TypeScript

Mohamed Mukthar Ahmed

41

TypeScript

Outline

- Introduction
- What TS Offers?
- Types
  - Built-in Types, Special Types
- Trying in REPL
- Function - Enhancements
- Object-Oriented Features
  - Classes
    - Defining a class – Properties, Constructor and Methods
    - Creating Instances
    - Invoking Methods
- Inheritance
- Utilities
  - Fat Arrow Functions
  - Template Strings

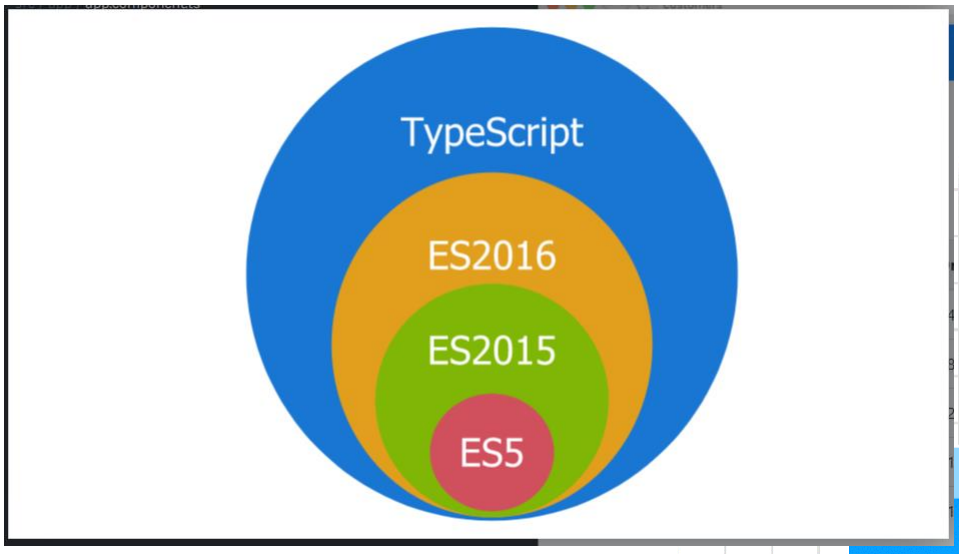
Mohamed Mukthar Ahmed

42

TypeScript

- Angular is built in a JavaScript-like language called **TypeScript**
- There are a lot of great reasons to use TypeScript instead of plain JavaScript.
- TypeScript isn't a completely new language, it's a superset of ES6.

TypeScript



TypeScript – More

- We have **transpilers** (or sometimes called **transcompiler**).
- The TypeScript **transpiler** takes our TypeScript code as input and outputs ES5 code that nearly all browsers understand.
- To convert ES6 code (not TypeScript) to ES5 there are two major ES6-to-ES5 transpilers:
  - `**traceur**` by Google and
  - `**babel**` created by the JavaScript community.is small.
- TypeScript is an official collaboration between **Microsoft** and **Google**.

Mohamed Mukthar Ahmed

45

What TypeScript Offers?

- There are **SIX** big improvements that TypeScript bring over ES5:
  - Types
  - Function :: Enhancements
  - Classes :: Object-Oriented Features
  - Decorators
  - Imports
  - Language Utilities (ex. de-structuring)

Mohamed Mukthar Ahmed

46

Types

- The **major improvement** of TypeScript over ES6, that gives the language its name, is the typing system.
- One of the great things about type checking is that:
  - it helps when writing code because it can prevent bugs at compile time and
  - it helps when reading code because it clarifies your intentions
- It's also worth noting that **types** are **optional** in TypeScript.
- TypeScript's basic types are the same ones we've been using implicitly when we write "normal" JavaScript code: **strings**, **numbers**, **booleans**, etc.

Types

- The new **TypeScript** syntax is a natural evolution from **ES5**, we still use var but now we can optionally provide the variable type along with its name:

```
var fullName: string;
```

- When declaring functions we can use types for arguments and return values:

```
function greetText(name: string): string {
 return "Hello " + name;
}
```

- What happens if we try to write code that doesn't conform to our declared typing?

Types

- What happens if we try to write code that doesn't conform to our declared typing?

```
function hello(name: string): string {
 return 12;
}
```

- If we try to compile it, we'll get the following error:
- **filename.ts(2,12): error TS2322: Type 'number' is not assignable to type 'string'.**
- By using types it can save us from a lot of bugs down the road.

Mohamed Mukthar Ahmed

Type – Error

```
1   let firstName: string;
2
3   firstName = 123;
4   console.log( firstName );
```

```
ex_01.ts:3:1 - error TS2322: Type 'number' is not assignable to type 'string'.

3 firstName = 123;
  ~~~~~~~~~~

Found 1 error in ex_01.ts:3
```

Mohamed Mukthar Ahmed

Trying it out with a REPL

- To understand the syntax of TypeScipt, let's install a nice little utility called **TSUN** (TypeScript Upgraded Node):

```
C:\>npm install -g tsun
```

```
Command Prompt - tsun

C:\>tsun
TSUN : TypeScript Upgraded Node
type in TypeScript expression to evaluate
type :help for commands in repl

>
```

- That little **>** is the prompt indicating that **TSUN** is ready to take in commands.

Mohamed Mukthar Ahmed

Built-in Types

- **String**: holds text and is declared using the `string` type.

