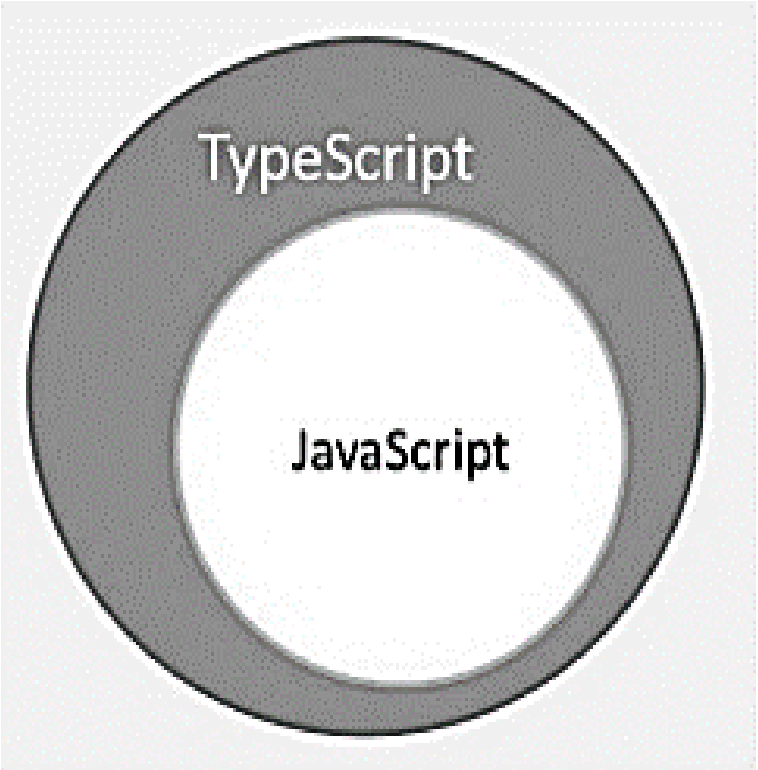
Introduction to TypeScript

About TypeScript

- TypeScript is well-know as typed-superset of JavaScript.
- 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 or Deno.
- TypeScript needs an ECMAScript 3 or higher compatible environment to compile.
- This is a condition met by all browsers and JavaScript engines today.
- Syntax and semantics are similar to JavaScript

- What is TypeScript?
- By definition, “TypeScript is JavaScript for application-scale development.”
- TypeScript is a strongly typed, object oriented, compiled language. It was designed by **Anders Hejlsberg** (designer of C#) at Microsoft. TypeScript is both a language and a set of tools. TypeScript is a typed superset of JavaScript compiled to JavaScript. In other words, TypeScript is JavaScript plus some additional features.


TypeScript
JavaScript

Features

- Cross platform
- Static type checking
- Optional type checking
- Object oriented support
- DOM manipulation
- ES6 features

