Angular 1.x Features: Released in Jun 2012

* A JavaScript MVW Framework
* Extends HTML by adding attributes, tags, and expressions
* Events Handling
* Powerful Data Binding
* Built-In Template Engine and Routing
* Form Validations and Animations
* Dependencies Injection etc.

Angular 2.x Features: Released in Sep 2016

* Modern, faster and highly scalable framework
* One framework for web, mobile and desktop apps
* Not an update to Angular1.x, but a complete rewrite
* Written in Typescript
* Simple and Expressive
* Web components based architecture
* Hierarchical Dependency Injection

**Important Information**

Angular 3.0 was not released since angular router package version was 3.0 in Angular 2. So development team skipped Angular version 3 and directly named it version 4 to maintain the same version number for all the Angular packages.

Angular 4.x Features: Released in Mar 2017

* Smaller and Faster
* Reduce the size of generated bundled code up to 60%
* Animation moved out to a separated package @angular/animations
* Supports for if/else statement
* Supports for email validator

Angular 5.x Features: Released in Nov 2017

* Make AOT the default
* Easier to build progressive web apps
* Type checking in templates
* Internationalized Number, Date, and Currency Pipes
* An update to Httpclient
* Better error messages
* Zone speed improvements

Angular 6.x Features: Released in May 2018

* ng add and ng update
* Bazel Compiler
* Angular Elements
* RxJS 6.0 Supports
* Better URL Serialization

Angular 7.x Features: Released in Oct 2018

* CLI Prompts
* Improve Application Performance
* Angular compilation options
* Dependency Updates
  + TypeScript 3.1
  + RxJS 6.3
  + Node 10

Angular 8.x Features: Released in May 2019

* Easier to read and debug the generated code at runtime
* Web Workers Support
* Lazy Loading Improvements
* Improvement to ngUpgrade
* Faster re-build
* Improved template type checking



he features of TypeScript are enforced only at compile-time.

Some of the compelling benefits of TypeScript are following

* Starts from JavaScript, Ends with JavaScript  
  The syntax of TypeScript is almost similar to JavaScript that most of the web developer already know. Existing JavaScript code can be used along with the TypeScript generated JavaScript code.
* Strong tools for large applications  
  Types enable developers to use existing development environment and practices like static checking, statement completion, refactoring etc.
* Scalable  
  TypeScript offers classes, interfaces, modules and data type that help to take advantage of existing robust programming language features. These code gets compiled into simple JavaScript code that runs in any browser.

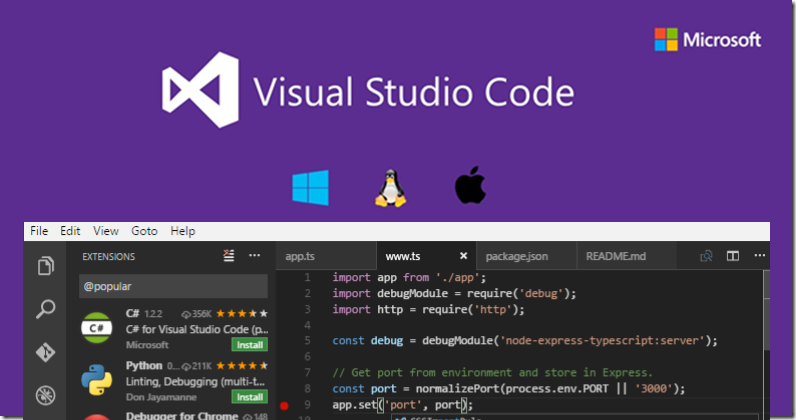
The importance of TypeScript is going to be more and more with the popularity of several JavaScript frameworks like Angular2, Knockout etc.

As you know TypeScript is developed by Microsoft, which is integrated into Visual Studio, a specialized IDE for developing .NET applications. If Visual Studio is available you can get started with TypeScript without any other installation.

* [Install Visual Studio 2015 on Windows](https://o7planning.org/10815/install-visual-studio-2015-on-windows)

Although Visual Studio is a great tool, it is very heavy and slow. You do not need to use IDE just to learn or develop small TypeScript applications. There are many lightweight source code editors that support TypeScript, such as Visual Studio Code, Atom, etc. However, you need a few additional installations.

