Week 1

coursera.org/learn/single-page-web-apps-with-angularjs/discussions/weeks/1/threads/jZ6PtQmXEee36g5uvZCkbA

AngularJS - Week #1

1) Use **CTRL+** to toggle (open/close) the File Browser in Atom (or **CTRL+B** to toggle the Side Bar in Visual Studio Code);

2) You **may** but **should NOT** name variables starting with **$** since AngularJS uses it by convention to name its native services(**$scope**, **$filter**, **$injector**, **$timeout**, etc.);

3) **AngularJS 1.5.8** (NOT 1.5.7!): You will see ahead in "*Lecture 33, part 4: Components & Component-based Architecture*" that we use the new life-cycle method called $doCheck(), introduced in Angular version **1.5.8**. So, you can avoid future compatibility issues by downloading **angular.min.js** (or angular.js) straight from https://code.angularjs.org/1.5.8/. (2.x are for Angular 2)

4) Add ***'use strict';*** at the top of (and inside) your IIFE function (**.js** files) to avoid accidentally declaring variables in the global scope. By doing so, you will be forced to use **var** when declaring a variable name (**var x = 1;** instead of **x = 1;**) otherwise you will get an error message - NOT adding *'use strict';* would allow you to declare variables without *var* but then those would be set in the global scope instead of your local scope.

```
(function({

  'use strict';

  ...

})();
```

5) **Code Indentation**, **High Cohesion** and **Low Cohesion** (Lecture 2, Part 1): Use spacing (or tabulation) for best writing (indent) your code, making it more readable and understandable (not only for yourself but for others whom you might ask to help you out finding bugs or typos in your code somewhere ahead during this course); High Cohesion: how well does one thing stick to doing just one thing; Loose Coupling: independence between components: changing something in one component doesn't affect another component (least dependency of one component on another).

6) **MVVM**: In the design pattern MVVM (Model-View-ViewModel), the *angular.module("myApp").**controller***'s function is responsible for implementing the presentation logic in **ViewModel** (the state of the view), which helps processing and providing the right response (hold data) from the **Model** (raw data) to the **View** (html/css) according to user's request (entry/selection/interaction). This is possible thanks to the **Declarative Binder**, which binds the *Model* (business logic)to the *View* (*{{ }}*, *ng-model*) but doing so through the

ViewModel (**$scope**). In other words, the Declarative Binder binds the ViewModel's logic data (which in turn connects to business Model's raw data) to the View. More info about Controllers, **MVVM vs MVC**, how to ask questions in the forums as well as more details on the content of weeks 2, 3 and so on (Service, Provider, Factory, etc.): https://github.com/jhu-ep-coursera/fullstack-course5/blob/master/FAQ.md

7) **Where does AngularJS start (ng-app) in the HTML**? You AngularJS App starts depending on where you place the **ng-app** directive in your html template (index.html). It can be inside the **htmltag** (), inside the **body tag** (), inside a **div tag** (

), generally inside some container. But keep in mind one thing: wherever you choose to place your **ng-app** directive/property, remember that your **ng-controller** has to be below and nested to it.

```
<body ng-app="MyApp">

  <div ng-controller="MainController as ctrl">

  </div>

</body>
```

8) **Data Binding**: Every communication between the template (HTML) and Angular (property bindings: "**ng-model"**, "**ng-click"**, or string interpolation: "**{{ }}**") must be nested between tags with the **ng-controller** directive in order for AngularJS to be able to recognize those while sharing the same scope. In the example bellow, *ng-controller* was placed in a **div tag** and so all Angular "data bindings" were placed (nested) between the opening and closing div tags: as property binding in an input tag (***ng-model***), as string interpolation (***{{}}***), and in a button's property binding (***ng-click***).

```
  <div ng-controller="MainController as ctrl">

    <input type="text" ng-model="ctrl.name">

    &lt; You entered: {{ ctrl.name }}

    <br>

    <button ng-click="ctrl.method1()">Go</button>

  </div>
```

Notice they all refer to the **controller** (MainController) through its alias "**ctrl.**" (which could have been any word of your choice - other than "ctrl"), and that is the connection between the **View** (the html template inside ng-controller) and the **ViewModel** (the controller function that is referred by ng-controller).

9) **$scope & controller function**: That connection between the View and the ViewModel is made through an Angular object called "scope service" (**$scope**). Every data sent or received through data binding is set as a property to that object instantiated by Angular. To be able to process its data from the ViewModel, we must pass **$scope** into the **controller function**.

```
(function({

  'use strict';

  angular.module("MyApp", [])

    .controller("MainController", MainController);

  function MainController($scope) {

    $scope.method1 = function () {

      $scope.message = "Welcome, " + $scope.name + "!";

    }

  }

})();
```

At same time, we can also place new properties in this $scope object from inside the ViewModel (the controller function), and they will automatically be exposed to the View (anywhere in the template inside ng-controller). All we have to do is to reference them in the html, like adding the string interpolation **{{ message }}**, but nothing is shown until the button is clicked ($scope.**message** only exists after $scope.**method1()** runs).

10) **Dependency Injection (DI) & Minification**: DI is a design pattern that implements inversion of control (IoC) for resolving dependencies (more on Lectures 8 and 9). DI needs to be minification proof through inline array with function or by attaching **$inject** property to the function object.

Minification (Lecture 10) is the process of reducing code size (which involves removing empty spaces and extra characters, and replacing variable names) for saving bandwidth when deploying it to the server and for faster page loading when opening it in the browser. The code becomes unreadable but its functionality remains unchanged. Variable names can be protected from minification (or "uglification") by injecting those into the function right before its declaration: