**Typescript**

**Getting Started**

1. [Introduction To TypeScript](https://www.tektutorialshub.com/typescript/introduction-to-typescript-what-is-typescript/)
2. [TypeScript Installation and Environment Setup](https://www.tektutorialshub.com/typescript/typescript-installation-environment-setup/)
3. [TypeScript Hello World Example](https://www.tektutorialshub.com/typescript/typescript-getting-started-with-hello-world-example/)
4. [Enable Compile on Save in Visual Studio Code](https://www.tektutorialshub.com/typescript/typescript-compile-on-save-in-visual-studio-code/)
5. [Syntax and Basic Rules](https://www.tektutorialshub.com/typescript/typescript-syntax/)

**Typescript Basics**

1. Typescript Data Types
2. Type Annotations
3. Variable Declaration
4. Identifiers & keywords
5. Variable Scope
6. Let, Var & Const
7. Constants
8. Type Inference

**Strings**

1. [String Data Type](https://www.tektutorialshub.com/typescript/typescript-string/)
2. [Template strings/Literal strings](https://www.tektutorialshub.com/typescript/template-literals-template-strings-in-typescript/)
3. [Tagged Templates](https://www.tektutorialshub.com/typescript/typescript-tagged-templates/)

**Numbers**

1. [Number Data Type](https://www.tektutorialshub.com/typescript/typescript-number/)
2. [NaN in Typescript](https://www.tektutorialshub.com/typescript/nan-in-typescript/)
3. [Min, Max & Safe Values](https://www.tektutorialshub.com/typescript/typescript-number-min-max-safe-values/)
4. [EPSILON & Floating Point Precision](https://www.tektutorialshub.com/typescript/typescript-epsilon-floating-point-precision/)
5. [Infinity](https://www.tektutorialshub.com/typescript/typescript-infinity/)

**BigInt**

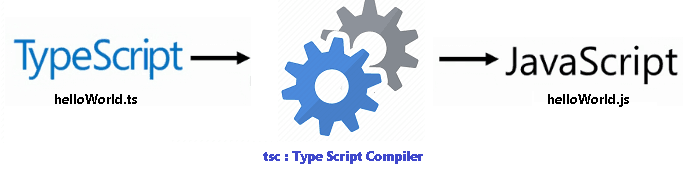
1. [BigInt data type](https://www.tektutorialshub.com/typescript/typescript-bigint/)
2. [BigInt Vs Number](https://www.tektutorialshub.com/typescript/typescript-bigint-vs-number/)

**Boolean**

1. [Boolean Data Type](https://www.tektutorialshub.com/typescript/typescript-boolean/)
2. **Introduction to typescript**

Typescript is superset language written over javascript .It added only syntactic suger over javascript to add static type instead dynamic type of javascript .

we can write the javascript and Still typescript still compile it just as it is javascriprt only .



**Note : TypeScript** is a primary language used in **Angular** application development.

## Benefits of TypeScript

Since in the end Typescript produces Javascript, you may wonder why not use javascript directly. The Benefit of TypeScript comes with its type system, which provides several benefits over javascript.

* 1. Optional Type System

TypeScript provides the static type system which provides great help in catching programming errors at compile time.

Javascript is a dynamically typed system. The variables can hold any values. The type of variable is determined on the fly. The javascript implicitly converts types for example string to a number. This is ok for a small app, but large apps this can be a lot of headaches. It is difficult to test to see if the proper types are passed and errors always happen at runtime.

* 1. Intellisense & syntax checking

The static Type system helps in provide better tooling support in IDE. The intellisense, syntax checking & code completion are few of the major benefits you get with the tooling support. This speeds up the development time and also ensures that the programmers make fewer mistakes with typos. All the major editors like VSCode, atom, sublime text includes the tooling support for Typescript.

* 1. Maintainable code

Typescript brings Types, Classes, interfaces & modules. it makes the code more maintainable and scalable. It much easier to organize the Typescript code, than a Javascript code.