Visual Studio Code is a free source-code editor for Windows, Linux and macOS. Visual Studio Code is developed by Microsoft. It is considered as a perfect combination between IDE and Code Editor.

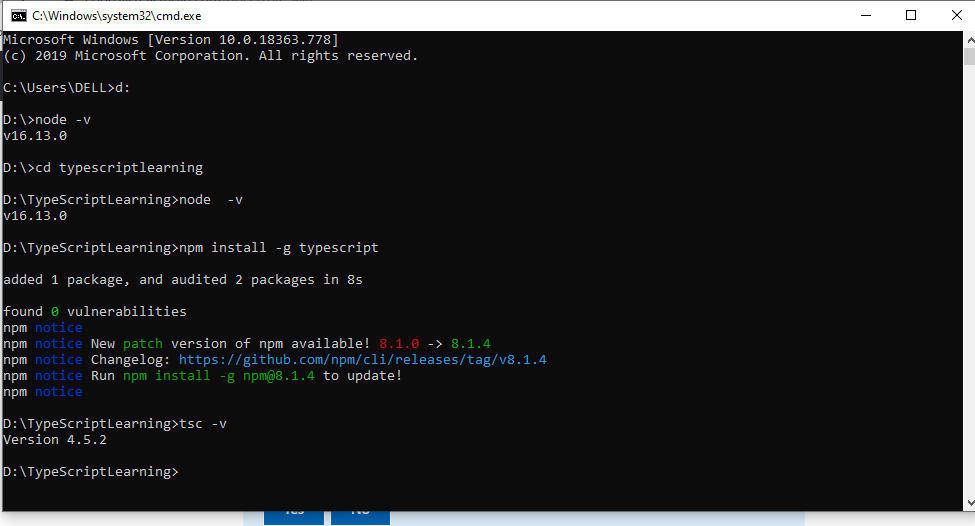


After being installed, by default Visual Studio Code supports many programming languages such as C/C++, C#, F#, Visual Basic, HTML, CSS, JavaScript, etc. In case you want to use another programming language which is not listed above, you can download an extension. There are many extensions written for Visual Studio Code.

To sum up, here are some reasons you should use Visual Studio Code:

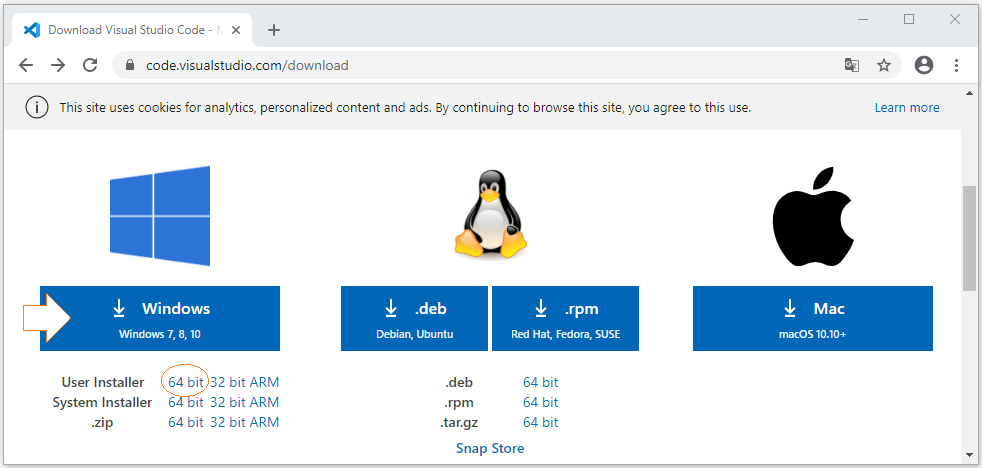
* Multi-platform support: Windows, Linux, Mac
* Multi-language support: C/C++, C#, F#, Visual Basic, HTML, CSS, JavaScript, JSON, etc.
* Small size
* Powerful features
* Professional Intellisense
* User-friendly interface

*IntelliSense is an abbreviation of Intelligent Sense. A source-code editor featuring IntelliSense means that it is able to recommend code to the user, detect incomplete code, or even supplement missing syntax. Thanks to this, Visual Studio Code is really outstanding.*



