Submission Instructions:

Submit all of your answers in the spaces provided within this document. Submit your completed document to the dropbox for this inclass document.

**Weight:**

10% of your final mark

**Due:**

March 13, 6PM (6:01PM is late)

20% will be deducted each day this is late.

Contents

[\*\* Tip \*\* 2](#_Toc413424060)

[Starting AngularJS 2](#_Toc413424061)

[Advantages of the AngularJS Framework 2](#_Toc413424062)

[Terminology Introduction 2](#_Toc413424063)

[Templates 3](#_Toc413424064)

[Directives 3](#_Toc413424065)

[Models 3](#_Toc413424066)

[Scope 3](#_Toc413424067)

[Expressions 3](#_Toc413424068)

[View 3](#_Toc413424069)

[Router 3](#_Toc413424070)

[Filter 3](#_Toc413424071)

[Data Binding 4](#_Toc413424072)

[Controller 4](#_Toc413424073)

[Dependency Injection 4](#_Toc413424074)

[Module 4](#_Toc413424075)

[Services 4](#_Toc413424076)

[Directives 4](#_Toc413424077)

[Sorting 9](#_Toc413424078)

[Search Filters 11](#_Toc413424079)

[Modules, Routing and Views 14](#_Toc413424080)

[Services 17](#_Toc413424081)

[Route Parameters 18](#_Toc413424082)

[Managing Inputs 19](#_Toc413424083)

[Text Boxes 19](#_Toc413424084)

[Buttons 21](#_Toc413424085)

[Check Boxes 22](#_Toc413424086)

[Radio Option List 24](#_Toc413424087)

[Select Controls 25](#_Toc413424088)

[Hiding, Showing, Disabling, and Enabling Inputs 27](#_Toc413424089)

[Table 2: Hiding, Showing, Disabling, and Enabling Inputs 27](#_Toc413424090)

[Debugging 28](#_Toc413424091)

[Terminology Review 28](#_Toc413424092)

# \*\* Tip \*\*

If data does not appear to work properly, try running the example in Firefox. I personally like to run the from a web server so I can use the Google Chrome debugging tools - however Firefox offers a competitive set of debugging tools for JavaScript.

# Starting AngularJS

This section introduces essential structures of AngularJS to help you quickly ramp up with this technology for professional, data-driven development.

## Advantages of the AngularJS Framework

* Excellent separation of presentation and logic for data driven applications.
* Excellent support for one page application development.
* Convenient two way data binding and validation.
* Developed and supported by Google.
* Excellent browser compatibility.

## Terminology Introduction

To help start this discussion, here are some general definitions:

### Templates

* Contain HTML elements plus directives and expressions that create the view.

### Directives

* Invoke custom behavior for HTML elements.
* Are pre-defined in the AngularJS framework but they can also be developed from scratch.
* The following sample shows how a directive may appear in a div tag;

<div ngDirective="**{{**expression**}}**"></div>

### Models

* Define and store data for the view. Here is an example that shows the creation of a card model with that has number and suit attributes:

$scope.card = { "number": "5", "suit": "hearts" };

Here is a declaration for the color model:

$scope.color = { "blue" };

### Scope

* Is a service that binds the model data for controllers and templates. Any updates to the model in the view are applied to the same model in the controller through the scope and vice versa.

### Expressions

* Are JavaScript-like snippets that are usually placed in double curly braces within a template.
* May be used to read and write model attributes from a model to a template; {{card.suit}}
* May be used to perform calculations; 1+2={{1+2}}
* May be used to evaluate Boolean conditions like in the ng-disabled directive below;

Click me to toggle: <input type="checkbox" ng-model="checked"><br/>

<button ng-model="button" ng-disabled="checked">Button</button>

### View

* Is the output of a fully processed template in the browser.

### Router

* Receives page requests and selects the appropriate template and controller.

### Filter

* Is a construct that is used to select a subset of items from an array. For example, when a selector with an orderProp model is used for a collection of cards;

<select ng-model="orderProp">

<option value="suit">Suit</option>

<option value="numOrd">Number</option>

</select>

We could include a filter to sort items with the selector results;

<tr ng-repeat="card in cards | orderBy:orderProp ">

### Data Binding

* Synchronization between the model and view is managed by the scope. If the view changes, the model changes immediately and vice versa.

### Controller

* Encapsulates functions and data.
* Sets the view model's initial state.

### Dependency Injection

* Is a development pattern that AngularJS typically uses to pass references for services to controllers and other components. Services are registered in the AngularJS application so they can be injected with this pattern.
* Here are three common syntax implementation examples for injecting services with AngularJS:

**// Inferring Dependencies**

function myController($scope, greeter) { ... }

**// $inject Annotation**

var myController = function($scope, $greeter) { ... }

myController.$inject = ['$scope', 'greeter'];

**// Inline Annotation**

myModule.factory('greeter', ['$window', function(renamed$window)

{ ... }]);

### Module

* Is a reference to your AngularJS application.
* Is a collection of services, directives, filters, and models. An angular application declares only-one main module in the ng-app directive and registers it and any others with angular.module(…)

### Services

* Is a singleton object/function that carries out specific web application task.
* Can be developed from scratch, however, AngularJS provides many pre-defined services like an HTTP service to manage get, post, put, and delete web service operations.

## Directives

AngularJS directives extend HTML attributes to trigger specific behavior. The standard naming convention uses the ‘ng-‘ prefix. However, you can prefix the directives with ‘data-ng-‘ to make it HTML5 compliant.

Table 1: Introductory AngularJS Directives

|  |  |
| --- | --- |
| **ng-app** | Sets the AngularJS application root. This directive may also indicate which module to load. |
| **ng-bind** | Binds the application data to the view. Example:  <p ng-bind="name"></p> |
| **ng-controller** | Indicates which controller gets/sets the data model. |
| **ng-init** | Initializes model data from HTML. Example:  <body ng-app='' ng-init="message='Hello world!'"> |
| **ng-model** | Binds the value of the HTML control to the application data. Example:  <input type="text" ng-model="name"> |
| **ng-repeat** | Iterates through an array of objects to populate an HTML template. For example;  <div ng-repeat="item in items">  <p>**{{** item **}}**</p>  </div> |

Example 1

🞑 This first example demonstrates how to use the *ng-app*, *ng-init*, *ng-model* and *ng-bind* directives. Notice how *ng-init* initializes the *message* model with the value of ‘Hello world’. Any change to the text box where *ng-model* is referenced is bound to the paragraph tags through *ng-bind* and double curly brace references. While it is common to see double curly braces, the ng-bind directive eliminates the unwanted curly braces from appearing while the page is loading data.

