

Developing Web Applications Using AngularJS

Student Guide

Version 1.7

presented by
Hands On Technology Transfer, Inc.
Chelmsford, Massachusetts (USA)

Copyright © 2016-2018 Hands On Technology Transfer, Inc. All Rights Reserved.

This student guide is copyrighted. This document may not, in whole or in part, be copied, duplicated, reproduced, translated, electronically stored, or reduced to machine readable form without prior written consent from Hands On Technology Transfer, Inc. Training programs are available through Hands On Technology Transfer, Inc. For information, contact HOTT in Chelmsford, Massachusetts at (800) 413-0939. All trademarks, product names and company names are the property of their respective owners.

Although the material contained herein has been carefully reviewed, HOTT does not warrant it to be free of errors or omissions. HOTT reserves the right to make corrections, updates, revisions or changes to the information contained herein. HOTT does not warrant the material described herein to be free of patent infringement.

Do Not Duplicate

Do Not Duplicate

Table of Contents

Developing Web Applications Using AngularJS Course Introduction	xvii
Course Goals and Objectives.....	xviii
Module 1 Introduction to AngularJS	1-1
Module Goals and Objectives	1-2
Section 1-1 AngularJS	1-3
AngularJS	1-4
AngularJS <i>cont'd</i>	1-5
AngularJS vs jQuery	1-6
Using jQuery with AngularJS	1-7
Versions of Angular JS.....	1-8
Section 1-2 MVC vs MV*	1-9
Model-View-Controller	1-10
Model-View-Controller and AngularJS.....	1-11
Model-View-ViewModel.....	1-12
MVVM and AngularJS	1-13
MV* (Model-View-Whatever).....	1-14
Section 1-3 Working with AngularJS.....	1-15
Key Concepts in AngularJS.....	1-16
AngularJS Directives	1-17
Modules.....	1-18
Controllers.....	1-19
Controllers <i>cont'd</i>	1-20
Binding Data using Expressions	1-21
Binding Data using Expressions <i>cont'd</i>	1-22
Services.....	1-23
Dependency Injection.....	1-24
Dependency Injection <i>cont'd</i>	1-25
Single Page Applications.....	1-26
Adding Animations	1-27
Section 1-4 Using AngularJS in an Application	1-28
Referencing AngularJS	1-29
Referencing AngularJS <i>cont'd</i>	1-30
Module Summary	1-31
Module 2 Data Binding	2-1
Module Goals and Objectives	2-2
Section 2-1 AngularJS Data Binding	2-3
AngularJS and Data Binding.....	2-4
Defining the Application.....	2-5
Defining Application Variables	2-6
Data Binding using ngBind.....	2-7
AngularJS Expressions	2-8
Displaying Collections using ngRepeat	2-9
Properties Exposed via ngRepeat	2-10
Properties Exposed via ngRepeat <i>cont'd</i>	2-11
Defining Properties using ngModel.....	2-12
Section 2-2 Working with Model Data	2-13
Working with Controllers	2-14
Creating a Module	2-15
Defining a Controller.....	2-16
Using \$scope	2-17
Associating a Controller with a View.....	2-18
Example: Data Binding with a Controller	2-19
Defining Methods	2-20

Calling Methods using <code>ngClick</code>	2-21
Example: Calling Methods using <code>ngClick</code>	2-22
Example: Calling Methods using <code>ngClick</code> <i>cont'd</i>	2-23
Using <code>ngCloak</code> to Improve Initial UI Appearance.....	2-24
Using <code>ngCloak</code> to Improve Initial UI Appearance <i>cont'd</i>	2-25
Using <code>ngCloak</code> to Improve Initial UI Appearance <i>cont'd</i>	2-26
Using <code>ngCloak</code> to Improve Initial UI Appearance <i>cont'd</i>	2-27
Section 2-3 Working with Scope and the Data Model.....	2-28
Using Scope to Define the Data Model.....	2-29
Example: Binding Data to a Complex Data Model.....	2-30
Example: Binding Data to a Complex Data Model <i>cont'd</i>	2-31
Adding Methods to the Data Model.....	2-32
Passing Arguments to Methods.....	2-33
Passing Arguments to Methods using <code>ngClick</code>	2-34
Passing Arguments to Methods using <code>ngClick</code> <i>cont'd</i>	2-35
Passing User Input as Arguments.....	2-36
Issues with <code>ngRepeat</code>	2-37
Using a Custom Tracking Function with <code>ngRepeat</code>	2-38
Section 2-4 How Data Binding Works.....	2-39
Interpolation.....	2-40
Data Binding and the Digest Cycle.....	2-41
Watchers and the Digest Cycle.....	2-42
Using the <code>\$watch</code> Service.....	2-43
Understanding the <code>\$watch</code> Method.....	2-44
Example: Using the <code>\$watch</code> Method.....	2-45
Example: Using the <code>\$watch</code> Method <i>cont'd</i>	2-46
Using <code>\$apply</code>	2-47
Using <code>\$apply</code> <i>cont'd</i>	2-48
Using <code>\$apply</code> <i>cont'd</i>	2-49
Using AngularJS <code>\$timeout</code>	2-50
Example: Using AngularJS <code>\$timeout</code>	2-51
Module Summary.....	2-52
Module 3 Working with Filters.....	3-1
Module Goals and Objectives.....	3-2
Section 3-1 AngularJS Filters.....	3-3
AngularJS Filters.....	3-4
Section 3-2 Using Filters to Format Data.....	3-5
Using Filters to Format Data.....	3-6
The "Case" Filters.....	3-7
The Number Filter.....	3-8
The Currency Filter.....	3-9
The Currency Filter <i>cont'd</i>	3-10
Example: Using Filters.....	3-11
Filtering Dates.....	3-12
Filtering Dates <i>cont'd</i>	3-13
Example: Filtering Dates.....	3-14
Section 3-3 Using Filters to Sort or Restrict Data.....	3-15
Sorting and Restricting Data.....	3-16
Sorting Data.....	3-17
Sorting Data <i>cont'd</i>	3-18
Sorting Data <i>cont'd</i>	3-19
Dynamic Sorting.....	3-20
Dynamic Sorting <i>cont'd</i>	3-21
Toggling Sort Order.....	3-22
Toggling Sort Order <i>cont'd</i>	3-23
Example: Toggling Sorting.....	3-24
Example: Toggling Sorting <i>cont'd</i>	3-25

Example: Toggling Sorting <i>cont'd</i>	3-26
Restricting Data with <code>limitTo</code>	3-27
Example: Using <code>limitTo</code>	3-28
Example: Using <code>limitTo</code> <i>cont'd</i>	3-29
Restricting Data with Custom Text using <code>filter</code>	3-30
Restricting Data with Custom Text using <code>filter</code> <i>cont'd</i>	3-31
Restricting Data with Custom Text using <code>filter</code> <i>cont'd</i>	3-32
Chaining Filters	3-33
Section 3-4 Displaying Data as JSON	3-34
JSON Data	3-35
Section 3-5 Using Filters with JavaScript	3-36
Using Filters with JavaScript	3-37
Using Filters with JavaScript <i>cont'd</i>	3-38
Using Filters with JavaScript <i>cont'd</i>	3-39
Example: Using the <code>\$filter</code> Service	3-40
Example: Using the <code>\$filter</code> Service <i>cont'd</i>	3-41
Example: Using the <code>\$filter</code> Service <i>cont'd</i>	3-42
Injecting Filters into the Controller	3-43
More Examples	3-44
Module Summary	3-45
Module 4 Directives	4-1
Module Goals and Objectives	4-2
Section 4-1 Custom HTML Elements and Attributes	4-3
HTML Elements and Attributes	4-4
Custom Attributes in HTML5	4-5
Section 4-2 Overview of AngularJS Directives	4-6
Overview – Directives	4-7
What are Directives?	4-8
Why Use Directives?	4-9
Why Use Directives? <i>cont'd</i>	4-10
AngularJS Directives	4-11
AngularJS Directives <i>cont'd</i>	4-12
Directives and the HTML Compiler	4-13
AngularJS Built-in Directives	4-14
Section 4-3 Core Functionality Directives	4-15
Directives for Core Functionality	4-16
Directives for Core Functionality <i>cont'd</i>	4-17
Example: Using <code>ngSrc</code>	4-18
Example: Using <code>ngSrc</code> <i>cont'd</i>	4-19
Example: Using <code>ngPluralize</code>	4-20
Example: Using <code>ngReadOnly</code> and <code>ngDisabled</code>	4-21
Example: Using <code>ngReadOnly</code> and <code>ngDisabled</code> <i>cont'd</i>	4-22
Section 4-4 Data Binding Directives	4-23
Directives that Bind to the Model	4-24
<code>ng</code> Module Directives <i>cont'd</i>	4-25
<code>ng</code> Module Directives <i>cont'd</i>	4-26
Example: Using <code>ngBindHtml</code>	4-27
Example: Using <code>ngBindHtml</code> <i>cont'd</i>	4-28
Example: Using <code>ngShow</code>	4-29
Example: Using <code>ngShow</code> <i>cont'd</i>	4-30
Using <code>ngIf</code>	4-31
Using <code>ngIf</code> <i>cont'd</i>	4-32
Example: Using <code>ngIf</code>	4-33
Using <code>ngSwitch</code>	4-34
Example: Using <code>ngSwitch</code>	4-35
Using <code>ngClass</code>	4-36

Example: Using <code>ngClass</code>	4-37
Example: Using <code>ngClassEven</code> and <code>ngClassOdd</code>	4-38
Using <code>ngStyle</code>	4-39
Section 4-5 Event Handling Directives	4-40
Binding Page Events with Directives	4-41
<code>ng</code> Module Directives <i>cont'd</i>	4-42
Accessing Event Information	4-43
Accessing Event Information <i>cont'd</i>	4-44
Using <code>ngChange</code>	4-45
Using <code>ngClick</code>	4-46
Module Summary	4-47
Module 5 AngularJS Modules	5-1
Module Goals and Objectives	5-2
Section 5-1 Dependency Injection	5-3
Separation of Concerns in AngularJS	5-4
Dependency Injection	5-5
Dependency Injection <i>cont'd</i>	5-6
Dependency Injection in AngularJS	5-7
Dependency Injection in AngularJS <i>cont'd</i>	5-8
Dependency Injection in AngularJS <i>cont'd</i>	5-9
Dependency Injection and Modules	5-10
Dependency Annotation	5-11
Implicit Annotation	5-12
Implicit Annotation <i>cont'd</i>	5-13
Inline Array Annotation	5-14
<code>\$inject</code> Property Annotation	5-15
<code>\$inject</code> Property Annotation <i>cont'd</i>	5-16
Section 5-2 Working with Modules	5-17
JavaScript's Global Namespace	5-18
Encapsulating your Code in an Object	5-19
AngularJS Modules	5-20
AngularJS Modules <i>cont'd</i>	5-21
The <code>angular.module()</code> Function	5-22
Defining a Module	5-23
Retrieving an Existing Module	5-24
Retrieving an Existing Module <i>cont'd</i>	5-25
Linking a View to a Module	5-26
Modules and Dependencies	5-27
Modules and Dependencies <i>cont'd</i>	5-28
Example: Injecting One Module into Another	5-29
Example: Injecting One Module into Another <i>cont'd</i>	5-30
Example: Injecting One Module into Another <i>cont'd</i>	5-31
Another Way to Declare Modules	5-32
Section 5-3 Problems with Minification and Dependency Injection	5-33
Code Minification	5-34
Code Minification <i>cont'd</i>	5-35
Code Minification <i>cont'd</i>	5-36
Code Minification <i>cont'd</i>	5-37
Solving the Minification Problem	5-38
Using the <code>\$inject</code> Property	5-39
Using the <code>\$inject</code> Property <i>cont'd</i>	5-40
Passing an Array of Dependencies	5-41
Passing an Array of Dependencies <i>cont'd</i>	5-42
Section 5-4 (Reference) Built-in AngularJS Modules	5-43
List of Built-in AngularJS Modules	5-44
The <code>ng</code> Module	5-45
<code>ng</code> Module Functions	5-46

ng Module Directives	5-47
ng Module Directives <i>cont'd</i>	5-48
ng Module Providers and Types	5-49
ng Module Services	5-50
Overview of ngSanitize	5-51
Example: Using ngSanitize	5-52
Module Summary	5-53
Module 6 Controllers	6-1
Module Goals and Objectives	6-2
Section 6-1 AngularJS Controllers.....	6-3
AngularJS Controllers.....	6-4
Overview – AngularJS Controllers <i>cont'd</i>	6-5
Moving Data and Logic to a Controller.....	6-6
Moving Data and Logic to a Controller <i>cont'd</i>	6-7
Section 6-2 Defining Controllers	6-8
Defining a Controller	6-9
The <code>controller()</code> Method	6-10
Defining the Controller in a Module Object.....	6-11
Defining the Controller in a Separate File	6-12
Defining the Controller in a Separate File <i>cont'd</i>	6-13
Defining a Controller within an IIFE	6-14
Section 6-3 Scope and Controllers.....	6-15
What are Scopes?	6-16
Initializing <code>\$scope</code>	6-17
Adding Behaviors to <code>\$scope</code>	6-18
Section 6-4 Linking Controllers to Views	6-19
Attaching a Controller to a View	6-20
Example: Attaching a Controller to a View.....	6-21
Example: Attaching a Controller to a View <i>cont'd</i>	6-22
Section 6-5 Controllers and Dependencies	6-23
Dependencies for Controllers	6-24
The <code>\$log</code> Service	6-25
The <code>\$log</code> Service <i>cont'd</i>	6-26
The <code>\$window</code> Service	6-27
The <code>\$http</code> Service.....	6-28
Section 6-6 Nesting Controllers.....	6-29
Nested Controllers.....	6-30
Nested Controllers <i>cont'd</i>	6-31
Nested Controllers <i>cont'd</i>	6-32
Referencing Parent Scope with <code>\$parent</code>	6-33
Handling Nested Controllers.....	6-34
Creating a Nested Object.....	6-35
"Controller As" Syntax.....	6-36
"Controller As" Syntax <i>cont'd</i>	6-37
Example: Nested Controllers	6-38
Example: Nested Controllers <i>cont'd</i>	6-39
Module Summary	6-40
Module 7 Single Page Applications	7-1
Module Goals and Objectives	7-2
Section 7-1 Single Page Applications	7-3
What are Single Page Applications?	7-4
What are Single Page Applications? <i>cont'd</i>	7-5
Challenges of SPAs	7-6
Building SPAs in Angular JS.....	7-7
Section 7-2 Routing in AngularJS.....	7-8
SPA Behavior	7-9

