Contents

[Due March 20, 6PM 1](#_Toc414028353)

[Loading Additional Partial Views 2](#_Toc414028354)

[Validating Form Inputs 4](#_Toc414028355)

[Validating Forms 4](#_Toc414028356)

[Validating Inputs 4](#_Toc414028357)

[Table 1: Input Directives 4](#_Toc414028358)

[Applying Validation Styles 8](#_Toc414028359)

[Directives 12](#_Toc414028360)

[Naming: 13](#_Toc414028361)

[Restriction Options 13](#_Toc414028362)

[Template Options 14](#_Toc414028363)

[Scope Options 14](#_Toc414028364)

[Link Option 14](#_Toc414028365)

[The $q Service 22](#_Toc414028366)

# Due March 20, 6PM

Worth 10% of your mark.

20% will be deducted each day this is late.

# Loading Additional Partial Views

In addition to having a parent view combined with a view that is specified by the router, it is possible to load more than one view with the ng-include directive:

<div ng-include="'./views/delme.html'"></div>

Example 1: Using the ng-include Directive

🞑 Here is a short sample of how to add a partial view. First, add the partial view to your project.

**views/partialView.html**

|  |
| --- |
| <h3>I am a partial view.</h3> |

Next, add the following code the following file. This code creates your application module and creates a *title* model in the scope.

**js/app.js**

|  |
| --- |
| // Application module that injects routing, controller, and directive dependencies.  var myApp = angular.module('myApp', ['ngRoute', 'myControllers'])  .config(function ($routeProvider) {  $routeProvider.when("/home", {  // templateUrl: 'views/list.html',  // controller: 'ListCtrl'  })  // If no route is selected then use the 'home' route.  .otherwise({ redirectTo: '/home' });  });  // Controller - dispatches inputs and outputs.  var myControllers = (function () {  var myControllers = angular.module('myControllers', []);  // Controllers are defined by the controller function.  myControllers.controller('AppCtrl', ['$scope', function ($scope) {  $scope.title = "AngularJS Tutorial";  }]);  return myControllers;  }()); |

Finally, add the index.html file which references the partial view with the *ng-include* directive.

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <body ng-app='myApp'>  <div ng-controller="AppCtrl">  <div ng-include="'./views/partialView.html'"></div>  </div>  <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>  <script src="js/app.js"></script>  </body>  </html> |

Exercise 1

🖍 Using Example 1, add a second ng-include directive to your HTML which references another partial view in the path ./views/anotherPartial.html. In the new partial view show your name in bold letters. Show your code for the second partial view:

|  |
| --- |
| <p style="font-weight: bold;">Alex Dai</p> |

Show the revised index.html file with both ng-include directives in it:

|  |
| --- |
| <!doctype html> <html> <head>  <title>Starting Angular</title> </head> <body ng-app='myApp'>  <div ng-controller="AppCtrl">  <div ng-include="'./views/partialView.html'"></div>  <div ng-include="'./views/anotherPartial.html'"></div>  </div>  <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>  <script src="js/app.js"></script> </body> </html> |

## Validating Form Inputs

Like any good data driven framework, AngularJS helps to automate input validation on the client. When user inputs are not in the correct format, AngularJS makes it easy to notify users immediately so they can quickly get the information they need to continue. Validation not only ensures accuracy of data inputs, it also helps to keep your site secure by narrowing the range of attacks that can be leveraged against an application. Your application users will also appreciate any effort to implement client side validation to guide them while also keeping their data safe.

### Validating Forms

It is possible to determine quickly if a form is valid with the expression that uses the Boolean $valid property of the form; **formName.$valid**. It is also possible to determine quickly if the control of a form is valid with the expression **formName.inputName.$valid**.

### Validating Inputs

Most of the input controls implement a **required directive**. Table 1 lists several common **validation directives** that we can apply to a text input along with expressions that report their validity status.

### Table 1: Input Directives

|  |  |
| --- | --- |
| **Directive** | **Boolean Error Expression (True when invalid)** |
| required | formName.inputName.$error.**required** |
| ng-minlength="3" | formName.inputName.$error.**minlength** |
| ng-maxlength="10" | formName.inputName.$error.**maxlength** |
| ng-pattern="/^[A-z]+$/" | formName.inputName.$error.**pattern** |

Example 2: Validation Directives and Expressions

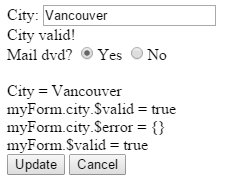
🞑 The following example shows how we can validate a text box field using directives for constraints such as minimum length, maximum length, regular expression and required. Each type of directive displays an error message when the invalid state is true. As well, the example displays the valid state for the control and the form.

The controller sets the default *userAddress* model to Vancouver. When the user clicks the update button, the *update()* function within the controller for the form launches an alert box with the update information. When the user clicks the cancel button the *cancel()* function within the controller displays an alert box with the cancelled information.

