**Typescript:**

Typescript is an open-source object-oriented language developed and maintained by Microsoft, licensed under Apache 2 license. It is a typed superset of Javascript that compiles to plain JavaScript.

**Why TypeScript**?

JavaScript has seen a vast development in the last few years. It is the most versatile cross-platform language that is used to develop modern web applications. It can be used to develop both the client side of an application, with frameworks like Angular or React.js, as well as the server side, with frameworks such as Node.js. It is now being considered the most opted for language to build end-to-end applications. But, JavaScript was never meant for such large-scale application development. JavaScript is a dynamic programming language with no type system, unlike well-structured and refined languages such as C# or Java. A *no type system* means that a variable in JavaScript can have any type of value such as string, number, boolean etc. The type system increases the code quality, readability and makes it an easy to maintain and refactor code base. More importantly, errors can be caught at compile time rather than at run time.

Without the type system, it is difficult to scale JavaScript to build complex applications with large teams working on the same code. Hence, the reason to use TypeScript is that it allows JavaScript to be used at scale. TypeScript compiles into simple JavaScript. The TypeScript compiler is also implemented in TypeScript and can be used with any browser or JavaScript engines like Node.js. TypeScript needs an ECMAScript 3 or higher compatible environment to compile. This is a condition met by all browsers and JavaScript engines today. Some of the most popular JavaScript frameworks like Angular.js and WinJS are written in TypeScript.

**How to use TypeScript?**

A Typescript code is written in a file with .ts extension and then compiled into JavaScript using the TypeScript compiler. A TypeScript file can be written in any code editor. A TypeScript compiler needs to be installed on your platform. Once installed, the command tsc <filename>.ts compiles the TypeScript code into a plain JavaScript file. JavaScript files can then be included in the HTML and run on any browser.

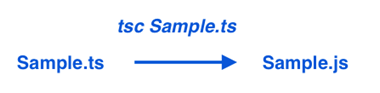
[](https://www.tutorialsteacher.com/Content/images/typescript/typescript-workflow.png)

Fig. Compile TypeScript to JavaScript

TypeScript installation for various platforms and the compilation process is detailed in the next section, along with the code examples.

# TypeScript - Setup Development Environment

In this chapter, you will learn how to install TypeScript, create and build your first TypeScript code, and compile your code using the TypeScript compiler.

There are two ways to install TypeScript:

1. Install TypeScript using [Node.js package manager](https://www.tutorialsteacher.com/nodejs/what-is-node-package-manager) (npm).
2. Install the TypeScript Plug-in in your IDE (Integrated Development Environment).

## Install TypeScript using Node.js Package Manager

Node.js package manager is used to set up TypeScript on your local environment.

To run TypeScript locally you will need: Node.js, TypeScript, and a code editor of your choice. Learn how to install Node.js in the [install Node.js](https://www.tutorialsteacher.com/nodejs/setup-nodejs-development-environment) chapter.

We will be using Node.js package manager (npm) to install TypeScript. To install TypeScript, open command prompt on Windows and type the following command:

> npm install -g typescript

Once installed, you can check the TypeScript version using the following command:

> tsc -v   
Version 2.8.1

As of writing this tutorial, the latest TypeScript version is 2.8.1.

## Install TypeScript Plug-in in your IDE

You will write your TypeScript programs in your IDE and save your file with .tsextension. You can use any IDE of your choice such as Visual Studio, Sublime Text, Eclipse, Atom, WebStorm etc.

# TypeScript - First Program

In this chapter, we will write a simple program in TypeScript and compile it. We will also make some modifications to our program to see how JavaScript and TypeScript treat the code. We will see how TypeScript compiles the program and shows the errors, if any.

Let's write a simple program which adds two numbers and displays the output to the console.