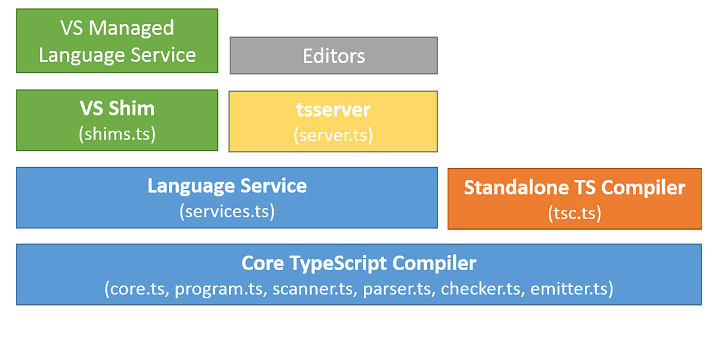
* 1. Language Enhancement

Typescript comes with several language features. It supports Encapsulation through classes and modules. Supports constructors, properties & functions. It has support for Interfaces. You can make use of Arrow functions or lambdas or anonymous functions.

The Typescript is still Javascript. It retains most of the javascript syntax. This means that all valid JavaScript code is valid TypeScript code. Hence the learning curve is lean. If you are not aware of OOP concepts like classes, interfaces, etc, you may need to learn them to get the best out of Typescript

## Typescript Components

The architecture of TypeScript is neatly organized in different layers as shown in the image below. The three major layers are



a.Core TypeScript Compiler.

b.Typescript Standalone Compiler

c. Typescript Language Services

Core TypeScript Compiler

The TypeScript compiler manages the task of type-checking our code and converting it into valid JavaScript code. The compiler is made up of several different layers like core, program, scanner, parser, checker & emitter, etc

Standalone compiler (tsc)

The batch compilation CLI. Mainly handle reading and writing files for different supported engines (e.g. Node.js)

Language Service

The Language Service supports the editors and other tools to provide better assistance in implementing features such as IntelliSense, code completion, formatting and outlining, colorization, code re-factoring like rename, Debugging interface helpers like validating breakpoints, etc.

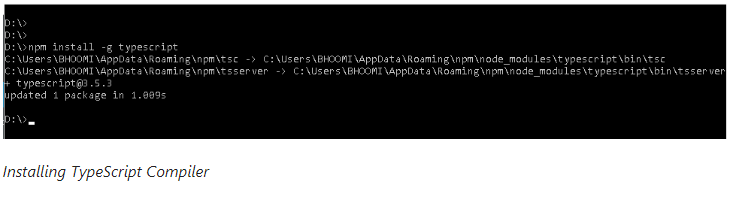
1. **Installation guide**

The Typescript is installed using the npm package manager or NPM. It is a default package manager for Node.js and is installed when the Node.js is installed.

You can download and install Node.js from downloading it from the official website https://nodejs.org/.

Once you install the Node, you can install the typescript compiler using the following command from the command prompt.

Npm install –g typescript



Installing Visual Studio Code

Next, we will need a Code editor. You have many choices here. For this tutorial, we will use Visual Studio Code.

You can download it from the official <https://code.visualstudio.com/download>

1. **Hello world first program using typescriprt**

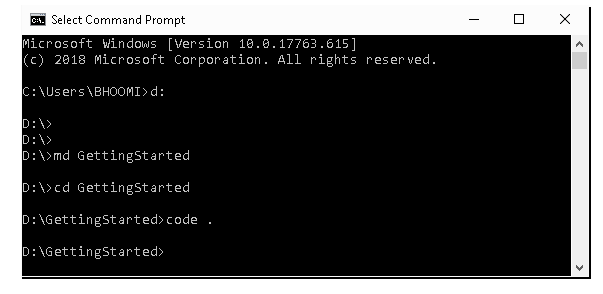
Typescript Hello World Example

Open the command prompt and create a folder GettingStarted and then cd into it.

md GettingStarted

cd GettingStarted

Now, Open Visual Studio Code or any text editor. Go to the GettingStarted folder**.**



You can open the Visual Studio Code just by typing the following command in the GettingStarted folder

code .

Copy the following code and save the file as helloWorld1.ts



The message is a variable of type string. A variable is a storage for the data. “message” is the name of the variable. In the code above, it stores the value “Hello World”, which is a string

We use let(or var) to create variables in typescript. You will learn more about the variable declaration in our next tutorial

console.log() writes the whatever the message passed onto it to the console.

