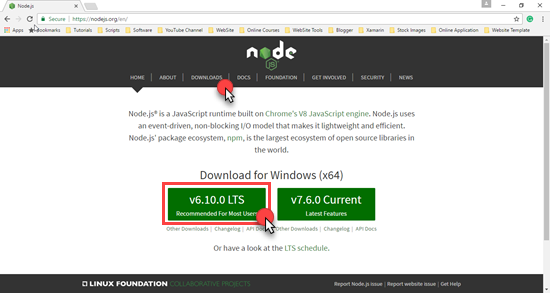
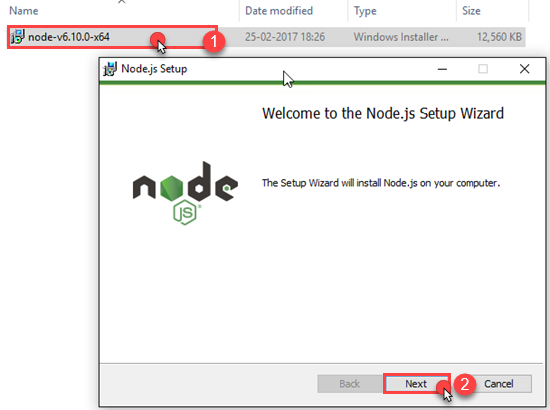
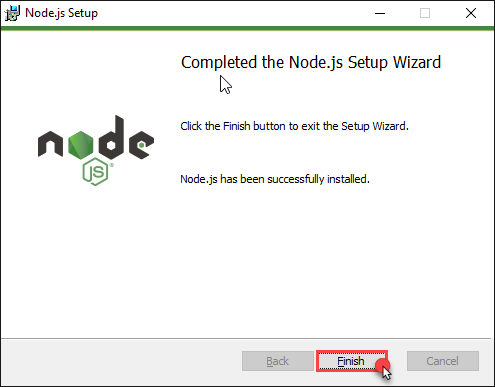
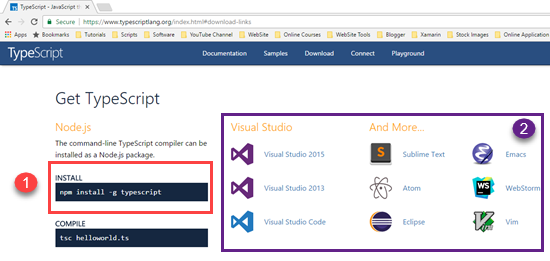
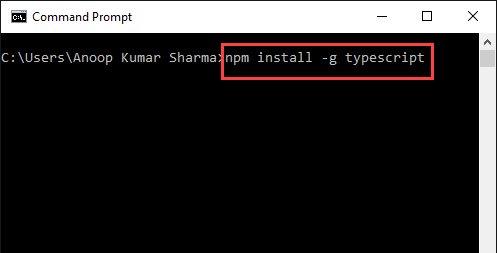
**What is TypeScript?**  
“TypeScript is a typed superset of JavaScript that compiles plain JavaScript”.  
  
We can download this from the official [Website](https://www.typescriptlang.org/play/index.html).

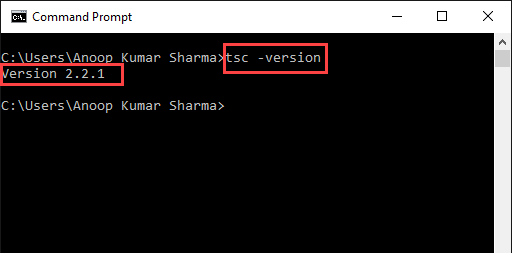
Setting up the environment

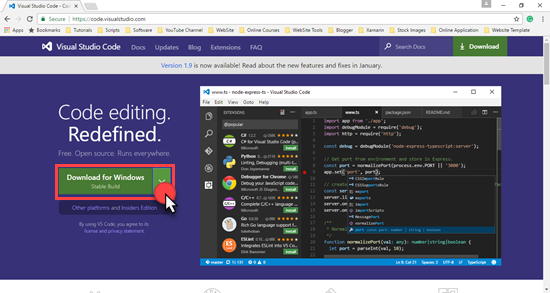
**Let’s begin**Go to *https://nodejs.org/en/* and Download Node.js (which is a JavaScript runtime built on Chrome’s V8 runtime engine).  
  


