V skills
CERTIFIED
www.vskills.com

Certified Angular 2 Developer Sample Material VS-1427

Vskills Certifications

Vskills Reading Material

Skills for a secure future

1. TYPESCRIPT

TypeScript is a superset of ES6, which means all ES6 features are part of TypeScript, but not all TypeScript features are part of ES6. Consequently, TypeScript must be transpiled into ES5 to run in most browsers.

One of TypeScript's primary features is the addition of type information, hence the name. This type information can help make JavaScript programs more predictable and easier to reason about.

Types let developers write more explicit "contracts". In other words, things like function signatures are more explicit.

Without TS:

```
function add(a, b) {
  return a + b;
}

add(1, 3);   // 4
add(1, '3'); // '13'
```

With TS:

```
function add(a: number, b: number) {
  return a + b;
}

add(1, 3);   // 4
// compiler error before JS is even produced
add(1, '3'); // '13'
```

1.1. TypeScript Basics

Install the TypeScript transpiler using npm:

```
$ npm install -g typescript
```

Then use tsc to manually compile a TypeScript source file into ES5:

```
$ tsc test.ts
$ node test.js
```

TypeScript is more demanding than ES6 and it expects instance properties to be declared:

```
class Pizza {
  toppings: string[];
  constructor(toppings: string[]) {
    this.toppings = toppings;
  }
}
```

Note that now that we've declared toppings to be an array of strings, TypeScript will enforce this. If we try to assign a number to it, we will get an error at compilation time.

If you want to have a property that can be set to a value of any type, however, you can still do this: just declare its type to be "any":

```
class Pizza {
  toppings: any;
  //...
}
```

Typings

Astute readers might be wondering what happens when TypeScript programmers need to interface with JavaScript modules that have no type information. TypeScript recognizes files labelled *.d.ts as definition files. These files are meant to use TypeScript to describe interfaces presented by JavaScript libraries.

There are communities of people dedicated to creating typings for JavaScript projects. There is also a utility called typings (npm install --save-dev typings) that can be used to manage third party typings from a variety of sources. (Deprecated in TypeScript 2.0)

In TypeScript 2.0, users can get type files directly from @types through npm (for example, npm install --save @types/lodash will install lodash type file).

Linting

Many editors support the concept of "linting" - a grammar check for computer programs. Linting can be done in a programmer's editor and/or through automation.

For TypeScript there is a package called tslint, (npm install --save-dev tslint) which can be plugged into many editors. tslint can also be configured with a tslint.json file.

Webpack can run tslint before it attempts to run tsc. This is done by installing tslint-loader (npm install --save-dev tslint-loader) which plugs into webpack like so:

```
// ...
module: {
  preLoaders: [
    { test: /\.ts$/, loader: 'tslint' }
  ],
  loaders: [
    { test: /\.ts$/, loader: 'ts', exclude: /node_modules/ },
    // ...
  ]
  // ...
}
```

1.2. TypeScript Types

TypeScript uses .ts as a file extension. We can compile TypeScript code into JavaScript by invoking the TypeScript compiler using the following command:

```
$ t sc <filename.ts>
```

JavaScript is a loosely typed language, and we do not need to declare the type of a variable ahead of time. The type will be determined automatically while the program is being executed. Because of the dynamic nature of JavaScript, it does not guarantee any type safety. TypeScript provides an optional type system to ensure type safety.

String

In TypeScript, we can use either double quotes (") or single quotes (') to surround strings similar to JavaScript, as

```
var bookName: string = "Angular";
bookName = 'Angular UI Development';
```

Number

As in JavaScript, all numbers in TypeScript are floating point values, as

```
var version: number = 4;
```

Boolean

The boolean data type represents the true / false value, as

```
var isCompleted: boolean = false;
```

Array

We have two different syntaxes to describe arrays, and the first syntax uses the element type followed by [], as

```
var fw: string[] = ['Angular', 'React', 'Ember'];
```

The second syntax uses a generic array type, Array<elementType> , as

```
var fw: Array<string> = ['Angular', 'React', 'Ember'];
```

Enum

TypeScript includes the enum data type along with the standard set of data types from JavaScript. In languages such as C# and JAVA, an enum is a way of giving more friendly names to sets of numeric values, as

```
enum Frameworks { Angular, React, Ember };
var f: Frameworks = Frameworks.Angular;
```

The numbering of members in the enum type starts with 0 by default. We can change this by manually setting the value of one of its members. As illustrated earlier, in code snippet, the Frameworks.Angular value is 0 , the Frameworks.React value is 1 , and the Frameworks.Ember value is 2 .

We can change the starting value of enum type to 5 instead of 0 , and remaining members follow the starting value, as

```
enum Frameworks { Angular = 5, React, Ember };
var f: Frameworks = Frameworks.Angular;
var r: Frameworks = Frameworks.React;
```

In the preceding code snippet, the Frameworks.Angular value is 5 , the Frameworks.React value is 6 , and the Frameworks.Ember value is 7.