Routing.....	7-10
Referencing <code>ngRoute</code>	7-11
Configuring Routes.....	7-12
Configuring Routes <i>cont'd</i>	7-13
Configuring Routes <i>cont'd</i>	7-14
Referencing Routes.....	7-15
SPAs and the Hash (#).....	7-16
SPAs and the Hash (#) <i>cont'd</i>	7-17
Section 7-3 Designing a SPA in AngularJS.....	7-18
View and Controllers.....	7-19
Specifying View Placement using <code>ngView</code>	7-20
Building the HTML Shell Page.....	7-21
Example: Simple SPA.....	7-22
Example: Simple SPA <i>cont'd</i>	7-23
Example: Simple SPA <i>cont'd</i>	7-24
Example: Simple SPA <i>cont'd</i>	7-25
Example: Simple SPA <i>cont'd</i>	7-26
Section 7-4 Advanced Routing.....	7-27
Passing Values via URLs.....	7-28
Defining a Route with Parameters.....	7-29
Defining a Route with Parameters <i>cont'd</i>	7-30
Accessing Route Parameters.....	7-31
Accessing Route Parameters <i>cont'd</i>	7-32
Example: More Sophisticated SPA.....	7-33
Example: More Sophisticated SPA <i>cont'd</i>	7-34
Example: More Sophisticated SPA <i>cont'd</i>	7-35
Example: More Sophisticated SPA <i>cont'd</i>	7-36
Example: More Sophisticated SPA <i>cont'd</i>	7-37
Example: More Sophisticated SPA <i>cont'd</i>	7-38
Example: More Sophisticated SPA <i>cont'd</i>	7-39
Example: More Sophisticated SPA <i>cont'd</i>	7-40
Example: More Sophisticated SPA <i>cont'd</i>	7-41
Example: More Sophisticated SPA <i>cont'd</i>	7-42
Example: More Sophisticated SPA <i>cont'd</i>	7-43
Section 7-5 The <code>\$location</code> Service.....	7-44
The <code>\$location</code> Service.....	7-45
<code>\$location</code> Service Modes.....	7-46
<code>\$location</code> Service Modes <i>cont'd</i>	7-47
<code>\$location</code> Service Modes <i>cont'd</i>	7-48
Working with <code>\$location</code> Service.....	7-49
Code Based Navigation.....	7-50
Code Based Navigation <i>cont'd</i>	7-51
Passing Additional Data via URL.....	7-52
Receiving Data Passed using Path Extensions.....	7-53
Module Summary.....	7-54
Module 8 Custom Services.....	8-1
Module Goals and Objectives.....	8-2
Section 8-1 Services.....	8-3
Code Reuse.....	8-4
Services in AngularJS.....	8-5
Services in AngularJS <i>cont'd</i>	8-6
Built-in AngularJS Services.....	8-7
Defining Custom Services.....	8-8
Services and Dependency Injection.....	8-9
Section 8-2 Creating a Custom Factory.....	8-10
Factories.....	8-11
Creating a Service with <code>factory()</code>	8-12

Example: Creating a Simple Factory.....	8-13
Example: Creating a More Complex Factory.....	8-14
Example: Creating a More Complex Factory <i>cont'd</i>	8-15
Example: Creating a More Complex Factory <i>cont'd</i>	8-16
Moving Controller Code to a Factory.....	8-17
Moving Controller Code to a Factory <i>cont'd</i>	8-18
Moving Controller Code to a Factory <i>cont'd</i>	8-19
Moving Controller Code to a Factory <i>cont'd</i>	8-20
Moving Controller Code to a Factory <i>cont'd</i>	8-21
Section 8-3 Creating a Custom Service.....	8-22
Services.....	8-23
Creating a Service with <code>service()</code>	8-24
Example: Creating a Simple Service.....	8-25
Example: Creating a More Complex Service.....	8-26
Example: Creating a More Complex Service <i>cont'd</i>	8-27
Example: Creating a More Complex Service <i>cont'd</i>	8-28
Moving Controller Code to a Service.....	8-29
Section 8-4 Creating a Custom Provider.....	8-30
AngularJS Providers.....	8-31
AngularJS Providers <i>cont'd</i>	8-32
Creating a Provider.....	8-33
Example: Creating a Simple Provider.....	8-34
Example: Creating a Simple Provider <i>cont'd</i>	8-35
Section 8-5 Creating Values and Constants.....	8-36
Registering Existing Data Items as Services.....	8-37
Creating a Value.....	8-38
Creating a Constant.....	8-39
Injecting Values and Constants.....	8-40
Injecting Values and Constants <i>cont'd</i>	8-41
Module Summary.....	8-42
Module 9 Using AngularJS Built-In Services.....	9-1
Module Goals and Objectives.....	9-2
Section 9-1 AngularJS Built-in Services.....	9-3
Services and Data.....	9-4
Built-in Services.....	9-5
Working with Services.....	9-6
Promises.....	9-7
Deferreds.....	9-8
AngularJS <code>\$q</code> Service.....	9-9
Example: Using the <code>\$q</code> Service.....	9-10
Example: Using the <code>\$q</code> Service <i>cont'd</i>	9-11
Example: Using the <code>\$http</code> Service.....	9-12
Using the Result.....	9-13
Using the Result <i>cont'd</i>	9-14
Chaining Promises.....	9-15
Section 9-2 Using the <code>\$http</code> Service.....	9-16
<code>\$http</code> Service.....	9-17
Configuring the <code>\$http</code> Request.....	9-18
<code>\$http</code> Shortcut Methods.....	9-19
Using <code>\$http</code> to GET Data.....	9-20
Using <code>\$http</code> to GET Data <i>cont'd</i>	9-21
Using <code>\$http</code> to POST Data.....	9-22
Using <code>\$http</code> to PUT Data.....	9-23
Using <code>\$http</code> to DELETE Data.....	9-24
Section 9-3 Using the <code>\$resource</code> Service.....	9-25
The <code>\$resource</code> Service.....	9-26

Configuring \$resource.....	9-27
\$resource Action Methods	9-28
Using \$resource to Make Requests.....	9-29
Executing the Actions.....	9-30
Executing the Actions <i>cont'd</i>	9-31
Executing the Actions <i>cont'd</i>	9-32
Simplifying the \$resource Syntax	9-33
Section 9-4 (Self-Study) Introduction to RESTful Services	9-34
What is REST?	9-35
RESTful Web Service Constraints.....	9-36
Building RESTful Web Services	9-37
Defining an ASP.NET Web API RESTful Service	9-38
Request Mapping to ASP.NET Web API.....	9-39
The GET Method	9-40
The POST Method	9-41
The PUT Method	9-42
The DELETE Method.....	9-43
Testing RESTful Web Services	9-44
Testing RESTful Services with Postman.....	9-45
Testing RESTful Services with Postman <i>cont'd</i>	9-46
Module Summary	9-47
Module 10 Defining Custom Directives	10-1
Module Goals and Objectives	10-2
Section 10-1 Custom Directives.....	10-3
Custom Directives.....	10-4
Custom Directives <i>cont'd</i>	10-5
Section 10-2 Creating Custom Directives	10-6
Defining a Custom Directive.....	10-7
Configuring the DDO.....	10-8
Example: Custom <code>searchResult</code> Directive	10-9
Restricting a Directive's Use	10-10
Defining a Directive's Markup.....	10-11
Specifying Markup with <code>template</code>	10-12
Specifying Markup with <code>template</code> <i>cont'd</i>	10-13
Specifying Markup with <code>templateUrl</code>	10-14
Specifying Markup with <code>templateUrl</code> <i>cont'd</i>	10-15
Specifying Markup with <code>templateUrl</code> <i>cont'd</i>	10-16
Section 10-3 Custom Directives and Scope.....	10-17
Custom Directives and Scope	10-18
Custom Directives and Scope <i>cont'd</i>	10-19
Controlling a Directive's Scope.....	10-20
Using Shared Scope	10-21
Example: Using Shared Scope.....	10-22
Example: Using Shared Scope <i>cont'd</i>	10-23
Using Child Scope.....	10-24
Example: Using Child Scope.....	10-25
Example: Using Child Scope <i>cont'd</i>	10-26
Using Isolate Scope.....	10-27
Using Isolate Scope <i>cont'd</i>	10-28
Using Isolate Scope <i>cont'd</i>	10-29
Configuring Isolate Scope	10-30
One-Way Data Binding with <code>@</code>	10-31
One-Way Data Binding with <code>@</code> <i>cont'd</i>	10-32
One-Way Data Binding with <code>@</code> <i>cont'd</i>	10-33
Two-Way Binding with <code>=</code>	10-34
Two-Way Binding with <code>=</code> <i>cont'd</i>	10-35
Passing a Function to the Page with <code>&</code>	10-36

Passing a Function to the Page with & <i>cont'd</i>	10-37
Passing a Function to the Page with & <i>cont'd</i>	10-38
Repeated Directives	10-39
Repeated Directives <i>cont'd</i>	10-40
Repeated Directives <i>cont'd</i>	10-41
Section 10-4 Advanced Topics	10-42
compile and link Properties	10-43
Using the compile Property	10-44
Using the compile Property <i>cont'd</i>	10-45
Using the compile Property <i>cont'd</i>	10-46
Using the compile Property <i>cont'd</i>	10-47
Using the compile Property <i>cont'd</i>	10-48
Pre- and Post-link Functions	10-49
Pre- and Post-link Functions <i>cont'd</i>	10-50
Using the link Property	10-51
Overview of Transclusion	10-52
Overview of Transclusion <i>cont'd</i>	10-53
Using Transclusion	10-54
Using Transclusion <i>cont'd</i>	10-55
Module Summary	10-56
Module 11 Defining Custom Filters	11-1
Module Goals and Objectives	11-2
Section 11-1 Custom Filters	11-3
Why Define Custom Filters	11-4
Creating Custom Filters	11-5
Example: Creating a Custom Filter	11-6
Example: Creating a Custom Filter <i>cont'd</i>	11-7
Chaining Filters	11-8
Injecting a Filter	11-9
Types of Custom Filters	11-10
Example: Creating a Static, Single Use Filter	11-11
Example: Creating a Static, Single Use Filter <i>cont'd</i>	11-12
Example: Creating a Filter for Repeats	11-13
Example: Creating a Filter for Repeats <i>cont'd</i>	11-14
Example: Creating a Filter for Repeat with Arguments	11-15
Example: Creating a Filter for Repeat with Arguments <i>cont'd</i>	11-16
Example: Creating a Filter for Repeat with Arguments <i>cont'd</i>	11-17
Using \$filterProvider	11-18
Example: Using \$filterProvider	11-19
Example: Using \$filterProvider <i>cont'd</i>	11-20
Example: Creating a Set Filter	11-21
Example: Creating a Set Filter <i>cont'd</i>	11-22
Example: Creating a Set Filter <i>cont'd</i>	11-23
Module Summary	11-24
Module 12 Integrating Forms with AngularJS	12-1
Module Goals and Objectives	12-2
Section 12-1 AngularJS Forms	12-3
Working with Forms in AngularJS	12-4
Traditional Forms vs AngularJS Forms	12-5
Section 12-2 Working with AngularJS Forms	12-6
Understanding AngularJS Forms	12-7
Creating an AngularJS Form	12-8
Using the novalidate Attribute	12-9
Binding Form Fields to \$scope Properties	12-10
Using FormController Properties to Validate Form Fields	12-11
Example: Building an AngularJS Form	12-12

Example: Building an AngularJS Form <i>cont'd</i>	12-13
Other FormController Properties	12-14
Section 12-3 Forms and Data Binding.....	12-15
Data Binding With Forms.....	12-16
Data Binding With ngModel	12-17
Data Binding With ngModel <i>cont'd</i>	12-18
Example: One-Way Data Binding.....	12-19
Example: One-Way Data Binding <i>cont'd</i>	12-20
Example: Two Way Data Binding With a Controller	12-21
Example: Two Way Data Binding With a Controller <i>cont'd</i>	12-22
Section 12-4 AngularJS Input Elements.....	12-23
AngularJS Input Elements.....	12-24
Enhanced Form Controls.....	12-25
Working with input Controls	12-26
Configuring AngularJS input Fields	12-27
Configuring AngularJS input Fields <i>cont'd</i>	12-28
Example: Using Text Fields.....	12-29
Example: Using Text Fields <i>cont'd</i>	12-30
Working with Radio Buttons	12-31
Working with Radio Buttons <i>cont'd</i>	12-32
Working with Radio Buttons <i>cont'd</i>	12-33
Working with Checkboxes	12-34
Example: Using Radio Buttons and Checkboxes	12-35
Example: Using Radio Buttons and Checkboxes <i>cont'd</i>	12-36
Using ngChecked with Checkboxes.....	12-37
Working with Textareas	12-38
Working with Select Lists.....	12-39
Working with Select Lists <i>cont'd</i>	12-40
Dynamically Generating List Options.....	12-41
Understanding the ngOptions Directive	12-42
Sample ngOptions Expressions.....	12-43
Sample ngOptions Expressions <i>cont'd</i>	12-44
Sample ngOptions Expressions <i>cont'd</i>	12-45
Sample ngOptions Expressions <i>cont'd</i>	12-46
Section 12-5 Submitting Forms with AngularJS.....	12-47
Submitting Forms with AngularJS	12-48
Example: Submitting a Form to a Server	12-49
Example: Calling an AngularJS Method on Form Submission.....	12-50
Module Summary	12-51
Module 13 Form Validation With AngularJS.....	13-1
Module Goals and Objectives	13-2
Section 13-1 Form Validation	13-3
Form Validation.....	13-4
Section 13-2 (Self-Study) Validating Data with HTML5.....	13-5
Validating Data with HTML5	13-6
HTML5 Form Validation.....	13-7
Disabling Validation.....	13-8
pattern Attribute	13-9
pattern Attribute <i>cont'd</i>	13-10
required Attribute	13-11
CSS for Validation.....	13-12
Example: Validation Pseudo-Class	13-13
Example: HTML5 Form Validation	13-14
Example: HTML5 Form Validation	13-15
HTML5 Form Validation Results.....	13-16
HTML5 Form Validation Results <i>cont'd</i>	13-17