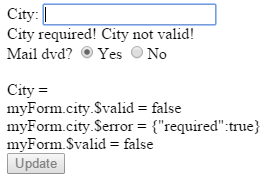
**index.html**

|  |
| --- |
| <!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.userAddress = 'Vancouver';  $scope.update = function (input) {  alert("Updated address to: " + input);  };  $scope.cancel = function (input) {  alert("Cancelled changes to: " + input);  };  }]);  </script>  </head>  <body ng-app="myApp">  <form name="myForm" ng-controller="ExampleController">  <!-- Input with validation -->  City: <input name="city" ng-model="userAddress"  ng-minlength="3"  ng-maxlength="10"  ng-pattern="/^[A-z]+$/"  required><br />  <!-- Show validation error messages -->  <span ng-show="myForm.city.$error.minlength">Minimum 3 characters required for city!</span>  <span ng-show="myForm.city.$error.maxlength">More than 10 characters not permitted for city!</span>  <span ng-show="myForm.city.$error.pattern">Only alphabetical characters allowed for city!</span>  <span ng-show="myForm.city.$error.required">City required!</span>  <span ng-show="myForm.city.$valid">City valid!</span>  <span ng-show="!myForm.city.$valid">City not valid!</span><br>  Mail dvd?  <input type="radio" name="mail" ng-model="dvd" value="yes" ng-required="!dvd">Yes  <input type="radio" name="mail" ng-model="dvd" value="no" ng-required="!dvd">No<br />  <!-- Validation message for mail -->  <span ng-show="!dvd">Mail DVD not selected<br></span>  <!-- Show input and form validation status -->  City = **{{**userAddress**}}**<br>  myForm.city.$valid = **{{**myForm.city.$valid**}}**<br>  myForm.city.$error = **{{**myForm.city.$error**}}**<br>  myForm.$valid = **{{**myForm.$valid**}}**<br>  <button ng-click="update(userAddress)" ng-disabled="myForm.$invalid">Update</button>  <button ng-click="cancel('Cancelled')" ng-hide="myForm.$invalid">Cancel</button>  </form>  </body>  </html> |

When the form is valid, all invalidation expressions evaluate to false. The update button is enabled since the form must be valid. The cancel button is displayed since the form is valid.



When the form is invalid, the invalid expression evaluates to true. The update button is disabled in this case and the cancel button is hidden.



Exercise 2

🖍 Which directive and expression in Example 2 enables the update button when the form is valid? (1 mark)

|  |
| --- |
| The ng-disabled directive enables the update button when the form is valid. |

Exercise 3

🖍 Starting with Example 2 change the *required* attribute from the input text box to *ng-required*. Examine the use of *ng-required* with the radio options as an example of how to use the *ng-required* directive. (1 mark)

|  |
| --- |
| ng-required allows us to set the required attribute depending on a boolean expression. |

Exercise 4

🖍 Adjust the code in Example 17 from day 1 to show the validation status for the mail option. If “Mail hard copy” is checked the form should only be valid if either UPS or Fedex is selected. None are selected by default. This validation message should declare whether the mail control is valid or invalid. Use the form’s $valid property value to show this message. Show the revised example with your adjustment. (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.update = **function** (input) {  $scope.mailService = "";  }  }]);  </script> </head> <body ng-app="myApp" ng-controller="ExampleController">  <form name="myForm">  Mail hard copy?  <input name="mailHardCopy" type="checkbox" ng-model="checked" ng-change="update(checked)" required><br />  <span ng-show="myForm.mailHardCopy.$valid">Mail hard copy selected!</span>  <span ng-show="!myForm.mailHardCopy.$valid">Mail hard copy not selected!</span>  <br />   Select Mail Service:  <select name="mailService" id="transferTo" ng-disabled="!checked" ng-model="mailService"  ng-required="!mailService">  <option>UPS</option>  <option>Fedex</option>  </select>  <br />  myForm.mailHardCopy.$valid = {{myForm.mailHardCopy.$valid}}<br>  myForm.mailHardCopy.$error = {{myForm.mailHardCopy.$error}}<br>  myForm.mailService.$valid = {{myForm.mailService.$valid}}<br>  myForm.mailService.$error = {{myForm.mailService.$error}}<br>  myForm.$valid = {{myForm.$valid}}<br>  <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>  </form> </body> </html> |

## Applying Validation Styles

We can apply validation styles by several means. To help with this effort, AngularJS actually automatically generates several CSS classes for elements that are using AngularJS validation directives.

* @property {boolean} $**untouched** True if control has not lost focus yet.
* @property {boolean} $**touched** True if control has lost focus.
* @property {boolean} $**pristine** True if user has not interacted with the control yet.
* @property {boolean} $**dirty** True if user has already interacted with the control.

