Angular

Benjamin Longearet - @blongearet

Part 5
TypeScript

- Why using TypeScript?
- Installation & first usage
- Typings
- Functions
- Interfaces
- Classes
- Modules
- ES2016 Syntax: Constant, template string, arrow function, spread
- operator

Part 5 - TypeScript

Why using TypeScript

- TypeScript has great tools!
- TypeScript is a superset of JavaScript
- Typings prevents early from stupid error ¯_(ツ)_/¯
- Allows us to transpile in ESx with same codebase
- Large adoption in the Frontend ecosystem
- Useful when working on a large codebase

Part 5 - TypeScript

Installation & first usage

- Quick tutorial from typescriptlang.org
- Use npm to install TypeScript
  ```
  $ npm install -g typescript
  ```
- Use `tsc` to compile your *.ts file
- Use *tsconfig.json* to configure the compiler
- Write a file with some code to try the compilation

Typings

Part 5 - TypeScript

Typings

- JavaScript

```
var n = 3;
```

- TypeScript

```
let n: number = 3;
```

Part 5 - TypeScript

Typings are great!

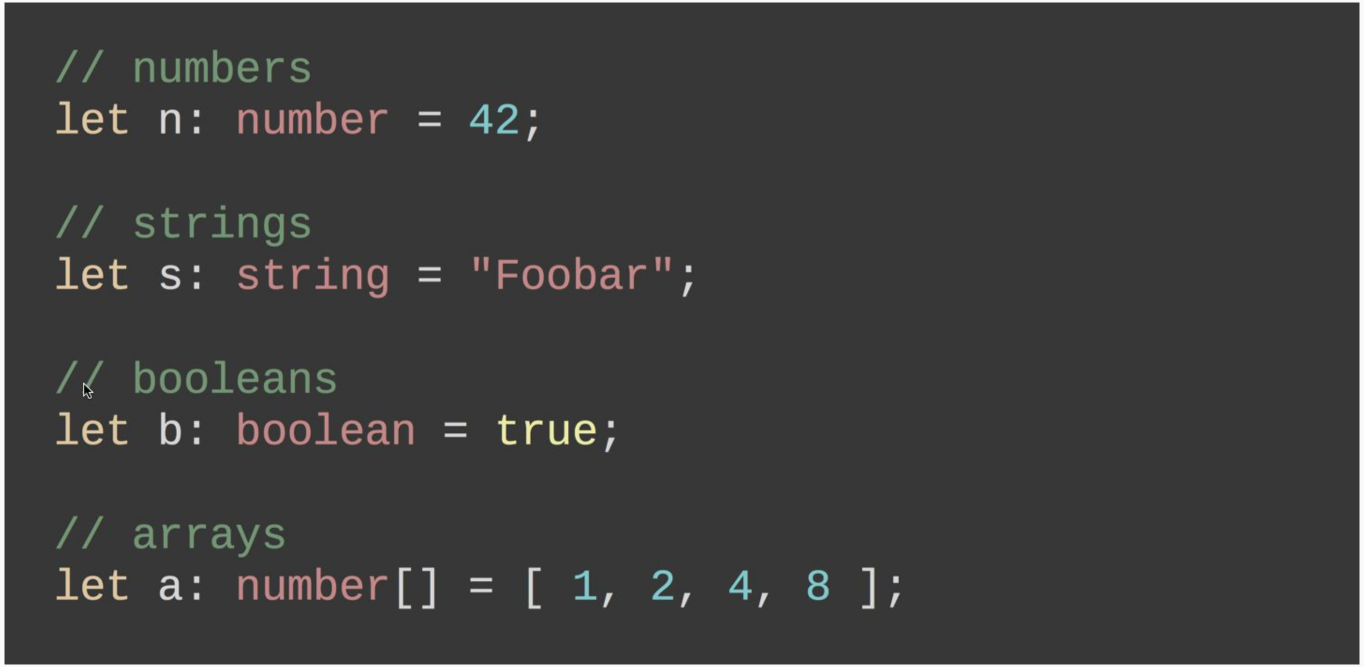
```
let n: number = 1;
// > 1

n = 2;
// > 2

n = "foobar";
// Error: Type 'string' is not assignable
//        to type 'number'.
```