Any

If we need to opt out type-checking in TypeScript to store any value in a variable whose type is not known right away, we can use any keyword to declare that variable, as

```
var eventId: any = 7890;
eventId = 'event1';
```

In the preceding code snippet while declaring a variable, we are initializing it with the number value, but later we are assigning the string value to the same variable. TypeScript compiler will not report any errors because of any keyword. Here is one more example of an array storing different data type values, as

```
var myCollection:any[] = ["value1", 100, 'test', true];
myCollection[2] = false;
```

Void

The void keyword represents not having any data type. Functions without return keyword do not return any value, and we use void to represent it, as

```
function simpleMessage(): void {
alert("Hey! I return void");
}
```

Let's write our first TypeScript example and save it into the example1.ts file, as

```
var bookName: string = 'Angular UI Development';
var version: number = 2;
var isCompleted: boolean = false;
var frameworks1: string[] = ['Angular', 'React', 'Ember'];
var frameworks2: Array<string> = ['Angular', 'React', 'Ember'];
enum Framework { Angular, React, Ember };
var f: Framework = Framework.Angular;
var eventId: any = 7890;
eventId = 'event1';
var myCollection:any[] = ['value1', 100, 'test', true];
myCollection[2] = false;
```

Let's compile the example1.ts file using the TypeScript command-line compiler, as

```
$ tsc example.ts
```

The preceding command will compile TypeScript code into plain JavaScript code into the example1.js file; this is how it will look, and it is plain JavaScript, as

```
var bookName = 'Angular UI Development';
var version = 2;
var isCompleted = false;
var frameworks1 = ['Angular', 'React', 'Ember'];
var frameworks2 = ['Angular', 'React', 'Ember'];
var Framework;
(function (Framework) {
Framework[Framework["Angular"] = 0] = "Angular";
Framework[Framework["React"] = 1] = "React";
Framework[Framework["Ember"] = 2] = "Ember";
})(Framework || (Framework = {}));
;
var f = Framework.Angular;
var eventId = 7890;
eventId = 'event1';
var myCollection = ['value1', 100, 'test', true];
myCollection[2] = false;
```

Functions

Functions are the fundamental building blocks of any JavaScript application. In JavaScript, we declare functions in two ways.

Function declaration – named function - The following is an example of function declaration:

```
function sum(a, b) {
return a + b;
}
```

Function expression – anonymous function - The following is an example of function expression:

```
var result = function(a, b) {
return a + b;
}
```

In JavaScript, unlike any other concept, there is no type safety for functions also. We do not have any assurance on data types of parameters, return type, the number of parameters passed to function, TypeScript guarantees all this. It supports both the syntaxes. Here are the same functions written in TypeScript, as

```
function sum(a: number, b: number): number {
return a + b;
}
```

```
var result = function(a: number, b: number): number {
return a + b;
}
```

The preceding TypeScript functions are very similar to the JavaScript syntax except parameters, and return type has type on them, which ensures type safety while invoking them.

Classes

The ES5 does not have the concept of classes. However, we can mimic the class structure using different JavaScript patterns. The ES2015 does support classes. However, today, we can already write them in TypeScript. In fact, the ECMAScript 2015 class syntax is inspired by TypeScript.

The following example shows a person class written in TypeScript, as

```
class Person {
name: string;
constructor(name: string) {
this.name = name;
}
sayHello() {
return 'Hello ' + this.name;
}
}
var person = new Person('Shravan');
console.log(person.name);
console.log(person.sayHello());
```

The preceding Person class has three members - a property named name , a constructor, and a method sayHello . We should use this keyword to refer to the properties of the class. We created an instance of the Person class using the new operator. In the next step, we invoke the sayHello() method of the Person class using its instance created in the previous step.

Save the preceding code into the person.ts file and compile it using the TypeScript command-line compiler. It will compile TypeScript code into plain JavaScript code into the person.js file, as

```
$ tsc person.ts
```

Here is the plain JavaScript code, which was compiled from the TypeScript class, as

```
var Person = (function () {
function Person(name) {
this.name = name;
}
Person.prototype.sayHello = function () {
return 'Hello ' + this.name;
};
return Person;
}());
```

```
var person = new Person('Shravan');
console.log(person.name);
console.log(person.sayHello());
```

1.3. tsc

So far tsc has been used to compile a single file. Typically programmers have a lot more than one file to compile. Thankfully tsc can handle multiple files as arguments.