Figure 1 displays several class names that Angular dynamically generates in the browser at run time:

Figure 1



Example 3: Validation Styling

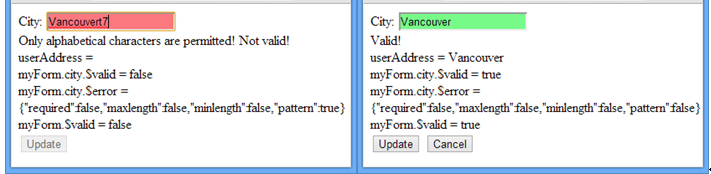
🞑 This example applies auto-generated css styles to the input box in Example 2.

|  |
| --- |
| <style type="text/css">  .css-form input.ng-invalid.ng-dirty { background-color: #FA787E; }  .css-form input.ng-valid.ng-dirty { background-color: #78FA89; }  </style> |

Then, replace the form directive with this one to set the form’s class for styling:

|  |
| --- |
| <form name="myForm" ng-controller="ExampleController" class="css-form"> |

When running the application now, the input box background changes based on validation status:



Exercise 5

🖍 Modify Example 3 to so the city shows a gray or slightly off white background to the city input if no modifications have been made:

|  |
| --- |
| <!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.userAddress = 'Vancouver';   $scope.update = **function** (input) {  alert("Updated address to: " + input);  };  $scope.cancel = **function** (input) {  alert("Cancelled changes to: " + input);  };  }]);  </script>  <style type="text/css">  .css-form input.ng-invalid {  background-color: #FA787E;  }   .css-form input.ng-valid {  background-color: #78FA89;  }   .css-form input.ng-pristine {  background-color: #AAAAAA;  }  </style>  </head> <body ng-app="myApp">  <form name="myForm" ng-controller="ExampleController" class="css-form">  *<!-- Input with validation -->* City: <input name="city" ng-model="userAddress"  ng-minlength="3"  ng-maxlength="10"  ng-pattern="/^[A-z]+$/"  required><br />   *<!-- Show validation error messages -->* <span ng-show="myForm.city.$error.minlength">Minimum 3 characters required for city!</span>  <span ng-show="myForm.city.$error.maxlength">More than 10 characters not permitted for city!</span>  <span ng-show="myForm.city.$error.pattern">Only alphabetical characters allowed for city!</span>  <span ng-show="myForm.city.$error.required">City required!</span>  <span ng-show="myForm.city.$valid">City valid!</span>  <span ng-show="!myForm.city.$valid">City not valid!</span><br>   Mail dvd?  <input type="radio" name="mail" ng-model="dvd" value="yes" ng-required="!dvd">Yes  <input type="radio" name="mail" ng-model="dvd" value="no" ng-required="!dvd">No<br />   *<!-- Validation message for mail -->* <span ng-show="!dvd">Mail DVD not selected<br></span>   *<!-- Show input and form validation status -->* City = {{userAddress}}<br>  myForm.city.$valid = {{myForm.city.$valid}}<br>  myForm.city.$error = {{myForm.city.$error}}<br>  myForm.$valid = {{myForm.$valid}}<br>   <button ng-click="update(userAddress)" ng-disabled="myForm.$invalid">Update</button>  <button ng-click="cancel('Cancelled')" ng-hide="myForm.$invalid">Cancel</button>  </form> </body> </html> |

Exercise 6

🖍 Starting with Example 3, apply formatting to the first set of messages that report erroneous input data for the city. Adjust the formatting so the font is displayed using red font. Show your code here:

|  |
| --- |
| <!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.userAddress = 'Vancouver';   $scope.update = **function** (input) {  alert("Updated address to: " + input);  };  $scope.cancel = **function** (input) {  alert("Cancelled changes to: " + input);  };  }]);  </script>  <style type="text/css">  .css-form input.ng-invalid {  background-color: #FA787E;  }   .css-form input.ng-valid {  background-color: #78FA89;  }   .css-form .errors {  color: red;  }  </style>  </head> <body ng-app="myApp">  <form name="myForm" ng-controller="ExampleController" class="css-form">  *<!-- Input with validation -->* City: <input name="city" ng-model="userAddress"  ng-minlength="3"  ng-maxlength="10"  ng-pattern="/^[A-z]+$/"  required><br />   *<!-- Show validation error messages -->* <span class="errors" ng-show="myForm.city.$error.minlength">Minimum 3 characters required for city!</span>  <span class="errors" ng-show="myForm.city.$error.maxlength">More than 10 characters not permitted for city!</span>  <span class="errors" ng-show="myForm.city.$error.pattern">Only alphabetical characters allowed for city!</span>  <span class="errors" ng-show="myForm.city.$error.required">City required!</span>  <span ng-show="myForm.city.$valid">City valid!</span>  <span ng-show="!myForm.city.$valid">City not valid!</span><br>   Mail dvd?  <input type="radio" name="mail" ng-model="dvd" value="yes" ng-required="!dvd">Yes  <input type="radio" name="mail" ng-model="dvd" value="no" ng-required="!dvd">No<br />   *<!-- Validation message for mail -->* <span ng-show="!dvd">Mail DVD not selected<br></span>   *<!-- Show input and form validation status -->* City = {{userAddress}}<br>  myForm.city.$valid = {{myForm.city.$valid}}<br>  myForm.city.$error = {{myForm.city.$error}}<br>  myForm.$valid = {{myForm.$valid}}<br>   <button ng-click="update(userAddress)" ng-disabled="myForm.$invalid">Update</button>  <button ng-click="cancel('Cancelled')" ng-hide="myForm.$invalid">Cancel</button>  </form> </body> </html> |

Example 4: Form Inputs with Validation

🞑 Now that we have discussed several form automation techniques, let’s create a form that adds a playing card to the collection in our cards application. To start, begin with the card application solution from example 9 during day 1. Next, add the following service which adds a playing carding card.

**js/services.js**

|  |
| --- |
| addCard : function ($scope, cardNum, cardSuit) {  **/\***  **// LIVE WEB SERVICE**  **var url = "http://localhost:1353/api/cards?number=" + cardNum + "&suit=" + cardSuit;**  **$http.post(url).success(function(data) {**  **$scope.message = data;**  **});**  **\*/**  $scope.message = { "Msg": cardNum.toUpperCase() + ' '  + cardSuit + ' has been added.'};  }, |

Next, add the following controller code. Note that we define the two functions within the scope of the *AddCtrl* controller. The *add()* function redirects the application to the *AddConfirmCtrl* controller which then adds the card and displays a success or fail message in the confirmation view. The *reset()* function clears the object data in the form.

**js/controllers.js**

|  |
| --- |
| **// AddConfirmCtrl calls the service to add the card.**  cardAppControllers.controller('AddConfirmCtrl', ['$scope', '$routeParams', 'cardService',  function ($scope, $routeParams, cardService) {  cardService.addCard($scope, $routeParams.numberID, $routeParams.suitID);  }]);  **// The $location service is injected to enable the redirect.**  cardAppControllers.controller('AddCtrl', ['$scope', '$location', 'cardService',  function AddCtrl($scope, $location, cardService) {  **// Create an empty object.**  $scope.master = {};  **// When add is clicked, redirect to the confirm view and controller where**  **// the card is actually added added.**  $scope.add = function (card) {  $location.path('/addconfirm/number/' + card.number + '/suit/' + card.suit);  };  **// When reset is clicked clear the ‘card’ model defined within the scope to**  **// clear the form data.**  $scope.reset = function () {  $scope.card = angular.copy($scope.master);  };  }]); |

The *add.html* view allows us to input a card and suit combination. The *ng-pattern* directive ensures that the number ranges from A, 1, 2…J, Q, or K. When the number input is incorrect, Angular displays an error message using the *ng-show* directive inside a span tag. Clicking the reset button calls the *reset()* function in the *AddCtrl* controller to clear the data from the form. The save button is only enabled if the form is valid. Clicking the save button calls *add()* to store the new card.

**views/add.html**

|  |
| --- |
| <form name="form" class="css-form">  Number:  **<!-- Apply card number validation with the ng-pattern and required directives -->**  <input type="text" ng-model="card.number" name="number"  ng-pattern="/^([aAkKjJQq23456789]{1}|(10){1})$/" required/><br />  **<!-- When the number input is invalid show text explaining the error. -->**  <span ng-show="form.number.$error.pattern" class="invalidText">  Card numbers can only be A,a,2-10,j,J,q,Q,k,K!</span><br/>  Suit:  <input type="radio" ng-model="card.suit" value="Clubs" required/>Clubs  <input type="radio" ng-model="card.suit" value="Hearts" />Hearts  <input type="radio" ng-model="card.suit" value="Spades" />Spades  <input type="radio" ng-model="card.suit" value="Diamonds" />Diamonds  <br />  **<!-- Call reset() function within AddCtrl controller. -->**  <button ng-click="reset()">RESET</button>  **<!-- Call add() function within AddCtrl controller. -->**  <button ng-click="add(card)" ng-disabled="form.$invalid ">SAVE</button>  </form> |

When the add button is clicked, the application goes to the add() function. The add function redirects the application to the confirm.html view.

**confirm.html**

|  |
| --- |
| **{{**message.Msg**}}** |

Next, add two conditions to the router that select appropriate views and controllers for adding a playing card. When there is a request for the *add.html* view, Angular will use the *AddCtrl* controller both to manage data and to handle click events. The *add()* function will receive post data and then redirect to the *confirm.html* page. Angular will use the *AddConfirmCtrl* controller to show either a success or fail message.