```
var fullName: string = 'Mukthar Ahmed';
```

- **Number**: any type of numeric value.
  - In TypeScript, all numbers are represented as floating point.

```
var age: number = 54;
```

- **Boolean**: holds either true or false as the value.

```
var married: boolean = true;
```

- **Array**: a collection to which we specify the type of the objects in the Array

Mohamed Mukthar Ahmed

Built-in Types

- **Array**: a collection to which we specify the type of the objects in the Array
  - Done either the Array<type> or type[] notations:

```
var jobs: Array<string> = ['IBM', 'Microsoft', 'Google'];
var jobs: string[] = ['Apple', 'Dell', 'HP'];
```

- Or similarly with a number:

```
var chickens: Array<number> = [1, 2, 3];
var chickens: number[] = [4, 5, 6];
```

- **Enums**: Enums work by naming numeric values.

Mohamed Mukthar Ahmed

53

Built-in Types

- **Enums**: Enums work by naming numeric values.
- For instance, if we wanted to have a fixed list of roles a person may have we could write this:

```
enum Role {Clerk, Manager, Admin};
var role: Role = Role.Clerk;

enum Role {Clerk = 3, Manager, Admin};
var role: Role = Role.Clerk;

enum Role {Clerk=3, Manager=5, Admin=7};
var role: Role = Role.Clerk;
```

Mohamed Mukthar Ahmed

54

Built-in Types

- You can also look up the name of a given enum by using its value:

```
enum Role {Clerk, Manager, Admin};
console.log('Roles:  ',  Role[0],  ',',
Role[1], 'and', Role[2]);
```

- **Any**: any is the default type if we omit typing for a given variable.
- Having a variable of type any allows it to receive any kind of value:

```
var something: any = 'as string';
something = 1;
something = [1, 2, 3];
```

Mohamed Mukthar Ahmed

55

Built-in Types

- **Void**: Using `void` means there's no type expected.
- This is usually in functions with no return value.

Mohamed Mukthar Ahmed

56

Functions

- TypeScript functions have some **awesome** features:
  - Default parameters
  - Optional parameters
  - Named parameters
  - Rest parameters
  - Union for different types of parameters
- Let's look into them one-by-one

```
// [2] Function with default parameter
function power(value: number, exponent: number = 1) {
    return value ** exponent;
}
result = power(2, 3);
console.log(result);
result = power(12);
console.log("Using Default parameter:" + result);
```

Mohamed Mukthar Ahmed

57

Functions

```
// [3] Function with optional parameter
function add(a: number, b: number, c?: number) {
    return a + b + (c || 0);
}
result = add(2, 3, 5);
console.log(result);
result = add(10, 20);
console.log("Using Optional parameter:" + result);
```

```
// [4] Function with named parameters
function divide({ dividend, divisor }:
    { dividend: number, divisor: number }) {
    return dividend / divisor;
}
result = divide({dividend:10, divisor:3});
console.log(result);
result = divide({divisor:3, dividend:20});
console.log("Using Named parameters:" + result);
```

Mohamed Mukthar Ahmed

58

Functions

```
// [5] Function with 'rest' parameters
function adding(a: number, b: number, ...rest: number[]) {
    return a + b + rest.reduce((p, c) => p + c, 0);
}
result = adding(1, 2, 3, 4, 5);
console.log("The rest parameters:" + result);
```

```
// [6] Function using UNION (|) to pass different type of data
function printStatusCode(code: string|number): void {
    console.log(`Status Code is ${code}`);
}

printStatusCode('404');
printStatusCode(404);
```


Classes

- In JavaScript ES5 object oriented programming was accomplished by using prototype-based objects.
- In ES6 we finally have built-in classes in JavaScript.
- To define a class we use the new `**class**` keyword and give our class a name and a body:

```
class Vehicle {
}
```

- Classes may have properties, methods, and constructors.

Properties

- Properties define data attached to an instance of a class.
- For example, a class named `Person` might have properties like first_name, last_name and age.

```
class Person {
 first_name: string;
 last_name: string;
 age: number;
}
```

Mohamed Mukthar Ahmed

61

Methods

- Methods are functions that run in context of an object.
- If we wanted to add a way to greet a Person using the class above, we would write something like:

```
class Person {
 first_name: string;
 last_name: string;
 age: number;
 greet() {
    console.log("Hello", this.first_name);
  }
}
```

Mohamed Mukthar Ahmed

62

Methods

- In order to invoke the greet method, you would need to first have an instance of the Person class.
- Here's how we do that:

```
// declare a variable of type Person
var p: Person;
// instantiate a new Person instance
p = new Person();
// give it a first_name
p.first_name = 'Mukthar';
// call the greet method
p.greet();
```

-

Mohamed Mukthar Ahmed

63

Constructors

- A constructor is a special method that is executed when a new instance of the class is being created.
- Usually, the constructor is where you perform any **initial setup** for new objects.
- Constructor methods must be named constructor.
- They can optionally take parameters but they can't return any value.
- **NOTE**: When a class has no constructor defined explicitly one will be created automatically.

Mohamed Mukthar Ahmed

64

Constructors

- **NOTE**: When a class has no constructor defined explicitly one will be created automatically.

