’, `hi`

//Boolean – true, false

tsc app.ts

TypeScript will put the restrictions on what kind of value it should accept.

Typescript has a buit in feature which is called type inference. This means that typescript does its best, to understand which type you have in a certain carriable or constant.

Const person : {

name:string;

age:number;

} = {

Name : “Lakshman”,

Age : 24

};

Console.log(person.name);

Or

Const person = {

Name : “Lakshman”,

Age : 24

};

Console.log(person.name);

Any type is a special type in typescript which accepts anytype.

Map method is available in arrays not in string.

Tuple is an array of fixed type and fixed length. [number, string]

Push is an exception which is allowed in tuples .

Enum assigns labels to the numbers

// enum Role { ADMIN, READ\_ONLY, AUTHOR};

By default if we didn’t assign the value to the enum then it will take 0 and then it will increment by itself and als we can define the different values to the enum.

Aliases - type n = number;

Then we can use n anywhere if we want to use in place of number.

**Function types** are the types that describes the which type of function specifically we want to use somewhere.

combieneValues : Function

Let combineValues: () => number // here combineValues accept the function and returns the number as the output.

Let combineValues : (a:number, b:number) =>number;

We will use **unknown** type when user doesn’t know what he will eventually enter. It is more restrictive than any type

For example:-

Let a :unknown;

Let b:string;

A = 5;

A = “hi”;

B = a;

This will give error since the unknown can have any type of datatypes.

Let a :any;

Let b:string;

A = 5;

A = “hi”;

B = a;

This will not give error since the any type is flexible kind of datatype.

**Never type :-** We use this type when the function is throwing an error or exception or while loop.

We will use **Union** and **literals** when we expect more than one type.

// tsc demo.ts -w

**Arrow function:-**

Const add = (a:number, b:number)=>a+b;

Const printOutput: (a:number|string) => void = outpu => console.log(output);

Default parameters has to be last in the parameters

Const add = (a:number, b:number = 1) => a+b;

Add(5) // gives 6

Array destructuring means pulling the array elements out. It won’t change the original array

Classes allow us to define how objects should look like which data they should hold, which methods they should have, so that we can easily build objects based on these classes.

Readonly in typescript is same as final in java. Won’t change the value once it is initialized.

Static properties and methods allow us to add properties and methods to classes which are not accessed on an instance of the class.

If we declare the access modifier of constructor as private then it will become singleton.

Interface describes the structure of the object

Emp = {

Name:”lakshman”,

Age:23,

}

If(Emp.Age) // typescript wont allow to access the object properties in order to do that we need to follow the following syntax below

If(“Age” in Emp)

! in typescript is used to know that parameter will never get null value.