**TYPESCRIPT**

What is TypeScript ?

* TypeScript is a Strongly typed progrmming language That builds on javaScript.
* TypeScript Developed and maintained by Microsoft
* TypeScript is javaScript with added Syntax for Types
* TypeScript add Features to JavaScript Without Changing it

Why we need TypeScript ?

* Detect Errors Without Running The Code “Static Type Checking”
* Analyze The Code As You Type
* Save Some Unit Tests As The Error Show While Writing
* Every Js File is a valid TS File
* Will help you when you write React,Vue or Angular Apps
* Gives You The missing Features In JS Likce “Interfaces, Generics, Decorators”

How TypeScript Works ?

* TypeScript Compiler Compile TS Code into Js code “This Called Transpilation”.
* How About The New Features? TS Try to simulate Them using Js **“Workaround”**

# TypeScript Getting Started .

## Instal Node.Js

For using node package manager and install the language on my stystem

You can dowland it from nodejs.org

You can make that instal it by : node -v for see the node version

## Instal TypeScript

You can install TypeScript on a project and you can install on the global system

For install it on the global system : npm i -g typescript

**Explanation :** node package manger install globally typscript

**nmp** is the world’s largest **Software Library** npm هي أكبر مكتبة برمجيات في العالم

**nmp** is also a software **Package Manager** and **Instaler**

**Check that is well installed** : tsc -v

## Transpiling Files

As we said, Ts detect errors without running the code.

console.log(Math.floor());

here we got a red line under floor, and a text : “Expected 1 arguments, but got 0.ts(2554) lib.es5.d.ts(714, 11): An argument for 'x' was not provided.”

Now you can see how ts is great.it is detect the error and tell you what is it and explain it to you for you can solve it , it really great !!

now, we have to **run the code**, so first we must compile it to a js file.

tsc tsFileName : tsc index.ts

so this code check if js file exist, if it exist, it update its content, if not, it will create a new one .

**Note:** you have to do the traspilation each time you modify ts file, because js file receive modifcation until doing transpilation

## Create Configuration and Watch Files

As we said in each modification,w have to compile the ts file to a js file using the compiler, if not no modificatin will be sended to our js file.

But this process seems hard espicially in the big project.

*That why watch mode comes*

Watch mode is a feature in TypeScript that allows you to automatically recompile your code whenever you make changes to your files. It continuously watches for changes in your TypeScript files and recompiles them in the background, so you can see the results of your changes quickly without having to manually run the compiler each time. This can be especially useful during development when you are making frequent changes to your code.

For turn this feature on , run this code in your terminal : tsc -w index.ts

**Note:** -w flag is a shortcut for -watch

This will open a node terminal that show also the errors before run the code.

***Configuration***

Configuration refers to the process of setting up and customizing the behaviour of the TS compiler, it includes specifying various options and settings to control how TS code is compiled into JS. These configurations can include specifying the target version of JS, enabling or disabling certain compiler features, The configuration file for TS is typically named tsconfig.json and is used to define these settings for particular TS project. (not recommended to use more than one tsconfig.js file.

For getting tsconfig.json file : tsc –init (initialization)

As we said you can controll many things:

Focus Here Pls :

* Src folder : is folder that contain all filles that will be compiler
* Dist folder (distination) : where the compiled folder are saved

For setting this folder you have to modify tsconfig.json.

More precisely, this property

// "rootDir": "./",                                  /\* Specify the root folder within your source files. \*/

So now create src folder and set it rootDir property .

"rootDir": "./src"

And now set dist folder (Note: you don’t have to create, it will be created auto)

 // "outDir": "./",                                   /\* Specify an output folder for all emitted files. \*/

And now :

"outDir": "./dist"

And now you don’t have to declare ts file name, tsc -w

And sure you have many other propeties :

// "removeComments": true,                           /\* Disable emitting comments. \*/

Now the commet will no be shown to the compiled file :

"removeComments": true

**Note** : you have to do exprence for more comprhension

## Statically vs Dynamically Typed Languages

Statically Typed Languages :

* Variables are صراحة explicitly declared with their data types at compile-time.
* Type Checking is performed during the compilation process
* Require explicit type annotations for variables,fuctions params, and return values
* Offers better perfomance as types are knonw in advance, so no need to check types dinamically.
* Error detected earlier.