Section 13–3 Form Validation with AngularJS	13-18
AngularJS Form Validation.....	13-19
Setting Validation Rules on Controls	13-20
Example: Setting AngularJS Form Validation Directives.....	13-21
Using ngPattern	13-22
Example: Using ngPattern.....	13-23
AngularJS Form Validation State Properties.....	13-24
Form Control Validation Properties and Classes	13-25
Using ngShow To Display Error Messages.....	13-26
Example: Using \$invalid And ngShow to Display Error Messages.....	13-27
Example: Using \$invalid And ngShow to Display Error Messages <i>cont'd</i>	13-28
Enhancing Error Validation Checks and Messages with \$touched.....	13-29
Example: Using \$invalid and \$touched To Display Error Messages	13-30
Example: Using \$invalid And \$touched To Display Error Messages <i>cont'd</i>	13-31
Validating HTML5 Type Fields	13-32
Example: Implementing HTML5 Style Validation with AngularJS	13-33
Example: Implementing HTML5 Style Validation With AngularJS <i>cont'd</i>	13-34
Dynamically Styling Forms Based On Validation State	13-35
Example: Dynamically Styling Forms Based On Validation State.....	13-36
Example: Dynamically Styling Forms Based On Validation State <i>cont'd</i>	13-37
The \$error Property.....	13-38
The \$error Property <i>cont'd</i>	13-39
Example: Working with \$error.....	13-40
Example: Working with \$error <i>cont'd</i>	13-41
Section 13–4 Using ngMessages	13-42
What is ngMessages ?.....	13-43
ngMessages Directives	13-44
Loading ngMessages.....	13-45
Defining Error Messages	13-46
Example: Using ngMessages	13-47
Example: Using ngMessages <i>cont'd</i>	13-48
Using ngMessagesInclude	13-49
Using ngMessagesInclude <i>cont'd</i>	13-50
Example: Using ngMessagesInclude	13-51
Example: Using ngMessagesInclude <i>cont'd</i>	13-52
Module Summary	13-53
Module 14 Creating Animations in AngularJS	14-1
Module Goals and Objectives	14-2
Section 14–1 Overview of Animation.....	14-3
Overview of Animation.....	14-4
Section 14–2 Adding Animations to AngularJS Applications	14-5
Overview of Animations in AngularJS.....	14-6
Referencing ngAnimate	14-7
Using Animation with Directives	14-8
CSS Naming Conventions.....	14-9
Section 14–3 Creating Animations with CSS3 Transitions.....	14-10
Overview of CSS3 Transitions	14-11
Defining CSS3 Transitions.....	14-12
Example: Adding Animations with Transitions.....	14-13
Example: Animating a ToDo List.....	14-14
Example: Animating a ToDo List <i>cont'd</i>	14-15
Example: Creating a Select Menu	14-16
Example: Creating a Select Menu <i>cont'd</i>	14-17
Example: Creating a Select Menu <i>cont'd</i>	14-18
Section 14–4 Creating Animations with CSS3 Animations.....	14-19
Overview of CSS3 Animations.....	14-20
Defining CSS3 Animations.....	14-21

Example: Adding Animations	14-22
Example: Adding Animations <i>cont'd</i>	14-23
Example: Animating an Accordion	14-24
Example: Animating an Accordion <i>cont'd</i>	14-25
Example: Animating an Accordion <i>cont'd</i>	14-26
Example: Transitioning Between Pages Using Animations.....	14-27
Example: Transitioning Between Pages Using Animations.....	14-28
Example: Transitioning Between Pages Using Animations.....	14-29
Section 14-5 Using the <code>Animate.css</code> Library	14-30
Overview of <code>Animate.css</code>	14-31
Overview of <code>Animate.css</code> <i>cont'd</i>	14-32
Example: Adding Animations with <code>Animate.css</code>	14-33
Example: Adding Animations with <code>Animate.css</code>	14-34
Section 14-6 Animating Custom Directives.....	14-35
Adding Animations to Custom Directives.....	14-36
Using the <code>enter()</code> Method	14-37
Using the <code>leave()</code> Method	14-38
Example: Adding Animations to a Custom Directive.....	14-39
Example: Adding Animations to a Custom Directive.....	14-40
Using the <code>move()</code> Method.....	14-41
Using the <code>addClass()</code> Method.....	14-42
Using the <code>removeClass()</code> Method.....	14-43
Module Summary	14-44
Module 15 Using UI Bootstrap.....	15-1
Module Goals and Objectives	15-2
Section 15-1 Overview of Angular UI.....	15-3
What is AngularUI?	15-4
Section 15-2 Using UI Bootstrap.....	15-5
Overview of UI Bootstrap.....	15-6
Downloading UI Bootstrap	15-7
Downloading UI Bootstrap <i>cont'd</i>	15-8
Creating a Custom Build	15-9
Referencing UI Bootstrap in a Page	15-10
Section 15-3 Using the Accordion Directive.....	15-11
Overview of the Accordion Directive.....	15-12
Creating an Accordion.....	15-13
Creating an Accordion <i>cont'd</i>	15-14
Creating an Accordion <i>cont'd</i>	15-15
Example: Creating a Basic Accordion.....	15-16
Example: Creating a Basic Accordion <i>cont'd</i>	15-17
Example: Creating a Dynamic Accordion.....	15-18
Example: Creating a Dynamic Accordion <i>cont'd</i>	15-19
Section 15-4 Using the Carousel Directive	15-20
Overview of the Carousel Directive	15-21
Creating a Carousel	15-22
Creating a Carousel <i>cont'd</i>	15-23
Example: Creating a Carousel.....	15-24
Example: Creating a Carousel <i>cont'd</i>	15-25
Example: Creating a Dynamic Carousel.....	15-26
Example: Creating a Dynamic Carousel <i>cont'd</i>	15-27
Section 15-5 Using the Tabs Directive	15-28
Overview of the Tabs Directive	15-29
Creating Tabs	15-30
Creating Tabs <i>cont'd</i>	15-31
Example: Creating Tabs.....	15-32
Example: Creating Dynamic Tabs	15-33

Example: Creating Dynamic Tabs <i>cont'd</i>	15-34
Module Summary	15-35
Module 16 Additional Angular UI Directives.....	16-1
Module Goals and Objectives	16-2
Section 16-1 Using the UI Calendar Directive	16-3
Overview of the UI Calendar Directive	16-4
FullCalendar Dependencies	16-5
Creating a Calendar	16-6
Calendar Event Source Object.....	16-7
Example: Creating a Basic Calendar	16-8
Example: Customizing a Calendar with Time	16-9
Example: Customizing a Calendar with Time <i>cont'd</i>	16-10
Section 16-2 Using the UI Chart Directive	16-11
Overview of the UI Chart Directive	16-12
jqPlot Dependencies	16-13
Using the UI Chart Directive	16-14
Configuring jqPlot	16-15
Configuring jqPlot <i>cont'd</i>	16-16
Example: Creating a Pie Chart	16-17
Example: Creating a Pie Chart <i>cont'd</i>	16-18
Section 16-3 Using the UI Leaflet Directive.....	16-19
Overview of the UI Leaflet Directive.....	16-20
Leaflet Dependencies.....	16-21
Leaflet Dependencies <i>cont'd</i>	16-22
Referencing UI Leaflet.....	16-23
Leaflet Options.....	16-24
Example: Creating a Basic Map	16-25
Example: Creating a Basic Map <i>cont'd</i>	16-26
Module Summary	16-27
Module 17 Working with AngularStrap	17-1
Module Goals and Objectives	17-2
Section 17-1 Overview of AngularStrap.....	17-3
Overview of AngularStrap.....	17-4
Dependencies	17-5
Downloading AngularStrap.....	17-6
Referencing AngularStrap.....	17-7
Section 17-2 Using the Modal Module.....	17-8
Overview of Modal.....	17-9
Overview of Modal <i>cont'd</i>	17-10
Loading Individual Modules	17-11
Modal Options	17-12
Modal Options <i>cont'd</i>	17-13
Modal Methods.....	17-14
Example: Creating a Modal.....	17-15
Example: Creating a Modal <i>cont'd</i>	17-16
Example: Creating a Modal Using a Template	17-17
Section 17-3 Using the Tooltip Module	17-18
Overview of Tooltip.....	17-19
Loading Individual Modules	17-20
Tooltip Options	17-21
Tooltip Options <i>cont'd</i>	17-22
Tooltip Methods.....	17-23
Example: Creating a Tooltip.....	17-24
Example: Creating a Tooltip <i>cont'd</i>	17-25
Section 17-4 Using the Typeahead Module.....	17-26
Overview of Typeahead	17-27

Overview of Typeahead <i>cont'd</i>	17-28
Loading Individual Modules	17-29
Typeahead Options.....	17-30
Typeahead Options <i>cont'd</i>	17-31
Example: Creating a Typeahead	17-32
Example: Creating a Typeahead <i>cont'd</i>	17-33
Section 17-5 Using the Datepicker Module.....	17-34
Overview of Datepicker	17-35
Loading Individual Modules	17-36
Datepicker Options	17-37
Datepicker Options <i>cont'd</i>	17-38
Example: Creating a Datepicker	17-39
Example: Creating a Datepicker <i>cont'd</i>	17-40
Section 17-6 Using the Dropdown Module	17-41
Overview of Dropdown	17-42
Overview of Dropdown <i>cont'd</i>	17-43
Loading Individual Modules	17-44
Dropdown Options	17-45
Dropdown Options <i>cont'd</i>	17-46
Example: Creating a Dropdown	17-47
Example: Creating a Dropdown <i>cont'd</i>	17-48
Module Summary	17-49
Appendix A Working with jqLite	A-1
jqLite Methods	A-2
jQuery Methods in jqLite.....	A-3
jQuery Methods in jqLite <i>cont'd</i>	A-4
<code>angular.element()</code>	A-5
DOM Manipulation with jqLite	A-6
DOM Manipulation with jqLite <i>cont'd</i>	A-7
Appendix B Bibliography	B-1
Bibliography.....	B-2

Do Not Duplicate

Developing Web Applications Using AngularJS

Course Introduction

Do Not Duplicate

Course Goals and Objectives

Major Course Goals

Understand the role of AngularJS in web development

Understand AngularJS data-binding and the `$digest` cycle

Understand AngularJS application architectures and how they use the MVC and MV* design patterns

Understand the role of controllers, directives and services in AngularJS

Understand how to build Single Page Applications with AngularJS

Understand how to communicate with remote services using AngularJS services

Understand how to use third-party directives in AngularJS applications

Understand how AngularJS applications can be animated

Specific Objectives

On completion of this course, students will be able to:

Bootstrap an AngularJS application

Use AngularJS directives in a web page

Use data-binding to display values defined on the scope

Create controllers and custom services to supply data to a view

Use `$scope` to link controllers and views together

Use AngularJS filters to format bound data

Create single page applications using routing

Make asynchronous requests to remote services using `$http` and `$resource` services

Display error messages using AngularJS form validation features and `ngMessages`

Provide animations using `ngAnimate`

Do Not Duplicate

Module 1

Introduction to AngularJS

Do Not Duplicate

Module Goals and Objectives

Major Goals

Understand the role of AngularJS

Understand the AngularJS architecture

Understand the architecture of a Single Page Application

Specific Objectives

On completion of this module, students will be able to:

Describe the role of AngularJS

Describe the AngularJS architecture

Describe the architecture of a Single Page Application

Download AngularJS and include it in a Web Site

Do Not Duplicate

Section 1–1

AngularJS

Do Not Duplicate

AngularJS

- **AngularJS is an open source, JavaScript-based web application development framework**
 - Originally developed in 2009 by Misko Hevery and Adam Abrons
 - Now maintained by Google
- **Angular's official documentation states:**

"AngularJS is a structural framework for dynamic web applications. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application components cleanly and succinctly. Its data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology."
- **AngularJS allows you to build cross-browser, client-side applications that follow the principles of the MVC design pattern**
 - AngularJS does not implement MVC in the traditional sense
 - AngularJS is closer to the MVVM (Model-View-ViewModel) pattern
 - It is often referred to as MV* (Model-View-Whatever)

Do Not Duplicate

AngularJS *cont'd*

- **AngularJS attempts to minimize the amount of code needed to create interactive, web applications by adding new capabilities to HTML**
 - Done using special constructs called *directives*
- **These directives extend the HTML syntax by providing new behaviors which include:**
 - Data binding
 - DOM manipulation
 - Form validation
 - Event handling

Do Not Duplicate

AngularJS vs jQuery

- **jQuery is one of the most popular JavaScript libraries around**
 - It contains functions that make it easy for web developers to code commonly performed tasks
 - One of the major uses of jQuery is DOM manipulation
- **AngularJS is a framework**
 - Frameworks provide features that configure an application's infrastructure
 - * That is, they specify exactly how applications should be coded
 - AngularJS also provides a library of functions that AngularJS developers can use to perform AngularJS-specific tasks

Do Not Duplicate

Using jQuery with AngularJS

- According to AngularJS documentation, AngularJS can use jQuery if it is present in your application when the application is being bootstrapped
 - AngularJS 1.3+ supports only jQuery 2.1 and above
 - * jQuery 1.7 and newer may work correctly, but it is not guaranteed

Example

```
<script src="scripts/jquery-2.1.0.min.js"></script>
<script src="scripts/angular.min.js"></script>
```

- If jQuery is not present when AngularJS is loaded, it uses its own implementation of jQuery called jqLite
- jqLite is a stripped-down version of jQuery that is built directly into AngularJS
 - Its purpose is to provide several features of jQuery while keeping it constrained within Angular's separation of responsibilities paradigm
 - In an attempt to keep the size of AngularJS small, AngularJS implements only a subset of selectors in jqLite
- Refer to the appendices for a brief jqLite discussion

Do Not Duplicate

Versions of Angular JS

- **There are now two distinct versions of Angular JS**
- **Angular JS 1.x was introduced in 2008**
 - The official website and documentation is `docs.angularjs.org`
- **Angular 2/4/5/6 (now called Angular) introduced major changes to syntax and will not work with Angular 1.x**
 - The official website and documentation is `angular.io`
- **This course will discuss AngularJS 1.x**
 - If you need training on Angular, HOTT offers a separate course called Developing Web Applications using Angular

Do Not Duplicate

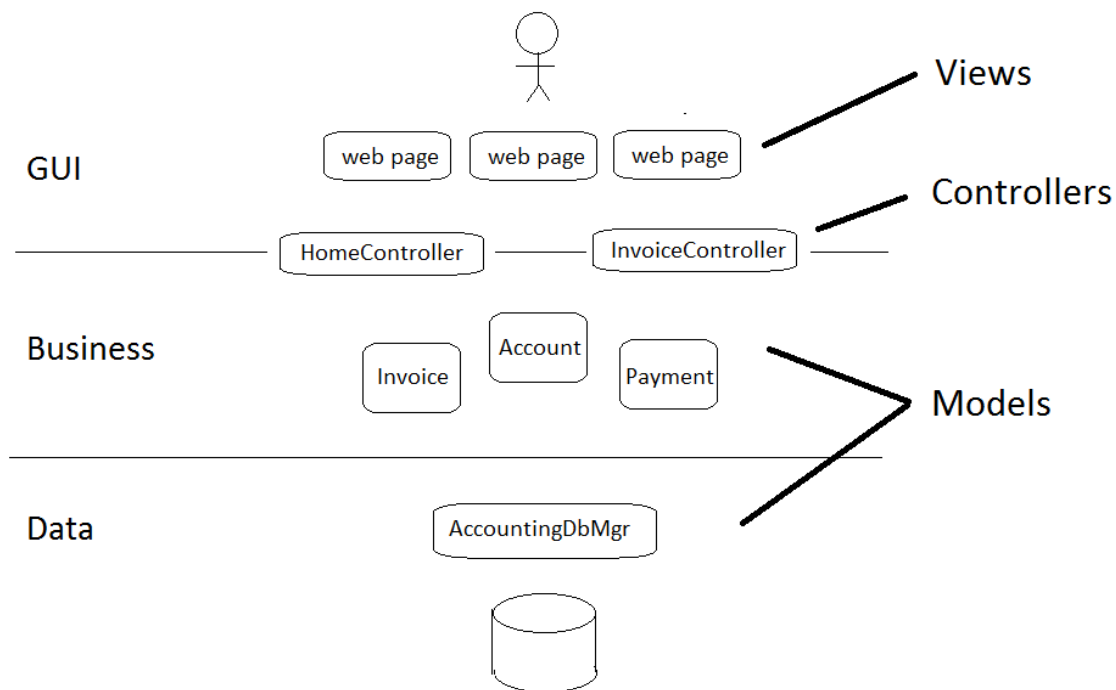
Section 1–2

MVC vs MV*

Do Not Duplicate

Model-View-Controller

- MVC is a design pattern used in desktop and web applications that enforces "separation of concerns"
 - That is, the presentation layer is separated from the management of the data



- The *model* classes are responsible for managing the application's data
- The *view* classes are responsible for generating the UI presented to the user
- The *controller* classes process the HTTP requests from the user and sequence the movement of data between the model(s) and the view(s)

Do Not Duplicate

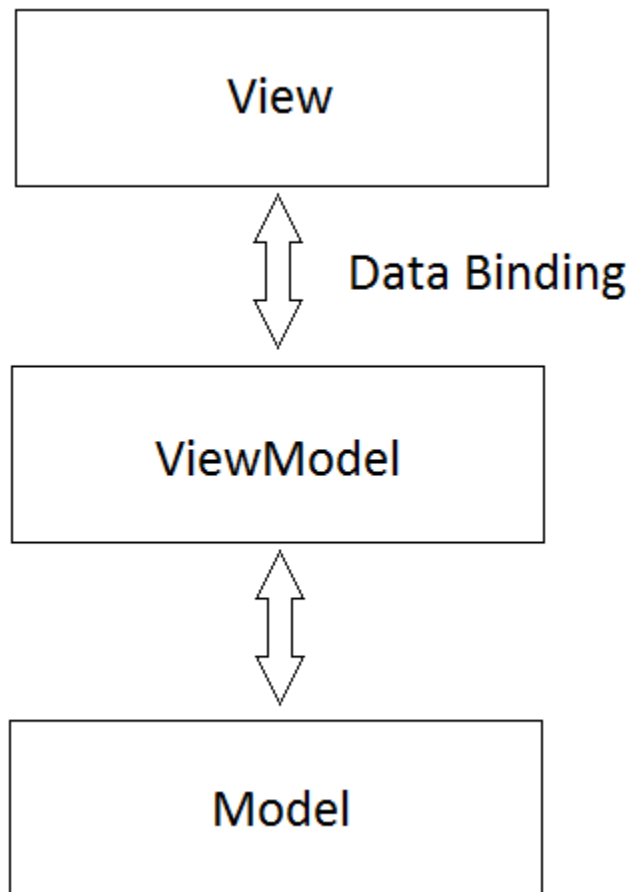
Model-View-Controller and AngularJS

- **Many MVC applications use model classes that run on the web server**
 - They have access to server-side components such as a database access layer or a remote service
- **AngularJS is a client-side technology and cannot directly access the server-side component**
 - As such, it must use Ajax to access server-side resources

Do Not Duplicate

Model-View-ViewModel

- MVVM is a design pattern popular in languages that support data-binding
- The model and the view are similar to that used in MVC
- The VM stands for ViewModel and represents an object that views bind to manage the movement of data



Do Not Duplicate

MVVM and AngularJS

- AngularJS uses the MVVM design pattern in one of its most useful features: data binding
- AngularJS developers can build data models that are bound to DOM elements
- The AngularJS infrastructure will manage the movement of data between ViewModel properties and the DOM elements to which they are bound

Example

If the ViewModel contains `name` and `age` properties, they could be bound to DOM elements in AngularJS as shown below.

```
<p>{{ name }} is {{ age }}</p>
```

Do Not Duplicate

MV* (Model-View-Whatever)

- The combination of the MVC and MVVM patterns is sometimes referred to as MV* (pronounced MV star)
 - Also called Model-View-Whatever
 - The "Whatever" stands for "whatever works for you"
- MV* refers to the blended approach of these different design patterns
- AngularJS uses the MV* design pattern, which gives you a lot of flexibility for separating the presentation logic from the business logic

Do Not Duplicate

Section 1–3

Working with AngularJS

Do Not Duplicate

Key Concepts in AngularJS

- When you build an application using AngularJS, you will typically use some or all of the following:

Concept	Description
Model	Data shown to the user in the view and with which the user interacts
View	The HTML page
Controller	Business logic behind a view
Scope	Context where the model is stored so that controllers, directives and data-binding expressions can access it
Data Binding	Syncs data between the model and the view
Service	Reusable business logic
Module	Container for different parts of an application including controllers, services and directives
Directive	Extends HTML with custom elements and attributes
Dependency Injection	Creates and wires objects and functions

Do Not Duplicate

AngularJS Directives

- **AngularJS extends HTML using *directives* that are typically applied using attributes**
 - However, some directives are implemented as elements, class names and comments
- **AngularJS directives are named with an `ng-` prefix**
 - In HTML, "words" in the directive are separated with hyphens
 - * For example, `ngApp` in JavaScript becomes `ng-app`
 - In JavaScript, the directives use camelCase notation
- **Some of the most common directives include:**

Directive	Description
<code>ngApp</code>	Used to define an AngularJS application. Should be placed on the root element that is available to the application
<code>ngModel</code>	Used to bind the value of an HTML control (input, select or textarea) to a property in the model
<code>ngController</code>	Used to attach a controller class to a view
<code>ngView</code>	Defines the area on the HTML page where a template should be rendered when routing is used
<code>ngClass</code>	Allows you to dynamically set CSS classes on an element using an expression

Do Not Duplicate

Modules

- **An AngularJS module defines the AngularJS application**
 - It is registered using the `angular.module()` method and provides a name and an empty array as arguments
 - * The array can list dependencies for the application

Example

```
var app = angular.module('myApp', []);
```

- **The AngularJS application is then associated with the HTML page using the ngApp directive**
 - This allows AngularJS directives to be used on any elements within the root element
 - Referred to as "bootstrapping" the application

Example

```
<body ng-app="myApp">  
...  
</body>
```

- **Modules are used to serve as containers for other AngularJS components**
 - You can register controllers, directives, filters and services with the `Module` object

Do Not Duplicate

Controllers

- **Controllers encapsulate the business logic for the views of an AngularJS application**
 - They are registered in a specific AngularJS application with the `controller()` method
- **The `controller()` method defines two parameters:**
 - The name of the controller
 - A reference to a function that defines properties accessible through a `$scope` object
 - * The `$scope` object is used to connect the model (or data) to the view (HTML page)

Example

One common convention is to define a variable to store a reference to the module after it has been defined. The variable is then used to reference the module.

```
var app = angular.module("myApp", []);

app.controller("simpleController", function($scope) {
    $scope.name = "Devan Marie";
    $scope.age = 1;
    $scope.toys = ["Minnie", "Hat", "Horse", "Bear"];
});
```

- **In AngularJS, the `$scope` object represents the data model**

Do Not Duplicate

Controllers *cont'd*

- **Variables and functions can be defined inside a controller as properties and methods of `$scope`**
 - They "live" within the scope of the controller
 - Each controller has its own `$scope` object
- **Controllers are associated with an HTML element using the `ngController` directive**
 - Within the element, you can access properties defined within the controller

Example

```
<body ng-app="myApp">  
  <div ng-controller="simpleController">  
    ...  
  </div>  
</body>
```

Do Not Duplicate

Binding Data using Expressions

- **AngularJS expressions are used to bind data to HTML elements**
- **They are written within double curly braces**
 - Can contain literals, operators, variables and functions
 - Conditionals, loops and exceptions are not permitted

Syntax

```
{{ expression }}
```

- **AngularJS expressions are interpolated and output the result of the expression where the expression is written**

Example

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

  <div ng-controller="simpleController">
    <p>{{ name }} is {{ age }}</p>
  </div>

</body>
```

- **When the data is in a collection, you must loop through the collection to retrieve individual items**
 - AngularJS has a directive called `ngRepeat` that simplifies looping
 - The element containing the directive is repeated for each item in a collection (like the JavaScript `for/in` loop)

Binding Data using Expressions *cont'd*

Example

The following example uses the `ngRepeat` directive to render a list item for each of the items in the specified collection.

```
<div ng-controller="simpleController">
  <h3>Toys</h3>
  <ul>
    <li ng-repeat="t in toys">{{ t }}</li>
  </ul>
</div>
```

- **AngularJS provides "live binding" to the expressions**
 - If the value of the expression changes, AngularJS updates the value automatically
 - Referred to as two-way data binding

Do Not Duplicate

Services

- **An AngularJS service is a singleton object that is responsible for carrying out some task**
 - For example, if you wanted to submit an Ajax request, you could use the `$http` service
- **Services allow us to keep communication logic out of controllers**
 - They can be used in controllers and other components by declaring them as dependencies
 - * Known as dependency injection
- **With a service, the first time the service runs, an instance is created and remains in memory**
 - With controllers, you get a new controller instance each time a view loads that controller
- **AngularJS includes several built-in factories and services**
 - Typically they are just called services

Service	Description
<code>\$http</code>	used to make Ajax calls
<code>\$window</code>	provides access to the JavaScript window object
<code>\$location</code>	provides access to the JavaScript location object
<code>\$timeout</code>	used in a similar way to <code>window.setTimeout()</code>
<code>\$interval</code>	used in a similar way to the <code>window.setInterval()</code> method
<code>\$q</code>	used with asynchronous processes, such as Ajax calls
<code>\$rootScope</code>	used behind the scenes to create new scope objects
<code>\$filter</code>	used to programmatically access custom filters
<code>\$log</code>	used for general logging purposes

Do Not Duplicate

Dependency Injection

- **Dependency Injection is a software design pattern that deals with how components are created and how they acquire their dependencies**
 - A dependency is an object (service)
 - The injection is the passing of a dependency to an object (client) that would use it
 - * The service is made part of the client's state
 - * Passing the service to the client, rather than allowing the client to build or find the service, is the fundamental requirement of the pattern
- **Dependency injection separates the creation of a client's dependencies from the client's behavior**
- **Dependency Injection is used throughout AngularJS**
 - When you follow the AngularJS best practice of separating your code into different components, there may be times when components must interact with one another
 - That's where dependency injection comes in
- **In AngularJS, dependency injection is used to provide a particular component with some other component(s) that it requires**

Do Not Duplicate

Dependency Injection *cont'd*

- For example, imagine that you are writing a controller which will obtain its data from a remote server using Ajax
 - In AngularJS, there is a `$http` service which is used to perform Ajax requests
 - But to use it in a controller, it must be injected

Example

```
var app = angular.module('demoApp', []);

app.controller('EventController', ['$scope', '$http',
    function($scope, $http) {

    // This uses the $http service to submit an HTTP request

    }]);
```

Do Not Duplicate

Single Page Applications

- **Single Page Applications (or SPAs) allow content (known as views) to be loaded into a container as a result of user interaction**
 - Typically a container (such as a `div`) is defined on a page
 - When the user interacts with the page (by clicking hyperlinks), the contents of the container are updated (instead of bringing the user to a new page)
 - * The entire page does not have to be reloaded
- **AngularJS makes it easy to create single page applications**
 - New content (views) are loaded into a shell page using routing
- **Routing connects a view with a controller**
 - When the page is first loaded, a default view can be specified
 - Other views are loaded dynamically as the user clicks on links

Do Not Duplicate

Adding Animations

- **Animations can be added to an AngularJS application to enhance the application**
 - Not required, but users may better enjoy their user experiences
 - * Can provide the user with a better idea that something has happened
- **AngularJS allows you to use animations using CSS3 or jQuery**
- **AngularJS uses directives to specify animations**

Do Not Duplicate

Section 1–4

Using AngularJS in an Application

Do Not Duplicate

Referencing AngularJS

- To use AngularJS, you reference a script file
 - Named `angular.js`
- AngularJS is available for download from the AngularJS website (`https://angularjs.org`)
 - There are multiple versions of AngularJS available
 - * Even-numbered versions are the stable releases, while the odd-numbered versions are intended for developers only
 - You can use a minified or uncompressed version of the library
 - * Minified versions are typically recommended for production and the uncompressed version in development

Example

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

- You can also reference the framework using a CDN

Example

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
```

Do Not Duplicate

Referencing AngularJS *cont'd*

- **Although the AngularJS script can be referenced in either the head or the body of the page, it is typically referenced at the bottom of the HTML code prior to the `</body>`**
 - The official documentation recommends placing the code at the bottom of the page to improve application load time
- **In addition to the core AngularJS file, there are additional modules available**
 - To avoid errors, make sure that you use the same AngularJS version for the additional modules to avoid an error
 - For example, if you were using version 1.4.8 of the core `angular.js` file, you should use version 1.4.8 of the `angular-animate.js`

Do Not Duplicate

Module Summary

- **AngularJS is an open source, JavaScript-based web application development framework**
 - Originally developed in 2009 by Misko Hevery and Adam Abrons
 - Now maintained by Google
- **AngularJS allows you to build cross-browser, client-side applications that follow the principles of the MVC design pattern**
 - AngularJS does not implement MVC in the traditional sense
 - AngularJS is closer to the MVVM (Model-View-ViewModel) pattern
 - It is often referred to as MV* (Model-View-Whatever)
- **AngularJS attempts to minimize the amount of code needed to create interactive, web applications by adding new capabilities to HTML**
 - Done using special constructs called *directives*
- **AngularJS uses several other components including modules, controllers and services**

Do Not Duplicate

Do Not Duplicate

Module 2

Data Binding

Do Not Duplicate

Module Goals and Objectives

Major Goals

Understand several techniques that can be used to define AngularJS application data

Understand how AngularJS uses data binding in views

Specific Objectives

On completion of this module, students will be able to:

Use the `ngInit` directive to define application data

Use the `ngBind` directive to display data using data binding

Use AngularJS expressions to display data using data binding

Use the `ngModel` directive to define application data

Display sets of data using the `ngRepeat` directive

Build simple AngularJS controllers defining application data and reference them in a view

Use data binding to display data from `$scope`

Use `$watch` to add watch behavior to the `$digest` cycle

Do Not Duplicate

Section 2–1

AngularJS Data Binding

Do Not Duplicate

AngularJS and Data Binding

- One of the strengths of AngularJS is its data binding features
- It allows the programmer to associate an HTML element with a data member from the application's data model
 - The bound HTML element will then "automatically" stay in sync with the data model
- AngularJS data binding uses *two-way data binding*
 - This means that not only will the HTML element stay in sync with the data model, but the data model will stay in sync with the HTML element
- With AngularJS, when the data model changes, the view changes and when the view changes, the data model changes!

Do Not Duplicate

Defining the Application

- **The ngApp directive defines the AngularJS application**
 - It is used to auto-bootstrap the AngularJS application
- **It is typically placed near the root element of the page**
 - For example, on the <html>, <body> or a container <div> tag

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>AngularJS Data Binding</title>
  <meta charset="utf-8" />
</head>
<body ng-app>

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

</body>
</html>
```

- **Although there are other ways to bootstrap an application, using ngApp is the most common way**

Do Not Duplicate

Defining Application Variables

- The `ngInit` directive defines and initializes variables that can be used within the AngularJS application
 - It uses a *key = value* syntax

Example

```
<!DOCTYPE html>
<html>
<head>  <title>AngularJS Data Binding</title>
        <meta charset="utf-8" />
</head>
<body ng-app>

    <div ng-init="courseTitle='Learning AngularJS'">

        <!-- you can access the variable courseTitle within here -->

    </div>

    <script src="scripts/angular.min.js"></script>
</body>
</html>
```

- Once we learn to use controllers, our data models will be more sophisticated and we won't use `ngInit`

Do Not Duplicate

Data Binding using ngBind

- The **ngBind** directive binds AngularJS application data to HTML

Example

In this example, we use **ngBind** to display the value of `courseTitle` in a ``

simplebinding1.html

```
<!DOCTYPE html>
<html>
<head>
  <title>AngularJS Data Binding</title>
  <meta charset="utf-8" />
</head>
<body ng-app>

  <div ng-init="courseTitle='Learning AngularJS'">

    <p>Welcome to the &quot;<span ng-bind="courseTitle"></span>
      &quot; course!</p>

  </div>

  <script src="scripts/angular.min.js"></script>
</body>
</html>
```

- Although **ngBind** can be used to bind our AngularJS data, it is more common to use AngularJS expressions

Do Not Duplicate

AngularJS Expressions

- **AngularJS expressions are a very easy way to display data**
 - They are written inside of double braces

Syntax

```
{{ expression }}
```

- **This syntax can be used to display the value of JavaScript expressions or AngularJS application data**

Example

This example displays the results of evaluating a JavaScript expression

[simplebinding2.html](#)

```
<p>1 plus 1 is {{ 1 + 1 }}</p>
```

Example

This example displays the value of an AngularJS application variable

[simplebinding3.html](#)

```
<div ng-init="courseTitle='Learning AngularJS'">  
  <p>Welcome to the &quot;{{ courseTitle }}&quot; course!</p>  
</div>
```

Do Not Duplicate

Displaying Collections using ngRepeat

- When applications store data in collections (ex: invoices, products, orders), you iterate through the collection to display the items
 - Rather than using the JavaScript `for/in` statement, AngularJS programmers use the `ngRepeat` directive

Syntax

`ng-repeat="item in collection"`

- **ngRepeat** creates a display template for each item in the collection

Example

The `ngInit` directive defines a `teams` property which references an array of strings. The `ngRepeat` directive repeats the `` tag for each item in the collection.

repeatexample1.html

```
<h3>NBA Atlantic Division Teams</h3>
<div
  ng-init="teams=['Celtics','Knicks','Nets','Raptors','Sixers']">
  <ul>
    <li ng-repeat="team in teams">
      {{ team }}
    </li>
  </ul>
</div>
```

NBA Atlantic Division Teams

- Celtics
- Knicks
- Nets
- Raptors
- Sixers

Do Not Duplicate

Properties Exposed via ngRepeat

- The **ngRepeat** directive exposes several properties on the current object in the collection, including:

Property	Data Type	Description
<code>\$index</code>	number	Is the index number of the current item in the collection
<code>\$first</code>	boolean	Returns <code>true</code> if the item is the first in the collection
<code>\$last</code>	boolean	Returns <code>true</code> if the item is the last in the collection
<code>\$middle</code>	boolean	Returns <code>true</code> if the item is between the first and last item in the collection
<code>\$even</code>	boolean	Returns <code>true</code> if <code>\$index</code> is even
<code>\$odd</code>	boolean	Returns <code>true</code> if <code>\$index</code> is false

Example

In this example, we use a JavaScript array of objects as our application data. `$index` is used to display the index of an item.

repeatexample2.html

```
<body ng-app>

  <div ng-init="employees = [
    { name:'Devan Marie',
      title:'Transportation Coordinator',
      pay:42000 },
    { name:'Zephaniah Hugh',
      title:'Demolition Expert',
      pay:39000 },
    { name:'Pursalane Faye',
      title:'Marketing Specialist',
      pay:42000}]">

    <h2>There are {{ employees.length }} employees:</h2>
    <ul>
      <li ng-repeat="emp in employees">
        Employee #{{ $index + 1 }} - {{ emp.name }}
          ({{ emp.title }} - Salary: {{ emp.pay }})
      </li>
    </ul>
  </div>

  <script src="scripts/angular.min.js"></script></body>
</body>
```

Do Not Duplicate

Properties Exposed via ngRepeat *cont'd*

There are 3 employees:

- Employee #1 - Devan Marie (Transportation Coordinator - Salary: 42000)
- Employee #2 - Zephaniah Hugh (Demolition Expert - Salary: 39000)
- Employee #3 - Pursalane Faye (Marketing Specialist - Salary: 42000)

Do Not Duplicate

Defining Properties using ngModel

- The **ngModel** directive is used to define a property in the AngularJS application model that is bound to a form field

Example

In this example, the `name` property is bound to a textbox. It can also be accessed using data binding expressions.

livebinding.html

```
<body ng-app>

  <p>Enter your name:
    <input type="text" ng-model="name"></p>

  <p>Hello {{ name }}</p>

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

</body>
```

- AngularJS uses two-way data binding, so your HTML automatically updates to reflect your data

Enter your name:

Hello

Enter your name:

Hello I

Enter your name:

Hello Ia

Enter your name:

Hello Ian

Section 2–2

Working with Model Data

Do Not Duplicate

Working with Controllers

- When you want to work with data of any size, you won't use `ngInit` or `ngModel` to create your variables
- Instead, you will want to use the MV* features of AngularJS
- Most AngularJS applications are made of:
 - `model(s)`
 - * A model represents the data being manipulated
 - `controller(s)`
 - * A controller is a JavaScript routine that defines the model
 - `view(s)`
 - * A view is described by writing HTML
 - * Views use data binding to display model data and to move user-entered values into the model

Do Not Duplicate

Creating a Module

- **Before we can define controllers, we must define an AngularJS module**
- **An AngularJS module defines the application and is a container for the different elements of your application**
 - It is registered using the `angular.module()` method and expects a name and an array of dependencies as arguments
 - * Right now, our array will be empty but a future module will address dependencies

Example

```
var app = angular.module('myApp', []);
```

- **The AngularJS application is then associated with an HTML page, or view, using the `ngApp` attribute**
 - This allows AngularJS directives to be used on any elements within the application
 - This is also referred to as "bootstrapping" the application

Example

```
<body ng-app="myApp">  
  
</body>
```

- **You can then register controllers with the module object**

Do Not Duplicate

Defining a Controller

- Controllers manage the data for an AngularJS application
- Controllers are registered using the module's `controller()` method by passing two arguments
 - The name of the controller
 - A reference to a function that defines the application's properties (data) and methods (functions that operate on the data)

Example

```
var app = angular.module('myApp', []);

app.controller('controllerName', function($scope) {
    // define model data and methods here
});
```

Do Not Duplicate

Using \$scope

- **\$scope** is an AngularJS object that references the model's properties and methods
 - \$scope is available in both the controller and view
 - However, the controller and view cannot see each other
- When you create a controller, its function reference is passed the \$scope object
- Within the controller, you can then associate the properties of your data model with the \$scope object

Example

```
<script>
  var app = angular.module('myApp', []);

  app.controller('CourseController', function($scope) {
    $scope.courseTitle = 'Learning AngularJS';
    $scope.instructor = 'Dr. Tech';
    $scope.location = 'Chelmsford, MA  USA';
  });
</script>
```

- You can also associate methods with the \$scope object
 - We will see this shortly

Do Not Duplicate

Associating a Controller with a View

- **Controllers are associated with views using the `ngController` directive**
 - You must also use the application name in the `ngApp` directive
- **Within the element, you can access all of the properties defined in the controller's `$scope`**

Example

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

  <div ng-controller="CourseController">

    <!-- You have access to the items in the
         controller's scope -->

  </div>

  <script src="scripts/angular.min.js"></script>
  <script>
    var app = angular.module('myApp', []);

    app.controller('CourseController', function ($scope) {
      $scope.courseTitle = 'Learning AngularJS';
      $scope.instructor = 'Dr. Tech';
      $scope.location = 'Chelmsford, MA USA';
    });
  </script>

</body>
```

- **Although you can put the JavaScript for a controller in the HTML file, it is more common to place it in a separate `.js` file**

- These files are often in a subfolder of `scripts` named `controllers`

DO NOT DUPLICATE

Example: Data Binding with a Controller

Example

scripts/controllers/CourseController.js

```
var app = angular.module('myApp', []);

app.controller('CourseController', function ($scope) {

    $scope.courseTitle = 'Learning AngularJS';
    $scope.instructor = 'Dr. Tech';
    $scope.location = 'Chelmsford, MA USA';

});
```

simplecontrollerview1.html

```
<!DOCTYPE html>
<html>
<head>
    <title>AngularJS Data Binding</title>
    <meta charset="utf-8" />
</head>
<body ng-app="myApp" >

    <div ng-controller="CourseController">

        <p>Course: {{ courseTitle }}</p>
        <p>Instructor: {{ instructor }}</p>
        <p>Location: {{ location }}</p>

    </div>

    <script src="scripts/angular.min.js"></script>
    <script src="scripts/controllers/CourseController.js"></script>
</body>
</html>
```

Course: Learning AngularJS

Instructor: Dr. Tech

Location: Chelmsford, MA USA

Do Not Duplicate

Defining Methods

- You can define methods within an AngularJS model using JavaScript functions
 - Methods are functions that operate on a model's data

Example

```
var app = angular.module('myApp', []);

app.controller('CourseController', function ($scope) {

    $scope.courseTitle = 'Learning AngularJS';
    $scope.instructor = 'Dr. Tech';
    $scope.location = 'Chelmsford, MA  USA';

    /* defines a method named setLocation */
    $scope.setLocation = function(newLocation) {
        $scope.location = newLocation;
    };

});
```

Do Not Duplicate

- You can use the `ngClick` directive to call the methods within the `$scope` when a button is clicked

```
<button ng-click="methodName(arguments)">Button Text</button>
```

- ## Example

Do Not Duplicate

Example

```
var app = angular.module('myApp', []);

app.controller('CourseController2', function ($scope) {

    $scope.courseTitle = 'Learning AngularJS';
    $scope.instructor = 'Dr. Tech';
    $scope.location = 'Chelmsford, MA    USA';

    $scope.setLocation = function (newLocation) {
        $scope.location = newLocation;
    };

});
```

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>AngularJS Data Binding</title>  
    <meta charset="utf-8" />  
  </head>  
  <body ng-app="myApp" >  
  
    <div ng-controller="CourseController2">  
  
      <p>Course: {{ courseTitle }}</p>  
      <p>Instructor: {{ instructor }}</p>  
      <p>Location: {{ location }}</p>  
      <p>  
        <button  
          ng-click="setLocation('Chelmsford, MA USA')">  
            Run class in USA</button>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
        <button ng-click="setLocation('London, England')" >  
            Run class in UK</button>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
        <button ng-click="setLocation('Toronto, Ontario')">  
            Run class in Canada</button>  
      </p>  
    </div>  
  
    <script src="scripts/angular.min.js"></script>  
    <bscript src="scripts/controllers/CourseController2.js"></bscript>  
  </body>  
</html>
```

Do Not Delete This Page

Do Not Duplicate

Example: Calling Methods using ngClick

cont'd

Course: Learning AngularJS

Instructor: Dr. Tech

Location: Chelmsford, MA USA

Run class in USA

Run class in UK

Run class in Canada

Course: Learning AngularJS

Instructor: Dr. Tech

Location: London, England

Run class in USA

Run class in UK

Run class in Canada

Course: Learning AngularJS

Instructor: Dr. Tech

Location: Toronto, Ontario

Run class in USA

Run class in UK

Run class in Canada

Do Not Duplicate

Using ngCloak to Improve Initial UI Appearance

- **When your AngularJS application is loading, the browser displays the raw, uncompiled HTML**
 - This will cause it to briefly display the HTML that contains the data binding expressions before flickering and displaying the bound data

Example

In the previous example, the browser window flashed the first image before displaying the second.

Course: {{ courseTitle }}

Instructor: {{ instructor }}

Location: {{ location }}

Run class in USA

Run class in UK

Run class in Canada

Course: Learning AngularJS

Instructor: Dr. Tech

Location: Chelmsford, MA USA

Run class in USA

Run class in UK

Run class in Canada

- **The ngCloak directive prevents the flicker effect**
 - It does this by using a CSS rule embedded within the AngularJS script file to set the `display` property to `none`

Do Not Duplicate

Using ngC1oak to Improve Initial UI Appearance *cont'd*

- When the AngularJS script compiles the template, it deletes the `ngCloak` attribute and the browser displays the element
- If you include the AngularJS script in the head tag, you can use `ngCloak` as a directive

Example

```
<!--DOCTYPE html>
<html>
<head>
  <title>AngularJS Data Binding</title>
  <meta charset="utf-8" />
  <script src="scripts/angular.min.js"></script>
</head>
<body ng-app="myApp">

  <div ng-controller="CourseController2" >

    <div ng-cloak>
      <p>Course: {{ courseTitle }}</p>
      <p>Instructor: {{ instructor }}</p>
      <p>Location: {{ location }}</p>
    </div>
    <p>
      <button ng-click="setLocation('Chelmsford, MA  USA')">
        Run class in USA</button>&nbsp;&nbsp;&nbsp;
      <button ng-click="setLocation('London, England')">
        Run class in UK</button>&nbsp;&nbsp;&nbsp;
      <button ng-click="setLocation('Toronto, Ontario')">
        Run class in Canada</button>
    </p>
  </div>

  <script src="scripts/controllers/CourseController2.js"></script>
</body>
</html>
```

Do Not Duplicate

Using ngCloak to Improve Initial UI Appearance *cont'd*

- **However, AngularJS best practices have us put the script at the bottom of the body so that the page loads faster**
 - This means the directive won't be "known" until after the browser has rendered the raw HTML that includes data binding
- **To correct this, we will have to add a CSS rule into our own .css file and use the class attribute**
 - The AngularJS script will still remove the CSS class after the application loads

Example

The following CSS rule uses attribute selectors to select the DOM elements that include different variations of the ngCloak directive. Within the rule, the display property is set to none with an !important rule.

css/CustomCSS.css

```
[ng\:cloak], [ng-cloak], [data-ng-cloak], [x-ng-cloak],
.ng-cloak, .x-ng-cloak {
  display: none !important;
}
```

ngcloakexample.html

```
<!DOCTYPE html>
<html>
<head>
  <title>AngularJS Data Binding</title>
  <meta charset="utf-8" />
  <link href="css/CustomCSS.css" rel="stylesheet" />
</head>
```

Do Not Duplicate

```
<body ng-app="myApp">  
  <div ng-controller="CourseController2" >  
  
    <div class="ng-cloak" >  
      <p>Course: {{ courseTitle }}</p>  
      <p>Instructor: {{ instructor }}</p>  
      <p>Location: {{ location }}</p>  
    </div>  
    <p>  
      <button ng-click="setLocation('Chelmsford, MA USA')">  
        Run class in USA</button>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
      <button ng-click="setLocation('London, England')">  
        Run class in UK</button>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
      <button ng-click="setLocation('Toronto, Ontario')">  
        Run class in Canada</button>  
    </p>  
  </div>  
  
  <script src="scripts/angular.min.js"></script>  
  <script src="scripts/controllers/CourseController2.js"></script>  
</body>  
</html>
```

2-27

Section 2–3

Working with Scope and the Data Model

Do Not Duplicate

Using Scope to Define the Data Model

- In AngularJS, the `$scope` object represents the data model
- The data model can define simple values, objects and arrays by associating them with the scope

Example

scripts/controllers/PersonController.js

```
var myApp = angular.module('myApp', []);

myApp.controller('PersonController', function($scope) {

    // simple type
    $scope.planet = 'Earth';

    // object
    $scope.person = {
        name : 'Devan',
        age : 1
    };

    // array of simple types
    $scope.toys = ['Minnie Mouse', 'Cardboard Box',
        'Rocking Horse', 'Teddy Bear'];

    // array of objects
    $scope.cousins = [
        {name : 'Brady', age : 10},
        {name : 'Alexis', age : 19},
        {name : 'Emma', age : 6}
    ];

});
```

Do Not Duplicate

Example: Binding Data to a Complex Data Model

- Binding data to a view from this "more complicated" data model is simply a combination of everything we have already seen

Example

personview.html

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

  <div ng-controller="PersonController">

    <h3>Planet {{ planet }} reports that its favorite person is
      {{ person.name }}</h3>

    <p>Her favorite toys are:</p>
    <ul>
      <li ng-repeat="toy in toys">
        {{ toy }}
      </li>
    </ul>

    <p>Her cousins are:</p>
    <table border="1">
      <tr>
        <th>Name</th>
        <th>Age</th>
      </tr>
      <tr ng-repeat="cousin in cousins">
        <td>{{ cousin.name }}</td>
        <td>{{ cousin.age }}</td>
      </tr>
    </table>
  </div>

  <script src="scripts/angular.min.js"></script>
  <script src="scripts/controllers/PersonController.js"></script>
</body>
```

Do Not Duplicate

Example: Binding Data to a Complex Data Model *cont'd*

Planet Earth reports that its favorite person is Devan

Her favorite toys are:

- Minnie Mouse
- Cardboard Box
- Rocking Horse
- Teddy Bear

Her cousins are:

Name	Age
Brady	10
Alexis	19
Emma	6

Do Not Duplicate

Adding Methods to the Data Model

- As you saw, methods are added to the data model by associating JavaScript functions with scope properties
 - To reference your properties within the method, prefix them with `$scope.`

Example

```
myApp.controller('PersonController2', function($scope){  
  
    $scope.planet = 'Earth';  
    $scope.person = {  
        name : 'Devan',  
        age : 1  
    };  
  
    $scope.getPersonDescription = function(){  
        return $scope.person.name + ' (Age: ' +  
            $scope.person.age + ' years old)';  
    };  
});
```

- You may also call the methods from within data binding expressions

Example

```
<body ng-app="myApp">  
  <div ng-controller="PersonController2">  
  
    <h3>Planet {{ planet }} reports that its favorite person  
      is:</h3>  
    <h3>{{ getPersonDescription() }}</h3>  
  
  </div>  
  
  <script src="scripts/angular.min.js"></script>  
  <script src="scripts/controllers/PersonController.js"></script>  
</body>
```

DO NOT DUPLICATE

Passing Arguments to Methods

- If your method defines parameters, you can pass values to them when you call the method in a data binding expression

Example

scripts/controllers/PersonController2.js

```
myApp.controller('PersonController2', function($scope) {

    $scope.planet = 'Earth';
    $scope.person = {
        name : 'Devan',
        age : 1
    };

    $scope.getPerson = function(verbose) {
        if (verbose == false) {
            return $scope.person.name;
        }

        return $scope.person.name + ' (Age: ' +
            $scope.person.age + ' years old)';
    };
});
```

Example

personview2.html

```
<body ng-app="myApp">
    <div ng-controller="PersonController2">

        <h3>Planet {{ planet }} reports that its favorite person
            is:</h3>
        <h4>{{ getPerson(true) }}</h4>

    </div>

    <script src="scripts/angular.min.js"></script>
    <script src="scripts/controllers/PersonController2.js"></script>
</body>
```

Passing Arguments to Methods using ngClick

- You can also pass values to parameters when you call a method using ngClick

Example

scripts/Controllers/ToyController.js

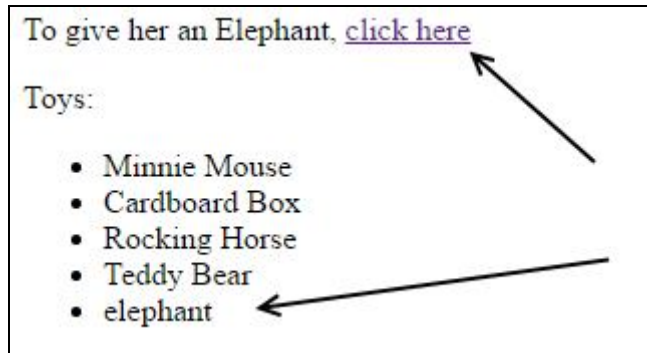
```
myApp.controller("ToyController", function($scope) {  
    $scope.toys = ["Minnie Mouse", "Cardboard Box",  
                  "Rocking Horse", "Teddy Bear"];  
  
    $scope.addToy = function(newToy) {  
        $scope.toys.push(newToy);  
    };  
});
```

Example

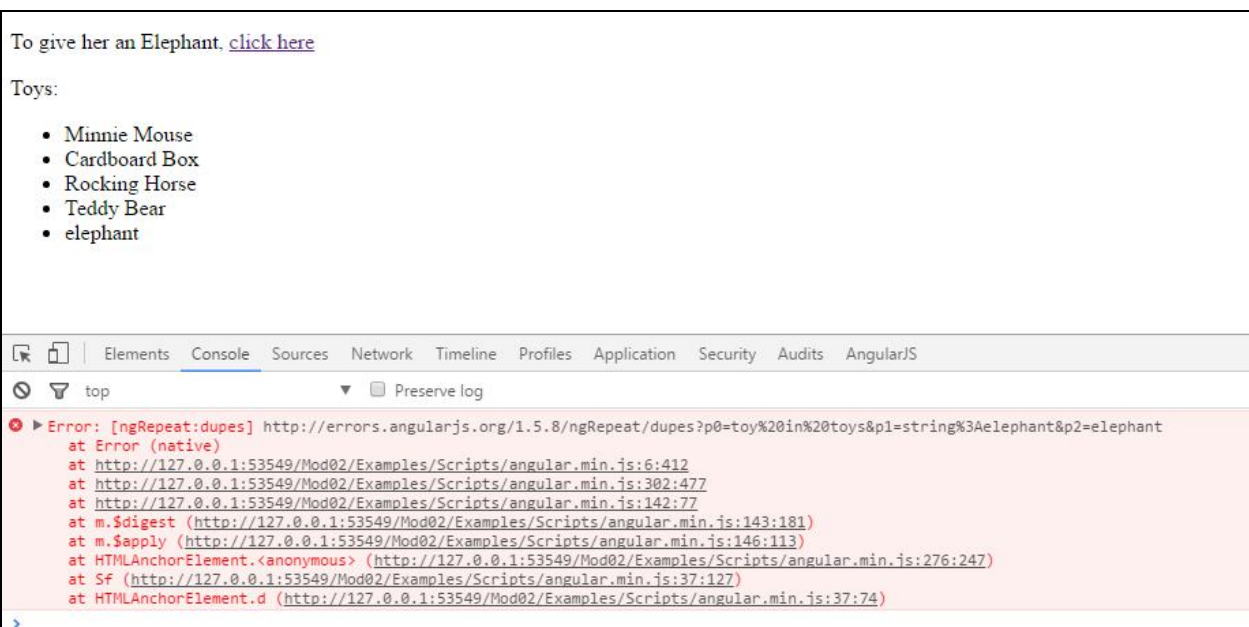
toyview1.html

```
<body ng-app="myApp">  
  <div ng-controller="ToyController">  
    <p>  
      To give her an Elephant,  
      <a href="#" ng-click="addToy('elephant')">  
        click here</a>  
    </p>  
  
    <p>Toys:</p>  
    <ul>  
      <li ng-repeat="toy in toys">  
        {{ toy }}  
      </li>  
    </ul>  
  
  </div>  
  
  <script src="scripts/angular.min.js"></script>  
  <script src="scripts/controllers/ToyController.js"></script>  
</body>
```

Passing Arguments to Methods using ngClick *cont'd*



- However, if the link is clicked a second time, an error occurs because the default behavior of `ngRepeat` does not allow duplicates in the collection
 - We will see how to resolve this in a few pages



Do Not Duplicate

Passing User Input as Arguments

- If you want to pass a value that was entered by the user in an HTML input element, you should use the `ngModel` directive along with `ngClick`

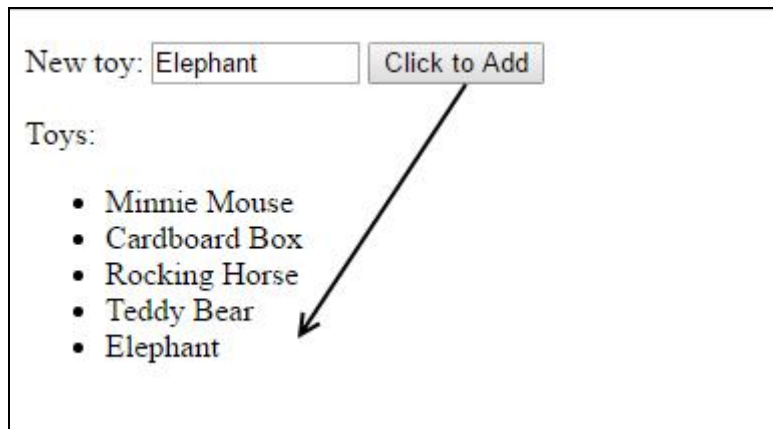
Example

toyview2.html

```
<body ng-app="myApp">
  <div ng-controller="ToyController">

    <p>New toy:
      <input ng-model="newToyName" style="width:100px"/>

      <input type="button" value="Click to Add"
        ng-click="addToy (newToyName) " />
    </p>
    <p>Toys:</p>
    <ul>
      <li ng-repeat="toy in toys">
        {{ toy }}
      </li>
    </ul>
  </div>
  <script src="scripts/angular.min.js"></script>
  <script src="scripts/controllers/ToyController.js"></script>
</body>
```



Do Not Duplicate

Issues with ngRepeat

- If we attempt to add a duplicate to the collection, we get the same error

The screenshot shows a web application interface with a form to add a new toy. The form has a text input labeled "New toy:" containing the word "Elephant" and a button labeled "Click to Add". Below the form, there is a list of toys: Minnie Mouse, Cardboard Box, Rocking Horse, Teddy Bear, and Elephant. An arrow points from the "Click to Add" button to the "Elephant" item in the list, with a text box stating: "If you try to add a second 'Elephant' to the collection, the data binding fails because ngRepeat doesn't allow duplicates". Below the application, the browser's developer console is open, showing an error message: "Error: [ngRepeat:dupes] http://errors.angularjs.org/1.5.8/ngRepeat/dupes? p0=toy%20in%20toys&p1=string%3AElephant&p2=Elephant". The error message is followed by a stack trace showing the call stack from the browser's HTMLInputElement to the AngularJS framework.

- Why? In order to maintain two-way binding, ngRepeat uses a *tracking function* that associates each collection element with its corresponding DOM element
 - The default tracking function (named `$id`) does not allow duplicate items in arrays

Do Not Duplicate

Using a Custom Tracking Function with ngRepeat

- If you need duplicate items in a collection, you can substitute your own tracking function
 - Use the `track by` option with `ngRepeat`

Example

In this example, we use the `$scope` property `$index` as our tracking function. It works with `ToyController.js`.

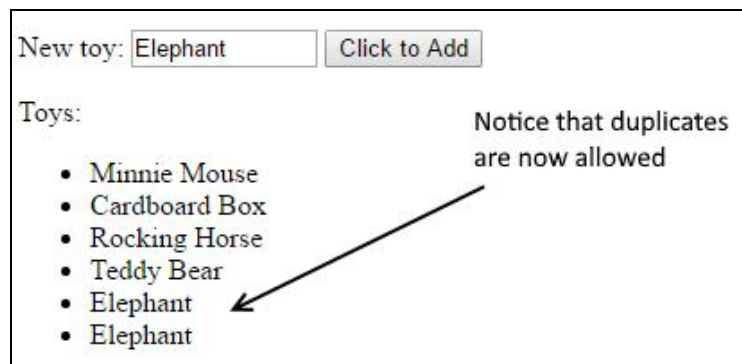
toyview3.html

```
<body ng-app="myApp">
  <div ng-controller="ToyController">
    <p>New toy:
      <input ng-model="newToyName" style="width:100px"/>

      <input type="button" value="Click to Add"
        ng-click="addToy(newToyName)" /></p>

    <p>Toys:</p>
    <ul>
      <li ng-repeat="toy in toys track by $index">
        {{ toy }}
      </li>
    </ul>
  </div>

  <script src="scripts/angular.min.js"></script>
  <script src="scripts/controllers/ToyController.js"></script>
</body>
```



Duplicate

Section 2–4

How Data Binding Works

Do Not Duplicate

Interpolation

- **In the world of programming, *interpolation* is the process of generating a string by using placeholders**
 - Interpolation takes a string literal containing one or more placeholders and replaces the placeholders with actual values
 - Many languages generate strings using this technique
- **AngularJS uses interpolation during data binding**
 - Whatever appears between the double sets of curly braces will be interpolated by AngularJS

Example

```
<div ng-controller="MainController">  
  <h1>Hello {{ name }}</h1>  
</div>
```

- **If you view the source for the page, the HTML that is downloaded will contain the data binding expression**
 - However, AngularJS interpolates the value and replaces the data binding expression with the interpolated value
 - * Visible in the Developer Tools
 - The value is then rendered on the page

Do Not Duplicate

Data Binding and the Digest Cycle

- As you've seen, AngularJS provides two-way data binding
- To accomplish this, AngularJS uses a "digest cycle" that consists of a loop that allows AngularJS to determine if a "watched" variable associated with the `$scope` has changed
 - If so, AngularJS re-syncs the variable with the DOM element
- Any variable you define in your controller using

```
$scope.myVariable = value;
```

is a *candidate* for being "watched"

- However, performance would be horrible if AngularJS had to watch every variable
- AngularJS provides a `$watch` service to allow you to identify which model properties support two-way binding

Do Not Duplicate

Watchers and the Digest Cycle

- For each "watched" property, the `$watch` service keeps track of the original value and a current value
- It looks at each variable in the watch list and compares the original value to the current value
 - If the value has changed, the property value is updated wherever it is being used
 - The update might occur in the DOM or in the code
- Then, another cycle of the digest cycle is executed to see if the first change affected some other value
 - The loop continues until all of the old values and the current values match
- The digest cycle then stops until another JavaScript event starts the process again
- The term *AngularJS execution context* is used to describe the things that occur within this digest cycle
 - If something happens outside of the execution context, then AngularJS is not aware of it and cannot correctly perform data binding

Do Not Duplicate

Using the \$watch Service

- **The \$watch service is the glue that supports data binding**
 - It maintains a list of *watch* expressions that identify the model properties being observed
- **To build a watch expression for \$scope variable, you can:**
 - Use it in your template via an expression: `{{ myVariable }}`
 - Use it in an AngularJS directive such as `ngRepeat`
- **Sometimes, you must manually add a property to the \$watch service**
 - To do this, you use the `$watch` method
 - * It registers a method that executes just after the controller method that modifies the specified variable finishes executing
 - Although rarely used, it can occasionally be useful when you need to respond to changes in watched variables through code

Do Not Duplicate

Understanding the \$watch Method

- The `$scope.watch()` method manually creates a watch for a variable
- When you register a watch, you pass two functions to the `$watch()` function:
 - A value function that returns the property being watched
 - * When the value changes, AngularJS calls the listener function
 - A listener function
 - * This function should do whatever it needs to do if the value has changed.

