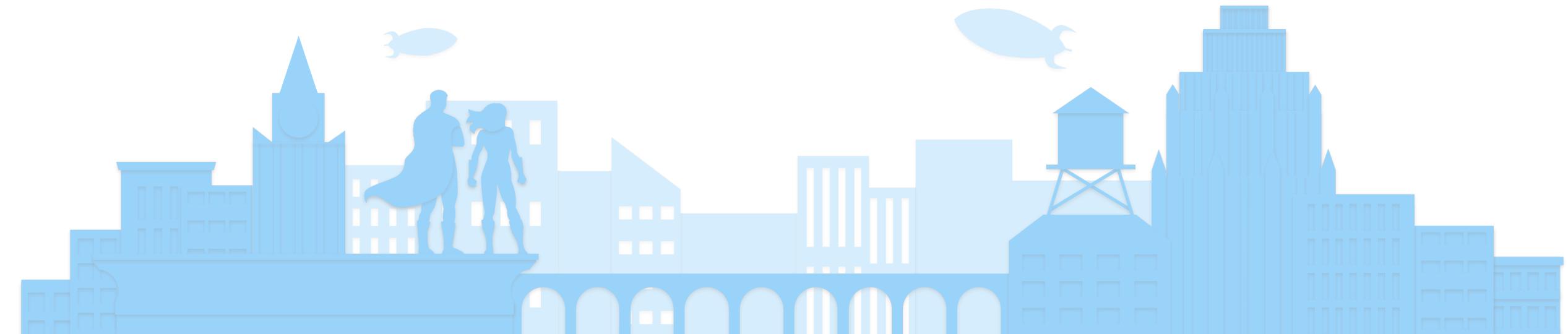


# Angular

TypeScript



# Section Overview

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

# Why using TypeScript?

- TypeScript Has Great Tools
- TypeScript is a Superset of JavaScript
- Typings prevents early from stupid errors
- Allows us to transpile ine ES5 or ES6 with same codebase
- Large adoption in the Frontend ecosystem
- Useful when working on a large codebase

# Installation

- Use NPM to install typescript  
`npm install -g typescript`
- Use `tsc` to compile your \*.ts file
- Use `tsconfig.json` to configure the compilation

# Typings

# Typings

- JavaScript

```
var n = 3;
```

- TypeScript

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

# Typings

- Types are great!

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

n = 2;
// > 2

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

# Typings : Basic types

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

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

// booleans
let b: boolean = true;

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

# 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'.
```

# 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]'.
```

# Typings : Any

```
let a: any;  
  
a = "Foobar";  
  
a = false;  
  
a = [ 42, "Foobar", true ];  
  
a = document.getElementById( "foobar" );
```

# Typings : Assertions

```
let value: any = "Christian";  
(<string>value).substring( 0, 5 );  
// > "Chris"
```

# Typings : Assertions

```
let value: any = "Christian";  
  
(<string>value).substring( 0, 5 );  
// > "Chris"
```

```
let value: any = "Christian";  
  
(value as string).substring( 0, 5 );  
// > "Chris"
```

# 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? :-((
}
```

# 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'.
```

# 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

# Functions: typed functions

```
function formatEuro( value: number ): string {  
    return value.toFixed( 2 ) + "€";  
}  
  
formatEuro( 42 );  
// > "42.00€"
```

# Functions: Optional parameters

```
function formatMoney( value: number,  
                      currency?: string ): string {  
  
    return value.toFixed( 2 ) + ( currency || "€" );  
  
}  
  
formatMoney( 42 );  
// > "42.00€"  
  
formatMoney( 42, "$" );  
// > "42.00$"
```

# Functions: Default parameters

```
function formatMoney( value: number,  
                      currency: string = "€" ): string {  
  
    return value.toFixed( 2 ) + currency;  
  
}  
  
formatMoney( 42 );  
// > "42.00€"  
  
formatMoney( 42, "$" );  
// > "42.00$"
```

# Interfaces

# Interfaces

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

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

# 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€
```

# Interfaces: Functions

```
interface AsStringFunc {  
    (): string;  
}  
  
interface Money {  
    amount: number;  
    currency: string;  
    asString: AsStringFunc;  
}  
  
let money: Money = { ... };  
  
money.asString();      // > 42.00€
```

# Interfaces: Extends

```
interface AsStringFunc {  
    (): string;  
}  
  
interface Printable {  
    asString: AsStringFunc;  
}  
  
interface Money extends Printable {  
    amount: number;  
    currency: string;  
}
```

# Interfaces: Structural sub-typings

```
interface Foo {  
    value: number;  
}  
  
interface Bar {  
    value: number;  
}  
  
let foo: Foo = {  
    value: 3  
};  
  
let bar: Bar = foo;    // OK
```

# Classes

# Classes: The 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€
```

# Classes: ES6

```
class Money {  
  
    constructor( amount, currency ) {  
        this.amount = amount;  
        this.currency = currency;  
    }  
  
   asString() {  
        return this.amount.toFixed( 2 ) + this.currency;  
    }  
  
}  
  
let money = new Money( 42, "€" );
```

# Classes: TypeScript

```
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;  
    }  
}
```

# Classes: Parameter properties

```
class Money {  
  
    constructor( private amount: number,  
                private currency: string ) {  
        // empty  
    }  
  
    asString(): string {  
        return this.amount.toFixed( 2 ) + this.currency;  
    }  
}
```

# 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

# 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
```

# 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
```

# 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: 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

# 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>`;
```

# 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 ]
```

# 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 ]
```

# 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
```

# 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"
```

# How play with!

- TypeScript playground  
<http://www.typescriptlang.org/play/>
- ES6 standards  
[LINK]