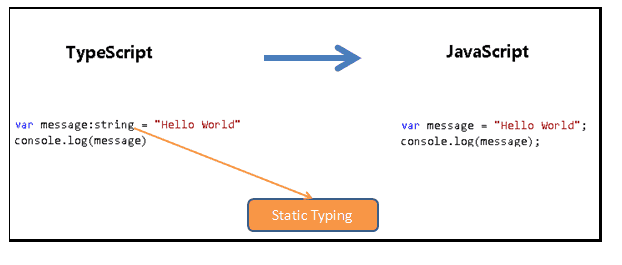
Compile/Execute TypeScript Programs

Now, go back to the command prompt or click on View -> Terminal in Visual Studio Code menu, which will open the PowerShell terminal window

Typescript codes are compiled using the Typescript compiler ie. tsc. Run the following command to compile

tsc helloWorld1

The tsc will look for the helloWorld1.ts file and compiles and generates the helloWorld1.js file. Open the file and inspect its content.



Running the Hello World using nodeJs

We will use the node.js to execute it. Run the following in the command window

node helloWorld1



Running TypeScript in the web app

Create the new typescript file hellowWorld2.ts

|  |  |
| --- | --- |
| Letmessage: string = "Hello World"  alert(message) |  |

The only change we have from the above code is we removed console.log() to alert(),

The alert() method displays the message passed onto it in a window, with an **ok** button. This method works only in the browser, hence we did not use in the previous code.

Compile the helloWorld2 as shown below



The above will generate the helloWorld2.js file. Now you can refer this in your HTML file. Create the index.html with the script tag pointing to the hellowWorld2.js



|  |  |
| --- | --- |
|  |  |

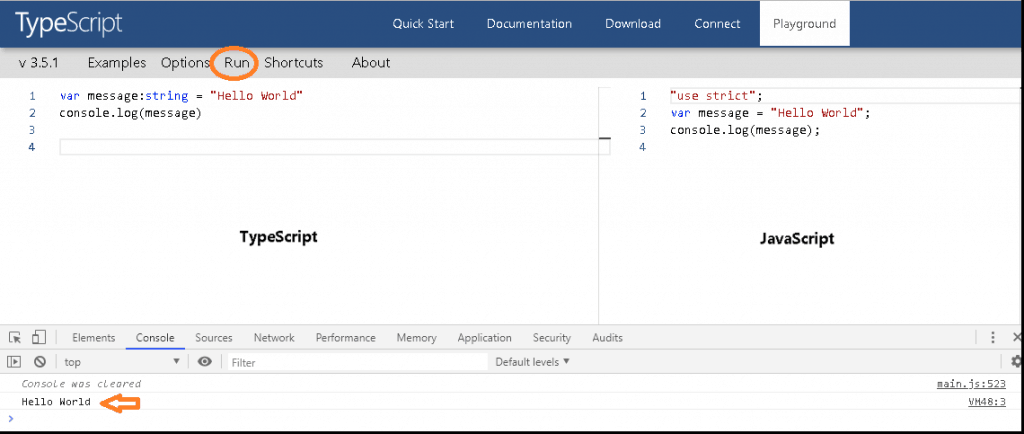
Open the file using the chrome or any browser, you should be able to see the alert window.

Running in Typescript Playground

You can also run typescript in using the TypeScript Playground. Head over to Typescript Playground <https://www.typescriptlang.org/play/index.html#code/G4QwTgBAtgpgznEBzGAuOAXMBLAdkiAXggCIAJGAG0oHsIB1GsSgExIgCgBjG3OGyjAB0tJAApYCZDACUHDkA>

As you type the code, you will see the javascript code is automatically generated in the right-hand side pane.

Open the Developer Console (ctrl-shit-I in chrome) and click on run and see the Hello World appearing in the console window.



## Compile Error

As we mentioned in the previous article on, the typescript compiler warns us if there are any **type** mismatches.

For Example, create helloWorld3.ts and add the following code. Here we are assigning a string to a number variable.



Instantly the Visual Studio Code underlines the message in red, indicating the error in the code. Also, when you compile, the compiler will throw the following error.





|  |  |
| --- | --- |
|  |  |

Typescript will generate the javascript code even if there is a compile error. The helloWorld3.ts shown above contains an error but the helloWorld3.js is still produced. This is intentional. This allows us to export our javascript code to typescript and progressively update it to Typescript.