Example

```
$scope.cartTotal = 0;  
  
$scope.$watch('cartTotal', function() {  
    alert('Cart has changed');  
});
```

Do Not Duplicate

Example: Using the \$watch Method

Example

scripts/controllers/CartController.js

```
var myApp = angular.module('myApp', []);

myApp.controller('CartController', function ($scope) {

    $scope.cartTotal = 0;

    $scope.$watch('cartTotal', function () {

        if ($scope.cartTotal != 0) {
            alert('There are ' + $scope.cartTotal +
                ' items in your cart');
        }
    });

    $scope.addItem = function () {
        $scope.cartTotal += 1;           // triggers the watch
    };

    $scope.removeItem = function () {
        if ($scope.cartTotal > 0) {
            $scope.cartTotal -= 1;       // triggers the watch
        }
    };
});
```

watchexample.html

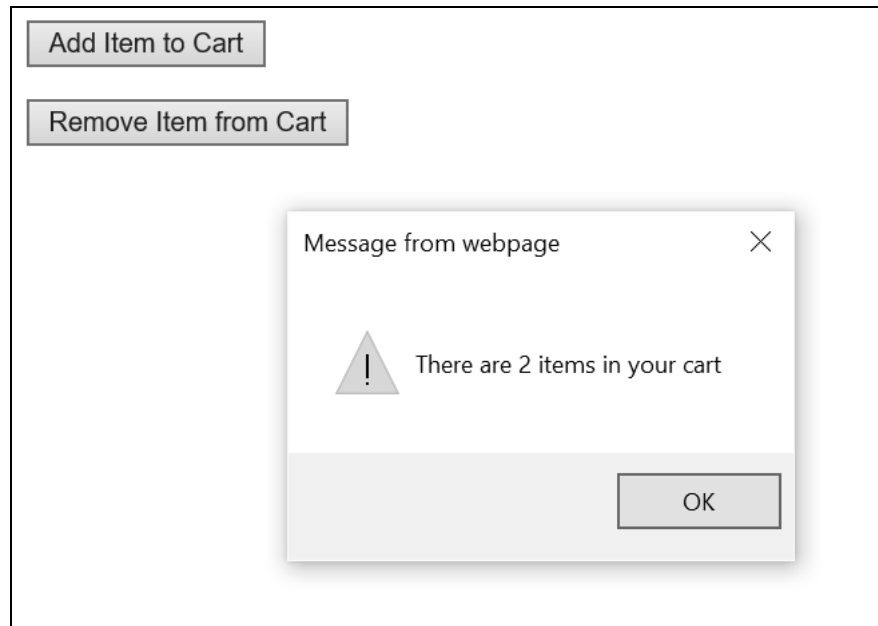
```
<body ng-app="myApp">

    <div ng-controller="CartController">
        <p>
            <input type="button" ng-click="addItem()"
                value="Add Item to Cart" />
        </p>
        <p>
            <input type="button" ng-click="removeItem()"
                value="Remove Item from Cart" />
        </p>
    </div>

    <script src="scripts/angular.min.js"></script>
    <script src="scripts/controllers/CartController.js"></script>
</body>
```

DO NOT DUPLICATE

Example: Using the \$watch Method *cont'd*



Do Not Duplicate

Using \$apply

- **When a browser calls a JavaScript function directly, it executes outside the AngularJS execution context**
 - This means AngularJS is unaware of model changes
- **To ensure that the digest cycle runs correctly, AngularJS requires methods to be executed using \$apply**
 - \$apply executes the function that is passed to it and then calls \$digest to perform the digest cycle
- **If you change the model outside of the AngularJS context, the change will not be propagated through data binding**

Example

Although this code updates the model, the changes are not propagated.

```
var myApp = angular.module('myApp', []);

myApp.controller('TimerController', function($scope){

    $scope.message = 'Processing...';

    setTimeout(function() {
        $scope.message = 'Process complete!';
    }, 3000);
});
```

Example

The updated message won't be shown after 3 seconds.

```
<p>{{ message }}</p>
```

Do Not Duplicate

Using \$apply *cont'd*

- You call the `$scope.$apply()` to propagate the changes
 - The `$apply()` method calls `$digest()` under the hood, which forces a new digest cycle
- There are two ways to use `$apply()`
 - You can call the method directly and it will force a digest cycle

Example

When this code updates the model, the changes are propagated.

scripts/controllers/TimerController.js

```
var myApp = angular.module('myApp', []);

myApp.controller('TimerController', function($scope) {

    $scope.message = 'Processing...';

    setTimeout(function() {
        $scope.message = 'Process complete!';
        $scope.$apply();
    }, 3000);
});
```

timerview.html

```
<body ng-app="myApp">
  <div ng-controller="TimerController">

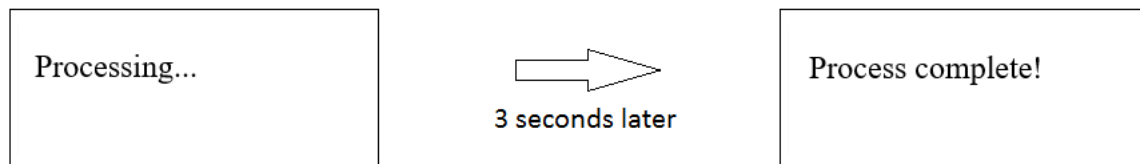
    <p>{{ message }}</p>

  </div>

  <script src="scripts/angular.min.js"></script>
  <script src="scripts/controllers/TimerController.js"></script>
</body>
```

Do Not Duplicate

Using `$apply` *cont'd*



- Although this way of calling `$apply` works, if your code throws an exception, it will do so outside of the AngularJS context
 - The digest cycle might not run
- A better way of using `$apply()` is to pass it a function that wraps the changes
 - This ensures that the `$digest` cycle will run

Example

This example illustrates how to ensure the `$digest` cycle will run.

```
setTimeout(function() {  
  $scope.$apply(function() {  
    $scope.message = 'Process complete!';  
  });  
}, 3000);
```

Do Not Duplicate

Using AngularJS \$timeout

- AngularJS has a `$timeout` service that can replace calls to the JavaScript `setTimeout` function
- What makes the `$timeout` service useful is that it executes within the AngularJS context
 - This means you don't have to call the `$apply()` method to force the `$digest` cycle
- The `$timeout` service must be injected into the controller method using the same technique as `$scope`
 - This is called dependency injection and is discussed in detail in a future module

Example

```
var myApp = angular.module('myApp', []);

myApp.controller('TimerControllerController',
  function($scope, $timeout){

    // within here, you can use $timeout

  });
```

- It is invoked using the same arguments as `setTimeout`

Do Not Duplicate

Example: Using AngularJS \$timeout

Example

scripts/controllers/TimerController2.js

```
var myApp = angular.module('myApp', []);

myApp.controller('TimerController2', function($scope, $timeout){

    $scope.message = 'Processing...';

    $timeout(function() {
        $scope.message = 'Process complete!';
    }, 3000);
});
```

timerview2.html

```
<body ng-app="myApp">
  <div ng-controller="TimerController2">

    <p>{{ message }}</p>

  </div>

  <script src="scripts/angular.min.js"></script>
  <script src="scripts/controllers/TimerController2.js"></script>
</body>
```

Do Not Duplicate

Module Summary

- **AngularJS data binding uses *two-way data binding***
 - This means that not only will the HTML element stay in sync with the data model, but the data model will stay in sync with the HTML element
- **Data binding can be accomplished using the `ngBind` directive or through AngularJS expressions**
 - AngularJS expressions are an easy way to display data and are written inside of double braces
- **Although you can define application data using the `ngInit` directive, it is more common to use an AngularJS controller**
- **When applications store data in collections, you iterate through the collection to display the items**
 - The `ngRepeat` directive creates a display template for each item in the collection
- **AngularJS uses a `$digest` cycle to manage its two-way binding**
 - A loop determines if "watched" variables have changed
 - If so, AngularJS re-syncs the variable and the bound DOM element

Do Not Duplicate

Module 3

Working with Filters

Do Not Duplicate

Module Goals and Objectives

Major Goals

Understand how AngularJS filters are used to transform data in an application

Understand the various types of AngularJS filters

Specific Objectives

On completion of this module, students will be able to:

Use AngularJS in data binding expressions

Use AngularJS filters to format data

Use AngularJS filters to order data

Use AngularJS filters to restrict data

Use AngularJS filters in JavaScript

Do Not Duplicate

Section 3–1

AngularJS Filters

Do Not Duplicate

AngularJS Filters

- **In AngularJS, filters provide a convenient mechanism for transforming data**
- **There are several types of built-in filters, including:**
 - filters that format data for output
 - filters that sort data
 - filters that restrict the data in a collection
- **You can even build custom filters**
- **These filters are used extensively in combination with AngularJS's data binding features**
- **They can also be used in JavaScript**

Do Not Duplicate

Section 3–2

Using Filters to Format Data

Do Not Duplicate

Using Filters to Format Data

- **AngularJS expressions can contain filters**
 - The filters we are about to examine are used to format data
- **Filters are added to an expression using the pipe (|) character and the filter**

Syntax

```
{{ expression | filter }}
```

- **Filters that format the appearance of data include:**

Filter	Description
lowercase	Formats a string to lowercase
uppercase	Formats a string to uppercase
number	Formats a number as text
currency	Formats a number to a currency format
date	Formats a date to a string based on the requested format.

Do Not Duplicate

The "Case" Filters

- The **uppercase** and **lowercase** filters can be used to transform data to the specified case

Example

```
<p>  
  Hello {{ name | uppercase }} !!!  
</p>
```

Example

```
<p>  
  Hello {{ name | lowercase }} !!!  
</p>
```

Do Not Duplicate

The Number Filter

- **The number filter formats a number as text**
 - Typically, it is used to control the number of decimal places to which the number is rounded for display purposes
 - * If the number of decimal places is not specified, it will use the default value 3
 - It also displays a thousands separator for the number

Example

```
<p>  
The cargo's weight is {{ weight | number : 1 }} pounds.  
</p>
```

OUTPUTS MIGHT BE:

```
The cargo's weight is 162.3 pounds.  
The cargo's weight is 1,162.3 pounds.  
The cargo's weight is 1,101,162.3 pounds.
```

- **If the input is infinite, it returns the ∞ symbol**
- **If the input is not a number, it returns the empty string**

Do Not Duplicate

The Currency Filter

- The **currency** filter displays a number as currency

Syntax

```
{{number | currency :symbol :fractionsize}}
```

- It can be applied to the value from a variable

Example

```
<p>
  {{ name }} earns {{ salary | currency }}
</p>
```

- It can also be applied to a computed value

Example

```
<p>
  Your Total Is: {{ quantity * price | currency }}
</p>
```

- By default, the local currency format is used and 2 digits are displayed to the right of the decimal point
- To use a different currency symbol, provide an argument to the filter

Example

```
{{price | currency :'€'}}
```

Do Not Duplicate

The Currency Filter *cont'd*

- To change the number of digits displayed to the right of the decimal point, pass a number as an argument to the filter

Example

```
{{ price | currency :3 }}
```

Example

currencyfilter.html

```
<body ng-app>
  <div ng-init="products=[
    {name: 'Football', price: 59.99},
    {name: 'Basketball', price: 25.99},
    {name: 'Soccer Ball', price: 19.99},
    {name: 'Baseball', price: 5.49} ]">
    <ul>
      <li ng-repeat="product in products">
        {{product.name}} - {{product.price | currency}}
      </li>
    </ul>
  </div>
  <script src="scripts/angular.min.js"></script>
</body>
```

- Football - \$59.99
- Basketball - \$25.99
- Soccer Ball - \$19.99
- Baseball - \$5.49

Do Not Duplicate

Example: Using Filters

Example

scripts/controllers/SimpleFiltersController.js

```
var myApp = angular.module('myApp', []);

myApp.controller('SimpleFiltersController', function ($scope) {
    $scope.phrase = 'Hello world';
    $scope.amount = 432.56;
});
```

simplefiltersview.html

```
<body ng-app="myApp" >
  <div ng-controller="SimpleFiltersController">

    <p>The phrase is: {{ phrase }} </p>
    <p>The phrase (in lower case) is: {{ phrase | lowercase }}</p>
    <p>The phrase (in upper case) is: {{ phrase | uppercase }}</p>
    <hr />
    <p>The amount is: {{ amount | currency }}
  </div>

  <script src="scripts/angular.min.js"></script>
  <script src="scripts/controllers/SimpleFiltersController.js">
  </script>
</body>
```

The phrase is: Hello world	
The phrase (in lower case) is: hello world	
The phrase (in upper case) is: HELLO WORLD	
<hr/>	
The amount is: \$432.56	

Do Not Duplicate

Filtering Dates

- The **date** filter formats a date as a string based on a format specified

Syntax

```
{{ expression | date: format }}
```

- Format strings can include:

