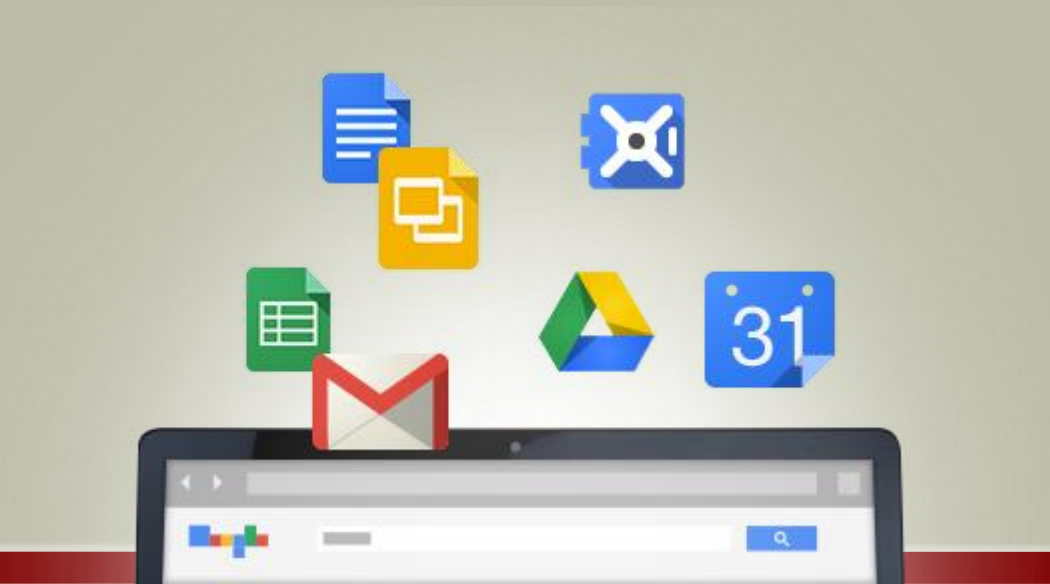
An Introduction To Angular 2

Ode to Code

History

- Started in 2009 @ Google
- Built from experience with large web applications



Concepts and Goals

Separation of Concerns

Testability

Extensibility

Forward Looking

Tools & Languages

Getting Started : Packages

- angular 2 – the framework
- typescript – the language
- systemjs - the module loader

```
npm init –y
npm install angular2 typescript rxjs
            systemjs reflect-metadata --save
```

Getting Started: Shell Page

```
<html>
    <head>
        <title>ng2</title>
    </head>
    <body>
        <app>loading...</app>
    </body>
    <script src="/node_modules/systemjs/dist/system.src.js"></script>
    <script src="/node_modules/reflect-metadata/reflect.js"></script>
    <script src="/node_modules/typescript/lib/typescript.js"></script>
    <script src="/node_modules/rxjs/bundles/rx.js"></script>
    <script src="/node_modules/angular2/bundles/angular2.dev.js"></script>
    <script>

        System.config({
            transpiler: "typescript",
            typescriptOptions: { emitDecoratorMetadata: true}
        });

        System.import("./app/main.ts");

    </script>
</html>
```

Getting Started: First Script

```
import {bootstrap} from "angular2/platform/browser";
import {Component} from "angular2/core";

@Component({
    selector: "app",
    template: "<h1>{{message}}</h1>"
})
class App {
    message: string
    constructor() {
        this.message = "Hello, from ng2!";
    }
}

bootstrap(App);
```

Other Starting Points

- **Yeoman generators**
  - generator-angular2
  - generator-angular2-gulp-webpack
- **Seed projects**
  - angular2-seed (npm)
  - ng2-play (github:pkozlowski-opensource)
  - angular-starter (github:eladrk)

Directives

- **Directives extend HTML**
  - Add behavior or change the appearance of the DOM
  - Angular compiles markup to process directives
- **You can build your own directives**
  - In fact, this is encouraged

```
<div>

    <h1>{{title}}</h1>

    <div *ngIf="firstName">Hello, {{firstName}}</div>

</div>
```

Components

- **Components include a template**
  - Like directives, can customize HTML

```
import {bootstrap} from "angular2/platform/browser";
import {Component} from "angular2/core";

@Component({
    selector: "app",
    templateUrl: "/app/main.html"
})
class App {

    title: string
    firstName: string

    constructor() {
        this.title = "ng2";
        this.firstName = "Scott";
    }
}

bootstrap(App);
```