Dynamically Typed Languages :

* Variables do not have exmolicit type declarations and ca hold values of any .
* Type checking is performed at runtime.
* Type annotations are optional and not strictly required (or no possible in origin)
* Provides flexibility for changing that types
* Error can be detected after execution (or can not be detected)

## Type (Annotations || Signature)

There are 3 mian primitives in js and ts.

**Boolean – number – string**

There are also 2 less common primitives used in later version of js and ts

* **Bigint** : as number but it allows larger numbers
* **Symbol** : are used to create a globally unique identifier

**Type Assignment :**

When creating a variable, There are 2 main ways Typscript assigns a type :

* Explicit
* Implicit

**Explicit** type assignement are easier to read and more intentional تعمدًا

let theName: string = "Bilal";

let theAge: number = 17;

let hire: boolean = true;

you can set many types to a var

let all: string | number | boolean = "Bilal";

all = 17; // no error

all = true; // no error

**Implicit** – TypeScript will “guess” the type, based on the assigned value.

let firstName = "Ossama";

**Note** : Having TypeScript “guess” the type of a value is caller **infer**

**Implicit** type assignment are shorter, faster to type

But some times “any” type can cause problems, particularly in function parameters:

function add(a, b) {

    parseInt(a);

    return a + b;

}

const result = add(5, "10");

console.log(result); // 510

so the function does not add numbers but it make concatenation .

and there no error that can throw out attention to the problem,

but in TS, it necessarily to define functon params type to avoid all this problem.

In other hand, you have the choice of disable this feature

\*\* **There is no better type assignmet, it depends on the dev style.**

## Type Annotations With Arrays

As we know, Ts can infer type if we did not declare it.

So what about arrays type, let make an array, and let ts detecting it type

let array = ["Bilal", "Ahemd", 17, true];

// let array: (string | number | boolean)[]

let array2 = ["Bilal ", "Elzero"];

// let array2: string[]

It very easy just focus, it declare elements array types and then declare that in the end an array .

**Qu** : what is the benefit of declaring var types

**Answear** : declaring var types make us able to use var types properties

let x: string;

console.log(x.)

now vs code suggests for me all stirng propeties

let x: number;

console.log(x.)

and now vs code suggests for me all number propeties

type declaring has many other benefits :

let array = ["Bilal", "Ahemd", 17];

for (let i of array) {

  console.log(i.repeat());

}

// error under repeat : Property 'repeat' does not exist on type 'string | number'.

// Property 'repeat' does not exist on type 'number'

The error is very clear, but see js what logged on the console :

index.js:4 Uncaught TypeError: i.repeat is not a function

at index.js:4:19

## Type Annotations With Multidimensional Arrays

A multidimensional array is an array that contain nested arrays

let array1 = ["Bilal ", "Elzero", ["Bilal", 22], true];

//let array1: (string | boolean | (string | number)[])[];

I think it very easy , and if you want a deep comprehension, make exercises by declaring an array and guessing what type they are, and then take the correction by make ts guss the type

let array = ["Bilal ", "Elzero", ["Bilal", [true, "inas"], 22], true];

// let array : (string | boolean | (string | number | (string | boolean)[])[])[]

let array = ["Bilal ", "Elzero", [15, 17], true];

// let array : (string | boolean | number[])[]

## Type Annotations With Functions

Focus in this code :

let showMsg = true;

function showDattails(name, age, salary) {

  if (showMsg) {

    return `Hello ${name},your age is ${age} and your salary is`;

  }

}

console.log(showDattails("Bilal", 17));

if you run this code in js, you wont receive any error, but if you run it is TS you will get 5 errors:

* Parameter 'name' implicitly has an 'any' type.
* Parameter 'age' implicitly has an 'any' type.
* salary' is declared but its value is never read.
* Parameter 'salary' implicitly has an 'any'
* Expected 3 arguments, but got 2.
* An argument for 'salary' was not provided.

