AngularJS tutorial

Lexicon

To bootstrap:

To start up or initialize a system, program, or application.

Bootstrapping refers to starting up a web application/framework by loading code and resources for execution. This involves initializing, setting up and configuring the main module of the app, dependencies and settings.

In Angular, bootstrapping means, loading Angular framework and linking the app to the main module.

**Comes from the phrase "pull oneself up by one's bootstraps," which means to start from scratch and work your way up to a higher level.*

AngularJS vs Angular

	AngularJS	Angular
Info	Also known as Angular 1.	Also known as Angular 2.
Architecture	**MVC** based	**Component**-based
Language	**JavaScript ES5**	**TypeScript**
Dep. Injections	Own.	Provided by TypeScript.
Templating	HTML with custom directives and filters.	HTML with TypeScript code.
Tooling	–	**Webpack** and Angular CLI

Directives

Extends HTML attribute with **'ng-directives'**.
Angular directives are application variables.

Here is the list of the basic directives used in Angular.

BASIC DIRECTIVES		
DIRECTIVE	**DESCRIPTION**	**NOTE**
ng-**app**	Defines an AngularJS application.	Used to define the root element of an AngularJS application **or** the name of your controller.

ng-**model**	Binds the value of HTML controls (input, select, textarea) to application data.	
ng-**bind**	Binds application data to the HTML view.	

```
<div ng-app="">
    <p>Name: <input type="text" ng-model="name"></p>
    <p ng-bind="name"></p>
</div>
```

ng-**init**	Initialize variables, objects, arrays, ...	

```
<div ng-app="" ng-init="
                    quantity=1;cost=5;
                    firstName='John';lastName='Doe';
                    person={firstName:'John',lastName:'Doe'};
                    points=[1,15,19,2,40];
                "></div>
```

ng-**cotent**	Allows to pass content from a parent to a children component.	

```
@Component({
    selector: 'my-component',
    template: `
<div class="wrapper">
    <h2>My Component</h2>
    <ng-content></ng-content>
</div>
`,
})
export class MyComponent { }

@Component({
    selector: 'app-root',
    template: `
      <my-component>
        <p>Displayed content !</p>
      </my-component>
    `,
})
export class AppComponent { }
```

ng-**submit**	Execute a function on form submit.	ngSubmit in Angular

PREFIXES		
PREFIXES	**DESCRIPTION**	**NOTE**
ng	Default Angular directive prefix.	Used to identify AngularJS specific directives.

data-ng	The "data-" prefix in directives is used to comply with the HTML5 specification.	Also makes the directive unique in case another library would use the same module prefixes.
x-ng	Prefix for XHTML.	
xml-ng	For xml applications with custom attributes.	

```
<div data-ng-app="" data-ng-init="firstName='John'">
    <p>The name is <span data-ng-bind="firstName"></span></p>
</div>
```

Structural directives

Structural directives are a type of directive in Angular that can add, remove, or manipulate elements in the DOM based on certain **conditions**. They change the structure of the DOM elements.

STRUCTURAL DIRECTIVES

*ng**If**

Conditionally render elements in the DOM.

```
<div *ngIf="isLoggedIn">
    <p>logged</p>
</div>

<div *ngIf="!isLoggedIn">
    <p>not logged</p>A
</div>
```

*ng**If** with else and then

If condition is false.

```
<div *ngIf="isValid; then themTemplate else elseTemplate">
    Here is never showing
</div>
<ng-template #thenTemplate>
    If isValid is true
</ng-template>
<ng-template #elseTemplate>
    If isValid is false
</ng-template>
```

*ng**Class**

Add or remove CSS classes.

```
<div [ngClass]="'my-class'">
</div>
<!-- or -->
<div [ngClass]="['my-class']">
</div>
<!-- or -->
<div [ngClass]="{ 'my-class': true }">
</div>
```

* ng**For** (not type safe)

Iterate trough items.

```
<body ng-app="myApp" ng-controller="myCtrl">
    <ul>
        <li ng-repeat="item in items">{{ item }}</li>
    </ul>
    <script>
        angular.module('myApp', [])
            .controller('myCtrl', function ($scope) {
                $scope.items = ['apple', 'banana', 'orange'];
            });
    </script>
</body>
```