1. [**Enable Compile on Save in Visual Studio Code**](https://www.tektutorialshub.com/typescript/typescript-compile-on-save-in-visual-studio-code/)

Weinstructs the VS to auto save and auto compile after each modification in the ts file .in this case we don’t save and compile again the ts file again.

**Typescript configuration file**

To Enable Compile on save feature, we first need to create the Typescript Configuration file tsconfig.json.

The tsconfig.json is the file where TypeScript Compiler looks for project configuration and compiler options.

The easiest way to create the tsconfig.json file is to use the following command

tsc --init

This will create tsconfig.json. You can look at the generated file. It has listed all the possible compiler option, with almost all of them commented out.

Another way is to just to create the tsconfig.json and copy the following content.

{

  "compilerOptions": {

      "target": "es5",

      "module": "commonjs",

  }

}

**Compile on Save**

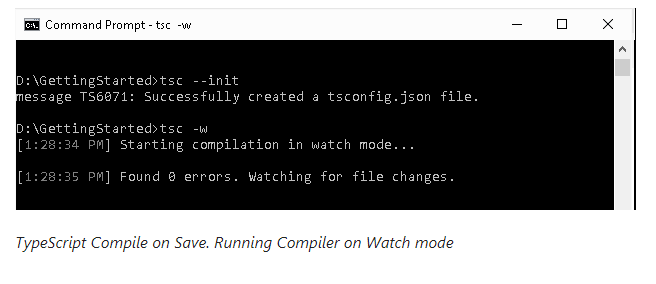
Once you created the tsconfig.json, You can enable compile on save feature. There are two ways you can achieve this result.

**Running the tsc in watch mode**

Open the command prompt and cd into the project folder or Click on View -> Terminal in the VS Code Menu. In the command prompt run the following command

Tsc –w

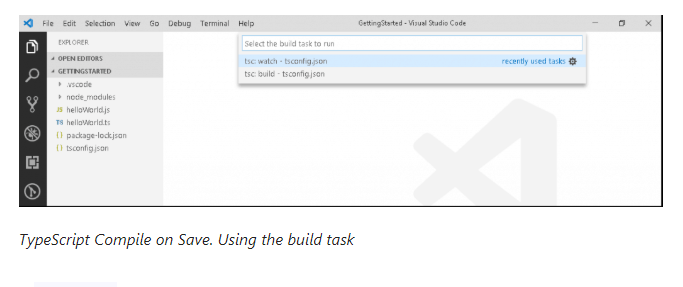
This will watch the folder for changes made to our typescript file and immediate run the compiler.



**Using the Build Task**

The second option is to hit the (Ctrl+Shift+B) from the VS Code. This will open the Execute Run Build Task window

Select tsc: watch. This will make the TypeScript compiler to watch for changes made to the TypeScript file and run the compiler on each change



1. **Syntax**

TypeScript is Case-sensitive

TypeScript is case sensitive. This means that foo is not the same as Foo.

Typescript Statements

A Typescript statement is an instruction to perform a specific action. A Typical Typescript program consists of several such sequences of statements and they control the flow of the program.

Here is an example of three statements

//statement 1 create a variable

var message;

//statement 2 assigns “Hello World” to message variable

message=”hello world”;

//statmenent 3 print out console log

console.log(message);

//statement 1 create a variable

var message;

//statement 2 assigns “Hello World” to message variable

message=”hello world”;

//statmenent 3 print out console log

console.log(message);

A single statement may span multiple lines.

Or you can write multiple statements in a single line, provided you separate each statement by a semicolon.

Semi-Colon

; the semicolon is used to indicate the end of a statement.

For Example

var message;

message=”hello world”;

console.log(message);

But, they are optional if you use a single line for each statement

For Example, this is also valid

var message

message=”hello world”

console.log(message)

But, they are required if you have multiple statements in a line

For Example

var message ; message=”hello world” ; console.log(message) ;

TypeScript Expressions

Expressions are units of code that produces value. They can appear anywhere in the code. They can be part of the function arguments or right side of an assignment, etc

Example of Expressions

5+7 //This is an arithmetic expression that evaluates to 12

I++ //Arithmetic expression

'Something' //Primary Expressions

a && b // Logical Expressions

The expressions can be of various types Arithmetic, String, Primary, Array & object Initialisation, Logical, Left-Hand side. Property access, object Creation, Function definition, invocation, etc.

