**Typescript Tutorial**

**1-In this tutorial you will be understanding :-**

**A-Type Annotations**

**B-Arrow Functions**

**C-Interfaces**

**D-Classes**

**E-Constructor**

**F-Access Modifiers**

**G-Properties**

**H-Modules**

**\*What is the Typescript?**

**Is superset JavaScript, has additional features that the JavaScript not supported**

**The property of Typescript:-**

**1-Strong Typing: -**

**This means that when assign value not suitable to the datatype it will make error, this make the application more predictable and easy to compile**

**2-Object Oriented features:-**

**It’s containing component not exist in the JavaScript such as:**

**Class, interface, property, access modifiers, constructor**

**3-A lot of errors can be detected at compile time, unlike the JavaScript that detected errors at runtime**

**4-Great Tooling (the intellisense is exist )**

**Trans pile**

**It’s compile the Typescript to JavaScript**

JavaScript

Typescript

**Steps of install typescript**

**1-on command prompt writes the following code to install npm**

**npm install –g typescript**

**2-to see the version of the Typescript, type this code**

**Tsc –version**

**3-make folder called ts-hello**

**Mkdir ts-hello**

**Cd ts-hello**

**Code main.ts 🡪 this will open main.ts folder to write Typescript File or create new file with the name main.ts if it’s not exist**

**Dir //To see the components on the folder on command prompt**

**Ls //To see the component on the folder on GitPash**

**If we write the following JavaScript Code like**

function log(msg)

{

console.log(msg);

}

var msg = "Hello world";

log(msg);

**And then type the following command which convert main.ts to main.js**

**tsc main.ts //it will compile the TypeScript folder to JavaScript Folder**

**So the manual translation from the Typescript to JavaScript happen when you compile your application with this command**

**ng serve**

**This means that the angular CLI order the Typescript Compiler to translate the JavaScript File to Typescript file**

**3-if we want to execute the code and show the result on node js we can set**

**Node main.js**

**Or**

**Node main.ts**

**JavaScript Versions**

**1-ES5 (Ecma Script 5) supported by all the browsers**

**2-ES6 (2015)**

**3-ES2016**

**4-ES2017**

**typescript variables**

**1-we have 2 ways to define variables**

**var x = 10**

**let x = 10**

**We will explain the differences between the 2 ways of define variables**

**1-by using var keyword**

**(By default the var keyword is accessible by the level of the nearest method, this means that it access inside and outside the for loop, so it will print the last number which is 11)s**

**We will put method and execute it on the node js**

function Dosomething()

{

//the variable i is accessible on inside the method (inside and outside for loop)

for(var i=0;i<=10;i++)

{

console.log(i);

}

console.log('The Final Number is '+i);

}

Dosomething();

//and in Git.Pash we will execute

Cd Desktop/TypeScriptPro

tsc main.ts

node main.js

**2-by using let keyword**

**(This means that it accessible in the for loop only not outside for loop, so it make compilation error)**

function Dosomething(){

for(let i=0;i<=10;i++){

console.log(i);}

console.log('The Final Number is '+i);}

Dosomething();

**3-we can remove the JavaScript file by using the command**

**rm main.js**

**tsc main.ts**

**node main.js**

**on the previous example we set let keyword , so it make compilation error , however it still make valid JavaScript File**

**//The Reason is that it’s execute on Ecma Script 5 on the most of browsers that is not defined the let keyword and automatically convert it to var keyword**

function Dosomething() {

//we see that it convert from let to var

for (var i = 0; i <= 10; i++) {

console.log(i);

}

console.log('The Final Number is ' + i);

}

Dosomething();

**4-if you don’t declare vale to the variable , it will be assigned as any , the any means that it carry any data in it and can change its value**

let str;

str = 1;

console.log(str);

str = ' ahmed ';

console.log(str);

str = true;

console.log(str);

**But this is make problems because the variable is accept any data type , so the solution is with using the Type definition**

//we define each variable with its suitable datatype

let no:number;

let str:string;

let state:boolean;

let intlst:number[] = [1,2,3,4,5];

let objlst:any[] = [1,true,'ali',10]

//this is const variable that is stored in the heap memory

const ColorRed = 0;

const ColorGreen = 1;

const ColorBule = 2;

//we create enum variable

//each element have it's default value

//enum colors {Red = 0, Green = 1, Blue = 2};

enum colors {Red , Green , Blue};

let col:colors = colors.Blue;

**Type Assertion**

**1-we have to know that the type assertion does not modify the structure of variable or instance in memory ,but it enable the intellisense of the variable**

**When we put code like this**

//the variable defined as any , so the intellisense it gone

let msg ;

msg = 'hello world';

let staet = msg.