**js/app.js**

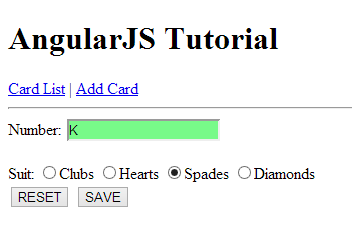
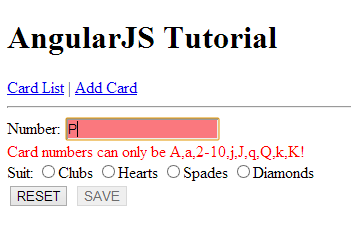
|  |
| --- |
| when('/add', {  templateUrl: 'views/add.html', controller: 'AddCtrl' }).  when('/addconfirm/number/:numberID/suit/:suitID', {  templateUrl: 'views/confirm.html', controller: 'AddConfirmCtrl' }). |

To apply styles that denote valid and invalid card additions, add the following css inside the head tag. The first two css classes apply colors to the background of the input box; this highlights the valid status. The third style applies red color to the error message when the number violates the required pattern.

**index.html**

|  |
| --- |
| <style type="text/css">  .css-form input.ng-invalid.ng-dirty { background-color: #FA787E; }  .css-form input.ng-valid.ng-dirty { background-color: #78FA89; }  span.invalidText { color: #ff0000; }  </style> |

An invalid form displays an invalid message and disables the save button. A valid form enables the save button. In either case, appropriate styling is applied.



# Directives

At a high level, directives are markers on a DOM element (such as an attribute, element name, or CSS class) that tell AngularJS's HTML compiler ($compile) to apply a specific behavior or transformation.

## Naming:

Directive names are written in camel case. There are two ways to reference a directive from the DOM. If we create an **ngModel** directive, then the proper way to reference it from the DOM is with one of the following:

1. <input **ng-model**="foo"> // The preferred way separates the name with hyphens.

2. <input **data-ng-model**="foo"> // This way is used to satisfy HTML validation tools.

Other formats outside these two are deprecated and are not recommended by AngularJS.

## Restriction Options

The ‘restrict‘ option of a directive specifies whether the directive is applied as an element, attribute or a class. By default, directives are restricted to the attribute option. The valid values for ‘restrict’ are typically set to:

* ‘**A**’ – Attribute (You want to use your directive as <div rating>)
* ‘**E**’ – Element (Use it as <rating>)
* ‘**C**’ – Class. (Use it like <div class=”rating”>)

Any combination of restrictions can be used to such as ‘AE’ for attributes and elements.

With a restriction of A and E for example:

|  |
| --- |
| .directive('helloWorld', function () {  return {  restrict: 'AE',  }  }); |

You can then implement the directive in an attribute or element:

|  |
| --- |
| <div hello-world person='Peter'></div>  <hello-world person="Jan" /> |

## Template Options

Directives provide a template option to define the html tags on the DOM.

## Scope Options

It is possible to define scope parameters that are managed inside the directive with the scope property.

## Link Option

The link option of a directive takes a function with the parameters; scope, element, attrs, and ctrl.

Example 5: Starting with a Basic Inline Directive

🞑 This simple example shows how a directive can be created to append HTML content. The directive is set up to be applied as an attribute with a restriction of ‘A’. Note that the JavaScript reference to the model is *welcomeMessage* but the reference to this in HTML is *welcome-message*:

**js/app.js**

|  |
| --- |
| // Application module that injects routing, controller, and directive dependencies.  var myApp = angular.module('myApp', ['ngRoute', 'myControllers', 'myDirectives'])  .config(function ($routeProvider) {  $routeProvider.when("/home", {  // templateUrl: 'views/list.html',  // controller: 'ListCtrl'  }  )  // If no route is selected then use the 'home' route.  .otherwise({ redirectTo: '/home' });  });  // Controller - dispatches inputs and outputs.  var myControllers = (function () {  var myControllers = angular.module('myControllers', []);  // Controllers are defined by the controller function.  myControllers.controller('AppCtrl', ['$scope', function ($scope) {  $scope.title = "AngularJS Directive Tutorial";  }]);  return myControllers;  }());  // Directive - Modifies HTML behaviour.  var myDirectives = (function () {  var myDirectives = angular.module('myDirectives', []);  // directive() is a factory method to create directives.  myDirectives.directive('welcomeMessage', function () {  return {  restrict: 'A', // restrict to attributes.  template: '<b>Hello from {{title}}</b>'  }  });  return myDirectives;  }()); |

Next replace the index.html file with the following. Note that the scope model *messageFromDirective* is visible only because the element references the directive *my-star-rating* which sets this model.

