Week 3

coursera.org/learn/single-page-web-apps-with-angularjs/discussions/weeks/3/threads/iUgcgQzzEeeoaxKMCL9POg

AngularJS - Directives

1) The Angular's special module method **.directive**('myCustom', MyCustomDirective) defines extension for HTML vocabulary (dynamic view) for your web application by placing a non-standard HTML tag on a DOM element (a special marker placed in your regular HTML code) that tells Angular's compiler to attach a specific behavior.

- myCustom: the normalized name to be used in the HTML;
- MyCustomDirective: the factory function returning a Directive Definition Object (DDO).

When you place the tag notation **custom>** in your HTML page, Angular's HTML compiler replaces it with the DDO "template" property's content defined in the factory function definition:

```
(function() {

  'use strict';

  angular.module("MyApp", [])

    .directive("myCustom", MyCustomDirective);

  function MyCustomDirective() {

    var ddo = {

      template: '<p>My <b>template</b> content.</p>'

    };

    return ddo;

  }

})();
```

So, the element "myCustom" defined in the factory function "MyCustomDirective()" becomes the special denormalized "<my-custom></my-custom>" HTML tag notation from which Angular is then able to match to the name of its directive by replacing each dash (-) with the capitalized following letter (every first letter that follows each dash), resulting to a normalized "**camelCase**" notation ("my-custom" becoming "myCustom").

```
<my-custom></my-custom>
```

Following the example in the first code snippet, <my-custom></my-custom> is compiled by Angular and replaced with " My template content. " in the html page, which in turn renders to "My **template** content." in your browser.

Using custom directives is possible for both, to reuse a HTML code and to nest templates, placing one inside another.

2) The "**restrict**" property on the DDO tells Angular compiler how to interpret your custom directive, whether as an attribute or an element. When not specified, the directive treats as "either one or another" (**restricted: 'AE'**). Restrict to an attribute (**restrict: 'A'**) when directive has no content and only extending behavior such as *ng-repeat*, or to an element (**restrict: 'E'**) when directive has content with possible behavior like a component with associated template such as .

3) Another property on the DDO called "**scope**" is an object used to signal an isolate scope, meaning the prototypal parent scope is not inherited. The data is now passed into our now more independent directive by explicitly biding predefined attributes to the isolate object scope's properties.

Inside the object, local scope properties (written in **camelCase** notation) are placed on the left while HTML template attribute names go on the right, both separated by a colons. The HTML template attribute names, surrounded by quotes, can be preceded with '**=**' indicating bidirectional (2-way) binding (change values go on both directions, which is not best practice since Angular sets extra watchers meaning wasting resources), '**@**' indicating a string value unidirectionally (1-way) binding an external value to the isolate scope property (which is on left side of colons), and finally '**<**' indicating one-way binding where only the parent's property is watched (not the directive's). '**=?**' / '**@?**' / '**<?**' indicates the property is optional (value might not be supplied).

For example, in **myProp: '@myTitle**', myProp becomes my-prop in the directive's template and myTitle becomes my-title the DOM's external custom tag's property. EXAMPLE 2: The same happens to **myItems: '<'** (not specifying a value name - e.g. "<items" - will automatically name it as "<myItems" as well):

Parent's Template:

```
...

<body>

  ...

  <my-custom my-title = "{{ctrl.title}}" my-items = "ctrl.items"></my

      -custom>

  ...

</body>

...
```

Updating the MyCustomDirective's function: (EXAMPLE 1)

```
...
function MyCustomDirective() {
  var ddo = {
    templateUrl: 'template.html',
    scope: {
      myProp1: '@myTitle',
      myProp2: '<myItems'
    }
  };
}
...
```

And an example for the Directive's Template: (EXAMPLE 1)

```
<h3> {{ ctrl.my-prop1 }} </h3>
<ol>
  <li ng-repeat="item in ctrl.my-prop2">
    {{ item.name }}
  </li>
</ol>
```

Updating the MyCustomDirective's function: (EXAMPLE 2)

```
...
function MyCustomDirective() {
  var ddo = {
    templateUrl: 'template.html',
    scope: {
      myTitle: '@',
      myItems: '<'
    }
  };
}
...
```

And an example for the Directive's Template: (EXAMPLE 2)

```
<h3> {{ ctrl.my-title }} </h3>

<ol>

  <li ng-repeat="item in ctrl.my-items">

    {{ item.name }}

  </li>

</ol>
```

4) The "**controller**", "**controllerAs**" and "**bindToController**" properties in the DDO adds functionality to the directive, defining that the directive's template should be able to refer to the "**controller: myDirectiveController**" instance through the label defined by "**controllerAs: ctrl**" where the properties in scope are bind by the "**bindToController: true**".

```
...

function MyCustomDirective() {

  var ddo = {

    templateUrl: 'template.html',

    scope: {

      myProp1: '@myTitle',

      myProp2: '<myItems'

    },

    controller: myDirectiveController,

    controllerAs: 'ctrl',

    bindToController: true

  };

}

...
```