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <!-- The ng-app directive -->  <body ng-app='' ng-init="message='Hello world!'">  Message: <input type="text" ng-model="message">  <p ng-bind="message"></p>  <p> **{{**message**}}**</p>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  </body>  </html> |

Exercise 1

🖍 Starting with the code from Example 1, create a second model named *ajsMessage*. Initialize this message with the string “Hello from AngularJS”. Include a second text box which displays this message and allows a person to edit it. Show the *ajsMessage* value below the new text box and display the text within <h1> tags. First show the new message using the ng-bind directive and then show the message with the expression inside double curly braces. Show your revised HTML page here: (1 mark)

|  |
| --- |
| <!doctype html> <html> <head>  <title>Starting Angular</title> </head> *<!-- The ng-app directive -->* <body ng-app='' ng-init="message='Hello world!'; ajsMessage='Hello from AngularJS'">  Message: <input type="text" ng-model="message">  <p ng-bind="message"></p>  <p> {{message}}</p>  AngularJS Message: <input type="text" ng-model="ajsMessage">  <h1 ng-bind="ajsMessage"></h1>  <h1>{{ajsMessage}}</h1>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script> </body> </html> |

Example 2: Using a Module and Controller

🞑 This example introduces basic AngularJS scope, controllers, views, and directives. Inside *controllers.js*, a dependency-injection pattern gives the controller access to the scope service. The controller function defines a model called *greeting* and assigns data to it.

**js/controllers.js**

|  |
| --- |
| // Register our controller with the angular module.  var cardApp = angular.module('cardApp', []);  // The controller is a constructor function that takes a $scope parameter.  // The controller lets us establish data binding between the controller and the view.  // The model is initialized with the $scope parameter.  cardApp.controller('MainController', ['$scope', function ($scope) {  // Scope ensures that any changes to the  // model are reflected in the controller.  // Here we create an initialize a 'greeting' model.  $scope.greeting = "AngularJS Hello World!";  }]); |

Inside the view, the **ng-app** directive sets the AngularJS application root. The **ng-controller** directive tells AngularJS that the div tag will be scoped by the *MainController* controller.

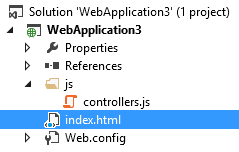
**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <!-- The ng-app directive triggers load and setup of AngularJS  after the DOM is loaded.  It sets the tag as the root of the AngularJS app. -->  <body ng-app="cardApp">  <!-- Identify the controller and model to be used. -->  <div ng-controller="MainController">  <!-- The ng-bind gets data from the model. -->  <h1 ng-bind="greeting"></h1>  <br />  </div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script src="js/controllers.js"></script>  </body>  </html> |

When the program runs the model, *greeting*, stores the data provided by the scope. The directive, **ng-bind**, is used to display value of the *greeting* model.

Exercise 2

🖍 Set up a Visual Studio empty web project with the code from Example 2. Your solution should appear as follows:



Next, add the following instruction to the end of your *MainController* function.

console.log($scope.greeting);

a) Show a screenshot of the console window in the Chrome debugger (Access by pressing ctrl+shift+I or F12) which displays the output from the log instruction. Or, you if you can’t deploy your code on a server just run it in Firefox and from the menu select **Developer | Debugger**. (1 mark)

|  |
| --- |
|  |

b) In the Developer | Debugger | Sources view of the Firefox debugger, open the controller.js file. Set a breakpoint beside your console.log instruction. Refresh the page in the browser and show a screenshot of the application while it is halted at the breakpoint. (1 mark)

|  |
| --- |
|  |

Exercise 3

🖍 Starting with Example 2, add a model named *subTitle* to the scope. Assign the model the value “AngularJS Intro”. Using the model reference, display your subtitle value in an H3 element so your output looks like the following: (1 mark)



Show your revised controllers.js file here:

|  |
| --- |
| // Register our controller with the angular module. **var** cardApp = angular.module('cardApp', []);  // The controller is a constructor function that takes a $scope parameter. // The controller lets us establish data binding between the controller and the view. // The model is initialized with the $scope parameter. cardApp.controller('MainController', ['$scope', **function** ($scope) {  // Scope ensures that any changes to the   // model are reflected in the controller.  // Here we create an initialize a 'greeting' model.  $scope.greeting = "AngularJS Hello World!";  $scope.subTitle = "AngularJS Intro."; }]); |

Show your revised index.html file here:

|  |
| --- |
| <!doctype html> <html> <head>  <title>Starting Angular</title> </head> *<!-- The ng-app directive triggers load and setup of AngularJS  after the DOM is loaded.  It sets the tag as the root of the AngularJS app. -->* <body ng-app="cardApp"> *<!-- Identify the controller and model to be used. -->* <div ng-controller="MainController">  *<!-- The ng-bind gets data from the model. -->* <h1 ng-bind="greeting"></h1>  <br />  <h3>{{subTitle}}</h3>  </div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script src="js/controllers.js"></script> </body> </html> |

Example 3: Adding a Repeater

🞑 This example demonstrates how to implement the **ng-repeater** directive to display attributes of a model that is defined with JSON. Starting with Example 2, replace the code inside *js/controller.js* with the following controller code. This code revision defines a new model called *cards* which stores a collection of JSON objects. Each object in the collection has a *number*, *suit*, and *numOrd* attribute.

