

TypeScript

Notes for Professionals

Chapter 6: Functions

Section 6.1: Optional and Default Parameters

Optional Parameters

In TypeScript, every parameter is assumed to be required by the function. You can add a ? to parameters name to set it as optional.

For example, the `lastName` parameter of this function is optional:

```
function buildName(firstName: string, lastName?: string) {  
    // ...  
}
```

Optional parameters must come after all non-optional parameters:

```
function buildName(firstName: string, lastName: string) // Invalid  
function buildName(firstName?: string, lastName: string) // Invalid
```

Default Parameters

If the user passes undefined or doesn't specify an argument, the default value will be default-initialized parameters.

For example, "Smith" is the default value for the `lastName` parameter:

```
function buildName(firstName: string, lastName = "Smith") {  
    // ...  
}
```

Section 6.2: Function as a parameter

Suppose we want to receive a function as a parameter, we can do it like this:

```
function Tool(etherFunc: Function): void {  
    // ...  
}
```

If we want to receive a constructor as a parameter:

```
function foo(constructorFunc: { new(): {} }) {  
    new constructorFunc();  
}
```

function foo(constructorWithParamFunc: { new(num: number): {} }) {
 new constructorWithParamFunc();
}

Or to make it easier to read we can define an interface describing the constructor:

```
interface IConstructor {  
    new(): {}  
}
```

TypeScript Notes for Professionals

Chapter 7: Classes

TypeScript, like ECMAScript 6, support object-oriented programming using classes. This contrasts with older JavaScript versions, which only supported prototype-based inheritance chain.

The class support in TypeScript is similar to that of languages like Java and C#, in that classes may inherit from other classes, while objects are instantiated as class instances.

Also similar to those languages, TypeScript classes may implement interfaces or make use of generics.

Section 7.1: Abstract Classes

```
abstract class Machine {  
    constructor(public manufacturer: string) {}  
    summary(): string {  
        // An abstract class can define methods of its own, or...  
        return `${this.manufacturer} makes this machine.`  
    }  
    // Require inheriting classes to implement methods  
    abstract moreInfo(): string;  
}
```

```
class Car extends Machine {  
    constructor(manufacturer: string, public position: number, protected speed: number) {  
        super(manufacturer);  
    }  
    move() {  
        this.position += this.speed;  
    }  
    moreInfo() {  
        return `This is a car located at ${this.position} and going ${this.speed} mph!`  
    }  
}
```

```
let myCar = new Car("Honda", 10, 70);  
myCar.move(); // position is now 80  
console.log(myCar.summary()); // prints "Honda makes this machine."  
console.log(myCar.moreInfo()); // prints "This is a car located at 80 and going 70mph"
```

Abstract classes are base classes from which other classes can extend. They cannot be instantiated themselves, you **cannot** do `new Machine("Honda")`.

The two key characteristics of an abstract class in TypeScript are

1. They can implement methods of their own.
2. They can define methods that inheriting classes **must** implement.

For this reason, abstract classes can conceptually be considered a combination of an interface and a class.

Section 7.2: Simple class

```
class Car {
```

TypeScript Notes for Professionals

Chapter 13: TypeScript basic examples

Section 13.1: 1 basic class inheritance example using extends and super keyword

A generic Car class has some car property and a description method

```
class Car {  
    name: string;  
    engineCapacity: string;  
    constructor(name: string, engineCapacity: string) {  
        this.name = name;  
        this.engineCapacity = engineCapacity;  
    }  
    describeCar() {  
        console.log(`${this.name} car comes with ${this.engineCapacity} displacement`);  
    }  
}
```

HondaCar extends the existing generic car class and adds new property.

```
class HondaCar extends Car {  
    seatingCapacity: number;  
    constructor(name: string, engineCapacity: string, seatingCapacity: number) {  
        super(name, engineCapacity);  
        this.seatingCapacity = seatingCapacity;  
    }  
    describeHondaCar() {  
        super.describeCar();  
        console.log(`this car's comes with seating capacity of ${this.seatingCapacity}`);  
    }  
}
```

Section 13.2: 2 static class variable example - count how many time method is being invoked

here countInstance is a static class variable

```
class StaticTest {  
    static countInstance = number: 0;  
    constructor() {  
        StaticTest.countInstance++;  
    }  
}
```

```
new StaticTest();  
new StaticTest();  
console.log(StaticTest.countInstance);
```

TypeScript Notes for Professionals

80+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with TypeScript	2
Section 1.1: Installation and setup	2
Section 1.2: Basic syntax	4
Section 1.3: Hello World	5
Section 1.4: Running TypeScript using ts-node	6
Section 1.5: TypeScript REPL in Node.js	6
Chapter 2: Why and when to use TypeScript	8
Section 2.1: Safety	8
Section 2.2: Readability	8
Section 2.3: Tooling	8
Chapter 3: TypeScript Core Types	9
Section 3.1: String Literal Types	9
Section 3.2: Tuple	12
Section 3.3: Boolean	12
Section 3.4: Intersection Types	13
Section 3.5: Types in function arguments and return value. Number	13
Section 3.6: Types in function arguments and return value. String	14
Section 3.7: const Enum	14
Section 3.8: Number	15
Section 3.9: String	15
Section 3.10: Array	16
Section 3.11: Enum	16
Section 3.12: Any	16
Section 3.13: Void	16
Chapter 4: Arrays	17
Section 4.1: Finding Object in Array	17
Chapter 5: Enums	18
Section 5.1: Enums with explicit values	18
Section 5.2: How to get all enum values	19
Section 5.3: Extending enums without custom enum implementation	19
Section 5.4: Custom enum implementation: extends for enums	19
Chapter 6: Functions	21
Section 6.1: Optional and Default Parameters	21
Section 6.2: Function as a parameter	21
Section 6.3: Functions with Union Types	23
Section 6.4: Types of Functions	23
Chapter 7: Classes	24
Section 7.1: Abstract Classes	24
Section 7.2: Simple class	24
Section 7.3: Basic Inheritance	25
Section 7.4: Constructors	25
Section 7.5: Accessors	26
Section 7.6: Transpilation	27
Section 7.7: Monkey patch a function into an existing class	28
Chapter 8: Class Decorator	29