As use see type any produce many errors, and if you take a deep look you will know that type “any” ruin the idea of Ts and static type checking and ……

But ‘any’ has some uses cases, and for this reason TS allow us to controll reporting on type ‘any’ on tsconfig.json using **noImplicitAny’ لا تقم بتضمين اني**

And know 1 errors stills :

* Expected 3 arguments, but got 2.
* An argument for 'salary' was not provided.

Althought ‘any’ produce no problem but under params we have some three dots,

Parameter 'name' implicitly has an 'any' type, but a better type may be inferred from usage

And a quick fix button, without view problem btn, because now ‘any’ wont produce any problem, quick fix : “infer type from usage”, this the result when doing quick fix:

function showDattails(name: string, age: number, salary: number) {

  if (showMsg) {

    return `Hello ${name},your age is ${age} and your salary is`;

  }

}

console.log(showDattails("Bilal", 17, 5000));

imagine that showMsg is false, so the function wont return any value, and this will not produce a problem, but TS give us the possibility of getting error in this case.

// "noImplicitReturns": true, /\* Enable error reporting for codepaths that do not explicitly return in a function. \*/

Let do an experience :

"noImplicitReturns": true

..

function showDattails(name: string, age: number, salary: number) {

  if (showMsg) {

    return `Hello ${name},your age is ${age} and your salary is`;

  }

}

// problem : Not all code paths return a value

// function showDattails(name: string, age: number, salary: number): string | undefined

As you can see, we got a problem, and take a look at type of returned value is *string* (condition is checked) or *undefined*

function showDattails(name: string, age: number, salary: number) {

  if (showMsg) {

    return `Hello ${name},your age is ${age} and your salary is`;

  }

  return `No Data To Show`;

}

// problem : no problem no error

// function showDattails(name: string, age: number, salary: number): string

Take a look at the next code :

function showDattails(name: string, age: number, salary: number) {

  let test = 10;

  if (showMsg) {

    return `Hello ${name},your age is ${age} and your salary is`;

  }

  return `No Data To Show`;

}

3 dots under test : 'test' is declared but its value is never read

Quick fix : will detete the var, because we do not use it.

But me, I want Ts to alert me, so I have to use noUnusedLocals

// "noUnusedLocals": true,       /\* Enable error reporting when local variables aren't read.

Let set it true and do an experience :

 "noUnusedLocals": true,

::

No TS alert us, and give us a problem : 'test' is declared but its value is never read.

In our func, you can noitce, that we don’t use salary params, for make TS alert us, we must use noUnusedParameters

// "noUnusedParameters": true,     /\* Raise an error when a function parameter isn't read. \*/

If you turn true TS will alert you, and give a problem

Problem : 'salary' is declared but its value is never read

function showDattails(name: string, age: number, salary: number) {

  if (showMsg) {

    return `Hello ${name},your age is ${age} and your salary is ${salary}`;

  }

  return `No Data To Show`;

}// problem : no problem no error

And now as we said, type annotation we use for input and output function

We alerady see the type annotation in a function input, and now output function

function showDattails(name: string, age: number, salary: number): string {

  if (showMsg) {

    return `Hello ${name},your age is ${age} and your salary is ${salary}`;

  }

  return `No Data To Show`;

}

## Optional and Default Parameters

Take a look at the next function

function showData(name , age, country ) {

  return `${name} ${age} ${country}`;

}

Imagine that one of them isn’t gived as an arg, so here we use default param

function showData(name = "Unknonw", age, country) {

  return `${name} ${age} ${country}`;

}

But can I skip the first argument when I call the funcition

console.log(showData(undefined, 17, "Egypt"));

but imagine that you will make an argument optional, without using dafault parameter

By default TypeScript will assume all parameters are required, but they can be explicitly marked as optional using the ‘?’ operator

function showData(name: string, age: number, country?: string) {

  return `${name} ${age} ${country}`;

}

console.log(showData("Bilal", 17));