Whitespace and Line Breaks

You can add spaces, tabs, and newline characters anywhere in the Typescript Program. The Compiler will ignore them. You can use them to indent your code so that it very easily readable.

Comments in TypeScript

The Comments can be applied to the line of code or to a block of code

Single-line comments ( // ) − Any text between a // and the end of a line is treated as a comment.

Example

//this is a single-line comment

Multi-line comments (/\* \*/) − These comments may span multiple lines.

Example

/\* This is a

Multi-line comment

\*/

**Typescript Basics**

1. Type script data type
2. Type Annotations
3. Type script data type
4. Variable Declaration
5. Identifiers & keywords
6. Variable Scope
7. Let, Var & Const
8. Constants
9. Type Inference

**What is a Type?**

The Type or the Data Type is an attribute of data which tells us what kind of value the data has. Whether it is a number, string, boolean, etc. The type of data defines the operations that we can do on that data. For example, you can perform arithmetic operations only on numbers and not on a string. The most commonly used data types are number, string, date, boolean, etc.

**Typescript Types**

TypeScript types introduce strong typing to JavaScript. The use of types is optional, but if use the types are analyzed by the compiler and errors are thrown. let us see it using an example.

Open Visual Studio Code and create a new file types.ts. If you are new to typescript you can refer to the tutorial Hello world example in Typescript

let num1 //no type is defined.

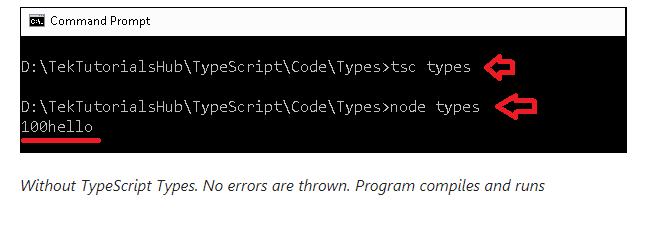
let num2

num1=100;

num2=”hello”

console.log(num1+num2)

We have declared two-variable num1 & num2. We want both the variable to hold numbers. But in the subsequent line, we assign a string to num2 variable instead of a number. Now open the command window and compile and run as shown below.



The program does not throw any errors. The result “100hello” is printed in the console as shown above

Now, modify the code and bring the Typescript Types as shown below. We are declaring that the num1 & num2 will hold the type number.

let num1 :number //defining type

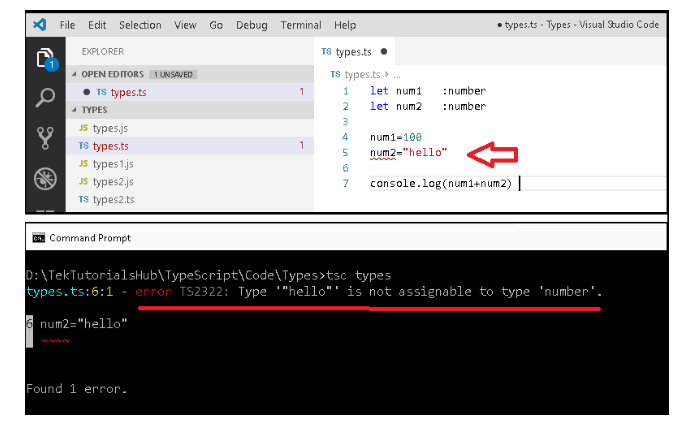
let num2 :number

num1=100

num2="hello"

console.log(num1+num2)

The visual Studio code instantly underlines the code with red as soon as we use the wrong type. We also have an error thrown at us at the time of compile as shown in the image below.



The number is one of the many data types that Typescript supports. but at the top of all types is any type. The any type is the base type for all other types.

any

any data type can hold any data. You can change the data type. We use this when we do not know the type of data. any is specific to typescript.