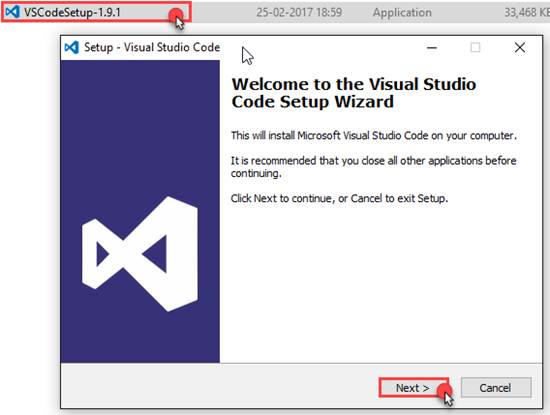
Once download is complete, install Node.js.  
  


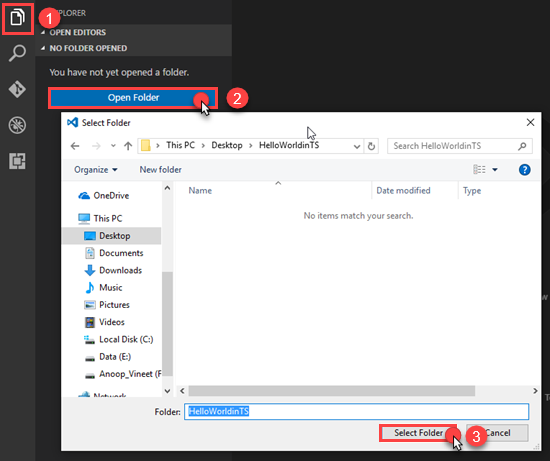


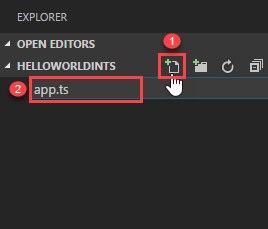
Now, open CMD prompt in order to install TypeScript. We will use NPM (Node Package Manager) to install TypeScript. (In image 1 given below, the command is used to install the latest stable TypeScript version and 2 is editors used for TypeScript).  
  
   
  
Type NPM install -g TypeScript command.  
  


Type tsc -version in order to check the installed version of TypeScript.  
  


There are several code editors available like Visual Studio code, Sublime, WebStorm, Eclipse, Vim, etc. In this series, we will use Visual Studio code as a code editor. Visit <https://code.visualstudio.com/> and click Download.  
  


Install Visual Studio code editor.  
  


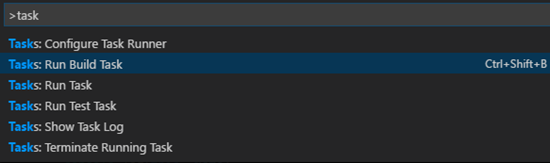
After installing, click Explorer icon and then click Open Folder. I have created a folder named as HelloWorldinTS. Click Select Folder.   
  


Click Add New File and name it as app.ts (You can name it, as you want).  
  


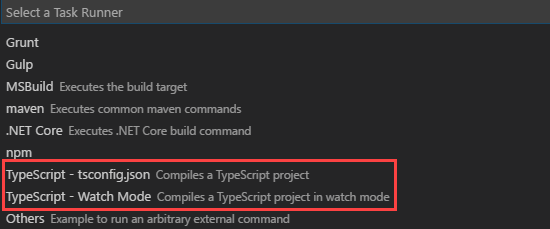
Add the code given below in app.ts.  
  