Section 8.1: Generating metadata using a class decorator	29
Section 8.2: Passing arguments to a class decorator	29
Section 8.3: Basic class decorator	30
Chapter 9: Interfaces	32
Section 9.1: Extending Interface	32
Section 9.2: Class Interface	32
Section 9.3: Using Interfaces for Polymorphism	33
Section 9.4: Generic Interfaces	34
Section 9.5: Add functions or properties to an existing interface	35
Section 9.6: Implicit Implementation And Object Shape	35
Section 9.7: Using Interfaces to Enforce Types	36
Chapter 10: Generics	37
Section 10.1: Generic Interfaces	37
Section 10.2: Generic Class	37
Section 10.3: Type parameters as constraints	38
Section 10.4: Generics Constraints	38
Section 10.5: Generic Functions	39
Section 10.6: Using generic Classes and Functions:	39
Chapter 11: Strict null checks	40
Section 11.1: Strict null checks in action	40
Section 11.2: Non-null assertions	40
Chapter 12: User-defined Type Guards	42
Section 12.1: Type guarding functions	42
Section 12.2: Using instanceof	43
Section 12.3: Using typeof	43
Chapter 13: TypeScript basic examples	45
Section 13.1: 1 basic class inheritance example using extends and super keyword	45
Section 13.2: 2 static class variable example - count how many time method is being invoked	45
Chapter 14: Importing external libraries	46
Section 14.1: Finding definition files	46
Section 14.2: Importing a module from npm	47
Section 14.3: Using global external libraries without typings	47
Section 14.4: Finding definition files with TypeScript 2.x	47
Chapter 15: Modules - exporting and importing	49
Section 15.1: Hello world module	49
Section 15.2: Re-export	49
Section 15.3: Exporting/Importing declarations	51
Chapter 16: Publish TypeScript definition files	52
Section 16.1: Include definition file with library on npm	52
Chapter 17: Using TypeScript with webpack	53
Section 17.1: webpack.config.js	53
Chapter 18: Mixins	54
Section 18.1: Example of Mixins	54
Chapter 19: How to use a JavaScript library without a type definition file	55
Section 19.1: Make a module that exports a default any	55
Section 19.2: Declare an any global	55
Section 19.3: Use an ambient module	56
Chapter 20: TypeScript installing typescript and running the typescript compiler tsc	57

Section 20.1: Steps	57
Chapter 21: Configure typescript project to compile all files in typescript.	59
Section 21.1: TypeScript Configuration file setup	59
Chapter 22: Integrating with Build Tools	61
Section 22.1: Browserify	61
Section 22.2: Webpack	61
Section 22.3: Grunt	62
Section 22.4: Gulp	62
Section 22.5: MSBuild	63
Section 22.6: NuGet	63
Section 22.7: Install and configure webpack + loaders	64
Chapter 23: Using TypeScript with RequireJS	65
Section 23.1: HTML example using RequireJS CDN to include an already compiled TypeScript file	65
Section 23.2: tsconfig.json example to compile to view folder using RequireJS import style	65
Chapter 24: TypeScript with AngularJS	66
Section 24.1: Directive	66
Section 24.2: Simple example	67
Section 24.3: Component	67
Chapter 25: TypeScript with SystemJS	69
Section 25.1: Hello World in the browser with SystemJS	69
Chapter 26: Using TypeScript with React (JS & native)	72
Section 26.1: ReactJS component written in TypeScript	72
Section 26.2: TypeScript & react & webpack	73
Chapter 27: TSLint - assuring code quality and consistency	75
Section 27.1: Configuration for fewer programming errors	75
Section 27.2: Installation and setup	75
Section 27.3: Sets of TSLint Rules	76
Section 27.4: Basic tslint.json setup	76
Section 27.5: Using a predefined ruleset as default	76
Chapter 28: tsconfig.json	78
Section 28.1: Create TypeScript project with tsconfig.json	78
Section 28.2: Configuration for fewer programming errors	79
Section 28.3: compileOnSave	80
Section 28.4: Comments	80
Section 28.5: preserveConstEnums	81
Chapter 29: Debugging	82
Section 29.1: TypeScript with ts-node in WebStorm	82
Section 29.2: TypeScript with ts-node in Visual Studio Code	83
Section 29.3: JavaScript with SourceMaps in Visual Studio Code	84
Section 29.4: JavaScript with SourceMaps in WebStorm	84
Chapter 30: Unit Testing	86
Section 30.1: tape	86
Section 30.2: jest (ts-jest)	87
Section 30.3: Alsation	89
Section 30.4: chai-immutable plugin	89
Credits	91
You may also like	93

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
<https://goalkicker.com/TypeScriptBook>

This *TypeScript Notes for Professionals* book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official TypeScript group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

[Click here to download full PDF material](#)