Part 5 - TypeScript

Typings: Basic Types



```typescript
// numbers
let n: number = 42;

// strings
let s: string = "Foobar";

// booleans
let b: boolean = true;

// arrays
let a: number[] = [ 1, 2, 4, 8 ];
```

Part 5 - TypeScript

Typings: Enums

```
enum Currency {
  EUR, USD, JPY, GBP
};

let c: Currency = Currency.EUR;

c = "FOOBAR";
// Error: Property 'FOOBAR' does not exist on
//        type 'typeof Currency'.
```

Part 5 - TypeScript

Typings: Tuples

```
let price: [ number, string ];

price = [ 12.99, "EUR" ];
// > OK

price = [ "EUR", 12.99 ];
// Error: Type '[string, number]' is not
//        assignable to type '[number, string]'.
```

Part 5 - TypeScript

Typings: Any

```
let a: any;

a = "Foobar";

a = false;

a = [ 42, "Foobar", true ];

a = document.getElementById( "foobar" );
```

Part 5 - TypeScript

Typings: Assertions

```
let value: any = "Christian";

(<string>value).substring( 0, 5 );
// > "Chris"
```

Part 5 - TypeScript

Typings: Assertions

```
let value: any = "Christian";

(<string>value).substring( 0, 5 );
// > "Chris"
```

```
let value: any = "Christian";

(value as string).substring( 0, 5 );
// > "Chris"
```

Part 5 - TypeScript

Typings: Inference

```
let n = 3;           // inferred type is 'number'

n = "foobar";
// Error: Type 'string' is not assignable
//        to type 'number'.
```

```
let n = null;          // inferred type is 'any'
if( something ) {
  n = 42;              // OK
  n = "foobar";        // OK? :-(
}
```

Part 5 - TypeScript

Typings: Advance types

```
let t: string|number;      // union type

t = 42;
// > OK

t = "foobar";
// > OK

t = true;
// Error: Type 'boolean' is not assignable to type
//          'string | number'.
```

Part 5 - TypeScript

Typings: Advance types

```
type MyType = string|number;    // type alias

let t: MyType = "foobar";
```

```
type Mode = "simple" | "advanced";

let mode: Mode = "simple";

mode = "foobar";
// Error: Type '"foobar"' is not assignable to
//          type 'Mode'
```

Functions

Part 5 - TypeScript

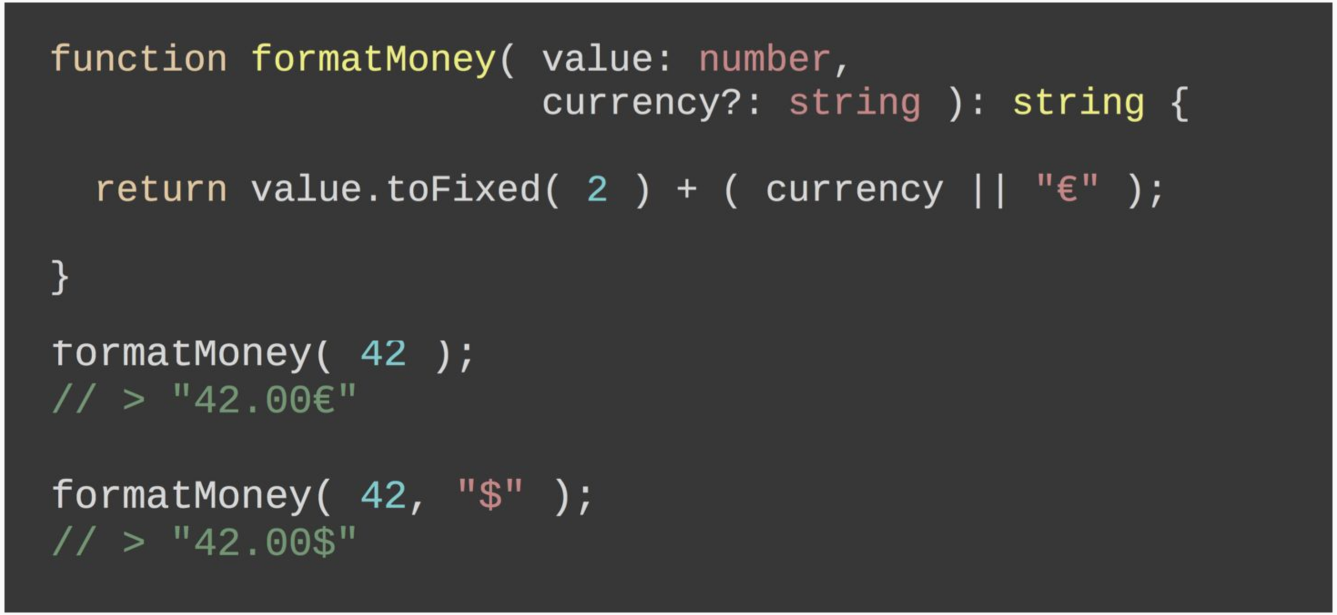
Functions: typed functions

```
function formatEuro( value: number ): string {
  return value.toFixed( 2 ) + "€";
}
```

```
formatEuro( 42 );
// > "42.00€"
```