Templates

- **Responsible for presenting the model**
  - Using directives and {{ interpolation }}

```
<h1>{{title}}</h1>

<div *ngIf="firstName">Hello, {{firstName}}</div>

<ul>
    <li *ngFor="#movie of movies">
        {{movie.title}}
    </li>
</ul>
```

Models

- **Plain Old JavaScript (2015)**
- **Component fields exposed for binding**

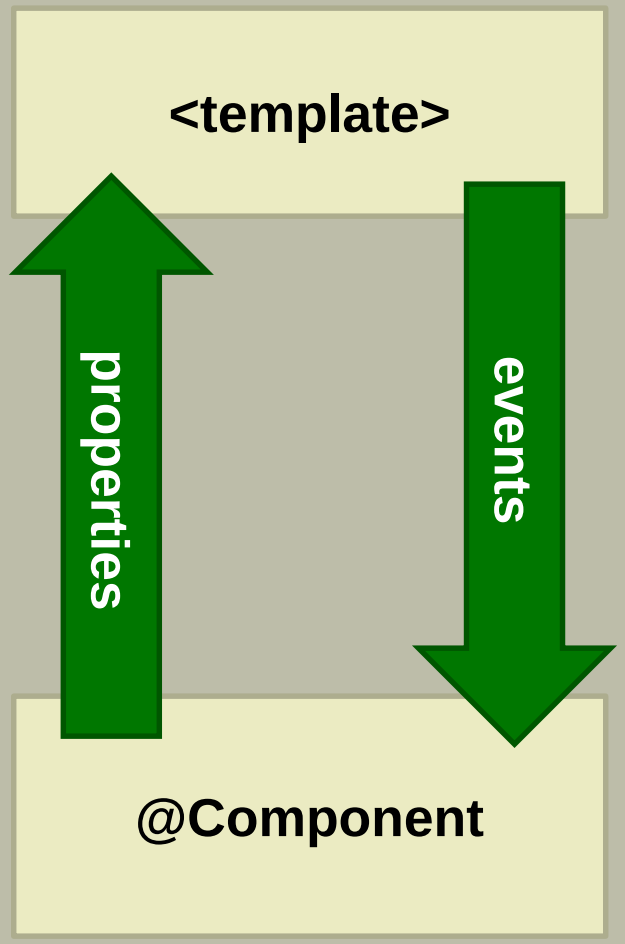
```
@Component({
    selector: "app",
    templateUrl: "/app/main.html"
})
export class App {

    title: string
    firstName: string
    movies: Array<Movie>

    constructor() {
        this.title = "ng2";
        this.firstName = "Scott";
        this.movies = [
            new Movie("Star Wars", 120, 1979),
            new Movie("Jurrasic Park", 130, 1992),
            new Movie("SP", 300, 2014)
        ];
    }
}
```

Essence

- **Components**
  - No direct DOM manipulation
- **Templates**
  - No serious model manipulation



Template Syntax

- **Display text using {{expressions}}**
- **Set properties using [expressions]**
- **Handle events using (expressions)**
  - [] () deal with properties, not attributes
  - Expressions limit side effects and global scope

```
<div>Count: {{movies.length}}</div>
<div>2 + 2 = {{ 2 + 2 }}</div>
<ul [style.color]="color">
    <li *ngFor="#movie of movies">
        {{movie.title}}
    </li>
</ul>

<button (click)="setColor('blue')">Change Color</button>
```

Inputs

- **Two way data binding and change notification**

```
<div *ngFor="#movie of movies">
    <input type="text" [(ngModel)]="movie.title">
    <button (click)="saveMovie(movie)">Save</button>
</div>
```

Forms

- **Support for validation and dirty flags**
- **Use ngModel in combination with:**
  - ngSubmit
  - ngControl
  - ngForm

```
<form (ngSubmit)="saveEdits()">

    ...

    <button type="submit">Save</button>
</form>
```

Angular versus Unobtrusive JavaScript

- Everyone is using JavaScript
- Angular behaves the same across browsers
- Expressions not evaluated in global scope

```
<form (ngSubmit)="saveEdits()">

    ...

    <button type="submit">Save</button>
</form>
```