// i dont get an error because country is optional

## Function Rest Parameter

function addAll(...nums: number[]): number {

  let result = 0;

  // let n of x => n is an element in nums

  // let n in x => n is an index element in nums

  for (let number of nums) result += number;

  return result;

}

console.log(addAll(2, 6, 8, 9, 7, +true));

## Anonymous And Arrow Function

const add = function (num1: number, num2: number) {

  return num1 + num2;

};

console.log(add(16, 16));

const addWithArrow = (num1: number, num2: number) => num1 + num2;

console.log(addWithArrow(15, 16));

from assignment :

"allowUnreachableCode": false /\* Disable error reporting for unreachable code. \*/,

## Data Types – Type Alias

Type aliases *create a new name for a type*.

type st = string;

let theName: st = "Bilal";

theName = true; // Here we got an error

// Type 'boolean' is not assignable to type 'string'.

You can also

type sn = string | number;

let all: sn = "Bilal";

all = 17;

all = true; // Type 'boolean' is not assignable to type 'sn'.

## Data Types – Type Alias Advanced

TypeScript allows types to be defined separately from the variables that use them.

But it necessarly to declare object key and use them all

type Buttons = {

  up: string;

  right: string;

  down: string;

  left: string;

  low: number;

};

const btn: Buttons = {

  up: "j",

  right: "",

  down: "",

  left: "",

  low: 6,

};

You will get and error if you delete in key of them, take a look at the next code:

const btn: Buttons = {

  up: "j",

  right: "",

  down: "",

  left: "",

};

/\* Property 'low' is missing in type '{ up: string; right: string; down: string;

   left: string; }' but required in type 'Buttons' \*/

but the solution is very easy, do you know what is it: make low an optional key

type Buttons = {

  up: string;

  right: string;

  down: string;

  left: string;

  low?: number;

};

const btn: Buttons = {

  up: "j",

  right: "",

  down: "",

  left: "",

};

/\* there no error now\*/

Another Examples :

type user = {

  id: number;

  fullName: string;

};

const user1 = {

  id: 4589265,

  fullName: "Bilal Elemrani",

};

console.log(showInfo(user1));

// Hello Bilal Elemrani, You ID : 4589265

function showInfo(user: user) {

  return `Hello ${user.fullName}, You ID : ${user.id}`;

}

## Data Types - Literal Types

Pls focus in the next function :

function compare(num1: number, num2: number): number {

  if (num1 === num2) return 0;

  else if (num1 > num2) return 1;

  else return -1;

}

console.log(compare(25, 50)); // -1

as you the previous function return number, but for being more exact it return 0 or 1 or -1.

A literal type is more exact that sub-type of a collective type, what this means is that “Hello World” is a stirng, but a string is not “Hello world”

function compare(num1: number, num2: number): 0 | 1 | -1 {

  if (num1 === num2) return 0;

  else if (num1 > num2) return 1;

  else return -1;

}

Let make for them an aliase and test it:

type nums = 0 | 1 | -1;

function compare(num1: number, num2: number): nums {

  if (num1 === num2) return 0;

  else if (num1 > num2) return 1;

  else return -1;

}

let x: nums;

x = 1;

x = 2;

// error : Type '2' is not assignable to type 'nums'

## Data Types - Tuple

A tuple is a typed array, with a pre-defined length and typees for each index.

Tuples are great because they allow each element in the array in the array to be a known type of value.

To define a tuple, specify the type of each element in the array

let article: [number, string, boolean] = [11, "Title One", true];

article = [12, "Title Two", false];

console.log(article); // [ 12, 'Title Two', false ]

and you will get an error if you try to initialize the array with more that the declared length

let article: [number, string, boolean] = [11, "Title One", true];

article = [12, "Title Two", false, "news"];

console.log(article);

// Type '[number, string, false, string]' is not assignable to type '[number, string, boolean]'.

 // Source has 4 element(s) but target allows only 3.

