**Coverage**

**ES6 JavaScript & TypeScript**

**Prerequisites:**

**=>Node**

[**https://nodejs.org/en/download**](https://nodejs.org/en/download)

Check the version in cmd

*node -v*

*npm -v*

**=> TypeScript**

***npm install -g typescript***

Check the version in cmd

tsc -v

**=>IDE**

<https://code.visualstudio.com/download>

**Note:**

node installation should be done through admin access only.

Tsc - -init ( tsconfig.json)

tsc FN.ts

node FN.js

tsc –w (watcher for Compilation process)

or

Cntrl+Shift+B

For Debug:

Cntrl+Shift+D

**Tsconfig.json**

{

"compilerOptions": {

"target": "es5",

"outDir": "jsScripts/",

"sourceMap": true

}

}

**DataTypes =>**

**Boolean**

let someVar: boolean = true | False;

**Number**

let someVar :number=7;

**String**

let someVar : string = "Hello";

You can also use **Template** strings, which can span multiple lines and have embedded expressions. These strings are surrounded by the backtick/backquote (`) character, and embedded expressions are of the form ${ expr }.

let someVar:string=”Training”;

let sentence:string=`Welcome to the ${someVar}` }` (Thru Variable Substitution)

let multi = `

hello

world

my

name

is

xyz`;

console.log(multi);

**Array**

let someVar :number [ ] = [1,2,3]

OR Generic Array Type

let someVar :Array<number>=[1,2,3]

**Tuple­­­­­­­­­**

letsomeVar :[ string,number,boolean]

someVar=[“Hello”,5,true];

**Enum (**allow us to define a set of named constants**)**

enum Color {Red, Green, Blue}

let c: Color = Color.Green;

OR

enum Direction {

Up = "UP",

Down = "DOWN",

Left = "LEFT",

Right = "RIGHT",

}

**Any**

let notSure: any = 4;

notSure = "maybe a string instead";

notSure = false;

let list: any[] = [1, true, "free"];

**Void**

**void is a little like the opposite of any: the absence of having any type at all. You may commonly see this as the return type of functions that do not return a value:**

function warnUser(): void {

console.log("This is my warning message");

}

**Null and Undefined**

**Null is an assignment value. It can be assigned to a variable as a representation of no value:**

var TestVar = null;

console.log(TestVar); // Null

**undefined means a variable has been declared but has not yet been assigned a value, such as:**

var TestVar;

console.log(TestVar); // undefined

**Object**

Object is a type that represents the non-primitive type, i.e. any thing that is not number, string, boolean, symbol, null, or undefined

var person = {

firstname:"Tom",

lastname:"Hanks"

};

console.log(person.firstname) //access the object values

console.log(person.lastname)

**Inferred Type** --declaring a variable without a type. In such cases, the compiler will determine the type of the variable on the basis of the value assigned to it.

var num = 2; // data type inferred as number

console.log("value of num "+num);

Num = "12"; // not assigned to a number

console.log(num);

**var** (Global Scope, Class Scope, Local Scope{within methods or loops} )

var foo = 123;

if (true) {

var foo = 456;

}

console.log(foo); // 456

**let** (Block Scope)

let foo = 123;

if (true) {

let foo = 456;

}

console.log(foo); // 123

var foo = 123;

function test() {

var foo = 456;

}

test();

console.log(foo);//123

**Re-declarations**

var x=10;

var x=20;

let x=10;

let x=20; // Error

**const (** are another way of declaring variables

they have the same scoping rules as let, but you can’t re-assign to them).