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:**

8%

**Due:**

June 11, 11:59PM

11:59PM

20% will be deducted each day this is late.

Contents

[\*\* Tip \*\* 2](#_Toc484168185)

[Debugging Tips 2](#_Toc484168186)

[Starting AngularJS 2](#_Toc484168187)

[Advantages of the AngularJS Framework 2](#_Toc484168188)

[Terminology Introduction 2](#_Toc484168189)

[Templates 3](#_Toc484168190)

[Directives 3](#_Toc484168191)

[Models 3](#_Toc484168192)

[Scope 3](#_Toc484168193)

[Expressions 3](#_Toc484168194)

[View 3](#_Toc484168195)

[Router 3](#_Toc484168196)

[Filter 3](#_Toc484168197)

[Data Binding 4](#_Toc484168198)

[Controller 4](#_Toc484168199)

[Dependency Injection 4](#_Toc484168200)

[Module 4](#_Toc484168201)

[Services 4](#_Toc484168202)

[Directives 4](#_Toc484168203)

[Sorting 10](#_Toc484168204)

[Managing Inputs 12](#_Toc484168205)

[Text Boxes 12](#_Toc484168206)

[Buttons 13](#_Toc484168207)

[Check Boxes 15](#_Toc484168208)

[Radio Option List 16](#_Toc484168209)

[Select Controls 18](#_Toc484168210)

[Hiding, Showing, Disabling, and Enabling Inputs 19](#_Toc484168211)

[Table 2: Hiding, Showing, Disabling, and Enabling Inputs 19](#_Toc484168212)

[Terminology Review 21](#_Toc484168213)

# \*\* Tip \*\*

If application does not appear to work properly, try running the example in Brackets (brackets.io).

## Debugging Tips

Hopefully you are all going to keep your debugger window open at all times when writing any sort of JavaScript. I have put together this video highlighting how to debug your application.

<https://www.youtube.com/watch?v=IDijSikrRZ0>

# 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 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://code.angularjs.org/1.5.8/angular.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>  asjMessage: <input type="text" ng-model="ajsMessage">  <h1 ng-bind="ajsMessage"></h1>  <h1> {{ajsMessage}}</h1>  <script src="https://code.angularjs.org/1.5.8/angular.js"></script>  </body>  </html> |

Example 2: Using a Module and Controller

🞑 This example introduces basic AngularJS modules, scope, and controllers. 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.  // In this case the controller takes a $scope parameter.  // Scope lets us establish data binding between the controller and the view.  cardApp.controller('MainController', ['$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 everything nested inside this 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://code.angularjs.org/1.5.8/angular.js"></script>  <script src="js/controllers.js"></script>  </body>  </html> |

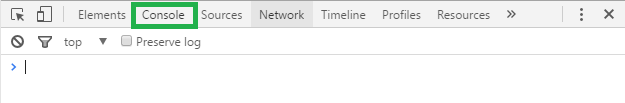
When the program runs the model, *greeting*, stores the data provided by the scope. The directive, **ng-bind**, is used to display value of the *greeting* model.

Exercise 2

🖍 Add the following instruction to the end of your *MainController* function from Example 2.

console.log($scope.greeting);

Show a screenshot of the console window in the Chrome debugger (Access by pressing ctrl+shift+I or F12 or choose More Tools | Developer Tools) and select console.

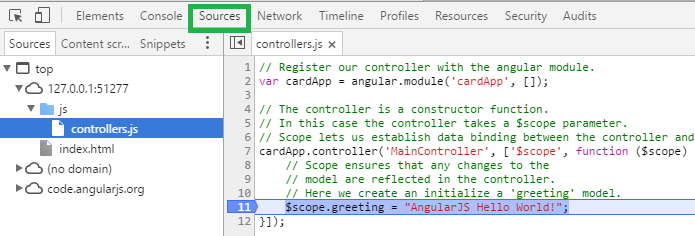


The output from the log instruction. Or, you if you can’t deploy your code on a server just run it in Firefox and from the menu select **Developer | Debugger** and then choose **Console**. (1 mark)

|  |
| --- |
|  |

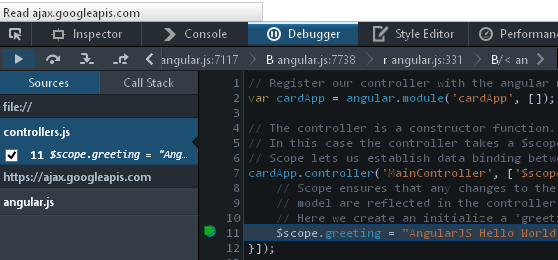
b)

In the More Tools | Developer Tools menu choose the Sources tab in Chrome. Then select the controllers.js file.