** Shorthand for ng-repeat and needs Angular import not AngularJS.*

*ng**Style**

Set CSS styles.

```
<div [ngStyle]="{'font-size': size + 'px'}">
</div>
<!-- or -->
<div [ngStyle]="{'color': textColor}">
</div>
```

*ng**Switch**

Based on specified value.

```
<div [ngSwitch]="fruit">
    <p *ngSwitchCase="'apple'">apple!</p>
    <p *ngSwitchCase="'banana'">banana!</p>
    <p *ngSwitchCase="'orange'">orange!</p>
    <p *ngSwitchDefault>select a fruit.</p>
</div>
```

*ng**Container**

A placeholder element to group elements.

```
<ng-container *ngIf="condition">
    <p>content</p>
</ng-container>
```

*ng**TemplateOutlet**

Render template based on context.

```
<body ng-controller="myController">
    <div ng-if="showTemplate">
        <ng-include src="'my-template.html'"></ng-include>
    </div>

    <script type="text/ng-template" id="my-template.html">
        <p>{{ content }}</p>
    </script>

    <script type="text/javascript">
        var app = angular.module('myApp', []);
        app.controller('myController', function ($scope) {
            $scope.content = 'This is the content of my template.';
        });
    </script>
</body>
```

Expressions

Binds data to HTML.

```
<div ng-app="">
    <p>My first expression: {{ 5 + 5 }}</p>
</div>
```

Expressions are written inside double braces **'{{ expression }}'**.

```
<div ng-app="">
    <p>Name: <input type="text" ng-model="name"></p>
    <p>{{ name }}</p>
</div>
```

AngularJS will "output" data exactly where the expression is written.

AngularJS expressions can also be written inside a directive: **ng-bind="expression"**.

```
<div ng-app="">
    <p>Name: <input type="text" ng-model="name"></p>
    <p ng-bind="name"></p>
</div>
```

Applications

The ng-app directive is used to bootstrap an Angular application. It tells AngularJS to start up and initialize the application by finding the root element of the application in the HTML document and linking it to the main module of the application.

Modules

AngularJS modules **define** AngularJS applications.
The module is a container for the different parts of an application and controllers.

Creating a module

```
<div ng-app="myApp">...</div>
<script>
    var app = angular.module("myApp", []);
</script>
```

The second parameter is an array that contains or not any dependencies on other modules.

Controllers

AngularJS controllers **control** AngularJS applications.
Controllers always belong to a module.

Creating a controller

```
<div ng-app="myApp" ng-controller="myCtrl">
    {{ firstName + " " + lastName }}
</div>
<script>
    var app = angular.module("myApp", []);
    app.controller("myCtrl", function ($scope) {
        $scope.firstName = "John";
        $scope.lastName = "Doe";
    });
</script>
```

Directives

Creating a directive

The function defines the behavior of the directive.

The template keyword inside the return statement.

```
<div ng-app="myApp" w3-test-directive></div>
<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
    return {
        template : "I was made in a directive constructor!"
    };
});
</script>
```

Return statement properties

- A directive is defined using a JavaScript object that specifies its behavior. e.g.: One of the properties that can be defined in this object is the template property.

PROPERTY	DESCRIPTION	NOTE
template	Used to specify the HTML markup that will replace the directive element in the DOM when the directive is used in the HTML.	
scope	Defines the directive's scope.	Part of AngularJS.

```
<body ng-app="myApp">
    <div my-directive name="Jhon1"></div>
    <my-directive name="John2"></my-directive>
    <script>
        angular.module('myApp', [])
            .directive('myDirective', function () {
                return {
                    template: '<div>Hello, {{name}}!</div>',
                    scope: {
                        name: '@'
                    }
                };
            });
    </script>
</body>
```

animations	An array of animations that can be used to animate the component's view.	

inputs	An array of strings that define the names of the component's input properties.	
outputs	An array of strings that define the names of the component's output.	
templateUrl	A URL or inline template that defines the component's view.	