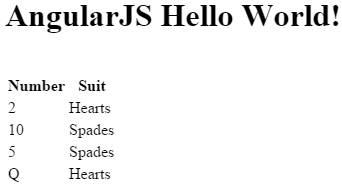
**js/controller.js**

|  |
| --- |
| // Register our controller with the angular module.  var cardApp = angular.module('cardApp', []);  // The controller is a constructor function that takes a $scope parameter.  // The controller lets us establish data binding between the controller and the view.  // The model is initialized with the $scope parameter.  cardApp.controller('MainController', ['$scope', function ($scope) {  // Scope ensures that any changes to the  // model are reflected in the controller.  // Here we create an initialize a 'title' model.  $scope.greeting = "AngularJS Hello World!";  // Define cards model which stores an array of objects.  $scope.cards = [  { "number": "2", "suit": "Hearts", "numOrd": 2 },  { "number": "10", "suit": "Spades", "numOrd": 10 },  { "number": "5", "suit": "Spades", "numOrd": 5 },  { "number": "Q", "suit": "Hearts", "numOrd": 12 }  ];  }]); |

Next, to define a template that Angular will populate with each object in the *cards* collection, add the following tags just after the title inside *index.html*. The *ng-repeat* directive allows us to define a *card* object for each item in the array. The card object is then used to populate *number* and *suit* attributes within an HTML template for a table row:

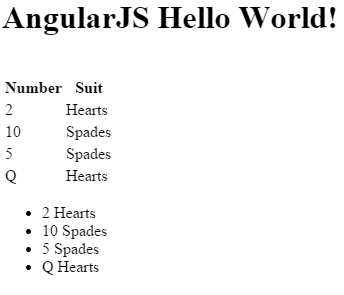
|  |
| --- |
| <table>  <tr><th>Number</th><th>Suit</th></tr>  <!-- Populate the HTML with each object in the cards model. -->  <tr ng-repeat="card in cards">  <td ng-bind ="card.number"></td>  <td ng-bind ="card.suit"></td>  </tr>  </table>  <br /> |

When running the application after these changes, the list of cards appears in the browser.



Exercise 4

🖍 Modify the code in Example 3 to also show the data in an unordered list using <ul> and <li> tags. Your output should resemble the following:



Show your modified HTML file here: (1 mark)

|  |
| --- |
| <!doctype html> <html> <head>  <title>Starting Angular</title> </head> *<!-- The ng-app directive triggers load and setup of AngularJS  after the DOM is loaded.  It sets the tag as the root of the AngularJS app. -->* <body ng-app="cardApp"> *<!-- Identify the controller and model to be used. -->* <div ng-controller="MainController">  *<!-- The ng-bind gets data from the model. -->* <h1 ng-bind="greeting"></h1>  <table>  <tr><th>Number</th><th>Suit</th></tr>  *<!-- Populate the HTML with each object in the cards model. -->* <tr ng-repeat="card in cards">  <td ng-bind ="card.number"></td>  <td ng-bind ="card.suit"></td>  </tr>  </table>  <br />  <ul ng-repeat="card in cards">  <li>{{card.number}} {{card.suit}}</li>  </ul>  </div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script src="js/controllers.js"></script> </body> </html> |

## Sorting

AngularJS provides an **orderBy** filter that allows you to sort items of an array. To use orderBy, you can define a model that includes the choices of sorting attributes:

<select ng-model="orderProp">

<option value="suit">Suit</option>

<option value="numOrd">Number</option>

</select>

Then, in the tag that contains your repeater, you set the *orderBy* filter with the designated sorting model:

<tr ng-repeat="card in cards | orderBy:orderProp ">

Example 4: Ordering Collection Results

🞑 This example shows how to order cards by either suit or number. To start, begin with the solution from Example 3. Then, add the following code just after the title <h1> header tag to input a user’s sort preference through a drop down selector.

|  |
| --- |
| Sort by:  <select ng-model="orderProp">  <option value="suit">Suit</option>  <option value="numOrd">Number</option>  </select> |

Notice that the *numOrd* property from the JSON model is used to sort the cards. The numOrd property is used for sorting since *orderBy* is not able to determine the difference between the Queen, Q, and its numeric equivalent of 12:

$scope.cards = [

{ "number": "2", "suit": "Hearts", "numOrd": 2 },

{ "number": "10", "suit": "Spades", "numOrd": 10 },

{ "number": "5", "suit": "Spades", "numOrd": 5 },

{ "number": "Q", "suit": "Hearts", "numOrd": 12 }

];

Next, inside index.html, replace the row tag and repeater directive with the following revision. This revision uses the *orderBy* filter to sort data by the selected option in the *orderProp* model:

|  |
| --- |
| <tr ng-repeat="card in cards | orderBy:orderProp "> |



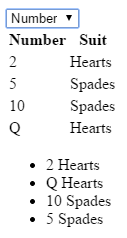
Exercise 5

🖍 How can you modify the code in the controller of Example 4 to set the default order so the sequence is by number? Show the line of code you can add to the controller to make this happen. (1 mark)

|  |
| --- |
| // Register our controller with the angular module. **var** cardApp = angular.module('cardApp', []);  // The controller is a constructor function that takes a $scope parameter. // The controller lets us establish data binding between the controller and the view. // The model is initialized with the $scope parameter. cardApp.controller('MainController', ['$scope', **function** ($scope) {  // Scope ensures that any changes to the   // model are reflected in the controller.  // Here we create an initialize a 'title' model.  $scope.greeting = "AngularJS Hello World!";   // Define cards model which stores an array of objects.  $scope.cards = [  { "number": "2", "suit": "Hearts", "numOrd": 2 },  { "number": "10", "suit": "Spades", "numOrd": 10 },  { "number": "5", "suit": "Spades", "numOrd": 5 },  { "number": "Q", "suit": "Hearts", "numOrd": 12 }  ];   $scope.orderProp = "numOrd"; }]); |

Exercise 6

🖍 Insert an un-ordered list in Example 4 with ul and li tags. Order the list though by suit and do this in your HTML with an AngularJS directive. Your output will resemble the following:



Show your new HTML code which inserts this list. (1 mark)

|  |
| --- |
| <!doctype html> <html> <head>  <title>Starting Angular</title> </head> *<!-- The ng-app directive triggers load and setup of AngularJS  after the DOM is loaded.  It sets the tag as the root of the AngularJS app. -->* <body ng-app="cardApp">  *<!-- Identify the controller and model to be used. -->* <div ng-controller="MainController">  *<!-- The ng-bind gets data from the model. -->* <h1 ng-bind="greeting"></h1>   Sort by:  <select ng-model="orderProp">  <option value="suit">Suit</option>  <option value="numOrd">Number</option>  </select>   <table>  <tr><th>Number</th><th>Suit</th></tr>  *<!-- Populate the HTML with each object in the cards model. -->* <tr ng-repeat="card in cards | orderBy:orderProp ">  <td ng-bind="card.number"></td>  <td ng-bind="card.suit"></td>  </tr>  </table>  <br />  <div ng-init="defaultOrder='suit'">  <ul ng-repeat="card in cards | orderBy:defaultOrder">  <li>{{card.number}} {{card.suit}}</li>  </ul>  </div>  </div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script src="js/controllers.js"></script> </body> </html> |

## Search Filters

AngularJS provides a nice way to quickly filter selection results. The user must define a search term:

Search: <input type="text" ng-model="modelName">

Then, the model name is included in the repeater:

<ul ng-repeat="item in items | filter:modelName">

<li>**{{** item **}}**</li>

</ul>

Example 5: Searching All Items in a Collection

🞑 This example demonstrates an implementation of the AngularJS filter. Starting with Example 4, under the ordering options selector, add the following input for filtering terms:

|  |
| --- |
| Search: <input type="text" ng-model="searchTerm"> |

Then, replace the tag that has the repeater with the following version where a search filter is included:

|  |
| --- |
| <tr ng-model="card" ng-repeat="card in cards | orderBy:orderProp | filter:searchTerm "> |

With hardly any code, the filter allows a user to narrow collection results with search terms:



The effect is impressive. However, the filter in Example 5 filters on all attributes in the collection. Given the data defined with JSON:

$scope.cards = [ { "number": "2", "suit": "Hearts", "numOrd": 2 },

{ "number": "10", "suit": "Spades", "numOrd": 10 },

{ "number": "5", "suit": "Spades", "numOrd": 5 },

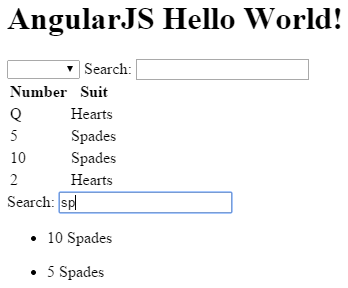
{ "number": "Q", "suit": "Hearts", "numOrd": 12 } ];

You will notice a search on ‘2’ would yield ‘Q Hearts’ and ‘2 Hearts’. ‘Q Hearts’ is selected because this item also contains a *numOrd* property of 12.



Exercise 7

🖍 Add an unordered list to Example 6. Adjust the code so you can search separately and independently on items in the list without affecting results that are displayed for the table. Do not sort your unordered list. Your output should resemble the following:



Show all of the code for your revised index.html page here: (1 mark)

|  |
| --- |
| <!doctype html> <html> <head>  <title>Starting Angular</title> </head> *<!-- The ng-app directive triggers load and setup of AngularJS  after the DOM is loaded.  It sets the tag as the root of the AngularJS app. -->* <body ng-app="cardApp">  *<!-- Identify the controller and model to be used. -->* <div ng-controller="MainController">  *<!-- The ng-bind gets data from the model. -->* <h1 ng-bind="greeting"></h1>   Sort by:  <select ng-model="orderProp">  <option value="suit">Suit</option>  <option value="numOrd">Number</option>  </select>   Search: <input type="text" ng-model="searchTerm">   <table>  <tr><th>Number</th><th>Suit</th></tr>  *<!-- Populate the HTML with each object in the cards model. -->* <tr ng-model="card" ng-repeat="card in cards | orderBy:orderProp | filter:searchTerm ">  <td ng-bind="card.number"></td>  <td ng-bind="card.suit"></td>  </tr>  </table>  <br />  Search: <input type="text" ng-model="searchField">  <ul ng-repeat="card in cards | filter:searchField">  <li>{{card.number}} {{card.suit}}</li>  </ul>  </div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script src="js/controllers.js"></script> </body> </html> |

Example 6: Refined Filtering

🞑 Given the flawed filtering in Example 5, this example binds search filter to specific model attributes. To begin, replace the search input in index.html with the following inputs:

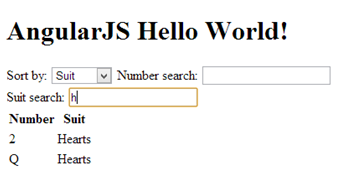
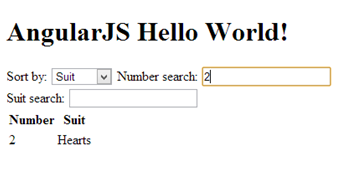
|  |
| --- |
| Number search: <input type="text" ng-model="searchTerm.card.number">  Suit search: <input type="text" ng-model="searchTerm.card.suit"><br/> |

These inputs allow you to specify the collection attributes that are used in a filter. For our example, the first input allows a user to define the search by card number. The second input allows a user to define the filter by suit.

To enable searching of the collection with both number and suit attributes, replace the existing row tag with the ng-repeat directive and ordering filter with a new one that also includes the improved search filter:

|  |
| --- |
| <tr ng-repeat="card in cards | orderBy:orderProp | filter:searchTerm.card "> |

Now the search results are reliable:



# Modules, Routing and Views

Now that we have examined some examples containing controllers, directives, and filters let’s see how to further separate our content and logic while improving the application scalability. In an AngularJS application it is common to separate the content and logic into the following structures:

* A **layout template** which manages the general presentation for the site.
* **Partial templates**, or **views**, where blocks of HTML are in separate files for specific presentations.
* An application **module** which contains a reference to the application and **routing** services.
* A series of **controllers** to populate the **models** with data.
* A **router** which receives page requests and selects the appropriate **template** and **controller**.

Example 7: Modules, Routing, and Templates

🞑 This example demonstrates how further to separate the logic and presentation using modules, routing and templates. This example is similar to the ones preceding it, but we will start from scratch to implement a more scalable development pattern for a one page application.

To create the first view, we will place the tags which filter and display the cards collection in a separate file called *list.html* in the **views** directory. To help later, some additional hyperlinks to detail and delete routes are included, but these new links are currently inactive.

**views/list.html**

|  |
| --- |
| Sort by:  <select ng-model="orderProp">  <option value="suit">Suit</option>  <option value="numOrd">Number</option>  </select>  Number search: <input type="text" ng-model="searchTerm.card.number">  Suit search: <input type="text" ng-model="searchTerm.card.suit"><br />  <table>  <tr><th><th><th>Number</th><th>Suit</th></tr>  <tr ng-repeat="card in cards | orderBy:orderProp | filter:searchTerm.card ">  <td><a href="#/home/number/{{card.number}}/suit/{{card.suit}}">View</a></td>  <td>  |  <a href="#/delete/number/{{card.number}}/suit/{{card.suit}}">Delete</a>  </td>  <td>**{{**card.number**}}**</td>  <td>**{{**card.suit**}}**</td>  </tr>  </table>  <br /> |

In this example, the controller assigns data to the model, the scope passes the populated model to the template, and the template renders the model data.

data 🡪 controller 🡪 model 🡪 scope 🡪 template 🡪 complete HTML

**js/controller.js**

|  |
| --- |
| **/\*global angular \*/**  var cardAppControllers = (function () {  'use strict';  var cardAppControllers = angular.module('cardAppControllers', []);  **// Declare the application controller and inject the scope reference.**  cardAppControllers.controller('AppCtrl', ['$scope', function ($scope) {  **// Define the title model.**  $scope.title = "AngularJS Tutorial";  }]);  **// Declare the controller that populates 'list.html' with data and inject the scope.**  cardAppControllers.controller('ListCtrl', ['$scope',  function ($scope) {  **// Define the cards model.**  $scope.cards = [  { "number": "2", "suit": "Hearts", "numOrd": 2 },  { "number": "10", "suit": "Spades", "numOrd": 10 },  { "number": "5", "suit": "Spades", "numOrd": 5 },  { "number": "Q", "suit": "Hearts", "numOrd": 12 }];  }]);  return cardAppControllers;  }()); |

Next, we will declare a module which references the services and routing structures in our application. To provide our application references to routing and controller services we will inject the AngularJS ‘ngRoute’ service and our own custom ‘cardAppControllers’ service into the module declaration. The router selects a template and controller based on the page request with a series of **when** and **otherwise** statements.

**js/app.js**

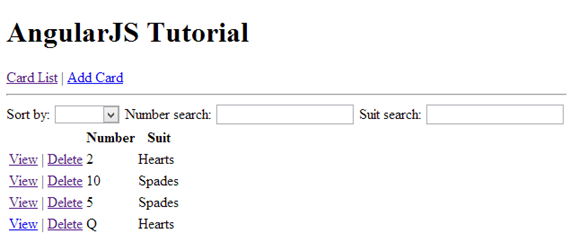
|  |
| --- |
| **// Declare module that references our controllers.**  var cardApp = angular.module('cardApp', ['ngRoute', 'cardAppControllers']).config(function ($routeProvider) {  **/\***  **Inject the AngularJS routing (ngRoute) service so we can**  **access the $routeProvider reference in our routing function.**  **Also inject the 'cardAppControllers' service which we will**  **define in 'controllers.js'.**  **\*/**  'use strict';  $routeProvider.when("/home", {  **/\* When 'home' route is selected**  **use the 'list.html' template and the 'ListCtrl' controller. \*/**  templateUrl: 'views/list.html',  controller: 'ListCtrl'  }).  **// If no route is selected then use the 'home' route.**  otherwise({ redirectTo: '/home' });  }); |

Now we can consolidate the main layout template of our one page application. The **ng-view** directive serves as a placeholder for the view that the controller selects.

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <!-- Notice here that ng-app references our 'cardApp' module -->  <body ng-app='cardApp'>  <!-- Reference the application level controller for the title model. -->  <div ng-controller="AppCtrl" ng-model="title">  <h1>**{{**title**}}**</h1>  <!-- The #home link selects routing for '/home'. Routing for  #add does not exist yet so the default for '/home' will be chosen.-->  <a href="#home">Card List</a> | <a href="#add">Add Card</a>  <hr>  <!-- Our one page app works by switching views here. -->  <div ng-view></div><br />  </hr>  </div>  <!--angular-->  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-route.min.js"></script>  <!--site-->  <script src="js/controllers.js"></script>  <script src="js/app.js"></script>  </body>  </html> |

When we run our project the output similar to the output in Example 6. However, this current example is more scalable because we have separated the content and into layout, view, module, routing, and controller components.



Exercise 8

🖍 The *AppCtrl* controller is referenced in the parent div tag within the index.html page. However, a reference to the *ListCtrl* controller cannot be found in index.html. How does the list.html template reference the controller than contains the array of cards? (1 mark)

|  |
| --- |
| The list.html template references the controller through the router. In AngularJS, the router receives page requests and selects the appropriate template and controller. |

# Services

Services allow us to divide the logic even further. By declaring a service, we can use the AngularJS dependency injection pattern to pass a reference to it into our controllers.

Example 8: Declaring a Custom Service

🞑 This example shows how to declare a custom service for retrieving data for the card application. Start with the solution for Example 7. Next, define a service that will retrieve the card data. Note that we are referencing the AngularJS $http with inline injection. This service provides access to the browser’s *XMLHttpRequest* object for making requests to the server. In the *services.js* file below, we have hard-coded the JSON data, but it is possible instead to use the commented-out commands to communicate with a live web service.

**js/services.js**

|  |
| --- |
| cardApp.factory("cardService", function ($http) {  'use strict';  return {  getCards: function ($scope) {  **/\***  **// LIVE WEB SERVICE**  **$http.get('http://localhost:1353/api/Cards').success(function(data) {**  **$scope.cards = data;**  **});**  **\*/**  $scope.cards = [  { "number": "2", "suit": "Hearts", "numOrd": 2 },  { "number": "10", "suit": "Spades", "numOrd": 10 },  { "number": "5", "suit": "Spades", "numOrd": 5 },  { "number": "Q", "suit": "Hearts", "numOrd": 12 }];  },  }  }); |

To reference our new services file, add the following script reference to the index.html. Make sure that you add it after the reference to js/app.js.

**index.html**

|  |
| --- |
| <script src="js/services.js"></script> |

To use the new service from our list controller, we use dependency injection to provide a reference to the card service. Here is the list controller replacement code with the injected card service:

**js/controllers.js**

|  |
| --- |
| **// Inject the scope and new cardService reference into the controller.**  cardAppControllers.controller('ListCtrl', ['$scope', 'cardService',  function ($scope, cardService) {  **// Define the cards model.**  cardService.getCards($scope);  }]); |

The output after implementing these changes is identical to the output from Example 7. This time though, when the application runs, Angular uses the card service to retreive the JSON data.

# Route Parameters

When providing routing for detailed views, it is often necessary to capture parameter values from the URL. Routers use the following syntax to identify parameter names and values:

when('/home/parameter0Name/:parameter0Value', {

templateUrl: 'views/partialTemplate.html', controller: 'CtrlName' }).