Create a new file in your code editor and name it "add.ts". Write the following code in it:

File: add.ts

function addNumbers(a: number, b: number) {

return a + b;

}

var sum: number = addNumbers(10,15)

console.log('Sum of the two numbers is: ' +sum);

Now, open the command prompt on Windows (or a terminal in your platform), navigate to the path where you saved add.ts and compile the above TypeScript program in your terminal using the following command:

> tsc add.ts

This will compile your TypeScript program and create the following Javascript code in the file named add.js at the same location of add.ts:

File: add.js

function addNumbers(a, b) {

return a + b;

}

var sum = addNumbers(10, 15);

console.log('Sum of the two numbers is: ' + sum);

As you can see, the TypeScript compiler compiled the TypeScript code into plain Javascript. Thus, if you now include the add.js file in your test.html file and run it in the browser, you will get the following output:

Sum of the two numbers is: 25

**What is typed and untyped programming language?**

**Untyped languages**, also known as dynamically **typed languages**, are programming **languages** that do not make you define the type of a variable. JavaScript is **untyped language**. This means that a JavaScript variable can hold a value of any data type. Whereas Typescript is typed language.

# TypeScript - Type Annotations

JavaScript is not a typed language. It means we cannot specify the type of a variable such as number, string, boolean etc. However, TypeScript is a typed language, where we can specify the type of the variables, function parameters and object properties.

We can specify the type using: Type after the name of the variable, parameter or property. There can be a space after the colon. TypeScript includes all the primitive types of JavaScript- number, string and boolean.

The following example declares variables with different data types:

Example: Type Annotation in TypeScript

var age: number = 32; // number variable

var name: string = "John";// string variable

var isUpdated: boolean = true;// Boolean variable

In the above example, each variable is declared with their data type. These are type annotations. You cannot change the value using a different data type other than the declared data type of a variable. If you try to do so, TypeScript compiler will show an error. This helps in catching JavaScript errors. For example, if you assign a string to a variable age or a number to name in the above example, then it will give an error.

Type annotations are used to enforce type checking. It is not mandatory in TypeScript to use type annotations. However, type annotations help the compiler in checking types and helps avoid errors dealing with data types. It is also a good way of writing code for easier readability and maintenance by future developers working on your code.

We can still follow the JavaScript way of declaring variables and have the TypeScript compiler infer the data type of the variable.

The following example demonstrates the type annotation of paramters.

Example: Type Annotation of Parameters

function display(id:number, name:string)

{

console.log("Id = " + id + ", Name = " + name);

}

Similarly, we can declare an object with inline annotations for each of the properties of the object.

Example: Type Annotation in Object

var employee : {

id: number;

name: string;

};

employee = {

id: 100,

name : "John"

}

Here, we declare an object employee with two properties id and name with the data type number and string respectively.

If you try to assign a string value to id then the TypeScript compiler will give the following error.

error TS2322: Type '{ id: string; name: string; }' is not assignable to type

'{ id:number; name: string; }'.Types of property 'id' are incompatible.

Type 'string' is not assignable to type 'number'.

# TypeScript - Variable

TypeScript follows the same rules as JavaScript for variable declarations. Variables can be declared using: var, let, and const.

## var Declaration

Variables in TypeScript can be declared using var keyword, same as in JavaScript. The scoping rules remains the same as in JavaScript.

## let Declaration

To solve problems with **var** declarations, ES6 introduced two new types of variable declarations in JavaScript, using the keywords **let** and **const**. TypeScript, being a superset of JavaScript, also supports these new types of variable declarations.

Example: Variable Declaration using let

let employeeName = "John";

// or

let employeeName:string = "John";

The let declarations follow the same syntax as var declarations. Unlike variables declared with var, variables declared with let have a block-scope. This means that the scope of let variables is limited to their containing block, e.g. function, if else block or loop block. Consider the following example.

Example: let Variables Scope

let num1:number = 1;

function letDeclaration() {

let num2:number = 2;

if (num2 > num1) {

let num3: number = 3;

num3++;

}

while(num1 < num2) {

let num4: number = 4;

num1++;

}

console.log(num1); //OK

console.log(num2); //OK

console.log(num3); //Compiler Error: Cannot find name 'num3'

console.log(num4); //Compiler Error: Cannot find name 'num4'

}

letDeclaration();

In the above example, all the variables are declared using let. num3 is declared in the if block so its scope is limited to the if block and cannot be accessed out of the if block. In the same way, num4 is declared in the while block so it cannot be accessed out of while block. Thus, when accessing num3 and num4 else where will give a compiler error.