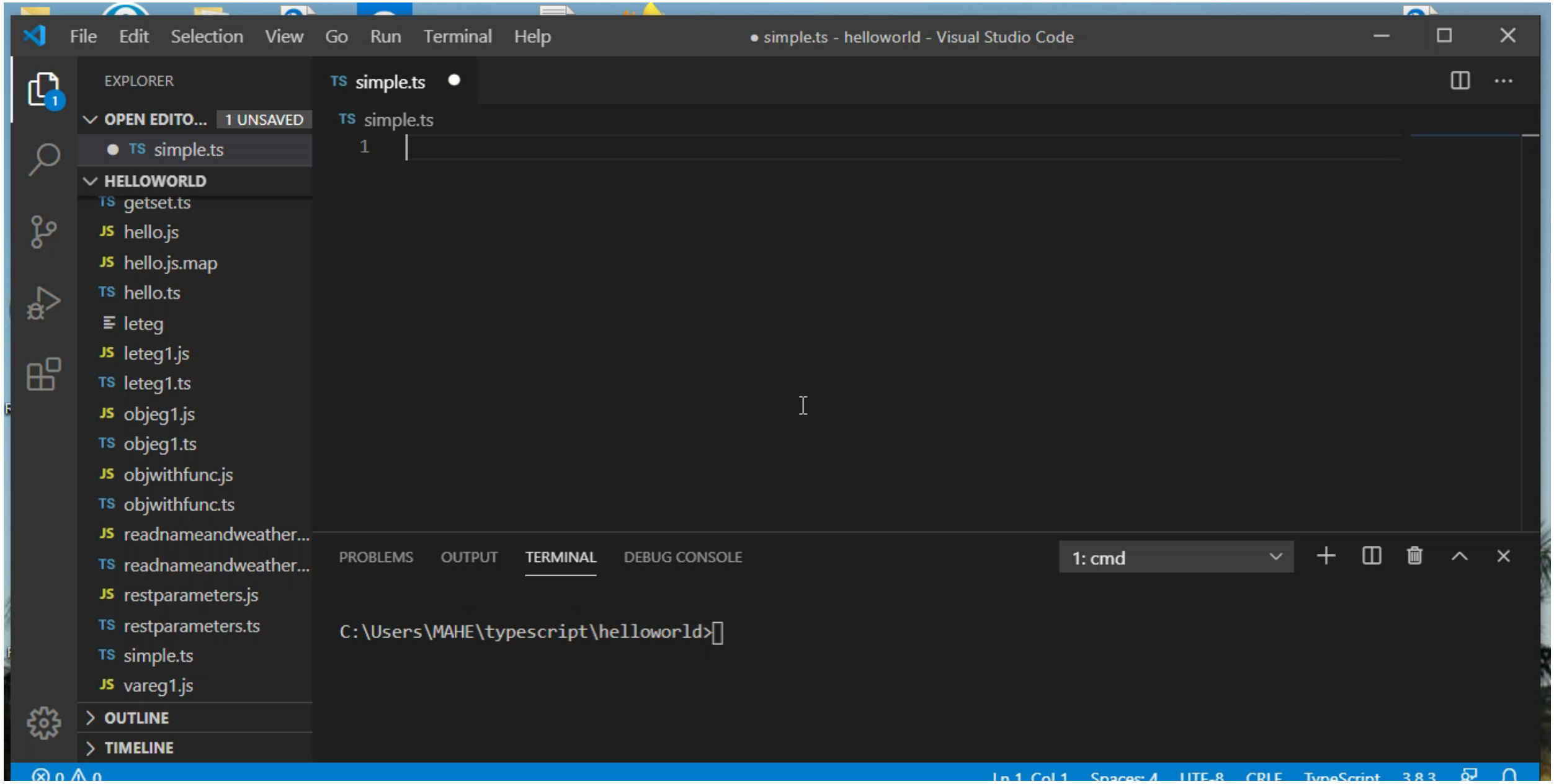
TypeScript - Setup Development Environment

There are two ways to install TypeScript:

- Install TypeScript using Node.js package manager (npm).
- Install the TypeScript Plug-in in your IDE (Integrated Development Environment).

Installing TypeScript using Command Prompt

- 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**
- you can check the TypeScript version using the following command:
- **tsc -v**


File Edit Selection View Go Run Terminal Help
simple.ts - helloworld - Visual Studio Code
EXPLORER
OPEN EDITO... 1 UNSAVED
simple.ts
HELLOWORLD
getset.ts
hello.js
hello.js.map
hello.ts
leteg
leteg1.js
leteg1.ts
objeg1.js
objeg1.ts
objwithfunc.js
objwithfunc.ts
readnameandweather...
readnameandweather...
restparameters.js
restparameters.ts
simple.ts
vareg1.js
OUTLINE
TIMELINE
simple.ts
1
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
1: cmd
C:\Users\MAHE\typescript\helloworld>

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.