The URL takes on the following format:

<a href="#/home/parameter0Name/**{{**parameter0Value**}}**">View</a>

The controller can then reference the parameter through the **routeParams** service:

**// Inject scope and routeParams services.**

appControllers.controller('CtrlName', ['$scope', '$routeParams',

function ($scope, $routeParams) {

alert('Parameter received: ' + $routeParams.parameter0Name);

}]);

Example 9: Adding a Detail View

🞑 To demonstrate how to implement route parameters we will add a detail view to the card application starting with the solution for Example 8. First, add a new partial template called *detail.html* which contains presentation tags for displaying a card’s number and suit data.

**views/detail.html**

|  |
| --- |
| <h3>Card Detail</h3>  <table>  <th></th><th>Number</th><th>Suit</th>  <tr ng-model= "card">  <td><a href="#/delete/number/**{{**card.number**}}**/suit/**{{**card.suit**}}**">Delete</a></td>  <td>**{{**card.number**}}**</td>  <td>**{{**card.suit**}}**</td>  </tr>  </table> |

Then, add a ‘getCard’ function (which is a service) to retrieve details for a single card. This service receives a reference to the scope, cardNum, and cardSuit. It will populate the data of scopes card object so that the scope can pass that data to the card object expressions in the view.

**js/services.js**

|  |
| --- |
| getCard: function ($scope, cardNum, cardSuit) {  **/\***  **// LIVE WEB SERVICE**  **var url = "http://localhost:1353/api/cards?number=" + cardNum**  **+ "&suit=" + cardSuit;**  **$http.get(url).success(function(data) {**  **$scope.card = data;**  **}); \*/**  $scope.card = { "number": cardNum, "suit": cardSuit };  }, |

Next, add a controller to call the new service. Notice how we need to inject the $scope, $routeParams, and cardService references. The $routeParams service provides us access to the parameters that are passed to the controller. The cardService is the service module that we defined.

**js/controller.js**

|  |
| --- |
| **// Inject scope, $routeParams, and cardService**  cardAppControllers.controller('DetailCtrl', ['$scope', '$routeParams', 'cardService',  function ($scope, $routeParams, cardService) {  cardService.getCard($scope, $routeParams.numberID, $routeParams.suitID);  }]); |

Next, in the router, add a condition to select the detail view when the home address is provided along with number and suit parameters.

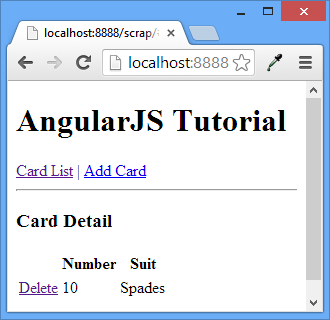
**js/app.js**

|  |
| --- |
| when('/home/number/:numberID/suit/:suitID', {  templateUrl: 'views/detail.html', controller: 'DetailCtrl' }). |

Note that back in Example 7, we added tags tags to the list view, to pass the card number and suit to the controller:

<td><a href="#/home/number/**{{**card.number**}}**/suit/**{{**card.suit**}}**">View</a></td>

When you run the program, it will function as before. This time though the detail view for each card is also enabled.



# Managing Inputs

AngularJS automates the management of user input. The functionality ranges from detecting changes in user inputs to validation of data.

## Text Boxes

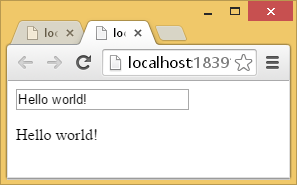
Text boxes are easy. We can pre-populate the text inputs with data from the controller using *ng-model*. Any input entered into the text box with an *ng-model* attribute automatically sets values in other controls that use the *ng-bind* directive to this model.

Example 10: Text Boxes

🞑 This sample shows how to take any text input and display it in a paragraph.

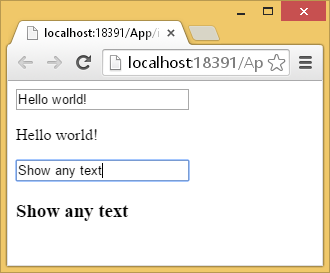
|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', function ($scope) {  $scope.greeting = "Hello world!";  }]);  </script>  </head>  <body ng-app="formSample" ng-controller="ExampleController">  <input type="text" ng-model="greeting" /> <br />  <p ng-bind="greeting"></p>  </body>  </html> |

Here is the sample output:



Exercise 9

🖍 Modify the code in Example 10 by adding a second input for text which is displayed within h3 tags. The message also needs to be managed by a different model that is defined in the controller script. Your output will resemble the following screenshot.



Show all of the contents for your revised html file (1 mark)

|  |
| --- |
| <!doctype html> <html> <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  **var** formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', **function** ($scope) {  $scope.greeting = "Hello world!";  $scope.sample = "Welcome to BCIT!"  }]);  </script> </head> <body ng-app="formSample" ng-controller="ExampleController">  <input type="text" ng-model="greeting" /> <br />  <p ng-bind="greeting"></p>  <input type="text" ng-model="sample" /> <br />  <h3>{{sample}}</h3> </body> </html> |

## Buttons

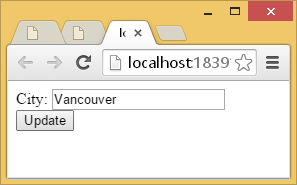
With buttons, we can use the AngularJS **ng-click** directive to trigger the click event. The attribute of the ng-click directive receives the function to call as a parameter. The function is defined within the controller and is accessible through the scope service.

Example 11: Handling Button Click Events

🞑 Here is a look at how to implement the *ng-click* directive which passes the *city* model to the *update()* function.