Or in the Developer | Debugger | Sources view of the Firefox debugger, open the controller.js file.

Whichever browser you choose, set a breakpoint beside your console.log instruction which you added in part a. Refresh the page in the browser and show a screenshot of the application while it is halted at the breakpoint like the following.



In your case though, place your breakpoint by the console.log instruction which you added in part a.

(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', []);  // Scope lets us establish data binding between the controller and the view.  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.  console.log($scope.greeting);  $scope.greeting = "AngularJS Hello World!";  // add new model 'subtitle'  $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>  <h3 ng-bind="subtitle"></h3>  </div>  <script src="https://code.angularjs.org/1.5.8/angular.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.

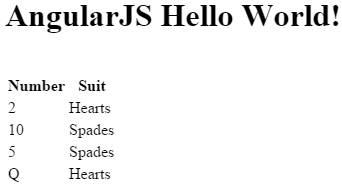
**js/controllers.js**

|  |
| --- |
| // 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 '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 h1 tag 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:

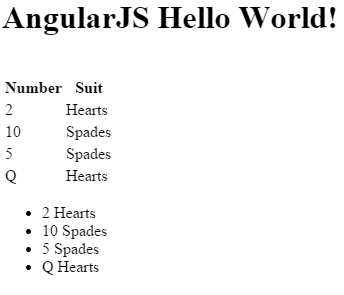
|  |
| --- |
| <table>  <tr><th>Number</th><th>Suit</th></tr>  <!-- Populate the HTML with each object in the cards model. -->  <tr ng-repeat="card in cards">  <td ng-bind ="card.number"></td>  <td ng-bind ="card.suit"></td>  </tr>  </table>  <br /> |

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 />  <!-- Populate unordered list with cards-->  <ul ng-repeat="card in cards">  <li>{{card.number}} {{card.suit}}</li>  </ul>  <h3 ng-bind="subtitle"></h3>  </div>  <script src="https://code.angularjs.org/1.5.8/angular.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 "> |



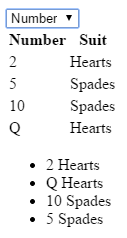
Exercise 5

🖍 How can you modify the code in the controllers.js file of Example 4 to set the default order so the sequence is by number? Show this short one line of code you can add to the controller to make this happen (remember you are sorting on a value that is stored in a model). (1 mark)

|  |
| --- |
| $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)

|  |
| --- |
| <ul>  <li ng-repeat="card in cards | orderBy: 'suit'">{{card.number}} {{card.suit}}</li>  </ul> |

# Managing Inputs

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

## Text Boxes

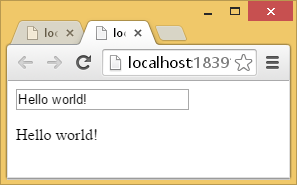
Text boxes are easy. We can pre-populate the text inputs with data from the controller using *ng-model*. Any input entered into the text box with an *ng-model* attribute automatically sets values in other controls that use the *ng-bind* directive to this model.

Example 5: Text Boxes

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

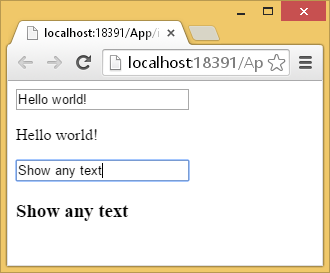
|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://code.angularjs.org/1.5.8/angular.js"></script>  <script>  var formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', function ($scope) {  $scope.greeting = "Hello world!";  }]);  </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 7

🖍 Modify the code in Example 5 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://code.angularjs.org/1.5.8/angular.js"></script>  <script>  var formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', function ($scope) {  $scope.greeting = "Hello world!";  $scope.largerGreeting = "Show any text";  }]);  </script>  </head>  <body ng-app="formSample" ng-controller="ExampleController">  <input type="text" ng-model="greeting" /> <br />  <p ng-bind="greeting"></p>  <!-- larger greeting -->  <input type="text" ng-model="largerGreeting" /> <br />  <h3>{{largerGreeting}}<h3>  </body>  </html> |

## Buttons

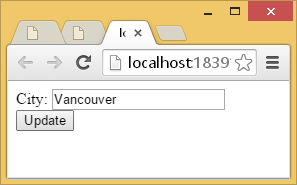
With buttons, we can use the AngularJS **ng-click** directive to trigger the click event. The attribute of the ng-click directive receives the function to call as a parameter. The function is defined within the controller and is accessible through the scope service.

Example 6: 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.

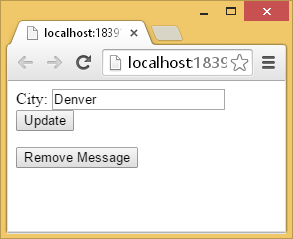
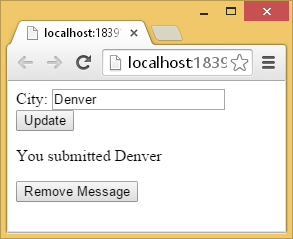
|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <script src="https://code.angularjs.org/1.5.8/angular.js"></script>  <script>  var myApp = angular.module('myApp', []);  myApp.controller('ExampleController', ['$scope', function ($scope) {  $scope.city = 'Vancouver';  // Execute some JavaScript  $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 8

🖍 Update the code from Example 6 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://code.angularjs.org/1.5.8/angular.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;  $scope.displayCity = "You submitted " + input;  alert("Updated address to: " + input);  };  $scope.remove = function () {  $scope.city = "";  $scope.displayCity = "";  alert("Removed address!");  }  }]);  </script>  </head>  <body ng-app="myApp" ng-controller="ExampleController">  City: <input ng-model="city"><br />  <button ng-click="update(city)">Update</button>  <p>{{displayCity}}</p>  <button ng-click="remove()">Remove</button>  </body>  </html> |

## Check Boxes

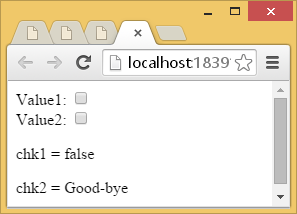
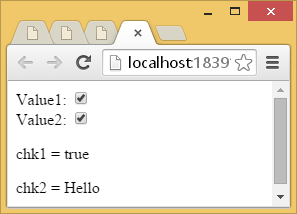
Check boxes which include the ng-model directive automatically update the assigned model with a Boolean value by default. However, you can customize the data associated with the ng-model directive in a checkbox with the *ng-true-value* and *ng-false-value* directives. The *ng-change* directive allows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

Example 7

🞑 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://code.angularjs.org/1.5.8/angular.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:

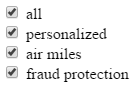


Exercise 9

🖍 Starting with the following code:

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Check boxes</title>  <script src="https://code.angularjs.org/1.5.8/angular.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.



Hint: You will probably want to create a for loop in the header. Here is an example of a for loop that iterates through the cards array:

for(var i=0; i<$scope.cards.length; i++) {

alert($scope.cards[i].suit);

}

Show your revised file contents here: (3 marks)

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Check boxes</title>  <script src="https://code.angularjs.org/1.5.8/angular.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() {  $scope.options.forEach(function(item) {  item.checked = $scope.all.checked;  }, this);  };  }]);  </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

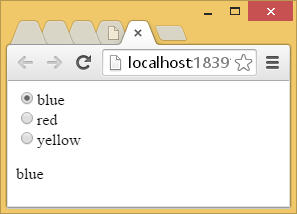
Radio inputs are easy too. Options are bound together by the ng-model value.

Example 8: 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://code.angularjs.org/1.5.8/angular.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.

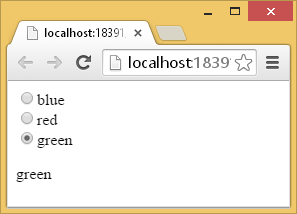


Example 9: Dyanmic Radio Option List

🞑 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://code.angularjs.org/1.5.8/angular.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

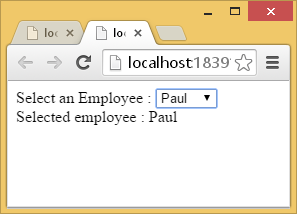
Select control options can also be bound to the model when the ng-model directive is included in the select element.

Example 10: 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://code.angularjs.org/1.5.8/angular.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 11: 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://code.angularjs.org/1.5.8/angular.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).

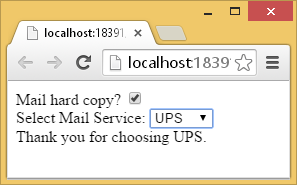
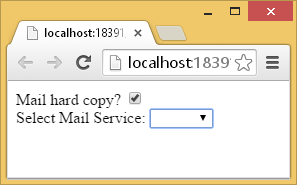
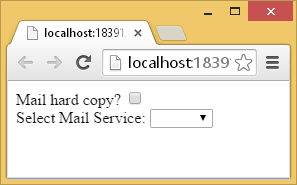
### Table 2: Hiding, Showing, Disabling, and Enabling Inputs

|  |  |
| --- | --- |
| **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 12: 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://code.angularjs.org/1.5.8/angular.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> |



# Terminology Review

Exercise 10

🖍 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 11

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

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Check boxes</title>  <script src="https://code.angularjs.org/1.5.8/angular.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)

|  |
| --- |
| ng-bind is used for output; instead you should use ng-model for input. |