**To resolve this issue we can use the Type Assertion**

**(Like conversion and casting in C#)**

**(We have 2 ways to define the Type Assertion)**

let state = (msg as string).endsWith('d');

let state2 = (<string>msg).endsWith('d');

**Arrow Function (Lamda Expression)**

**-1The lamda expression (Arrow Function) allow to simplify the assigning function in the variable and using it**

**2-we use the lamda expression when we want to make operation inside variable and we want to pass parameter on it , and this function is anonymous function**

//This is the regular way to assign function inside the variable

let f1 = function(message)

{

console.log('Hello '+message);

}

//This the lamda expression

let f2 = (message) => console.log('Hello 2 '+message);

//The way to execute variable that assigned lamda expression

f1('Mohammed');

f2('Ali');

**Inline Annotation**

**Instead to make no of parameters inside arrow functions like**

//not preferable

let f3 = function(x,y,z,f,w){

return ((x+y-z)\*f)/w;}

//we execute the inline function

f3(1,2,3,4,5);

**The Solution is with used the class and assigned parameters on it**

class shape{

constructor(xvalue:number,yvalue:number,shape:string){

this.x = xvalue;

this.y = yvalue;

this.shape = shape;

this.Area = xvalue \* yvalue;}

x:number;

y:number;

shape:string;

Area:number;}

//we make class and pass it as parameter to decrease the number of parameter

let f1 = (sh:shape) => console.log('the shape name is '+sh.shape+' and the area equal '+sh.Area);

let Rect:shape = new shape(10,20,'Rect1');

**Class**

**Is an group of variables (Prosperities) and functions (methods) that are highly related**

**(if you want to make function with name and can reused along the application life cycle , you can put it in the class)**

class Employee

{

constructor(fname:string,lname:string,gender:boolean,salary:number)

{

this.Fname = fname;

this.Lname = lname;

this.Gender = gender;

this.Salary = salary;

}

Fname:string;

Lname:string;

Gender:boolean;

Salary:number;

GetEmployeeInfo()

{

console.log('The Employee Full Name is '+this.Fname+' '+this.Lname+' , and his salary is '+this.CalcoulateSalary());

}

CalcoulateSalary(){

return this.Salary - (this.Salary\*0.1);}}

//we make instance of type Employee and assign object of type employee with = expression

let m1:Employee = new Employee('ali','ahmed',true,400);

//or you can declare it as (it will automatically define it as Employee type)

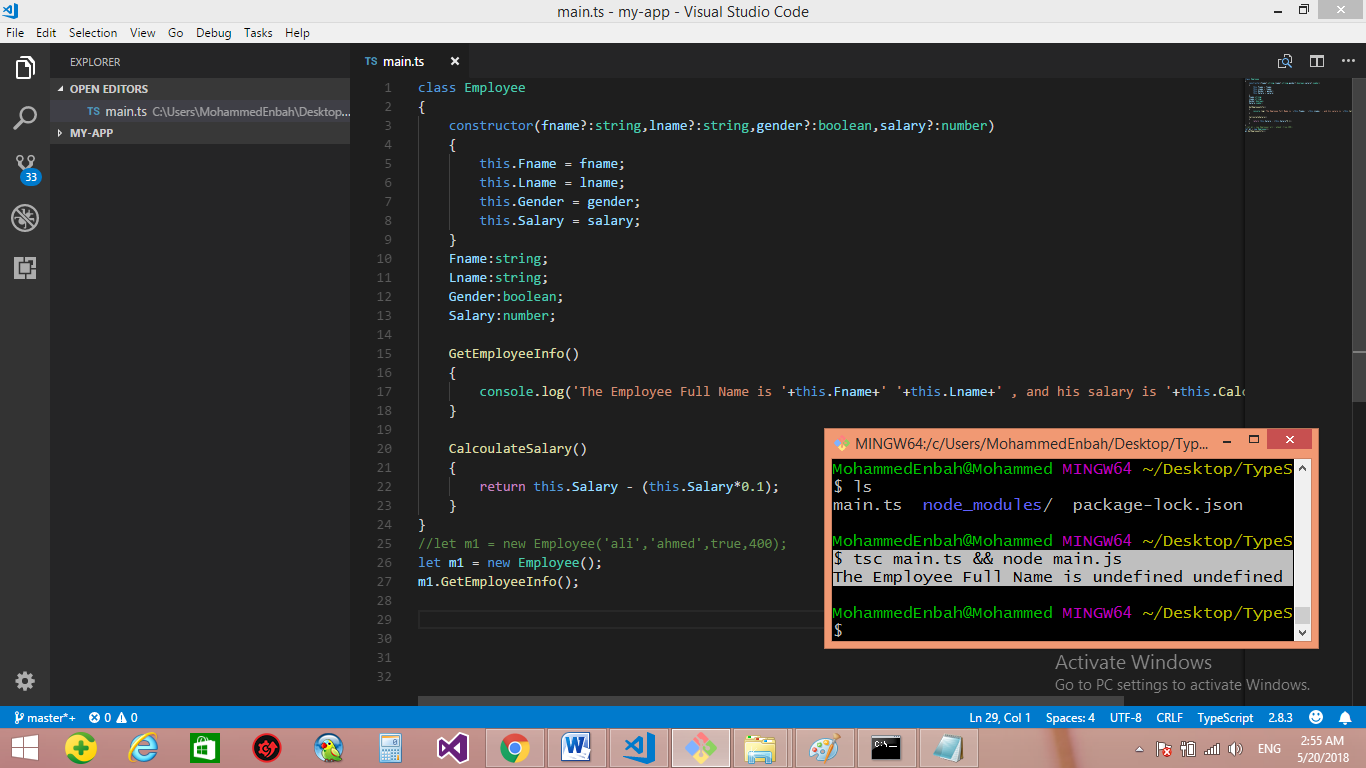
let m1 = new Employee('ali','ahmed',true,400);