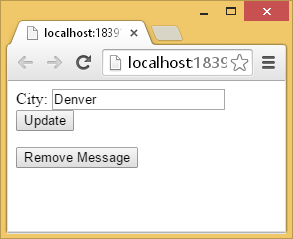
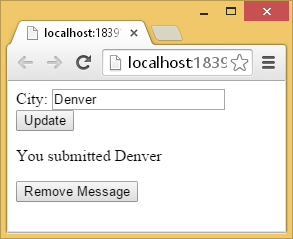
|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var myApp = angular.module('myApp', []);  myApp.controller('ExampleController', ['$scope', function ($scope) {  $scope.city = 'Vancouver';  // Execute some JavaScript  $scope.update = function (input) {  $scope.city = input;  alert("Updated address to: " + input);  };  }]);  </script>  </head>  <body ng-app="myApp" ng-controller="ExampleController">  City: <input ng-model="city"><br />  <button ng-click="update(city)">Update</button>  </body>  </html> |

This is the sample output:



Exercise 10

🖍 Update the code from Example 11 so the city submitted is printed in a paragraph tag but only after the button is clicked. Also, add a button that says “Remove Message”. When you click the new “Remove Message” button the message disappears. Your output will look similar to the following:



Show all contents from your revised index.html file: (2 marks)

|  |
| --- |
| <!doctype html> <html> <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  **var** myApp = angular.module('myApp', []);  myApp.controller('ExampleController', ['$scope', **function** ($scope) {  $scope.city = 'Vancouver';  $scope.isSubmitted = **false**;   // Execute some JavaScript  $scope.update = **function** (input) {  $scope.city = input;  $scope.isSubmitted = **true**;  alert("Updated address to: " + input);  };   $scope.remove = **function** () {  $scope.isSubmitted = **false**;  };  }]);  </script> </head> <body ng-app="myApp" ng-controller="ExampleController">  City: <input ng-model="city"><br />  <button ng-click="update(city)">Update</button>  <br />  <p ng-if="isSubmitted">You submitted {{city}}</p>  <br />  <button ng-click="remove()">Remove Message</button> </body> </html> |

## Check Boxes

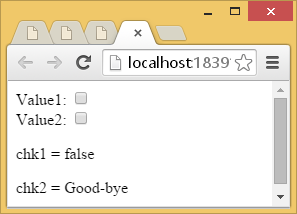
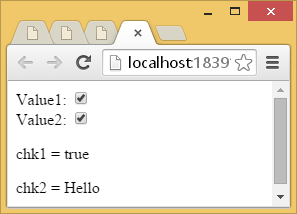
Check boxes which include the ng-model directive automatically update the assigned model with a Boolean value by default. However, you can customize the data associated with the ng-model directive in a checkbox with the *ng-true-value* and *ng-false-value* directives. The *ng-change* directive allows you to call a method that is defined in the controller whenever a checkbox is selected or deselected.

Example 12

🞑 Here is a sample that updates the model values chk1 and chk2 with separate check boxes. The first checkbox exhibits default behavior. The second checkbox demonstrates custom behavior.

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Check boxes</title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', function ($scope) {  $scope.chk1 = true;  $scope.chk2 = 'Hello';  $scope.update = function () {  alert('hi');  }  }]);  </script>  </head>  <body ng-app="formSample">  <form name="myForm" ng-controller="ExampleController">  Value1: <input type="checkbox" ng-model="chk1" ng-change="update()"> <br />  Value2: <input type="checkbox" ng-model="chk2"  ng-true-value="'Hello'" ng-false-value="'Good-bye'"> <br />  <p>chk1 = <label ng-bind="chk1"></label> </p>  <p>chk2 = **{{**chk2**}}**</p>  </form>  </body>  </html> |

The sample output looks like the following:

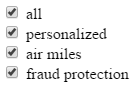


Exercise 11

🖍 Starting with the following code:

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Check boxes</title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', function ($scope) {  $scope.chk1 = true;  $scope.chk2 = 'Hello';  $scope.options = [{ "item": "personalized", "checked": true },  { "item": "air miles", "checked": false },  { "item": "fraud protection", "checked": false }];  $scope.all = { "item": "all", "checked": false };  }]);  </script>  </head>  <body ng-app="formSample" ng-controller="ExampleController">  <div>  <input type="checkbox"/> **{{**all.item**}}**  </div>  <div ng-repeat="option in options">  <input type="checkbox" ng-model="option.checked" /> **{{**option.item**}}**  </div>  </body>  </html> |

Adjust the code so all check boxes are automatically checked when *all* is checked. Hint: use an ng-change directive to trigger the update. Your output will resemble the following when *all* is checked.



Show your revised file contents here: (3 marks)

|  |
| --- |
| <!doctype html> <html> <head>  <title>Check boxes</title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  **var** formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', **function** ($scope) {  $scope.chk1 = **true**;  $scope.chk2 = 'Hello';  $scope.options = [{ "item": "personalized", "checked": **true** },  { "item": "air miles", "checked": **false** },  { "item": "fraud protection", "checked": **false** }];  $scope.all = { "item": "all", "checked": **false** };   $scope.update = **function** () {  **if** ($scope.all.checked) {  **for**(**var** i = 0; i < $scope.options.length; i++) {  $scope.options[i].checked = **true**;  }  }  };  }]);  </script> </head> <body ng-app="formSample" ng-controller="ExampleController">  <div>  <input type="checkbox" ng-model="all.checked" ng-change="update()" /> {{all.item}}  </div>  <div ng-repeat="option in options">  <input type="checkbox" ng-model="option.checked" /> {{option.item}}  </div> </body> </html> |

## Radio Option List

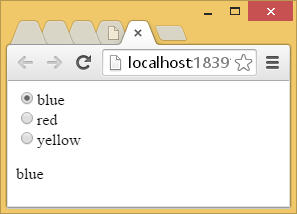
Radio inputs are easy too. Options are bound together by the ng-model value.

Example 13: Radio Option List

🞑 In this example, the model value is set when the option is selected.

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var myApp = angular.module('myApp', []);  myApp.controller('ExampleController', ['$scope', function ($scope) {  $scope.color = "blue";  }]);  </script>  </head>  <body ng-app="myApp" ng-controller="ExampleController">  <input type="radio" ng-model="color" value="blue" />blue<br />  <input type="radio" ng-model="color" value="red" />red<br />  <input type="radio" ng-model="color" value="yellow" />yellow<br />  <p ng-bind="color"></p>  </body>  </html> |

When you run the code you will see the option an selected item printed in a paragraph below the options as well.

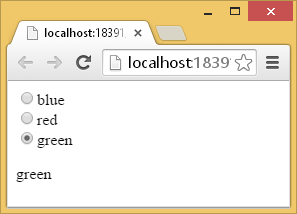


Example 14: Dyanmic Radio Option List

🞑 Here is another sample of a radio option list but this one is dynamically populated with the ng-repeat directive.