Hey but take a look to that :

let article: [number, string, boolean] = [11, "Title One", true];

article = [12, "Title Two", false];

article.push("news");

console.log(article); // [ 12, 'Title Two', false, 'news' ]

for avoiding this problem, you have to use readonly property

let article: readonly [number, string, boolean] = [11, "Title One", true];

article = [12, "Title Two", false];

article.push("news"); // Property 'push' does not exist on type 'readonly [number, string, boolean]'

## Data Types – Void And Never

TypeScript has special types that may not refer to any specific type of data.

**Type : any**

Any is a type that disables type chacking and effectively allows all types to be used.

Red Note : *any can be a useful way to get past errors since it disables type checking, but TypeScript will not be able to provide type safetly.*

**Type : Unkown**

Unknown is a similar, but safer alt to any

The main difference between “unkonwn” and “any” in TypeScript is that “unkonw” is a type-safe counterpart of “any”. “un” requires type-checking of type-casting before using a value, providing more type safetly. On the other hand , “any” is the least type-safe type, allowing any opertation without type-checking.

let x: unknown;

x = "Bilal";

x = 5;

// we want to make an operation on x so type checking is requied

if (typeof x === "number") {

  x++;

  // also when i make x. "vs code " show me all number properties.

}

x = x + 5;

// error :'x' is of type 'unknown'

// meaning i need type checking before any operation

So as you saw, ts depends a lot on type checking, and the next will proof that more :

let x: unknown;

x = 5;

// we want to make an operation on x so type checking is requied

if (typeof x === "string") {

    x.

  // also when i make x. "vs code " show me all string properties.

}

But also he has attention :

let x: unknown;

x = 5;

// we want to make an operation on x so type checking is requied

if (typeof x === "string") {

  x++;

  // error : An arithmetic operand must be of type 'any', 'number', 'bigint' or an enum type.

}

**Type : void**

The void type denotes the absence of having any type at all. It is a little like the opposite of the any type.

Typically, you use the void type as the return type of fuctions that no return a value for example :

function logging(msg: string): void {

  console.log(msg);

}

It is a good practice to to add the void type as the return type of a function or a method that does not return any value, By doing this, you can gain the following benefits :

* Imporve clarity of the code: you do not have to read the whole function body to see if it returns anytings
* Ensure type-safe : you will never assign the function with the void return type to a variable.

Note :

* Functions is JS that not return a value will show undefined.
* Undefined is not void

## Enums

An enum is a special “class” that represents a group of constants (unchangeable variables)

An enum, short for enumeration, is a way to define a set of named values. It allows you to create a collection of related constants that can be assigned to variables or used as a type

Enums come in two flavors *string* and *numeric*. Lets start with numeric

**Numeric Enums**

**Numeric Enums – Default**

By default, enums will initialize the first value to 0 and add 1 to each additional value.

enum Direction {

  North,

  South,

  East,

  West,

}

console.log(Direction.North); // 0

console.log(Direction.West); // 3

**Numeric Enums – Initialzed**

You can set the value of the first numeric enum and have it auto increment from that

enum Direction {

  North = 5,

  South,

  East,

  West,

}

console.log(Direction.North); // 5

console.log(Direction.West); // 8

**Numeric Enums – Fully Initialzed**

You can assign unique numer values for each enum value. Then the values will not incremented automatically :

enum StatusCodes {

  NotFound = 404,

  Success = 200,

  Accepted = 202,

  badRequest = 400,

}

console.log(StatusCodes.Accepted); // 202

console.log(StatusCodes.NotFound); // 404

**String Enums**

Enums can also contain strings. This is more common than numeric enums, because of their readability:

enum GameCoins {

  Apple = 2,

  snake = -3,

  water = 1,

}

let score = 0;

let handleMovement = (movement: string) => {

  if (movement === "apple") return GameCoins.Apple;

  else if (movement === "snake") return GameCoins.snake;

  else return GameCoins.water;

};

score += handleMovement("snake");

console.log(score); // -3

score += handleMovement("apple");

console.log(score); // -1

score += handleMovement("apple");

console.log(score); // 1

score += handleMovement("water");

console.log(score); // 2

**Red Note :** Technically, you can mix and match string and numeric enum values, but it is recommnded not to do so