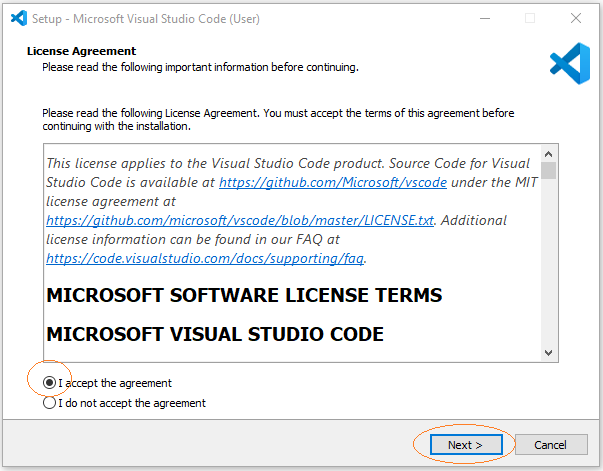
2- Downlooad Visual Studio Code

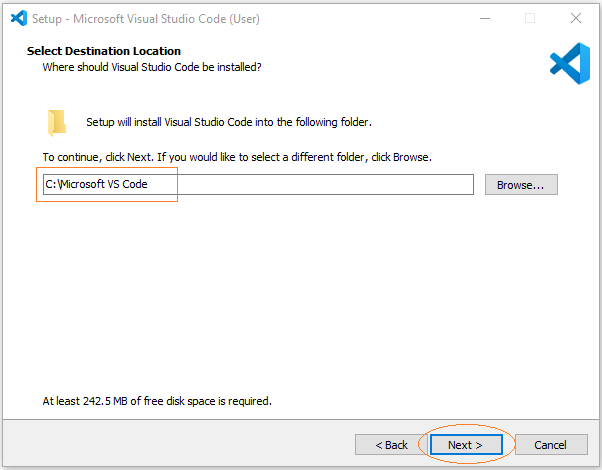
* <https://code.visualstudio.com/download>

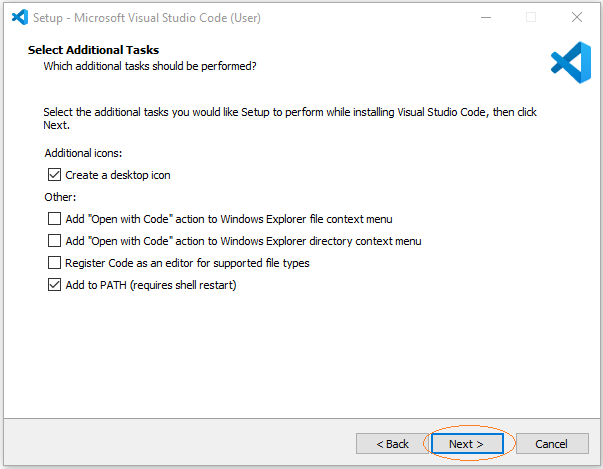


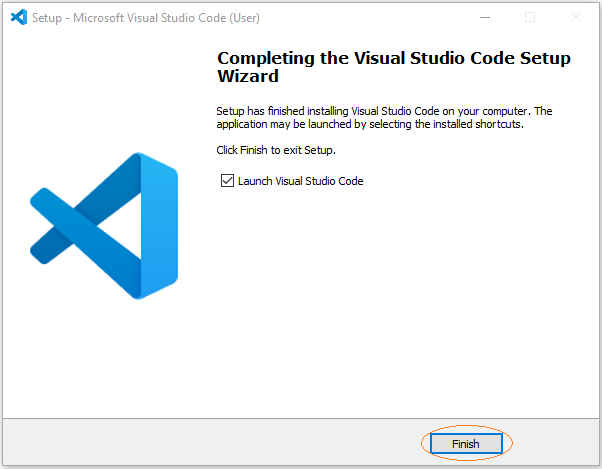
3- Visual Studio Code Installation

Installing Visual Studio Code is really simple. What you need is just a few mouse clicks like the illustrations below:

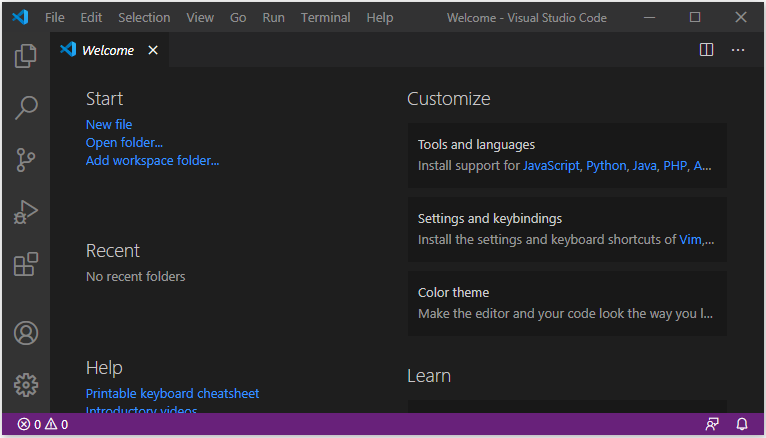








There you go, Visual Studio Code has been successfully installed. Here is the image of its first launch:



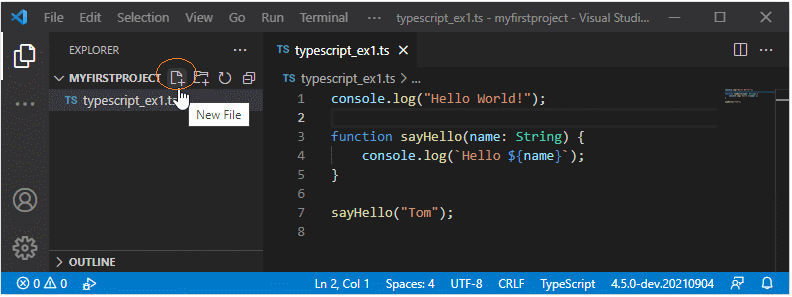
On your computer create a new empty folder to store the project. Such as:

* C:/TypeScript/myfirstproject

Next, on Visual Studio Code open the folder you just created in the previous step.

* File > Open Folder

Click the "New File" icon to create a new TypeScript file.



typescript\_ex1.ts

console.**log**("Hello World!");

**function** **sayHello**(name: String) {

console.**log**(`Hello ${name}`);

}

**sayHello**("Tom");

Next, we need to configure to run the above example.

3- Project Configuration

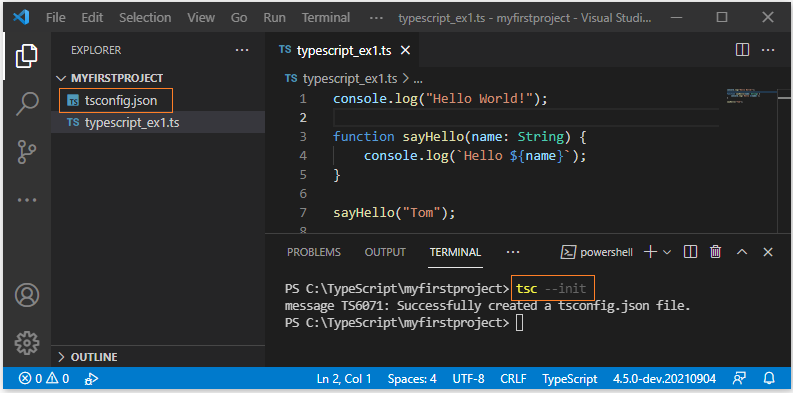
On Visual Studio Code open a Terminal window:

* View > Terminal

On a Terminal window, execute the following command to create a configuration file for your project:

tsc --init

After executing the above command, the tsconfig.json file is created:



*If you receive an error message like the one below, consider one of the solutions from Stackoverflow:*

* [*https://stackoverflow.com/questions/4037939/powershell-says-execution-of-scripts-is-disabled-on-this-system*](https://stackoverflow.com/questions/4037939/powershell-says-execution-of-scripts-is-disabled-on-this-system)

*tsc : File C:\Users\Windows10\AppData\Roaming\npm\tsc.ps1 cannot be*

*loaded because running scripts is disabled on this system. For more*

*information, see about\_Execution\_Policies at*

*https:/go.microsoft.com/fwlink/?LinkID=135170.*

*At line:1 char:1*

*+ tsc typescript\_ex1.ts*

*+ ~~~*

*+ CategoryInfo : SecurityError: (:) [], PSSecurityException*

*+ FullyQualifiedErrorId : UnauthorizedAccess*

Open the tsconfig.json file to change the values of some parameters.