Model

Validate user input

```
<form ng-app="" name="myForm">
    Email:
    <input type="email" name="myAddress" ng-model="text">
    <span ng-show="myForm.myAddress.$error.email">Not a valid e-mail address</span>
</form>
```

CSS classes

```
<style>
    input.ng-invalid {
        background - color: lightblue;
    }
</style>
<body>
    <form ng-app="" name="myForm">
        Enter your name:
        <input name="myName" ng-model="myText" required>
    </form>
</body>
```

ng-**empty**	ng-**not-empty**	ng-**dirty**
ng-**touched**	ng-**untouched**	ng-**pending**
ng-**valid**	ng-**invalid**	ng-**pristine**

Scopes

If we consider an AngularJS application to consist of:

- View, which is the HTML.
- Model, which is the data available for the current view.
- Controller, which is the JavaScript function that makes/changes/removes/controls the data.

Then the scope is the Model.

The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.

```
<div ng-app="myApp" ng-controller="myCtrl">
    <input ng-model="name">
    <h1>My name is {{name}}</h1>
</div>

<script>
    var app = angular.module('myApp', []);

    app.controller('myCtrl', function($scope) {
    $scope.name = "John Doe";
});
</script>
```

Filters

Filters allow you to transform data.

```
<div ng-app="myApp" ng-controller="myCtrl">
    <input ng-model="name">
    <h1>My name is {{name}}</h1>
</div>
<script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function ($scope) {
        $scope.name = "John Doe";
    });
</script>
```

FILTER	DESCRIPTION
currency	Format a number to a currency format.
date	Format a date to a specified format.
filter	Select a subset of items from an array.
Json	Format an object to a JSON string.
limitTo	Limits an array/string, into a specified number of elements/characters.
lowercase	Format a string to lower case.
Number	Format a number to a string.
orderBy	Orders an array by an expression.
uppercase	Format a string to upper case.

Custom filters

```
<ul ng-app="myApp" ng-controller="namesCtrl">
    <li ng-repeat="x in names">
        {{x | myFormat}}
    </li>
</ul>

<script>
    var app = angular.module('myApp', []);
    app.filter('myFormat', function () {
        return function (x) {
            var i, c, txt = "";
            for (i = 0; i < x.length; i++) {
                c = x[i];
                if (i % 2 == 0) {
                    c = c.toUpperCase();
                }
                txt += c;
            }
            return txt;
        };
    });
    app.controller('namesCtrl', function ($scope) {
        $scope.names = [
            'Jani',
            'Carl',
            'Margareth',
            'Hege',
            'Joe',
            'Gustav',
            'Birgit',
            'Mary',
            'Kai'
        ];
    });
</script>
```

Services

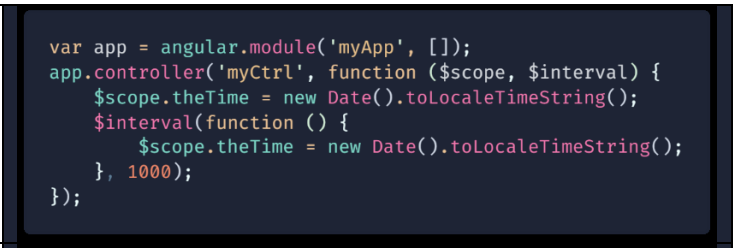
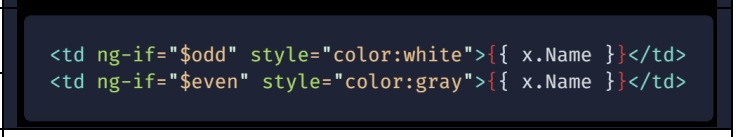
In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.

AngularJS has about 30 built-in services.

NAME	DESCRIPTION	NOTE

$**http**	Making http requests.	`var app = angular.module('myApp', []);`
		`app.controller('myCtrl', function ($scope, $http) {`
		`  $http.get("welcome.htm").then(function (response) {`
		`  $scope.myWelcome = response.data;`
		`  });`
		`});`

$**timeout**	Scheduling function calls to occur in the future.	`var app = angular.module('myApp', []);`
		`app.controller('myCtrl', function ($scope, $timeout) {`
		`  $scope.myHeader = "Hello World!";`
		`  $timeout(function () {`
		`  $scope.myHeader = "How are you today?";`
		`  }, 2000);`
		`});`

$**interval**	Scheduling functions calls to occur repeatedly.	
$**odd**	If odd.	
$**even**	If even.	
$**q**	Working with promises.	* About promises, look at my JavaScript tutorial.
$**location**	Working with the URL.	
$**rootScope**	The root scope of the application.	
$**scope**	The scope of a controller.	
$**compile**	Compiling and linking html templates.	
$**templateCache**	A cache for storing templates.	
$**routeParams**	Accessing parameters in the URL.	
$**filter**	Filtering and formatting data.	
$**log**	Logging messages.	
$**translate**	Internationalization and localization.	
$**injector**	For dependency injection.	* About dependency injection, look at my D.I. tutorial.
$**controller**	Instantiating controllers.	