Part 5 - TypeScript

Functions: Optional parameters



```
function formatMoney( value: number,
                      currency?: string ): string {

  return value.toFixed( 2 ) + ( currency || "€" );

}

formatMoney( 42 );
// > "42.00€"

formatMoney( 42, "$" );
// > "42.00$"
```

Part 5 - TypeScript

Functions: Optional parameters

```
function formatMoney( value: number,
                      currency: string = "€" ): string {

  return value.toFixed( 2 ) + currency;

}

formatMoney( 42 );
// > "42.00€"

formatMoney( 42, "$" );
// > "42.00$"
```

Part 5 - TypeScript

Functions: Default parameters

```
function formatMoney( value: number,
                      currency: string = "€" ): string {

  return value.toFixed( 2 ) + currency;

}

formatMoney( 42 );
// > "42.00€"

formatMoney( 42, "$" );
// > "42.00$"
```

Interfaces

Part 5 - TypeScript

Interfaces

```
let money = {
  amount: 42,
  currency: "€"
};
```

```
interface Money {
  amount: number;
  currency: string;
}
```

Part 5 - TypeScript

Interfaces: functions

```
interface Money {
  amount: number;
  currency: string;
  asString: () => string;
}

let money: Money = {
  amount: 42,
  currency: "€",
  asString: function(): string {
    return this.amount.toFixed( 2 ) + this.currency;
  }
};

money.asString();      // > 42.00€
```

Part 5 - TypeScript

Interfaces: functions

```
interface AsStringFunc {
  (): string;
}

interface Money {
  amount: number;
  currency: string;
  asString: AsStringFunc;
}

let money: Money = { ... };

money.asString();     // > 42.00€
```

Part 5 - TypeScript

Interfaces: extends

```
interface AsStringFunc {
  (): string;
}

interface Printable {
  asString: AsStringFunc;
}

interface Money extends Printable {
  amount: number;
  currency: string;
}
```

Part 5 - TypeScript

Interfaces: structural sub-typings

```
interface Foo {
  value: number;
}

interface Bar {
  value: number;
}

let foo: Foo = {
  value: 3
};

let bar: Bar = foo;   // OK
```

Classes

Part 5 - TypeScript

Classes: the old-old way

```
var Money = function ( amount, currency ) {
  this.amount = amount;
  this.currency = currency;
};

Money.prototype.asString = function () {
  return this.amount.toFixed( 2 ) + this.currency;
};

var money = new Money( 42, "€" );

money.asString();
// > 42.00€
```

Part 5 - TypeScript

Classes: the ES6 way

```
class Money {

  constructor( amount, currency ) {
    this.amount = amount;
    this.currency = currency;
  }

  asString() {
    return this.amount.toFixed( 2 ) + this.currency;
  }

}

let money = new Money( 42, "€" );
```

Part 5 - TypeScript

Classes: the TypeScript way

```
class Money {

  private amount: number;
  private currency: string;

  constructor( amount: number, currency: string ) {
    this.amount = amount;
    this.currency = currency;
  }

  asString(): string {
    return this.amount.toFixed( 2 ) + this.currency;
  }

}
```

Part 5 - TypeScript

Classes: parameter properties

```
class Money {

  constructor( private amount: number,
               private currency: string ) {
    // empty
  }

  asString(): string {
    return this.amount.toFixed( 2 ) + this.currency;
  }

}
```

Part 5 - TypeScript

Classes: implementing interface

```
interface Printable {
  asString(): string;
}

class Money implements Printable {

  constructor( private amount: number,
               private currency: string ) {
    // nothing here
  }

  asString(): string {
    return this.amount.toFixed( 2 ) + this.currency;
  }
}
```

There is more...