|  |  |
| --- | --- |
| Find line starting with.. | Change to |
| "target": "es5" | "target": "es6" |
| // "outDir": "./", | "outDir": "./dist", |

tsconfig.json

{

"compilerOptions": {

"target": "es6",

"outDir": "./dist",

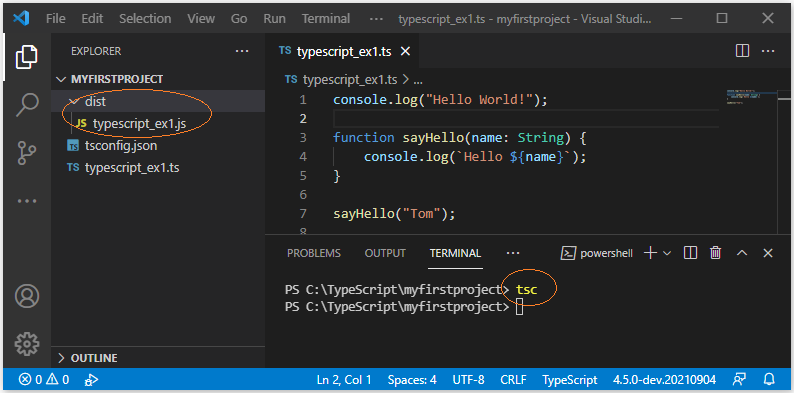
// Other codes...

}

}

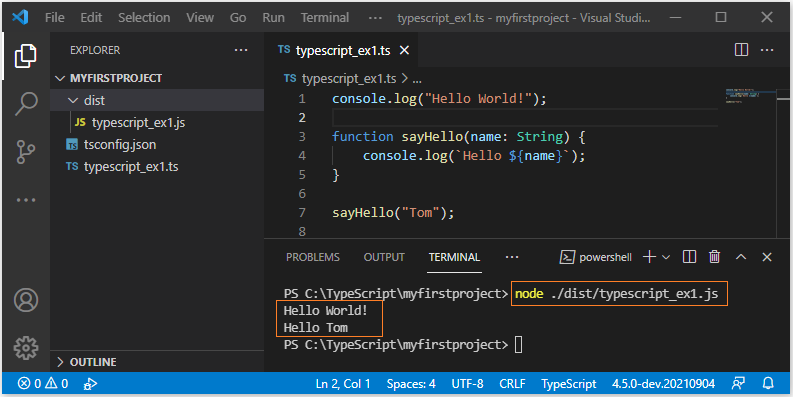
Next, on the Terminal window of Visual Studio Code, execute the command to compile all your typescript files into javascript files.

tsc



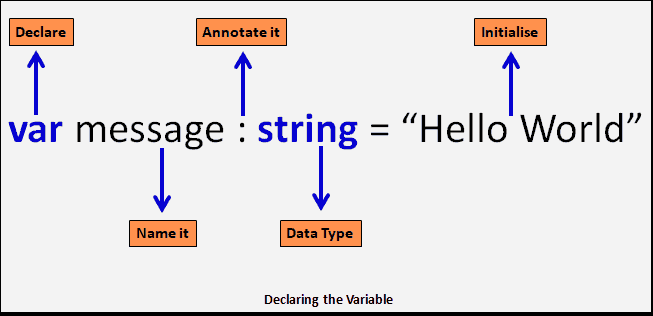
Finally, run the example.

node ./dist/typescript\_ex1.js



**Using the Type Annotation**

We annotate a variable by using a colon (:) followed by its type. There can be a space after the colon. For example, the following shows how to use the type annotation in variable declaration.

Example of Type Annotations

**Type Annotation in Variable declaration**

|  |  |
| --- | --- |
| 6 | var message : string = "hello world";       //string  var NumberVar: number = 100;                //number  var BooleanVar : boolean = true;           //boolean  var ArrayVar: string[]                     //arrays |

The keyword after the colon i.e string/number/boolean is a [Typescript type](https://www.tektutorialshub.com/typescript/typescript-data-types/). By including a type, we are letting the compiler know the type of the variable. The compiler will throw an error in case we attempt to assign a value different than the specified type.