m1.GetEmployeeInfo();

**We have to define the values of fields because we make object of type Employee, it does not initialize the fields.**

**//we see that there is no initialization when make create object with no assign values**

**//In TypeScript there is no multiple constructors , so the solution is used the optional parameter**



**Access Modifiers**

**1-Public**

**2-Private**

**3-Protected**

**By Default all member inside the class are public**

**The benefit of the access modifier is to restrict the access to the properties and fields on the class**

class Employee

{

constructor(fname?:string,lname?:string,gender?:boolean,salary?:number)

{

this.Fname = fname;

this.Lname = lname;

this.Gender = gender;

this.Salary = salary;

}

private Fname:string;

private Lname:string;

private Gender:boolean;

private Salary:number;

GetEmployeeInfo()

{

console.log('The Employee Full Name is '+this.Fname+' '+this.Lname+' , and his salary is '+this.CalcoulateSalary());

}

CalcoulateSalary()

{

return this.Salary - (this.Salary\*0.1);

}

}

let m1 = new Employee('ali','ahmed',true,400);

m1.GetEmployeeInfo();

**Automatic Generation Fields on the class**

**Instead to make fields and assign it in the constructors, the Typescript make amazing features called Automatic Generation Fields on the class which allow to automatic generate fields when passed it in the parameters**

class Employee{

//This amazing technique will make automatic generation of the fields when we make pass it as parameters

//you can set it as public or private

constructor(private fname?:string,private lname?:string,private gender?:boolean,public salary?:number)

{

}

//we see that it’s define the fields automatically

GetEmployeeInfo(){

console.log('The Employee Full Name is '+this.Fname+' '+this.Lname+' , and his salary is '+this.CalcoulateSalary());

}

CalcoulateSalary(){

return this.salary - (this.salary\*0.1);

}}

let m1 = new Employee('ali','ahmed',true,400);

m1.GetEmployeeInfo();

**Properties**

**1-in the general way , we can create function that perform the get and set and make logic business to it**

class Employee

{

constructor(private fname?:string,private lname?:string,private gender?:boolean,public salary?:number){}

GetEmployeeInfo(){

console.log('The Employee Full Name is '+this.fname+' '+this.lname+' , and his salary is '+this.CalcoulateSalary());}

CalcoulateSalary(){

return this.salary - (this.salary\*0.1);}

//This is the way to perform the Get / Set using functions

getFname(){

return this.fname;}

setFname(value:string){

this.fname = value;}

getSalary(){

return this.salary;}

setSalary(value:number){

if(value < 300){

throw new Error('value cannot be less than 300');}

this.salary = value;}}

let m1 = new Employee('ali','ahmed',true,400);

m1.GetEmployeeInfo();

let Fname = m1.getFname();

m1.setFname('Yosef');

let sal = m1.getSalary();

m1.setSalary(220);

**Instead of using function to access to private fields, we can use properties to achieve this requirement**

**So the Property is combination of getter and setter , or it’s the outside is field but in the inside it’s just method**

class Employee{

constructor(private fname?:string,private lname?:string,private gender?:boolean,private salary?:number){}

GetEmployeeInfo(){

console.log('The Employee Full Name is '+this.fname+' '+this.lname+' , and his salary is '+this.CalcoulateSalary());}