When a variable’s type is not given and typescript cannot infer its type from the initialization then it will be treated as an any type.

let notSure: any = 4; //defined as any

notSure = "maybe a string instead";

notSure = false;

let stillNotSure //No Type specifued. infered as any

stillNotSure=5

stillNotSure="may be a strng instead"

stillNotSure=false;

let notSure: any = 4; //defined as any

notSure = "maybe a string instead";

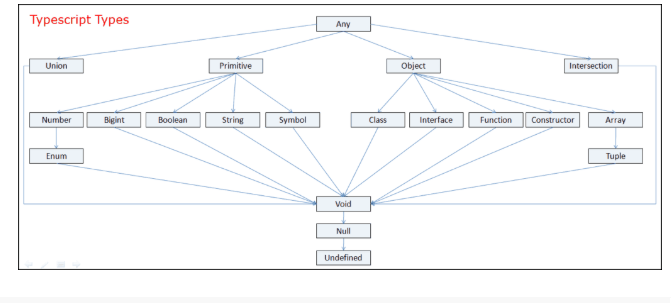
notSure = false;

let stillNotSure //No Type specifued. infered as any

stillNotSure=5

stillNotSure="may be a strng instead"

stillNotSure=false;



**Primitive Types**

TypeScript supports 7 primitive types number, string, boolean, bigint, symbol, undefined, and null. All other data types are objects in Typescript. A primitive data type is a data type that is not an object and has no methods. All primitives are immutable.

**string**

We use the string data type to store textual data. The string value is enclosed in double-quotes (“) or single quotes (‘).

Example

let message=”Hello World”

let color=”red"

**Multiline string**

The strings can span multiple lines in such cases the strings are surrounded by the backtick/backquote (`) character

let sentence: string =`Hello, welcome to the world of typescript,

the typed super of javascript`

**boolean**

The boolean type is a simple true/false value

Example

let isDone: boolean = false;

**number**

The number data type in TypeScript are 64-bit floating-point values and are used to represent integers and fractions. Typescript also supports the hexadecimal and decimal literals. It also supports the binary and octal literals introduced in ECMAScript 2015.

let decimal: number = 10;

let hex: number = 0xa00d; //hexadecimal number starts with 0x

let binary: number = 0b1010; //binary number starts with 0b

let octal: number = 0o633; //octal number starts with 0c

**bigint**

bigint is the new introduction in Typescript 3.2. This will provide a way to represent whole numbers larger than 253. You can get a bigint by calling the BigInt() function or by writing out a BigInt literal by adding an n to the end of any integer numeric literal as shown below.

let big1: bigint = BigInt(100); // the BigInt function

let big2: bigint = 100n; // a BigInt literal. end with n

You can use the bigint only if you are targeting version ESNext.

Remarks: what is ESNext

ESNext is a name that always indicates the next version of JavaScript.

The current ECMAScript version is ES2018. It was released in June 2018.

Proposals to the ECMAScript standard are organized in stages. Stages 1–3 are an incubator of new features, and features reaching Stage 4 are finalized as part of the new standard

**null and undefined**

The javascript has two ways to refer to the null. They are null and undefined and are two separate data types in Typescript as well. The null and undefined are subtypes of all other types. That means you can assign null and undefined to something like a number.

let u: undefined = undefined;

let n: null = null;

undefined Represents the intentional absence of object value.

null: Denotes value given to all uninitialized variables.

**symbol**

The symbol is the new primitive type introduced in ES6 and represents the javaScript symbol primitive type. it represents unique tokens that may be used as keys for object properties. it is created by the global Symbol() function. Each time the Symbol() function is called, a new unique symbol is returned.

**Typescript Special Types**

**never**

The never type represents the value which will never happen. We use it as the return type of a function, which does not return value. For example, the function that always throws an exception as shown below.

function doSomething(): never {

throw new Error("Error has occurred");

}

**void**

Void represents the absence of any return value. For example, the following function prints “hello” and returns without returning a value. Hence the return type is void.

It is different from never. never means it never returns a value.

function sayHello(): void {

console.log('Hello!')

}

**Object Types**

Everything that isn’t a primitive type in TypeScript is a subclass of the object type. Examples are class, Interface, function, constructor, array, tuple, etc. We will cover these in detail in the subsequent chapters

**Strings**

1. [Template strings/Literal strings](https://www.tektutorialshub.com/typescript/template-literals-template-strings-in-typescript/)
2. String Datatype
3. [Tagged Templates](https://www.tektutorialshub.com/typescript/typescript-tagged-templates/)