For Example, this will throw an error

Syntax:

let  [Indentifier] : [type-annotation]  = value ;

var  [Indentifier] : [type-annotation]  = value ;

const [Indentifier] : [type-annotation]  = value ;

|  |  |
| --- | --- |
|  | StringVar = NumberVar |

The types are optional in Typescript. If you do not want to use the types, then annotate it with any as shown below

|  |  |
| --- | --- |
|  | var StringVar : any= "hello world";  var NumberVar: any= 100; |

Then, the following example will not result in an error

|  |  |
| --- | --- |
|  | StringVar = NumberVar |

**Only the type**

Only the type is declared. Variable will get the value undefined. The const is not allowed here

**only the initial value**

Here, the variables are defined without type, but with an initial value. The Typescript infers the type from the value assigned to it.

|  |  |
| --- | --- |
|  | Syntax    let  [Indentifier] = value;  var  [Indentifier] = value;  const  [Indentifier] = value; |

In the example below, we have not declared the type but initialized message & num variable using a value. **Typescript infers the type of the variable** from the value assigned to it. Hence the variable message is created as string variable and num as number

**Example**:

|  |  |
| --- | --- |
|  | var message = "Hello World"    //Typescript infers the type as string becasue assigned value is string  var num =1000;                 //num is a number because 100 is assigned to it    console.log(message);  console.log(num);      message = num;    // this will result in compile error |

**output**

|  |  |
| --- | --- |
|  | Hello World  1000 |

## TypeScript if else – Conditional Statement

TypeScript **if else** is an extension to **if** conditional statement. **else** block is an optional block that can follow if block. **if else** block facilitates the branching of execution flow into one of the two blocks based on a condition.

If the return value of expression following **if** keyword is true, then the code block right after **if** statement is executed, otherwise the code block after **else** block is executed.

### Syntax

Following is the syntax of TypeScript **if-else** statement :

|  |
| --- |
| if(expression) {       /\* if block statements \*/   } else {       /\* else block statements \*/   } |

* **expression** should return a boolean value.

### Example 1 – TypeScript If Else

Following is an example TypeScript code to demonstrate if conditional statement.

**example.ts**

|  |
| --- |
| var a:number = 1  var b:number = 3    if(a == 1){      console.log("value of a is 1.")  } else {      console.log("value of a is not 1.")  }    if(a == b){      console.log("a and b are equal.")  } else {      console.log("a and b are not equal.")  } |

When the above code is compiled using typescript compiler, tsc example.ts , following JavaScript code is generated.

**example.js**

|  |
| --- |
| var a = 1;  var b = 3;  if (a == 1) {      console.log("value of a is 1.");  }  else {      console.log("value of a is not 1.");  }  if (a == b) {      console.log("a and b are equal.");  }  else {      console.log("a and b are not equal.");  } |

For this example, you might notice that there is no difference between TypeScript and JavaScript **if-else** code blocks.

### TypeScript If Else If

Instead of a code block following else keyword, another if statement could be placed forming an if-else-if ladder statement.

**example.ts**

|  |
| --- |
| var a:number = 1  var b:number = 3    if(a == b){      console.log("a and b are equal.")  } else if (a>b) {      console.log("a is greater than b.")  } else if (a<b) {      console.log("a is less than b.")  } |

switch (expression){

case constant1 : {

// this is first case block

break

}

case constant2 : {

// this is second case block

// and there can be any number of cases

break

}

default : {

// when no case is matched, this block executes

break

}

}

Typescript inheritance

class Person {

    name:string;

    constructor(Name:string)

     {

         this.name=Name;

    }

}

class Employee  extends Person

{

    empid:number;

   salary:number;

    age:number;

    constructor(employeeid:number,salary:number,age:number,empname:string)

    {

        super(empname);

        this.empid=employeeid;

        this.salary=salary;

        this.age=age;

    }

    getEmployeeDetails():void{

        console.log(this.empid);

        console.log(this.salary);

        console.log(this.age);

        console.log(this.name);

    }

}

let objemp=new Employee(100,89893,31,'john');

objemp.getEmployeeDetails();