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <!-- Notice here that ng-app references our 'myApp' module -->  <body ng-app='myApp'>  <div ng-controller="AppCtrl" ng-model="title">  <h1>**{{**title**}}**</h1>  <a href="#home">Home</a><br /><br />  <!--Implement small tags with directives-->  <div welcome-message></div><br />  </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/app.js"></script>  </body>  </html> |

Exercise 7

🖍 Notice that the directive declaration in Example 5 in JavaScript uses the name ‘welcomeMessage’ but the HTML reference is ‘welcome-message’. What happens if you change the HTML reference to be ‘welcomeMessage’ just like the directive?

|  |
| --- |
| The message is not shown. |

Exercise 8

🖍 Modify Example 5to restrict your directive for use in an element. After changing the restriction property in the directive to ‘E’ show what change is needed in the HTML to use this revised directive.

|  |
| --- |
| <!doctype html> <html> <head>  <title>Starting Angular</title> </head>  *<!-- Notice here that ng-app references our 'myApp' module -->* <body ng-app='myApp'>  <div ng-controller="AppCtrl" ng-model="title">  <h1>{{title}}</h1>  <a href="#home">Home</a><br /><br />   *<!--Implement small tags with directives-->* <welcome-message></welcome-message><br />  </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/app.js"></script> </body> </html> |

Exercise 9

🖍 Show what change in Example 5 is needed in the directive for the HTML to appear as “Hello from YourName”.

|  |
| --- |
| // Application module that injects routing, controller, and directive dependencies. **var** myApp = angular.module('myApp', ['ngRoute', 'myControllers', 'myDirectives'])  .config(**function** ($routeProvider) {  $routeProvider.when("/home", {  // templateUrl: 'views/list.html',  // controller: 'ListCtrl'  }  )  // If no route is selected then use the 'home' route.  .otherwise({ redirectTo: '/home' });  });  // Controller - dispatches inputs and outputs. **var** myControllers = (**function** () {  **var** myControllers = angular.module('myControllers', []);   // Controllers are defined by the controller function.  myControllers.controller('AppCtrl', ['$scope', **function** ($scope) {  $scope.name = "Alex Dai";  }]);  **return** myControllers; }());  // Directive - Modifies HTML behaviour. **var** myDirectives = (**function** () {  **var** myDirectives = angular.module('myDirectives', []);   // directive() is a factory method to create directives.  myDirectives.directive('welcomeMessage', **function** () {  **return** {  restrict: 'A', // restrict to attributes.  template: '<b>Hello from {{name}}</b>'  }  });  **return** myDirectives; }()); |

Example 6: Drawing Stars

🞑 This example shows how to create a directive that draws a specific number of stars. 

The *templateURL* option references the html for an unordered list of stars. The *scope* option states that the *rating* variable expects a value from the HTML. There is also another variable on the scope called stars inside the function. We have not declared it inside the scope option because it does not need to be passed in from the HTML.

**js/app.js**

|  |
| --- |
| // Application module that injects routing, controller, and directive dependencies.  var myApp = angular.module('myApp', ['ngRoute', 'myControllers', 'myDirectives'])  .config(function ($routeProvider) {  $routeProvider.when("/home", {  // templateUrl: 'views/list.html',  // controller: 'ListCtrl'  }  )  // If no route is selected then use the 'home' route.  .otherwise({ redirectTo: '/home' });  });  // Directive - Modifies HTML behaviour.  var myDirectives = (function () {  var myDirectives = angular.module('myDirectives', []);  // directive() is a factory method to create directives.  myDirectives.directive('myStars', function () {  return {  restrict: 'A',  scope: {  rating: '='  },  link: function ($scope, elem, attrs, ctrl) {  $scope.stars = [];  for (var i = 0; i < $scope.rating; i++) {  $scope.stars.push({}); // push empty objects onto an array  }  },  templateUrl: function (element, attr) { return 'views/stars.html' },  }  });  return myDirectives;  }());  // Controller - dispatches inputs and outputs.  var myControllers = (function () {  var myControllers = angular.module('myControllers', []);  // Controllers are defined by the controller function.  myControllers.controller('AppCtrl', ['$scope', function ($scope) {  $scope.title = "AngularJS Tutorial";  }]);  return myControllers;  }()); |

To line up and colour the stars properly add the following css.

**rating.css**

|  |
| --- |
| .rating{  margin: 0;  padding: 0;  color:#a9a9a9;  }  ul.rating {  display: inline-block;  }  .rating li {  list-style-type: none;  display: inline-block;  padding: 1px;  text-align: center;  font-weight: bold;  }  .filled {  color: #00ff21;  } |

Next, replace the existing html with the following. Notice how we have to refer to the camel case directive name with a hyphenated version in the html. In the same tag where the myStarRating directive is specified, notice how the ratingValue variable of the directive gets populated.