Create your own service

```
angular.module('myApp').service('rectangleAreaService', function () {
    this.calculateArea = function (width, height) {
        return width * height;
    }
});

angular.module('myApp').controller('MyController', function ($scope, rectangleAreaService) {
    $scope.width = 5;
    $scope.height = 10;
    $scope.area = rectangleAreaService.calculateArea($scope.width, $scope.height);
});
```

$HTTP

The response from the server is an object with these properties:

NAME	DESCRIPTION
.config	The object used to generate the request.
.data	A string, or an object, carrying the response from the server.
.headers	A function to use to get header information.
.status	A number defining the HTTP status.
.statusText	A string defining the HTTP status.

** The data you get from the response is expected to be in JSON format.*

Events

You can add AngularJS event listeners to your HTML elements by using one or more of these directives:

DIRECTIVE NAMES
ng-blur
ng-change
ng-click
ng-copy
ng-cut
ng-dblclick
ng-focus
ng-keydown
ng-keypress
ng-keyup
ng-mousedown
ng-mouseenter
ng-paste

Forms & inputs states

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

Input states

NAME	DESCRIPTION
$untouched	The field has not been touched yet.
$touched	The field has been touched.
$pristine	The field has not been modified yet.
$dirty	The field has been modified.
$invalid	The field content is not valid.
$valid	The field content is valid.

```
<form name="myForm">
    <input name="myInput" ng-model="myInput" required>
</form>

<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
```

Form states

NAME	DESCRIPTION
$submitted	The form is submitted.
$pristine	No fields have been modified yet.
$dirty	One or more have been modified.
$invalid	The form content is not valid.
$valid	The form content is valid.

AngularJS adds CSS classes to forms and input fields depending on their states:

```
<style>
    input.ng-invalid {
        background-color: pink;
    }

    input.ng-valid {
        background-color: lightgreen;
    }
</style>
```

Includes

Allows you to include html content with the **ng-include** directive.

```
<body ng-app="">
    <div ng-include="'myFile.htm'"></div>
</body>
```

Animations

You need to add the angular-animate module.

```
<body ng-app="myApp">
    <h1>Hide the DIV: <input type="checkbox" ng-model="myCheck"></h1>
    <div ng-hide="myCheck"></div>

    <script>
        var app = angular.module('myApp', ['ngAnimate']);
    </script>
</body>
```

Routing

The **ngRoute** module helps your application to become a Single Page Application.

You will need the angular route module.

```
<p><a href="#/!">Main</a></p>
<a href="#!red">Red</a>
<a href="#!green">Green</a>
<a href="#!blue">Blue</a>
<div ng-view></div>

<script>
    var app = angular.module("myApp", ["ngRoute"]);
    app.config(function ($routeProvider) {
        $routeProvider
            .when("/", {
                templateUrl: "main.htm"
            })
            .when("/red", {
                templateUrl: "red.htm"
            })
            .when("/green", {
                templateUrl: "green.htm"
            })
            .when("/blue", {
                templateUrl: "blue.htm"
            });
    });
</script>
```

Apps can only have one **ng-view** directive, and this will be the placeholder for all views provided by the route.

With the **$routeProvider** you can define what page to display when a user clicks a link. Also, it allows you to define a controller or/and a template for each view.

```
.when("/paris", {
  templateUrl: "paris.htm",
  controller: "parisCtrl"
});
```

```
.when("/", {
  template: "<h1>Main</h1><p>Click on the links to change this content</p>"
})
```

** routeProvider also has a .otherwise method.*

Made by :

Matthias Brat

Socials:

github.com/matthiasbrat

stackoverflow.com/users/17921879/matthias-b

matthiasbrat.dev