Typescript 101

Alternative to javascript

Allows us to use strict types

Support modern feature (arrow functions – let – const)

Extra features (generics – interfaces – tuples etc)

خواص

**TypeScript** is a relatively new language. It was launched for public use in October 2012, as the version 0.8 of the language. It was a result of two years of developed at Microsoft, with Anders Hejlsberg, the lead architect of c#, as well as the creator of Delphi and Turbo Pascal working on the project too.

**Parent language:**JavaScript

**Language designers:**Microsoft Corporation

## //src

<https://www.typescriptlang.org>

## //error

* When you compile the ts file to js file an error is present ‘cause about the same variable in the two files.
* If you try to change the datatype of a var example: nbr 🡪 string

Let n = 60; let n = “ahmad” 🡪

The ‘”y”’ is not assignable to type ‘number’

## //Compiler

To recompiler automatically => tsc file.ts -w (watch flag)

## //Syntax

Replace var/const by let:

* you can’t change the datatype of var the same of .js
* It’s not necessary to define integer or double for a number
* ! there are a value => not null
* ? optional
* ${name} 🡪 do the enter and the space
* Any function make it in a class call method
* Any variable make it in a class call property
* Instance = object

## //Either regular function = IRA function

Const circ = (diameter) => { //code } = const circ = function(diameter){//code}

## //array

It’s a kind of object

The .ts file watch in the contain of array and let you change in the same datatype which contain.

## //object

It doesn’t then match the structure of the initial object that we created:

Let ninja = {name: ’Mario’, age:’50’};

ninja = {name: ’ahmad’, age: ‘19’, major:IT };

## //initialize

**Explicit type:**

Let a : string,

**Array:**

Let ninjas : string [ ];

let ninjas : string[ ] = [ ]; (empty array to allowed the change on the array)

ninja =[ string, string, string ];

**union type:**

let ninjaOne: object;

let ninjaTwo: {

name: string,

age: number

};

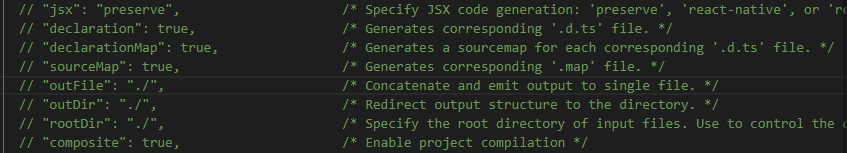
**Any: (dynamic type)**

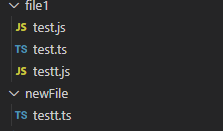
Let ahmad: any[ ] = [ ] ;

## //Set up files

To make a relation between two files on two different folder 🡪 tsc --init







## //function

Let greet : function; | Greet = () =>{ //code };

**? (optional)**

Ex:

Let click = (a : number , b? : number) => {

Console.log(“a”)

}

Click(3); (don’t difined b is acceptable ‘cause b is optional)

You can type it same that 🡪 let click = a:firstname => console.log(a)

## //Type aliases

This is a way to make the write of the data type pretty easier

//For example:

If an property data type is: name : string | number | bolean

🡪you make a type aliases:

Type Strinbrbol = string | number | bolean

UID : strinbrbol ;

And also we can do it for the object

//for example:

Type ObjWithName = {name: string , UID: Strinbrbol}

## //function signature

You type a general function

//example one

Let add =(nbrOne :number; nbrTwo: number) => void;

Add =(name : string, age : number)=>{

Console.log(nbrOne + nbrTwo)

}

//example two

//type aliases

Type nbr = number;

//fuction

let calc : (a: nbr , b : nbr , c ?: string ) => nbr;

calc = (nbrOne: number ,nbrTwo : number , action ?: string) =>{

if( action === "add"){

return nbrOne + nbrTwo;

}else {

return nbrOne - nbrTwo

}

}

calc(1 , 2 , "add" );

calc = (g:nbr , j: nbr , jo ?: string) => {

return g + j ;

}

calc(1 , 2 );

## //class

The class is a collection of properties

class person {

    name : string;

    age : number;

    registrtion : Date;

let x = new person(); // x called object / instance

 x.name = "ahmad";

 x.age = 55;

 x.printmyname();

 let y = new person();

 y.name = "omar";

 y.age = 80;

 y.printmyname();

extends relate one class to another class:

class student extends person{

    schoolID : number;

}

## //constructor

is the first function activated it run without you called it

export class person {

    name : string;

    age : number;

    registrtion : Date;

    printmyname(){

        console.log(this.age);

    }

    constructor(){

this.registrtion = new Date();

   console.log("ana l constructor")

   console.log(this.registrtion);

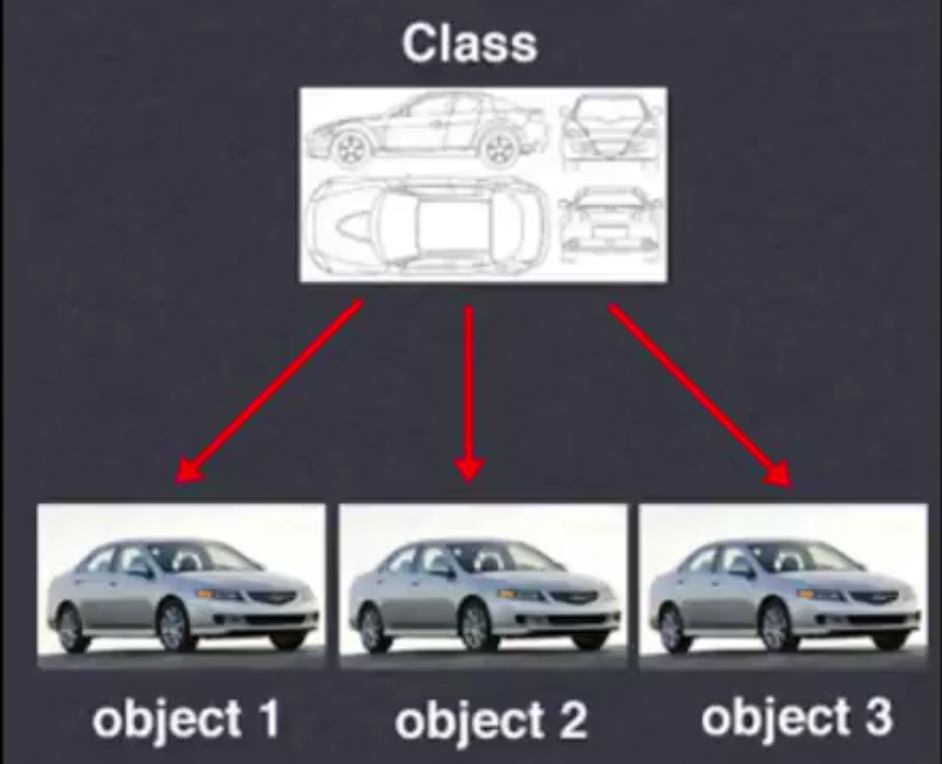
    }

}

## //module

Test.ts 🡪Export class className{//code}

Import {className} from {./test}



## //this

It used to call the class or a property in the class

## //Inheritance

The parent give the properties to her child

* For create a sub class

## //interface

It’s a way to abbreviate the parameter’s function

* Use our data in more places
* Descript an form of data
* Debug the code fast

Example:

Let descript = (name: string , age: number , eyescolor : string , haircolor: string , skincolor : string ) =>{ //code }

We can replace this code by an interface, that make it more easy;

// Interface persondesc {

name: string ,

age: number ,

eyescolor : string ,

haircolor: string ,

skincolor : string

}

// Let descript = (person : persondesc){ //code };