```
class Vehicle {
}
var v = new Vehicle();
// Explicit constructor is used.
```

- In TypeScript you can have **only one constructor** per class.

- Constructors can take parameters when we want to parameterize our new instance creation.

Mohamed Mukthar Ahmed

65

Constructors

- Constructors can take parameters when we want to parameterize our new instance creation.

```
class Person {
  first_name: string;
  last_name: string;
  age: number;
  constructor(first_name: string,
        last_name: string, age: number) {
    this.first_name = first_name;
    this.last_name = last_name;
    this.age = age;
  }
```

Mohamed Mukthar Ahmed

66

Constructors

```
class Person {
  :
  greet() {
    console.log("Hello", this.first_name);
  }
  ageInYears(years: number): number {
    return this.age + years;
  }
}
var p: Person =
new Person('Mukthar', 'Ahmed', 54);
p.greet();
```

Mohamed Mukthar Ahmed

67

Edit Selection View Go Run Terminal Help ex_03.ts - tscripts - Visual Studio ...

TS ex_03.ts

TS ex_03.ts > ...

```
// Defining a class, creating instance and invoking methods

class Person {
    firstName: string;
    lastName: string;
    age: number;

    constructor(firstName, lastName, age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    displayPersonDetails() : void {
        console.log("First Name : " + this.firstName);
        console.log("Last Name  : " + this.lastName);
        console.log("Age        : " + this.age);
    }
}

// Creating an instance -> i.e. Object
const p = new Person('Mukthar', 'Ahmed', 54);
// Invoking the method
p.displayPersonDetails();
```

Mohamed Mukthar Ahmed

68

Inheritance

- Another important aspect of object oriented programming is **inheritance**.
- Inheritance is a way to indicate that a class receives behavior from a parent class.
- Then we can override, modify or augment those behaviors on the new class.
- **TypeScript** fully supports **inheritance**.
- Achieved through the `**extends**` keyword.

Mohamed Mukthar Ahmed

69

Inheritance – Code Snippet

- The base/super class

```
// Understanding Inheritence
class Report {
    data: string[];

    constructor(data: string[]) {
        this.data = data;
    }

    run() {
        this.data.forEach(function(line) {
            console.log(line);
        });
    }
}
```

Mohamed Mukthar Ahmed

70

Inheritance – Code Snippet

- The derived/sub class

```
// Inheritence
class TabbedReport extends Report {
    headers: string[];

    constructor(headers: string[], data: string[]) {
        super(data);
        this.headers = headers;
    }

    run() {
        console.log(this.headers);
        super.run();
    }
}
```

Mohamed Mukthar Ahmed

Utilities

- **ES6**, and by extension **TypeScript** provides a number of syntax features that make programming really enjoyable.

- Two important ones are:
  - Fat Arrow Functions
  - Template Strings

Mohamed Mukthar Ahmed

Fat Arrow Functions

- Fat arrow => functions are a shorthand notation for writing functions..

```
// ES5-like example
var names = ['Amar', 'Akbar', 'Antony'];
names.forEach(function(name) {
console.log(name); });
```

- However with the => syntax we can instead rewrite it like so:

```
// Typescript example
var names: string[] = ['Amar', 'Akbar',
'Antony'];
names.forEach( (n) => console.log(n) );
```

Mohamed Mukthar Ahmed

73

Fat Arrow Functions

- Parentheses are optional when there's only one parameter.
- The => syntax can be used both as an expression or as a statement

```
// expression
var evens = [2,4,6,8];
var odds = evens.map(v => v + 1);
// statement
data.forEach( line => {
 console.log(line.toUpperCase())
});
```

- The fat arrow shares this with its surrounding code.

Mohamed Mukthar Ahmed

74

Template Strings

- In ES6 new template strings were introduced.
- The two great features of template strings are
  1. Variables within strings (without being forced to concatenate with +)
  2. Multi-line strings
- Variables in strings : This feature is also called “string interpolation.”

```
var firstName = "Mukthar";
var lastName = "Ahmed";
// interpolate a string
var g = `Hello ${firstName} ${lastName}`;
```


Template Strings

- In ES6 new template strings were introduced.
- The two great features of template strings are
  1. Variables within strings (without being forced to concatenate with +)
  2. Multi-line strings
- Multiline strings: Backtick strings can be in multi-lines.

```
var template = `
<div>
  <h1>Hello</h1>
  <p>This is a great website</p>
</div>
`;
```

Wrapping up!

- There are a variety of **other features** in **TypeScript**/ES6 such as:
  - Interfaces
  - Generics
  - Importing and Exporting Modules
  - Decorators
  - Destructuring

Mohamed Mukthar Ahmed

77

Q&A

Mohamed Mukthar Ahmed

78