Lists and Tables

- **Use ngFor**
  - Semantics similar to the new for-of loop in ES2015
  - Works with any iterable
  - Updates the screen if collection changes

```
<table>
    <tr>
        <th>Title</th>
        <th>Rating</th>
    </tr>
    <tr *ngFor="#movie of movies">
        <td>{{movie.title}}</td>
        <td>{{movie.rating}}</td>
    </tr>
</table>
```

Hiding and Showing

- **Several approaches**
  - Use *ngIf
  - Bind to the hidden property
  - Bind to the style.display property

```
<div *ngIf="showDiv">Show div</div>
<div [hidden]="!showDiv">Show div</div>
<div [style.display]="showDiv? '' : 'none'">Show div</div>
<button (click)="showDiv = !showDiv">Toggle</button>
```

Styles

- **Use the ngClass directive**
  - Adds class names for truthy values

```
<tr *ngFor="#movie of movies" [ngClass]="{ good:movie.isGood, bad:movie.isBad }">
    <td>{{movie.title}}</td>
    <td>{{movie.rating}}</td>
    <td>
```

TypeScript

- **Designed by Microsoft**
  - Anders Heilsberg
- **Open Source**
  - https://github.com/Microsoft/TypeScript
- **Superset of JavaScript**
  - Adds optional types and interfaces

Type Annotations

- **Declare the intended type of a variable**
  - Default is “any”
  - boolean, number, string, Array, enum, void

```
Type 'number' is not assignable to type 'string'.
let name: string
name = 123;
```

Types and Functions

- **Parameters can be typed**
- **Return value can also be typed**
  - Often can be inferred

```
function doWork(name: string) {

    let inner = (p: number) => {
        console.log(p);
    }

    inner(42);
    return name;
}

let result = doWork("Scott");
```

Interfaces

- **Focus on the shape**
  - Allows for duck typing
  - Can use optional properties
  - Can also describe functions

```
interface MovieData {
    title: string
    length?: number
}

function show(movie: MovieData) {
    // ....
}

show({ title: "Star Wars" });
```

Public and Private

- **Public is the default**
  - Compiler enforces private keyword

```
class Animal {
    constructor(private name: string) { }
    move(meters: number) {
        alert(this.name + " moved " + meters + "m.");
    }
}
```

Functions

- **Can have return types, optional and default parameters**

```
function getAdder() : (x:number, y?:number) => number {
    return (x:number, y = 3) => x + y;
}


let result = getAdder()(3,4);
```

Generics

- **Use generic types to parameterize a function or class**
  - Generic constraints can make type programmable

```
interface Ratable {
    rating: number
}

function recommend<T extends Ratable>(items: T[]) {
    for(let item of items) {
        if(item.rating > 4) {
            // ....
        }
    }
}
```

Decorators

- **@ symbol followed by a function**
- **Function can modify**
  - A class
  - A property
  - A method
  - A parameter

```
@readonly
class Person {
    constructor(name, admin) {
```

```
function readonly(Target) {
    let newConstructor = function () {
        Target.apply(this);
        Object.freeze(this);
    };

    newConstructor.prototype = Object.create(Target.prototype);
    newConstructor.prototype.constructor = Target;

    return newConstructor;
}
```

Declaration Files

- .d.ts files provide type metadata for 3rd parties

```
animate.js
animate.d.ts
bootstrap.js
bootstrap.d.ts
bootstrap_static.js
bootstrap_static.d.ts
common.js
common.d.ts
compiler.js
compiler.d.ts
core.js
core.d.ts
http.js
http.d.ts
```

Debugging

- **Source maps let you step through TypeScript**

system.src.js | angular2.dev.js | main.ts | movie.ts!transpiled | movie.ts ×

```
1  export class Movie {
2      constructor(title: string, length: number, year: number, rating: number) {
3          this.title = title;
4          this.length = length;
5          this.year = year;
6          this.rating = rating;
7      }
8
9      get isGood() {
10         return this.rating > 4;
11     }
12
13     get isBad() {
```

Summary

- **Angular2 is an application framework**
  - Similar to an MVVM design
- **Extensible, modular, testable**
- **TypeScript adds optional type annotations**
  - Types are structural
  - Type annotations useful for tooling and compile time checks