**views/stars.html**

|  |
| --- |
| <ul class="rating">  <li ng-repeat="star in stars" class="filled">  &#9733;  </li>  </ul> |

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  <link href="rating.css" rel="stylesheet" />  </head>  <body ng-app='myApp' ng-controller="AppCtrl">  <h3 ng-bind="title"></h3>  <div my-stars rating="5"></div>  <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>  <script src="js/app.js"></script>  </body>  </html> |

Example 7: Adding a Maximum Star Count

Now let’s show the star count out of total possible stars. 

Start with Example 6. To set this up, we add another variable called *max* to the *scope* option. In the link function we set the filled status when the star count is less than the total rating. The AngularJS, ng-switch directive series allows us to conditionally select elements based on the filled status. To proceed, replace the ‘myStars’ directive with this new version which includes two scope variables. The link function here iterates through all stars and fills the ones up to the rating count.

|  |
| --- |
| // directive() is a factory method to create directives.  myDirectives.directive('myStars', function () {  return {  restrict: 'A',  scope: {  rating: '=',  max: '='  },  link: function ($scope, elem, attrs, ctrl) {  $scope.stars = [];  for (var i = 0; i < $scope.max; i++) {  if (i < $scope.rating)  $scope.stars.push({ filled: true});  else  $scope.stars.push({ filled: false });  }  },  templateUrl: function (element, attr) { return 'views/stars.html' },  }  }); |

Here is the replacement code for the partial view. It performs a conditional check with ng-switch to determine whether to fill the star or not.

|  |
| --- |
| <ul class="rating">  <li ng-repeat="star in stars ">  <span ng-switch on="star.filled">  <div ng-switch-when="true" class="filled">&#9733;</div>  <div ng-switch-default class="rating">&#9733;</div>  </span>  </li>  </ul> |

This is replacement tag for index.html. Notice the values for *rating* and *max* are set.

**index.html**

|  |
| --- |
| <div my-stars rating="4" max="10"></div> |

Exercise 10

🖍 Modify the stars.html file so it dynamically shows the total stars awarded out of the maximum possible stars in text rather than with a visual graphic. For example;

If the element is <div my-stars rating="4" max="10"></div> the dynamically generated text displayed will be **4 out of 10 stars**.

If the element is <div my-stars rating="3" max="5"></div> the dynamically generated text displayed will be **3 out of 5 stars**.

Show your revised stars.html file here:

|  |
| --- |
| <p>{{rating}} out of {{max}} stars</p> |

Example 8: ng-click within a Directive

🞑 To allow the star rating to be set through a clickable interface, the ng-click directive can be used within the template to call a method in the link option. This example looks a lot like the previous one but the directive includes a function which gets called during the first load as well as every time a star is clicked.

To begin, start with the code from Example 8. Then replace the li tag inside views/stars.html with the following code:

|  |
| --- |
| <li ng-click="toggle($index)" ng-repeat=" star in stars "> |

Next, replace the directive with this new version which enables a toggle to update the stars whenever one is clicked.

|  |
| --- |
| // directive() is a factory method to create directives.  myDirectives.directive('myStars', function () {  return {  restrict: 'A',  replace: true,  scope: {  rating: '=',  max: '='  },  link: function ($scope, elem, attrs, ctrl) {  var updateStars = function () {  $scope.stars = [];  for (var i = 0; i < $scope.max; i++) {  if (i < $scope.rating)  $scope.stars.push({ filled: true });  else  $scope.stars.push({ filled: false });  }  };  $scope.toggle = function (index) {  $scope.rating = index + 1;  updateStars();  };  updateStars();  },  templateUrl: function (element, attr) { return 'views/stars.html' },  }  }); |

After making this change you will be able to change the rating by clicking on the stars.

Exercise 11

🖍 Create a directive that includes one text input and a button. The text input must have data before the submit button is enabled. When the submit button is clicked an alert message is displayed. Show your directive, controller, partial view and view code here:

|  |
| --- |
| Directive and controller:  // Application module that injects routing, controller, and directive dependencies. **var** myApp = angular.module('myApp', ['ngRoute', 'myControllers', 'myDirectives'])  .config(**function** ($routeProvider) {  $routeProvider.when("/home", {  // templateUrl: 'views/list.html',  // controller: 'ListCtrl'  }  )  // If no route is selected then use the 'home' route.  .otherwise({ redirectTo: '/home' });  });  // Directive - Modifies HTML behaviour. **var** myDirectives = (**function** () {  **var** myDirectives = angular.module('myDirectives', []);   // directive() is a factory method to create directives.  myDirectives.directive('myStars', **function** () {  **return** {  restrict: 'A',  scope: {},  link: **function** ($scope, elem, attrs, ctrl) {  $scope.showAlert = **function** () {  alert('Text data submitted: ' + $scope.textData);  };  },  templateUrl: **function** (element, attr) { **return** 'views/stars.html' }  }  });   **return** myDirectives; }());  // Controller - dispatches inputs and outputs. **var** myControllers = (**function** () {  **var** myControllers = angular.module('myControllers', []);   // Controllers are defined by the controller function.  myControllers.controller('AppCtrl', ['$scope', **function** ($scope) {  $scope.title = "AngularJS Tutorial";  }]);  **return** myControllers; }());  Partial View:  <form name="myForm">  Text Data:  <input name="textData" type="text" ng-model="textData" required>  <button ng-click="showAlert()" ng-disabled="myForm.textData.$invalid">Submit</button> </form>  View:  <!doctype html> <html> <head>  <title>Starting Angular</title> </head> <body ng-app='myApp' ng-controller="AppCtrl">  <h3 ng-bind="title"></h3>  <div my-stars></div><br /><br />   <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>  <script src="js/app.js"></script> </body> </html> |