Using Visual Studio Code

Steps to run a simple TypeScript Code on VSCode

- Create a folder to include the TypeScript files
- Write your TypeScript and save it with **.ts** extension (eg. **sample.ts**)
- Open the terminal
- You have the option of selecting (powershell, cmd or Git bash)
- To compile type: tsc <filename.ts>
- To run as a node: node <filename.ts>

Note:

- When using powershell, you may have to set appropriate execution policy.
- To get more information on Set-ExecutionPolicy, type **"Get-Help Set-ExecutionPolicy"** in the terminal

Defining functions with parameters

Sample.ts

```
let myFunc = function(name, weather)
{
   console.log("Hello " + name + ".");
   console.log("It is " + weather + " today");
};
myFunc("Adam", "sunny");
```

To check the output on the terminal:

tsc Sample.ts

node Sample.ts

Expected Output:

```
Hello Adam.
It is sunny today
```

Running typescript in angular environment

Steps

- Install node.js runtime
  - Download from https://nodejs.org/en/download/ and execute
- In the windows command prompt
  - Type: **node –v** (to check the version)
- Use npm to Install Angular CLI
  - command to install Angular CLI : **npm install -g @angular/cli**
  - To check Node and Angular CLI version, use **ng --version** command
- To create new app
  - ng new app_name (eg. ng new firstapp)
- Run your app
  - Use the command **ng serve --port 3000 --open**

Run the typescript in Angular environment

- Open the folder newly created app “firstapp”.
- In the src folder, you will find main.ts
- Replace the code in main.ts with the code in sample.ts and save
- In command prompt, (make sure you are in firstapp)
  - Run your app
  - Use the command **ng serve --port 3000 –open**
- Open console through browser developer tools to see the output

Brief on node JS platform

- Node.js brings event-driven programming to web servers, enabling development of fast web servers in JavaScript
- Node.js was built on the Google V8 JavaScript engine since it was open-sourced under the BSD license.
- It is proficient with internet fundamentals such as HTTP, DNS, TCP.
- JavaScript was also a well-known language, making Node.js accessible to the web development community.

Using Default Parameters

```
let myFunc1 = function (name, weather = "raining")
{
      console.log("Hello " + name + ".");
   console.log("It is " + weather + " today");
};
myFunc1("Adam");
```

Using Rest Parameters

```
let myFunc = function (name, weather, ...extraArgs)
{    console.log("Hello " + name + ".");
   console.log("It is " + weather + " today");
    for (let i = 0; i < extraArgs.length; i++) {
              console.log("Extra Arg: " + extraArgs[i]);
            }
 };
myFunc("Adam", "sunny", "one", "two", "three");
```

Output:

```
Hello Adam.
It is sunny today
Extra Arg: one
Extra Arg: two
Extra Arg: three
```

Using Arrow Functions

- Arrow functions—also known as fat arrow functions or lambda expressions
- Are an alternative way of defining functions.
- Fat arrow notations are used for anonymous functions i.e for function expressions
- Using fat arrow (=>) we drop the need to use the 'function' keyword. The function expression is enclosed within the curly brackets {}

Arrow function example

- **Eg1**

```
function sum(x,y)
{          return (x+y);
}
console.log(sum(4,5));
```

- **Eg2**

```
var sum=function(x,y){
   return (x+y); }
console.log(sum(6,5));
```

- **Eg3**

```
var sum=(x,y)=>{
   return (x+y);
}
console.log(sum(6,8));
```

- **Eg4 (typescript)**

```
var sum=(x:number,y:number):numb
er=>{
   return (x+y);
}
console.log(sum(6,9));
```

Variables

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

Advantages of using let over var

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

Let and Var

```
let messageFunction = function (name, weather) {
      let message = "Hello, Adam";

      if (weather == "sunny") {
            let message = "It is a nice day";
            console.log(message);
          } else {
                let message = "It is " + weather + " today";
                 console.log(message);    }    console.log(message);
              }
messageFunction("Adam", "raining");
```

Output:

It is raining today

Hello, Adam

Var eg

```
let messageFunction = function (name, weather) {
      var message = "Hello, Adam";
       if (weather == "sunny") {
                var message = "It is a nice day";
                console.log(message);
       } else {
                var message = "It is " + weather + " today";
                console.log(message);
            }    console.log(message);
}
messageFunction("Adam", "raining");
```

Output:

It is raining today

It is raining today

Objects

```
let myData = new Object();
myData.name = "Adam";
myData.weather = "sunny";
console.log("Hello " + myData.name + ".");
console.log("Today is " + myData.weather + ".");
```

Output (Ignore the errors put forward by tsc)

Hello Adam.