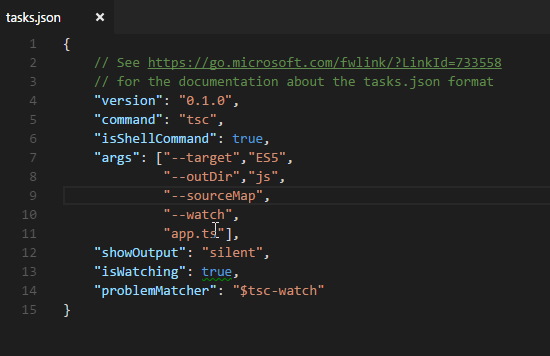

1. **class** MyFirstApp{
2. \_Name:string;
3. constructor(**public** Name:string){
4. **this**.\_Name=Name;
5. }
6. ShowHelloMessage()
7. {
8. console.log("Hello "+**this**.\_Name)
9. }
10. }
11. let objMyFirstApp=**new** MyFirstApp("Anoop");
12. objMyFirstApp.ShowHelloMessage();

Here, I have created a class with a constructor which takes name parameter of type string and a method, which prints Hello [Name] as an output on the console. Subsequently, I created objMyFirstApp object of that class and invoked ShowHelloMessage (We will learn in depth about each thing in my upcoming article on TypeScript).

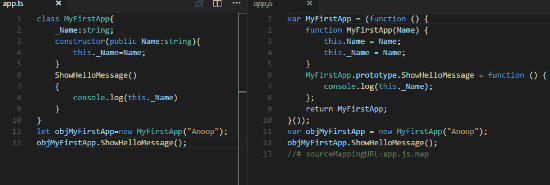
Now, press Crtl+Shift+P to open the command palette, type Tasks and select Run Build Task (We can compile it, using tsc in the command prompt but, in this article, I will show you how to compile or build it using Visual Studio code). Select Run Build Task (or Press Crtl+Shift+B).  
  


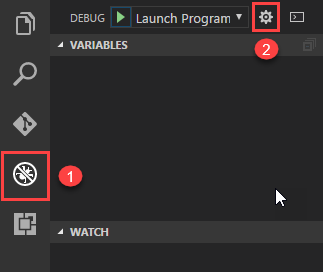
After clicking, you will see an alert, i.e. no task runner is configured. Click Configure Task Runner.  
  
https://csharpcorner-mindcrackerinc.netdna-ssl.com/article/temp/56583/Images/image14.png

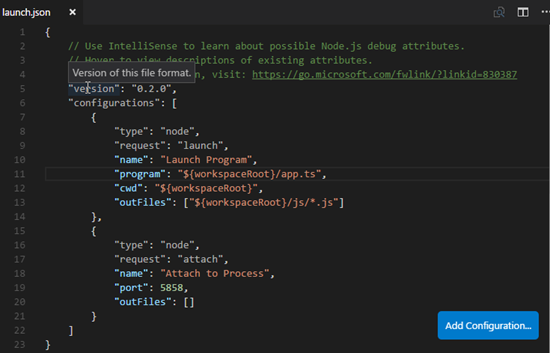
Now, select TypeScript to create tasks.json file in your Workspace .vscode folder.  
  


Whenever we build our code, it will use tsc command (The command tsc assumes that tsc has been installed, using npm install -g TypeScript) with several arguments like target ES5, outDir as js (I want to store compiled \*.js code in js folder), watch will detect the changes and compile the code on saving. I want app.ts file to be compiled.   
  


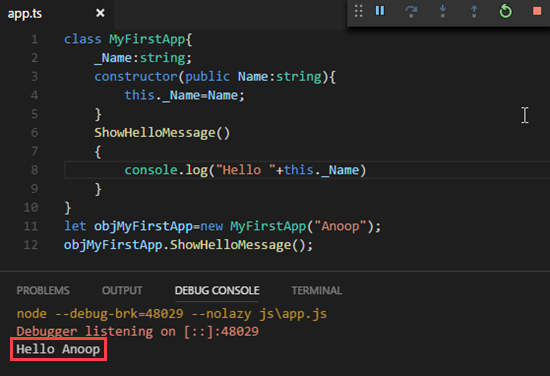
1. {
2. // See https://go.microsoft.com/fwlink/?LinkId=733558
3. // for the documentation about the tasks.json format
4. "version": "0.1.0",
5. "command": "tsc",
6. "isShellCommand": **true**,
7. "args": ["--target","ES5",
8. "--outDir","js",
9. "--sourceMap",
10. "--watch",
11. "app.ts"],
12. "showOutput": "silent",
13. "isWatching": **true**,
14. "problemMatcher": "$tsc-watch"
15. }