The same example with the var declaration is compiled without an error.

Example: var Variables Scope

var num1:number = 1;

function varDeclaration() {

var num2:number = 2;

if (num2 > num1) {

var num3: number = 3;

num3++;

}

while(num1 < num2) {

var num4: number = 4;

num1++;

}

console.log(num1); //2

console.log(num2); //2

console.log(num3); //4

console.log(num4); //4

}

varDeclaration();

### Advantages of using let over var

1) Block-scoped let variables cannot be read or written to before they are declared.

Example: let vs var

console.log(num1); // Compiler Error: error TS2448: Block-scoped variable 'num' used before its declaration

let num1:number = 10 ;

console.log(num2); // OK, Output: undefined

var num2:number = 10 ;

In the above example, the TypeScript compiler will give an error if we use variables before declaring them using let, whereas it won't give an error when using variables before declaring them using var.

2) Let variables cannot be re-declared

The TypeScript compiler will give an error when variables with the same name (case sensitive) are declared multiple times in the same block using let.

Example: Multiple Variables with the Same Name

let num:number = 1; // OK

let Num:number = 2;// OK

let NUM:number = 3;// OK

let NuM:number = 4;// OK

let num:number = 5;// Compiler Error: Cannot redeclared block-scoped variable 'num'

let Num:number = 6;// Compiler Error: Cannot redeclared block-scoped variable 'Num'

let NUM:number = 7;// Compiler Error: Cannot redeclared block-scoped variable 'NUM'

let NuM:number = 8;// Compiler Error: Cannot redeclared block-scoped variable 'NuM'

In the above example, the TypeScript compiler treats variable names as case sensitive. num is different than Num, so it won't give any error. However, it will give an error for the variables with the same name and case.

Variables with the same name and case can be declared in different blocks, as shown below.

Example: Same Variable Name in Different Blocks

let num:number = 1;

function demo() {

let num:number = 2;

if(true) {

let num:number = 3;

console.log(num); //Output: 3

}

console.log(num);//Output: 2

}

console.log(num); //Output: 1

demo();

Similarly, the compiler will give an error if we declare a variable that was already passed in as an argument to the function, as shown below.

function letDemo(a: number ) {

let a:number = 10 ; //Compiler Error: TS2300: Duplicate identifier 'a'

let b:number = 20 ;

return a + b ;

}

Thus, variables declared using let minimize the possibilities of runtime errors, as the compiler give compile-time errors. This increases the code readability and maintainability.

## Const Declaration

Variables can be declared using const similar to var or let declarations. The const makes a variable a constant where its value cannot be changed. Const variables have the same scoping rules as let variables.

Example: Const Variable

const num:number = 100;

num = 200; //Compiler Error: Cannot assign to 'num' because it is a constant or read-only property

Const variables must be declared and initialized in a single statement. Separate declaration and initialization is not supported.

const num:number; //Compiler Error: const declaration must be initialized

num = 100;

Const variables allow an object sub-properties to be changed but not the object structure.

Example: const Object

const playerCodes = {

player1 : 9,

player2 : 10,

player3 : 13,

player4 : 20

};

playerCodes.player2 = 11; // OK

playerCodes = { //Compiler Error: Cannot assign to playerCodes because it is a constant or read-only

player1 : 50, // Modified value

player2 : 10,

player3 : 13,

player4 : 20

};

Even if you try to change the object structure, the compiler will point this error out.

const playerCodes = {

player1: 9,

player2: 10,

player3: 13,

player4: 20

};

playerCodes = { //Compiler Error: Cannot assign to playerCodes because it is a constant or read-only

player1: 9,

player2: 10,

player3: 13,

player4: 20,

player5: 22

};

Full in detail tutorial link: <https://www.tutorialsteacher.com/typescript/typescript-number>