Today is sunny.

Functions as methods

```
let myData = {    name: "Adam",    weather: "sunny",
 printMessages: function () {
    console.log("Hello " + this.name + ". ");
    console.log("Today is " + this.weather + ".");
 }
};
 myData.printMessages();
```

Output:

Hello Adam.

Today is sunny

Classes

```
class MyClass {
   constructor(name, weather) {
            this.name = name;
            this.weather = weather;
          }
   printMessages() {
        console.log("Hello " + this.name + ". ");
        console.log("Today is " + this.weather + ".");
         }
        }
let myData = new MyClass("Adam", "sunny");
myData.printMessages();
```

Output:

Hello Adam.

Today is sunny

Class getters and setters

- Getter is represented using keyword **get**
  - The get syntax binds an object property to a function that will be called when that property is looked up.
  - **Syntax:**

```
{get prop() { ... } }
{get [expression]() { ... } }
```

- Setter is represented using keyword **set**
  - The set syntax binds an object property to a function to be called when there is an attempt to set that property
  - **Syntax**

```
{set prop(val) { . . . }}
{set [expression](val) { . . . }}
```

Defining Class Getter and Setter Properties

```
class Developer {
    private _language = '';
    private _tasks: string[] = [];

    get language() {
      return this._language;
    }
    set language(value: string) {
      this._language = value;
    }
    get tasks() {
      return this._tasks;
    }
    set tasks(value: string[]) {
      this._tasks = value;
    }
  }
```

```
const dev = new Developer();

dev.language = 'TypeScript';
console.log(dev.language); //  "TypeScript"

dev.tasks = ['develop', 'test'];
dev.tasks.push('ship');

console.log(dev.tasks); // ['develop', 'test', 'ship']
```

Exporting and import information

NameWeather.ts

```
export class Name {
      constructor(first, second) {
               this.first = first;
               this.second = second; }
   get nameMessage() {
           return `Hello ${this.first} ${this.second}`;
           } }
export class WeatherLocation {
      constructor(weather, city) {
               this.weather = weather;
               this.city = city;
            }
   get weatherMessage() {
           return `It is ${this.weather} in ${this.city}`;
          } }
```

.

- **DuplicateName.ts**

```
export class Name {
   get message() {
           return "Duplicate Name";
          }
}
```

Exporting and import information..

Readdata.ts

```
import{ Name, WeatherLocation } from "./nameandweather
";
import{Name as OthName} from "./DuplicateName";

let name = new Name("Adam", "Freeman");
 let loc = new WeatherLocation("raining", "London");
let othername= new OthName();
 console.log(name.nameMessage);
 console.log(loc.weatherMessage);
 console.log(othername.message);
```

Type annotation

- 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
- TypeScript includes all the primitive types of JavaScript- number, string and Boolean
- Eg:
  - var age: number = 32; // number variable
  - var name: string = "John";// string variable
  - var isUpdated: boolean = true;// Boolean variable

DataTypes in typescript

- Number
- String
- Boolean
- Array
- Tuples
- Enum
- Union
- Any
- Void
- Never

Type annotation example

```
class Name {
   first: string;
   second: string;
   constructor(first: string, second: string) {
            this.first = first;
            this.second = second;
          }
   get nameMessage() : string {
             return `Hello ${this.first} ${this.second}`;
           }
    }

let ename= new Name("Raj","kapoor");
console.log(ename.nameMessage);
```

References

- https://www.typescriptlang.org/
- *Adam Freeman, Apress, Pro Angular 6, Third Edition*
  (Chapter 5 and 6 for typescript)