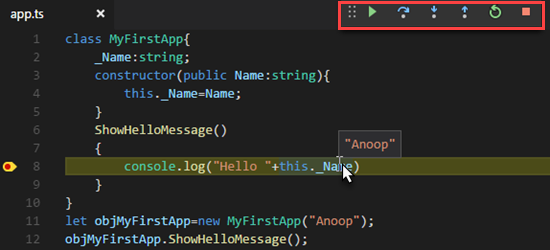
Now, press Crtl+Shift+B in order to run build task. You will see JS directory is created in your Workspace folder with app.js file. As we set tsc in watch mode, whenever we change and save our TypeScript Code, it will be compiled immediately.  
  


**Integrating Debugger in Visual Studio code**   
  
Now, I want to debug my code or want to see the output in Visual Studio code’s console. Click Debug icon on the left side of Visual Studio code and Click the little gear icon (i.e. Open launch.json file). Visual Studio code automatically detects your development environment.   
  


Visual Studio code supports debugging a program in launch or attaching it to the already running program. Here, I mentioned the program files and outFiles location in launch.json file.  
  


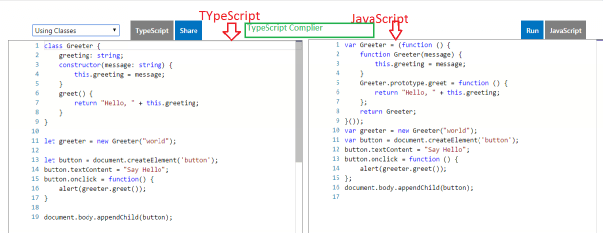
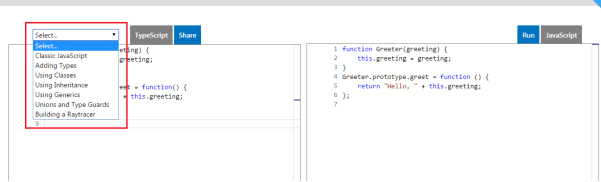
1. {
2. // Use IntelliSense to learn about possible Node.js debug attributes.
3. // Hover to view descriptions of existing attributes.
4. // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5. "version": "0.2.0",
6. "configurations": [
7. {
8. "type": "node",
9. "request": "launch",
10. "name": "Launch Program",
11. "program": "${workspaceRoot}/app.ts",
12. "cwd": "${workspaceRoot}",
13. "outFiles": ["${workspaceRoot}/js/\*.js"]
14. },
15. {
16. "type": "node",
17. "request": "attach",
18. "name": "Attach to Process",
19. "port": 5858,
20. "outFiles": []
21. }
22. ]
23. }

Now, hit F5. You will see the output on Debug console of VS code.  
  


We can also set the breakpoints in order to debug the code at a particular step and use step in/out, step over and other things for debugging purposes.  
  


**Key Features of TypeScript**

* Support standard JavaScript codes.
* Encapsulation through classes and modules.
* TypeScript supports interface and enum.
* Constructor, properties and functions.
* Error handling features.
* Generics and lambda.

**TypeScript to Javascript**  
TypeScript compiler (TSC) will convert your TypeScript code into a JavaScript code.   
  
  
  
The are different types of examples in the Playground options, which we can choose from the dropdown list.  
  
  
  
**TypeScript Playground**  
In this [URL](https://www.typescriptlang.org/play/index.html), we have the option to write the typescript code and we can see the same JavaScript code in the right hand side.  
  
**Basic Data Types in TypeScript**

* Boolean
* Number
* String
* Array
* Tuple
* Enum
* Any
* Void
* Null and Undefined
* Never

The code represents the basic data type declaration and syntax in Typescript.