Imagine two ultra simple files/modules:

a.ts
```
export const A = (a) => console.log(a);
```

b.ts
```
export const B = (b) => console.log(b);
```

Before TypeScript@1.8.2:

```
$ tsc ./a.ts ./b.ts
a.ts(1,1): error TS1148: Cannot compile modules unless the '--module' flag is provided.
```

Hmmm. What's the deal with this module flag? TypeScript has a help menu, let's take a look:

```
$ tsc --help | grep module
 -m KIND, --module KIND             Specify module code generation: 'commonjs', 'amd', 'system', 'umd' or 'es2015'
 --moduleResolution                 Specifies module resolution strategy: 'node' (Node.js) or 'classic' (TypeScript pre-1.6).
```

There are two help entries that reference "module", and --module is the one TypeScript was complaining about. The description explains that TypeScript supports a number of different module schemes. For the moment commonjs is desirable. This will produce modules that are compatible with node.js's module system.

```
$ tsc -m commonjs ./a.ts ./b.ts
```

Since TypeScript@1.8.2, tsc has a default rule for --module option: target === 'ES6' ? 'ES6' : 'commonjs' (more details can be found here), so we can simply run:

```
$ tsc ./a.ts ./b.ts
```

tsc should produce no output. In many command line traditions, no output is actually a mark of success. Listing the directory contents will confirm that our TypeScript files did in fact compile.

```
$ ls
a.js    a.ts    b.js    b.ts
```

Excellent - there are now two JavaScript modules ready for consumption.

Telling the tsc command what to compile becomes tedious and labor intensive even on small projects. Fortunately TypeScript has a means of simplifying this. tsconfig.json files let programmers write down all the compiler settings they want. When tsc is run, it looks for tsconfig.json files and uses their rules to compile JavaScript.

For Angular projects there are a number of specific settings that need to be configured in a project's tsconfig.json

```
{
 "compilerOptions": {
   "module": "commonjs",
   "target": "es5",
   "emitDecoratorMetadata": true,
   "experimentalDecorators": true,
   "noImplicitAny": false,
   "removeComments": false,
   "sourceMap": true
  },
  "exclude": [
   "node_modules",
   "dist/"
  ]
}
```

Target

The compilation target. TypeScript supports targeting different platforms depending on your needs. In our case, we're targeting modern browsers which support ES5.

Module

The target module resolution interface. We're integrating TypeScript through webpack which supports different interfaces. We've decided to use node's module resolution interface, commonjs.

Decorators

Decorator support in TypeScript hasn't been finalized yet but since Angular uses decorators extensively, these need to be set to true. Decorators have not been introduced yet, and will be covered later in this section.

TypeScript with Webpack

We won't be running tsc manually, however. Instead, webpack's ts-loader will do the transpilation during the build:

```
  // webpack.config.js
  //...
  rules: [
    { test: /\.ts$/, loader: 'ts', exclude: /node_modules/ },
   //...
 ]
```

This loader calls tsc for us, and it will use our tsconfig.json.

Certifications

Accounting, Banking & Finance
- Certified GST Professional
- Certified AML-KYC Compliance Officer
- Certified Business Accountant
- Certified BASEL III Professional
- Certified GAAP Accounting Standards Professional
- Certified Treasury Markets Professional

Big Data
- Certified Hadoop and Mapreduce Professional

Cloud Computing
- Certified Cloud Computing Professional

Design
- Certified Interior Designer

Digital Media
- Certified Social Media Marketing Professional
- Certified Inbound Marketing Professional
- Certified Digital Marketing Professional

Foreign Trade
- Certified Export Import (Foreign Trade) Professional

Health, Nutrition and Well Being
- Certified Fitness Instructor

Hospitality
- Certified Restaurant Team Member (Hospitality)

Human Resources
- Certified HR Compensation Manager
- Certified HR Staffing Manager
- Certified Human Resources Manager
- Certified Performance Appraisal Manager

Office Skills
- Certified Data Entry Operator
- Certified Office Administrator

Project Management
- Certified Master in Project Management
- Certified Scrum Specialist

Real Estate
- Certified Real Estate Consultant

Marketing
- Certified Marketing Manager

Quality
- Certified Six Sigma Green Belt Professional
- Certified Six Sigma Black Belt Professional
- Certified TQM Professional

Logistics & Supply Chain Management
- Certified International Logistics Professional
- Certified Logistics & SCM Professional
- Certified Supply Chain Management Professional

Legal
- Certified IPR & Legal Manager
- Certified Labour Law Analyst
- Certified Business Law Analyst
- Certified Corporate Law Analyst

Information Technology
- Certified Angular JS Professional
- Certified Basic Network Support Professional
- Certified Business Intelligence Professional
- Certified Core Java Developer
- Certified E-commerce Professional
- Certified IT Support Professional
- Certified PHP Professional
- Certified Selenium Professional

Mobile Application Development
- Certified Android Apps Developer
- Certified iPhone Apps Developer

Security
- Certified Ethical Hacking and Security Professional
- Certified Network Security Professional

Management
- Certified Corporate Governance Professional
- Certified Corporate Social Responsibility Professional
- Certified Leadership Skills Professional

Life Skills
- Certified Business Communication Specialist
- Certified Public Relations Officer

Media
- Certified Advertising Manager
- Certified Advertising Sales Professional

Sales, BPO
- Certified Sales Manager
- Certified Telesales Executive

& many more job related certifications

Contact us at:
V-Skills
011-473 44 723 or info@vskills.in
www.vskills.in