- Decorators
- Inheritance
- Abstract classes
- Static properties
- Visibility modifiers
- Accessors
- Generics

Modules

Part 5 - TypeScript

Modules: Export/Import

```
// math.ts
export function max( a: number, b: number ): number {
  return a > b ? a : b;
}

export let PI = 3.14156;
```

```
// foobar.ts
import { max, PI } from "./math.ts";

max(9, 13) === 13;          // > true
PI === 3.14156;             // > true
```

Part 5 - TypeScript

Modules: Export/Import

```
// math.ts
export function max( a: number, b: number ): number {
  return a > b ? a : b;
}

export let PI = 3.14156;
```

```
// foobar.ts
import * as math from "./math.ts";

math.max(9, 13) === 13   // > true
math.PI === 3.14156      // > true
```

Part 5 - TypeScript

Modules: Export/Import

```
// money.ts
export class Money {

  constructor( private amount: number,
               private currency: string ) {
  }

  asString(): string {
    return this.amount.toFixed( 2 ) + this.currency;
  }
}
```

```
import { Money } from "./money.ts";

let m = new Money( 42, "€" );
```

ES2016 (ES7)

Part 5 - TypeScript

ES6, ES2016, ES.next … Kezako

- JavaScript was originally named JavaScript to use success of Java
- ECMA is an organization that standardizes information
- Result is a new standard, known as ECMAScript (ES is shortcut)
- ES1: June 1997 --- ES2: June 1998 --- ES3: Dec. 1999 --- ES4: Abandoned
- ES5: Dec. 2009 (10 years later ¯_(ツ)_/¯)
- ES6 / ES2015: June 2015 (6 years later 💤 💪)
- ES2016 (ES7): June 2016
- ES2017 (ES8): June 2017 (yearly schedule)
- ES.Next (incoming version)

Part 5 - TypeScript

ES2016: Constants

```
const users = [ "Christian" ];

users.push( "Jim" );
// > 2

users = [ "Bob" ];
// Error: Left-hand side of assignment cannot
//        be a constant or a read-only property.
```

ES2016: Template string

```
let name = "Christian";
let count = 213;

let message =
    `Hello ${name}, you have ${count} messages.`;
```

```
let html =
    `<h1>Hello ${name}</h1>
     <p>
       You have ${count} unread messages
     </p>`;
```

Part 5 - TypeScript

ES2016: classic functions

```
let numbers = [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ];

numbers.filter( function(n) {
  return n % 2 !== 0;
} );
// > [ 1, 3, 5, 7, 9 ]
```

Part 5 - TypeScript

ES2016: arrow functions

```
numbers.filter( n => {
  return n % 2 !== 0;
} );
// > [ 1, 3, 5, 7, 9 ]
```

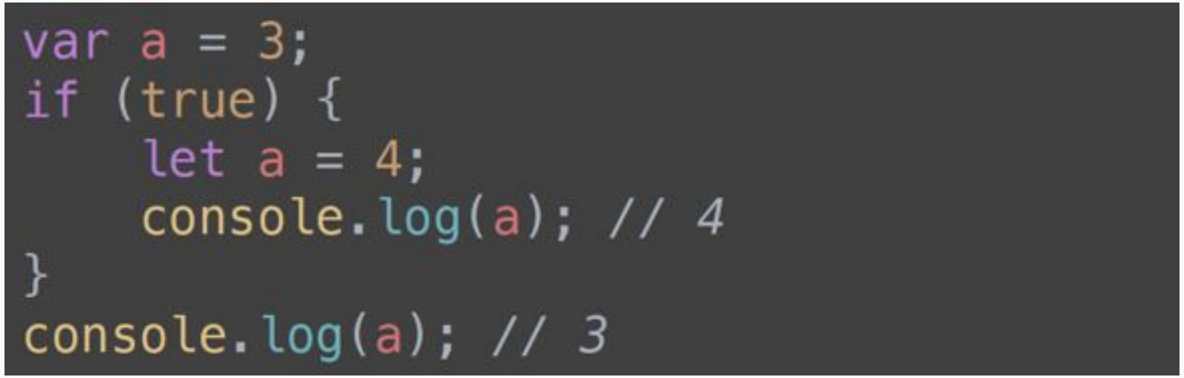
```
numbers.filter( n => n % 2 !== 0 );
// > [ 1, 3, 5, 7, 9 ]
```

```
numbers.filter( n => n % 2 );
// > [ 1, 3, 5, 7, 9 ]
```