# The $q Service

The $q service helps you run functions asynchronously. You can use it to return values (or exceptions) when processing is complete.

The $q service allows you to create a deferred object:

var deferred = $q.defer();

The deferred object allows you to expose an associated **Promise** instance as well as APIs that can signal the successful or unsuccessful completion of a task. A deferred promise essentially promises a resource once it is available. The promise delivery is announced with a call to the *resolve()* function of the deferred object. When an error or issue occurs, the promise can be cancelled with the deferred object’s reject() function.

var deferred = $q.defer();

$http.get("http://ssdprogram.ca/tetJson.php")

.success(function (data) {

deferred.resolve(data);

})

.error(function () {

console.log('bob' + $q.reject + 'bob');

deferred.reject("\*\*\* Rejected! \*\*\*\*");

});;

return deferred.promise;

Example 9: Asynchronous calls with $q

🞑 This example demonstrates how to use the q service to wait until numbers are retrieved from a REST service. Data is returned when available. In the controller, a then() function waits for the promise. If the promise is delivered the data is received in the first condition of the then() function. Otherwise, an error message is received by the second condition of the promise.

**js/app.js**

|  |
| --- |
| // Declare module that references our controllers.  var myApp = angular.module('myApp', ['ngRoute',  'myControllers']).config(function ($routeProvider) {  $routeProvider.when("/home", {  //templateUrl: 'views/list.html',  //controller: 'ListCtrl'  })  .otherwise({ redirectTo: '/home' });  });  var myControllers = (function () {  var myControllers = angular.module('myControllers', []);  myControllers.controller('AppCtrl', ['$scope', 'simpleService',  function ($scope, simpleService) {  $scope.title = "Angular JS!";  $scope.error = false;  var promise = simpleService.getData($scope);  promise.then(  function (numbers) {  $scope.numbers = numbers;  },  function (errorReason) {  $scope.error = true;  $scope.msg = "An error occurred while fetching data.";  console.log(errorReason);  });  }]);  return myControllers;  }());  myApp.factory("simpleService", ["$http", "$q", function ($http, $q) {  var service = {  getData: function ($scope) {  var deferred = $q.defer(); // Set up deferral.  $http.get("http://ssdprogram.ca/testJson.php")  .success(function (data) {  deferred.resolve(data); // Return resource.  })  .error(function () {  deferred.reject("\*\*\* Rejected! \*\*\*\*"); // Return rejection.  });;  return deferred.promise; // Promise to return something once available.  }  };  return service;  }]); |

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <!-- Notice here that ng-app references our 'myApp' module -->  <body ng-app='myApp'>  <!-- Reference the application level controller for the title model. -->  <div ng-controller="AppCtrl">  <h1 ng-bind="title"></h1>  <p ng-show="**{{**msg != ''**}}**" ng-bind="msg"></p>  <ul ng-repeat="number in numbers">  <li ng-bind="number"></li>  </ul>  </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>  <script src="js/services.js"></script>  </body>  </html> |

If you tamper with the URL so the data cannot be retrieved you will notice that an error message displays.

Exercise 12

🖍 Try Example 9 and ensure it receives data successfully. In the blank spaces provided beside the lines of code, enter a number to indicate the order which each instruction is called. 1 is the first instruction and 6 is the last instruction. Step through your code or use logging to verify the sequence: (3 marks)

\_\_\_1\_\_\_\_ var promise = simpleService.getData($scope);

\_\_\_4\_\_\_\_ promise.then(function (numbers) {

\_\_\_6\_\_\_\_ $scope.numbers = numbers;

},

\_\_\_2\_\_\_\_ var deferred = $q.defer(); // Set up deferral.

$http.get("http://ssdprogram.ca/testJson.php")

.success(function (data) {

\_\_\_5\_\_\_\_ deferred.resolve(data); // Return resource.

})

.error(function () {

deferred.reject("\*\*\* Rejected! \*\*\*\*"); // Return rejection.

});;

\_\_\_3\_\_\_\_ return deferred.promise; // Promise to return something once available