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var myApp = angular.module('myApp', []);  myApp.controller('ExampleController', ['$scope', function ($scope) {  $scope.colors = ["blue", "red", "green"];  $scope.color = "";  $scope.updateColor = function (input) {  $scope.color = input;  }  }]);  </script>  </head>  <body ng-app="myApp" ng-controller="ExampleController">  <div ng-repeat="col in colors">  <input type="radio" ng-model="color" value="col"  name="preference" ng-click="updateColor(col)">**{{**col**}}**<br>  </div>  <p ng-bind="color"></p>  </body>  </html> |

Here is the sample output:



## Select Controls

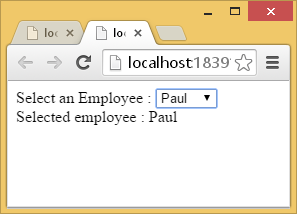
Select control options can also be bound to the model when the ng-model directive is included in the select element.

Example 15: Basic Select Control

🞑 This example demonstrates how to set up the select control with static data. The selected value is displayed in text output that follows the control.

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var myApp = angular.module('myApp', []);  myApp.controller('ExampleController', ['$scope', function ($scope) {  $scope.selectedEmployee = "";  }]);  </script>  </head>  <body ng-app="myApp" ng-controller="ExampleController">  Select an Employee :  <select ng-model="selectedEmployee">  <option>John</option>  <option>Paul</option>  <option>Ringo</option>  </select>  <br />  <!-- Show employee that is selected. -->  Selected employee : **{{**selectedEmployee**}}**  </body>  </html> |

This application allows you to select a person from the list:



Example 16: Dynamic Select Control

🞑 This example shows how to create a dynamically populated select control with the ng-repeat directive.

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var myApp = angular.module('myApp', []);  myApp.controller('ExampleController', ['$scope', function ($scope) {  $scope.employees = ["john", "paul", "ringo"];  }]);  </script>  </head>  <body ng-app="myApp" ng-controller="ExampleController">  Select an Employee :  <select ng-model="selectedEmployee">  <option ng-repeat="emp in employees"  ng-model="selectedEmployee">  **{{**emp**}}**  </option>  </select>  <p ng-bind="selectedEmployee"></p>  </body>  </html> |

It looks like the one shown above.

## Hiding, Showing, Disabling, and Enabling Inputs

AngularJS directives can help to automate the hiding, showing, enabling, and disabling of inputs through pre-defined directives (see Table 2).

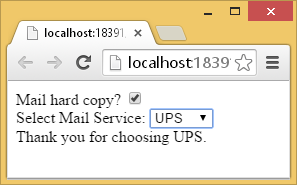
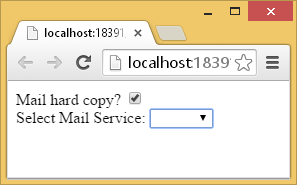
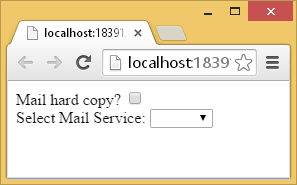
### Table 2: Hiding, Showing, Disabling, and Enabling Inputs

|  |  |
| --- | --- |
| **Directive** | **Function** |
| ng-hide | Hides an element when the expression is true. |
| ng-show | Shows an element when the expression is true. |
| ng-disabled | Disables an element when the expression is true. |

Example 17: Showing, Hiding, Enabling, and Disabling

🞑 This example demonstrates how to show, hide, enable, and disable controls with AngularJS directives. The form prompts the user to select a checkbox if they wish to select a mail service. A select input for a mail service is enabled if the checkbox checked. If a message is selected a thank message appears.

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var myApp = angular.module('myApp', []);  myApp.controller('ExampleController', ['$scope', function ($scope) {  $scope.update = function (input) {  $scope.mailService = "";  }  }]);  </script>  </head>  <body ng-app="myApp" ng-controller="ExampleController">  Mail hard copy?  <input type="checkbox" ng-model="checked" ng-change="update(checked)"><br />  Select Mail Service:  <select id="transferTo" ng-disabled="!checked" ng-model="mailService">  <option>UPS</option>  <option>Fedex</option>  </select>  <br />  <!-- Only show if checkbox is checked and mailService is selected -->  <div ng-show="checked && mailService"  ng-disabled="!checked">Thank you for choosing **{{**mailService**}}**.</div>  </body>  </html> |



# Debugging

Exercise 12

🖍 In the starter files for Assignment 1, I have written an instruction to output data to the console in either the Firefox debugging tools or Google Chrome debugging tools.

b) What is the instruction that I used to output the debug data? (1 mark)

|  |
| --- |
| console.log($scope.greeting); |

a) Take a screenshot of this data when it is printed to the console and show it here. (2 marks)

|  |
| --- |
|  |

# Terminology Review

Exercise 13

🖍 Fill in the blanks with each the following terms. Use each term only once: (0.5 marks each)

Controller, array, View(template), Router, object, Model, Service, Scope, Directive

1. \_\_Model\_\_ defines data.

2. \_\_Router\_\_ selects templates (partial views) and controllers.

3. \_\_Directive \_\_ modifies behavior of HTML.

4. \_\_Scope\_\_ enables two way data binding between the controller and view.

5. \_\_View (template) \_\_stores HTML.

6. \_\_Controller\_\_ encapsulates data and logic.

7. \_\_Service\_\_ encapsulates logic which can be used by many controllers.

8. Given the following JSON: (2 marks)

{"employees": [

{ "firstName":"John" , "lastName":"Doe" },

{ "firstName":"Anna" , "lastName":"Smith" },

{ "firstName":"Peter" , "lastName":"Jones" }]}

a. The data contained within the curly braces is a \_\_\_object\_\_\_ .

b. The data contained within the square braces is a \_\_array\_\_ .

Exercise 14

🖍 The goal of this code is to populate a text input with data.

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Check boxes</title>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script>  var formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', function ($scope) {  $scope.myValue = 'ABC';  }]);  </script>  </head>  <body ng-app="formSample" ng-controller="ExampleController">  <input type="text" ng-bind="myValue" />  </body>  </html> |

Explain why the code does not work: (1 mark)

|  |
| --- |
| Within the input tag, we are supposed to use ng-model instead of ng-bind  <input type="text" ng-model="myValue" /> |