Part 5 - TypeScript

ES2016: const/var/let

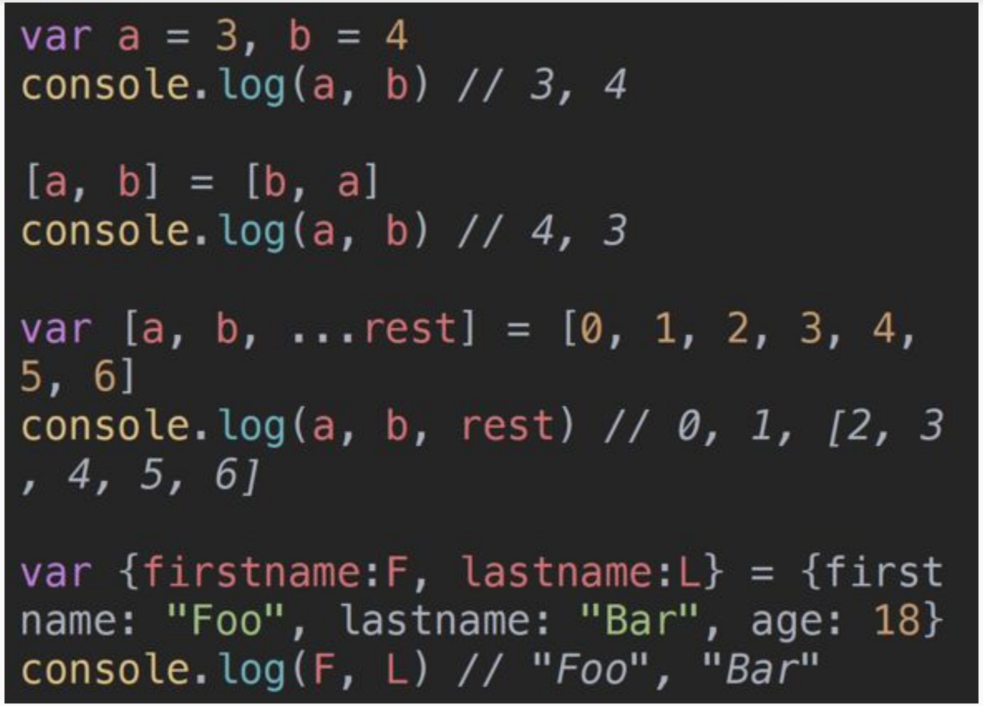
- **const**: use for read-only variable
- **var**: declare a variable
- **let**: declare a block-scoped variable



```
var a = 3;
if (true) {
    let a = 4;
    console.log(a); // 4
}
console.log(a); // 3
```

Part 5 - TypeScript

ES2016: destructuring



```
var a = 3, b = 4
console.log(a, b) // 3, 4

[a, b] = [b, a]
console.log(a, b) // 4, 3

var [a, b, ...rest] = [0, 1, 2, 3, 4,
5, 6]
console.log(a, b, rest) // 0, 1, [2, 3
, 4, 5, 6]

var {firstname:F, lastname:L} = {first
name: "Foo", lastname: "Bar", age: 18}
console.log(F, L) // "Foo", "Bar"
```

Part 5 - TypeScript

Play with it!

- TypeScript playground
  http://www.typescriptlang.org/play/
- ES6 Features
- ES7 features:
  - async/await function
  - Object destructuring
  - etc.

Part 5 - TypeScript

Ok, now your turn ;-)

- Clone my typescript project
  https://github.com/blongearet/angular-course-typescript
- Follow the README file, and most important … Enjoy it 💪