Week 4

coursera.org/learn/single-page-web-apps-with-angularjs/discussions/weeks/4/threads/HAdNNhM-Eee65wqYgwk_QA

** This is a review from **Lecture 35, Part 1** and **Part 2**.*

AngularJS – **Modules**

MODULE DECLARATIONS

1. CREATING Module: A module is created by specifying a second argument.

a) **angular.module('Spinner', []);** creates a module with no dependencies (no injections)

```
(function () {

'use strict';

angular.module('Spinner', []);

})();
```

b) **angular.module('Main', ["Spinner"]);** creates a module with a dependency – 'Main' module depends on 'Spinner' module; "Spinner" is being injected into the creating 'Main' module ('Spinner' module must be previously declared and linked).

```
(function () {

'use strict';

angular.module('Main', ["Spinner"]);

})();
```

c) angular.module('R2D2', ["Main", "StartWars"]) creates a module with dependencies – two modules are being injected into the 'R2D2' module being created ('Main' and 'StarWars' module declarations must exist and be linked).

```
(function () {

'use strict';

angular.module('R2D2', ["Main", "StarWars"]);

})();
```

2. RETRIEVING Module: A module is retrieved when no second argument is asked.

angular.module('Main') does not create a module but retrieves it when **omitting a second argument** – 'Main' is then being just referred to, which means it should be followed by an artifact declaration, for example:

```
(function () {
'use strict';
angular.module('Main')
  .component('myComponent', {
    templateUrl: 'my-component-template.html',
    controller: MyComponentController,
    bindings: { ... }
  });
function MyComponentController() { ... }
})();
```

WAYS OF WIRING YOUR ANGULAR APPLICATION TO A HTML PAGE

```
<!DOCTYPE html>
<html ng-app='Main'>
  <head> ... </head>
  <body>
    ...
    <script src="angular.min.js"></script>
    <script src="app.js"></script>
  </body>
</html>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body ng-app='Main'>
    ...
    <script src="angular.min.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

```
<!DOCTYPE html>

<html>

  <head> ... </head>

  <body>

    <div ng-app='Main'>

      ...

    </div>

    ...

    <script src="angular.min.js"></script>

    <script src="app.js"></script>

  </body>

</html>
```

HOW TO LINK MODULES AND ARTIFACTS SPLIT INTO SEPARATE JAVASCRIPT FILES

When split into separate files, **artifacts** of each module, must always be linked **after** the module to which they were declared. Among **modules** themselves, the order does not matter – AngularJS will figure it out by mapping their dependencies altogether.

Obviously, **libraries** must still come before **everything** else which used to be the **app.js** but now split into separate files.

```
<script src="lib/jquery.min.js"></script>

<script src="lib/angular.min.js"></script>

<script src="src/main/main-module.js"></script>

<script src="src/main/main-component.js"></script>

<script src="src/spinner/spinner-module.js"></script>

<script src="src/spinner/spinner-component.js"></script>

<script src="src/spinner/spinner-comp-controller.js"></script>
```

Here, spinner-module.js is linked **after** main-module.js (it can be), but because of the dependency injections AngularJS knows which **order** to load them all.

THE SPECIAL MODULE METHODS

1) The **.config(function () { ... })** method runs **before any other method** on the module.

    Only constants and providers can be injected into the .config method.

2) The **.run(function () { ... })** method is executed **right after the .config** method.

Only constants and instances can be injected into the .run method (**not providers**).

3) The .**config** method of every module run **before** the **.run** method of every module.

```
(function () {

'use strict';

angular.module('Main')

  .config(function () {

    console.log("Main config fired.");

  }).run(function () {

    console.log("Main run fired.");

  });

})();

(function () {

'use strict';

angular.module('Spinner')

  .config(function () {

    console.log("Spinner config fired.");

  }).run(function () {

    console.log("Spinner run fired.");

  });

})();
```

Considering the dependency injections previously declared, console will print:

```
Spinner config fired.

Main config fired.

Spinner run fired.

Main run fired.
```

The .config method of each module will execute in order of (according to) each module's dependencies – modules with no dependencies first, such as .module('Spinner', **[]**).**config()**, then following a dependency hierarchy: .module('Main', ["**Spinner**"]).**config()**, then .module('StarWars', ['**Main**']).**config()**, then .module('R2D2', ["**StarWars**"]).**config()** and so on.