1. let isAuthor: **boolean** = **true** //boolean data type
2. let ArticleCount: number = 100 // number data type
3. let AuthorNmae: string = "Shiju" //string data types
4. let ArrayList: number[] = [100, 101.102] // array list example
5. //Generic arraylist example
6. let GenericArrayList: Array < number >= [10, 20, 30]
7. The“
8. let” keyword is using instead of“
9. **var**” keyword

Tuple types allows you to represent a value as a pair of a string and a number or any other combination.  
  
**Examples for Tuples and enum data types are as follows.**

1. // Tuple
2. let articles: [string, number]
3. articles = ["shiju", 10]
4. //Enum
5. **enum** Technologies {
6. SharePoint,
7. SQL,
8. ASP
9. }
10. let Tech: Technologies = Technologies.ASP
11. **enum** TechnologiesUsed {
12. SharePoint = 1, SQL = 2, ASP = 3
13. }
14. let Techs: TechnologiesUsed = TechnologiesUsed.SharePoint

**Examples for “any “ are as follows.**

1. “
2. any” is a dynamic data type.once you declare a variable as any type, you can assign any type of data into that variable
3. // Any
4. // Here the value will assigned dyanamicallay
5. let dynamicdataType: any
6. dynamicdataType = **true**; //boolean type
7. dynamicdataType = "shiju"; //string Type
8. dynamicdataType = 100; // number type

**Using “Void”, “undefined” and “null”.**Example of function with string return type is as follows.

1. // Function
2. **function** ArticleNotification(): string {
3. let message: string = "This is my New article";
4. **return** message;
5. }
6. // Function with void type
7. **function** ArticleNotification(): **void** {
8. alert("This is my New article");
9. }

**Create a class in TypeScript**

1. // Exmaple of class
2. **class** Author {
3. **public** AuthorName: string = "Shiju";
4. }
5. // Create the object of the class
6. let authordetails = **new** Author();
7. let authorName: string = authordetails.AuthorName

**Access modifiers in TypeScript are as**

* Publicfollows.
* private
* protected

**Property of the class**

1. // Exmaple of class properties
2. **class** Author {
3. **private** \_AuthorName: string;
4. **public** set AuthorName(value: string) {
5. **this**.\_AuthorName = value;
6. }
7. **public** get AuthorName(): string {
8. **return** **this**.\_AuthorName;
9. }
10. }

**Methods in the class**

1. // Example of method in the class
2. **class** Author {
3. **public** AuthorName: string;
4. isActiveAthour(authorId: number): **boolean** {
5. **return** **true**;
6. }
7. }

**Inheritance**To inherit the class in TypeScript, use the “extends” keyword. See the example, mentioned below.

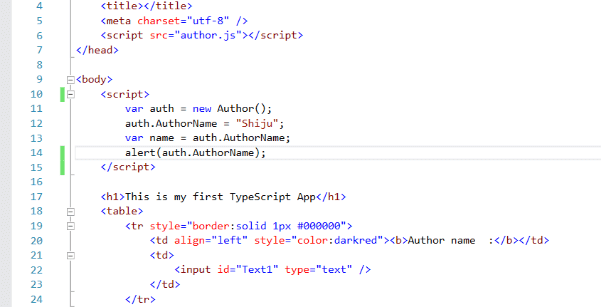
1. **class** Author {
2. **public** AuthorName: string;
3. isActiveAthour(authorId: number): **boolean** {
4. **return** **true**;
5. }
6. inheritExample(): **void** {
7. alert("Inheritance1")
8. }
9. }
10. **class** inheritanceOfClass **extends** Author {
11. inheritExample(): **void** {
12. alert("Inheritance example 2")
13. }
14. }

**Interface in TypeScript**Use the keyword “implements” for the interface implementation in TypeScript.

1. **interface** ITopAuthor {
2. GetTopAuthor()
3. }
4. **class** Author **implements** ITopAuthor {
5. **public** AuthorName: string = "";
6. GetTopAuthor() {
7. //Implimentation here ...
8. }
9. }

**Constructor of a class**  
The example, mentioned below indicates how to create a constructor of a class in the TypeScript.

1. **class** Author {
2. **public** AuthorName: string;
3. constructor(name: string) {
4. **this**.AuthorName = name;
5. }
6. }
7. let authorName = **new** Author("shiju");



Run the application