CalcoulateSalary(){

return this.salary - (this.salary\*0.1);}

//This called getter and it's operate as field not function

get Fname(){

return this.fname;}

//This called sitter and it's operate as field not function

set Fname(value:string){

this.fname = value;}

//This called gitter and it's operate as field not function

get Salary(){

return this.salary;}

//This called sitter and it's operate as field not function

set Salary(value:number){

if(value < 300){

throw new Error('value cannot be less than 300');}

this.salary = value;}}

let m1 = new Employee('ali','ahmed',true,400);

m1.GetEmployeeInfo();

//This is the way of using the property to get value

let Fname = m1.Fname;

//This is the way of using the property to set value

m1.Fname = 'Yosef';

//This is the way of using the property to get value

let sal = m1.Salary

//This is the way of using the property to set value

m1.Salary = 330;

**Modules**

**1-Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes, etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the**[**export forms**](https://www.typescriptlang.org/docs/handbook/modules.html#export)**. Conversely, to consume a variable, function, class, interface, etc. exported from a different module, it has to be imported using one of the**[**import forms**](https://www.typescriptlang.org/docs/handbook/modules.html#import)**.**

**2- At runtime the module loader is responsible for locating and executing all dependencies of a module before executing it**

**3-Well-known modules loaders used in JavaScript are the [CommonJS](https://en.wikipedia.org/wiki/CommonJS) module loader for Node.js and**[**require.js**](http://requirejs.org/)**for Web applications.**

**4-a file without any top-level import or export declarations is treated as a script whose contents are available in the global scope (and therefore to modules as well)**

**5-file with import and export is treated as module which can be used by other modules with import and export syntax**

**1-Export**

**1-any component like class , function , type , interface can be exported and see from the other modules such as**

**export interface StringValidator {**

**isAcceptable(s: string): boolean;**

**}**

**export const numberRegexp = /^[0-9]+$/;**

**export class ZipCodeValidator implements StringValidator {**

**isAcceptable(s: string) {**

**return s.length === 5 && numberRegexp.test(s);**

**}**

**}**

**Multiple Inheritance and Single Inheritance**

**We refer to multiple inheritance to interface , this is means that the class that implement the interface can implement more than one interface**

**All member of the interface are public by default and cannot be changed**

**We refer to single inheritance to Abstract class , this is means that the class the extend the abstract class can extend only one abstract class**

**The abstract class can have method with code , and abstract method , but abstract methods must be implemented in the child class**

interface IhomeContract{

FirstPart? : string;

SecondPart? : string;

HomePrice? : number;

Commistion? : number;

CalcoulateHomePrive() : number;

ShowHomeContract() : void;

}

abstract class Employee implements IhomeContract{

//This the implementation code of the interface

Commistion?: number = 0.02;

FirstPart?: string;

SecondPart?: string;

HomePrice?: number;

CalcoulateHomePrive(): number {

return this.HomePrice + (this.HomePrice\*this.Commistion);

}

ShowHomeContract(): void {

console.log('The First Part name is '+this.FirstPart+' and the second part name is '+this.SecondPart+' and the Home Price is '+this.CalcoulateHomePrive());\}

//we pass parameter to the constructor that will be called first when we create object in the subclass

constructor(public fname:string,public lname:string,public gender:boolean,public salary:number,SecondPart?:string,homeprice?:number){

this.FirstPart = fname +' '+lname;

this.SecondPart = SecondPart;

this.HomePrice = homeprice;}

//we make method with code

GetEmployeeInfo():string{

return 'The Employee Full Name is '+this.fname+' '+this.lname;}

//we make abstract method

public abstract CalcoulateSalary() : number;

}

//This is the implementation of the single inheritance

class Admin extends Employee {

Allownce:number ;

constructor(fname:string, lname:string, gender:boolean, salary:number, allownce:number = 0,SecondPart?:string,homeprice?:number)

{

//we see that it must be call the constructor of the super class before we call the execute the code

// inside the constructor on the sub class

super(fname,lname,gender,salary,SecondPart,homeprice);

this.Allownce = allownce;

}

//we see that it's optional to implement the method with code in the super class

GetEmployeeInfo():string

{

return super.GetEmployeeInfo() + 'and his salary is'+this.CalcoulateSalary();

}

//it must be call the mehtod of abstract of the super class

public CalcoulateSalary(): number {

return this.salary + (this.salary \* this.Allownce) ;

}

}

//we will make object of Admin that will call the constructor of Employee before executing its own code

let m1 = new Admin('ali','ahmed',true,400,50,'ITECH 2',30000);

//we implement the abstract method on the interface

m1.ShowHomeContract();

//we call the method on the super class Employee

console.log(m1.GetEmployeeInfo());