Format	Description	Example
YYYY	A four digit year	2016
YY	A two digit year	16
MMMM	A month spelled out in its entirety	JANUARY
MMM	A month abbreviation	JAN
MM	A month as a 2-digit number	01
M	A month as a number	1
dd	The day in the month as a 2-digit number	05
d	The day in the month	5
EEEE	The day of the week spelled out in its entirety	TUESDAY
EEE	The day of the week abbreviated	TUE
HH	The hour in the day (out of 24 hours) as a 2-digit number	17
H	The hour in the day (out of 24 hours)	17
hh	The hour in the day (based on AM/PM) as a 2-digit number	05
h	The hour in the day (based on AM/PM)	5
mm	The minute in the hour as a 2-digit number	00
m	The minute in the hour	0
ss	The second in the minute as a 2-digit number	00
s	The second in the minute	0
sss	The millisecond of the second as a 3 digit number	000
a	AM/PM marker	PM
Z	A four digit signed number representing the time zone offset	+0600
ww	Week of the year (00-53) where week 01 has the first Thursday	02
w	Week of the year (0-53) where week 1 has the first Thursday	2
GG	The era (AD)	AD

Example

```
<p>Date: {{ enrollTime | date: 'yyyy-MM-dd h:mm a' }}</p>
```

Do Not Duplicate

Filtering Dates *cont'd*

- A format string can also be one of the following predefined localizable formats:

Format	Description	en-US Equivalent
short	A short date/time.	M/d/yy h:mm a
medium	A medium date/time.	MMM d, y h:mm:ss a
shortDate	A short date.	M/d/yy
mediumDate	A medium date.	MMM d, y
longDate	A long date.	MMMM d, y
fullDate	A long date.	EEEE, MMMM d, y
shortTime	A short time.	h:mm a
mediumTime	A medium time.	h:mm:ss a

Example

```
<p>Date: {{ enrollTime | date:'short' }}</p>
```

- The format string can also contain literal values
 - They need to be surrounding with single quotes

Example

```
<p>Date: {{ enrollTime | date:"dd/MM/yy 'at' HH:mm Z" }}</p>
```

- This date filter does not convert the date to your browser locale
 - Refer to the following for more information about tricks you can use to display dates in browser locale format

<http://michelebusta.com/angular-tip-date-and-time-locale-formatting/>

Do Not Duplicate

Example: Filtering Dates

Example

The screen shot shown at the bottom of this example was taken from a computer whose operating system settings were "Central Time" in the United States.

scripts/controllers/DateFiltersController.js

```
var myApp = angular.module('myApp', []);

myApp.controller('DateFiltersController', function($scope) {
    $scope.enrollment = Date.now();
});
```

datefiltersview.html

```
<body ng-app="myApp">
  <div ng-controller="DateFiltersController">

    <p>Date using 'short': {{ enrollment | date:'short' }}</p>
    <p>Date using 'medium': {{ enrollment | date:'medium' }}</p>
    <p>Date using 'fullDate':
      {{ enrollment | date:'fullDate' }}</p>

    <hr />

    <p>Date using 'yyyy-MM-dd h:mm a':
      {{ enrollment | date:'yyyy-MM-dd h:mm a' }}</p>
    <p>Date using 'dd-MMM-yy HH:mm:ss':
      {{ enrollment | date:'dd-MMM-yy HH:mm:ss' }}</p>
    <p>Date using 'dd/MM/yy 'at' HH:mm Z':
      {{ enrollment | date:"dd/MM/yy 'at' HH:mm Z" }}</p>

  </div>

  <script src="scripts/angular.min.js"></script>
  <script
    src="scripts/controllers/DateFiltersController.js"></script>
</body>
```

Date using 'short': 1/6/16 12:53 AM	
Date using 'medium': Jan 6, 2016 12:53:04 AM	
Date using 'fullDate': Wednesday, January 6, 2016	
<hr/>	
Date using 'yyyy-MM-dd h:mm a': 2016-01-06 12:53 AM	
Date using 'dd-MMM-yy HH:mm:ss': 06-Jan-16 00:53:04	
Date using 'dd/MM/yy 'at' HH:mm Z': 06/01/16 at 00:53 -0600	

Section 3–3

Using Filters to Sort or Restrict Data

Do Not Duplicate

Sorting and Restricting Data

- **Filters can also be used to sort or restrict data**
 - These filters include:

Filter	Description
<code>orderBy</code>	Orders an array by an expression
<code>limitTo</code>	Creates a new array or string containing only a specified number of elements
<code>filter</code>	Selects a subset of items from an array

Do Not Duplicate

Sorting Data

- The **orderBy** filter can be applied to the **ngRepeat** directive to control the order in which items appear

Example

In this example, data defined in the `EmployeeController.js` is displayed in ascending order by city. The `$scope` object defines an `employees` property that is an array of objects. Each object has `name`, `jobTitle`, `city`, `pay`, and `startDate` properties.

The CSS that styles the table is found in the HTML file.

employeeview.html

```
<body ng-app='myApp'>
  <div ng-controller='EmployeeController'>
    <h2>There are {{ employees.length }} employees:</h2>
    <table>
      <tr>
        <th>Name</th>
        <th>Job Title</th>
        <th>City</th>
        <th>Pay</th>
        <th>Start Date</th>
      </tr>
      <tr ng-repeat="emp in employees | orderBy: 'name'">
        <td>{{ emp.name }}</td>
        <td>{{ emp.jobTitle }}</td>
        <td>{{ emp.city | uppercase }}</td>
        <td>{{ emp.pay | currency }}</td>
        <td>{{ emp.startDate | date:'dd-MMM-yy' }}</td>
      </tr>
    </table>
  </div>

  <script src="scripts/angular.min.js"></script>
  <script
    src="scripts /controllers/EmployeeController.js"></script>
</body>
```

Do Not Duplicate

Sorting Data *cont'd*

There are 3 employees:

Name	Job Title	City	Pay	Start Date
Brittany	Supervisor	FT. WORTH	\$52,580.00	17-Apr-06
Natalie	Technical Writer	FT. WORTH	\$67,521.25	25-Jun-05
Zachary	Construction Engineer	HOUSTON	\$74,115.75	14-Jan-16

- To indicate a reverse sort, you can place a hyphen in front of the sort field

Example

employeeview2.html

```
<tr ng-repeat="emp in employees | orderBy: '-pay'">
  <td>{{ emp.name }}</td>
  <td>{{ emp.jobTitle }}</td>
  <td>{{ emp.city | uppercase }}</td>
  <td>{{ emp.pay | currency }}</td>
  <td>{{ emp.startDate | date:'dd-MMM-yy' }}</td>
</tr>
```

There are 3 employees:

Name	Job Title	City	Pay	Start Date
Zachary	Construction Engineer	HOUSTON	\$74,115.75	14-Jan-16
Brittany	Supervisor	FT. WORTH	\$68,580.00	17-Apr-06
Natalie	Technical Writer	FT. WORTH	\$67,521.25	25-Jun-05

Do Not Duplicate

Sorting Data *cont'd*

- You can also use a model property to specify the sort field
 - The property is then used in the `orderBy` filter

Example

```
myApp.controller('EmployeeController', function ($scope) {  
    $scope.sortColumn = 'name';    // the default sort column  
  
    $scope.employees = [ ... ];  
  
});
```

Example

```
<p>Sort Column: {{ sortColumn }}</p>
```

```
<!-- not all code shown -->
```

```
<tr ng-repeat="emp in employees | orderBy:sortColumn">  
    <td>{{ emp.name }}</td>  
    <td>{{ emp.jobTitle }}</td>  
    <td>{{ emp.city | uppercase }}</td>  
    <td>{{ emp.pay | currency }}</td>  
    <td>{{ emp.startDate | date:'dd-MMM-yy' }}</td>  
</tr>
```

Sort Column: name

Name	Job Title	City	Pay	Start Date
Brittany	Supervisor	FT. WORTH	\$68,580.00	17-Apr-06
Natalie	Technical Writer	FT. WORTH	\$67,521.25	25-Jun-05
Zachary	Construction Engineer	HOUSTON	\$74,115.75	14-Jan-16

Do Not Duplicate

Dynamic Sorting

- You can allow the user to sort displayed data dynamically
 - It is typically done by clicking on the table column headers
- To do this, add `ngClick` to the `<th>` elements and assign the sort column property the name of the column on which to sort
 - That property is then used by the `orderBy` filter

Example

```
myApp.controller('EmployeeController', function ($scope) {  
  
    $scope.sortColumn = 'name'; // the default sort column  
    $scope.employees = [ ... ];  
  
});
```

Example

<p>Sort Column: {{ sortColumn }}</p>

```
<table>  
  <tr>  
    <th ng-click="sortColumn='name'">Name</th>  
    <th ng-click="sortColumn='jobTitle'">Job Title</th>  
    <th ng-click="sortColumn='city'">City</th>  
    <th ng-click="sortColumn='pay'">Pay</th>  
    <th ng-click="sortColumn='startDate'">Start Date </th>  
  </tr>  
  <tr ng-repeat="emp in employees | orderBy:sortColumn">  
    <td>{{ emp.name }}</td>  
    <td>{{ emp.jobTitle }}</td>  
    <td>{{ emp.city | uppercase }}</td>  
    <td>{{ emp.pay | currency }}</td>  
    <td>{{ emp.startDate | date:'dd-MMM-yy' }}</td>  
  </tr>  
</table>
```

Do Not Duplicate

Dynamic Sorting *cont'd*

Sort Column: name

Name	Job Title	City	Pay	Start Date
Brittany	Supervisor	FT. WORTH	\$68,580.00	17-Apr-06
Natalie	Technical Writer	FT. WORTH	\$67,521.25	25-Jun-05
Zachary	Construction Engineer	HOUSTON	\$74,115.75	14-Jan-16

Sort Column: pay

Name	Job Title	City	Pay	Start Date
Natalie	Technical Writer	FT. WORTH	\$67,521.25	25-Jun-05
Brittany	Supervisor	FT. WORTH	\$68,580.00	17-Apr-06
Zachary	Construction Engineer	HOUSTON	\$74,115.75	14-Jan-16

Do Not Duplicate

Toggling Sort Order

- AngularJS's `orderBy` filter supports an additional parameter called **reverse** whose value indicates whether the sort is ascending (**false**) or descending (**true**)
- You can provide clickable column headers that toggle between ascending and descending sorts when you click on a header
- To do this, add a property to the model to track the sort order (ascending/descending)

Example

```
myApp.controller('EmployeeController', function ($scope) {  
  
    $scope.sortColumn = 'name'; // the default sort column  
    $scope.reverseSort = false; // the default sort type  
  
    $scope.employees = [ ... ];  
  
});
```

- You can then add the **reverse** argument to the `orderBy` filter

Example

```
<tr ng-repeat="emp in employees | orderBy:sortColumn:reverseSort">  
  <td>{{ emp.name }}</td>  
  <td>{{ emp.jobTitle }}</td>  
  <td>{{ emp.city | uppercase }}</td>  
  <td>{{ emp.pay | currency }}</td>  
  <td>{{ emp.startDate | date:'dd-MMM-yy' }}</td>  
</tr>
```

Do Not Duplicate

Toggling Sort Order *cont'd*

- Finally, toggle the property that specifies the sort direction in the `ngClick` directive using ***reverse = !reverse***

Example

```
<th ng-click="sortColumn='name'; reverseSort = !reverseSort">
  Name</th>
<th ng-click="sortColumn='jobTitle'; reverseSort = !reverseSort">
  Job Title</th>
<th ng-click="sortColumn='city'; reverseSort = !reverseSort">
  City</th>
<th ng-click="sortColumn='pay'; reverseSort = !reverseSort">
  Pay</th>
<th ng-click="sortColumn='startDate'; reverseSort = !reverseSort">
  Start Date</th>
```

Do Not Duplicate

Example: Toggling Sorting

Example

scripts/controllers/EmployeeController2.js

```
var myApp = angular.module('myApp', []);

myApp.controller('EmployeeController2', function ($scope) {

    $scope.sortColumn = 'name'; // set the default sort column
    $scope.reverseSort = false; // set the default sort type

    $scope.employees = [
        { name: 'Natalie', jobTitle: 'Technical Writer',
          pay: 67521.25, city: 'Ft. Worth',
          startDate: '2005-06-25' },
        { name: 'Brittany', jobTitle: 'Supervisor',
          pay: 68580.00, city: 'Ft. Worth',
          startDate: '2006-04-17' },
        { name: 'Zachary', jobTitle: 'Construction Engineer',
          pay: 74115.75, city: 'Houston',
          startDate: '2016-01-14' }
    ];

});
```

employeewithtogglesortingview.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Using Filters</title>
    <meta charset="utf-8" />
    <style>
        table, td, th {
            border: 1px solid black;
            border-collapse: collapse;
            padding: 3px;
        }

        tr:nth-child(even) {
            background-color: #eee;
        }
    </style>
</head>
```

Do Not Duplicate

Example: Toggling Sorting *cont'd*

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

  <div ng-controller="EmployeeController2">
    <div>
      <p>Sort Column: {{ sortColumn}}</p>
      <p>Ascending Sort: {{ !reverseSort }}</p>
    </div>

    <table>
      <tr>
        <th ng-click="sortColumn='name';
                      reverseSort = !reverseSort">Name</th>
        <th ng-click="sortColumn='jobTitle';
                      reverseSort = !reverseSort">Job Title</th>
        <th ng-click="sortColumn='city';
                      reverseSort = !reverseSort">City</th>
        <th ng-click="sortColumn='pay';
                      reverseSort = !reverseSort">Pay</th>
        <th ng-click="sortColumn='startDate';
                      reverseSort = !reverseSort">Start Date</th>
      </tr>
      <tr ng-repeat="emp in employees |
                      orderBy : sortColumn : reverseSort">
        <td>{{ emp.name }}</td>
        <td>{{ emp.jobTitle }}</td>
        <td>{{ emp.city | uppercase }}</td>
        <td>{{ emp.pay | currency }}</td>
        <td>{{ emp.startDate | date:'dd-MMM-yy' }}</td>
      </tr>
    </table>
  </div>

  <script src="scripts/angular.min.js"></script>
  <script
    src="scripts/controllers/EmployeeController2.js"></script>
</body>
</html>
```

Sort Column: name

Ascending Sort: true

Name	Job Title	City	Pay	Start Date
Brittany	Supervisor	FT. WORTH	\$68,580.00	17-Apr-06
Natalie	Technical Writer	FT. WORTH	\$67,521.25	25-Jun-05
Zachary	Construction Engineer	HOUSTON	\$74,115.75	14-Jan-16

Example: Toggling Sorting *cont'd*

Sort Column: name

Ascending Sort: false

Name	Job Title	City	Pay	Start Date
Zachary	Construction Engineer	HOUSTON	\$74,115.75	14-Jan-16
Natalie	Technical Writer	FT. WORTH	\$67,521.25	25-Jun-05
Brittany	Supervisor	FT. WORTH	\$68,580.00	17-Apr-06

Sort Column: startDate

Ascending Sort: true

Name	Job Title	City	Pay	Start Date
Zachary	Construction Engineer	HOUSTON	\$74,115.75	14-Jan-16
Brittany	Supervisor	FT. WORTH	\$68,580.00	17-Apr-06
Natalie	Technical Writer	FT. WORTH	\$67,521.25	25-Jun-05

Do Not Duplicate

Restricting Data with `limitTo`

- In addition to using filters for formatting and sorting data, you can use filters to restrict data being displayed
 - The `limitTo` filter, used in conjunction with the `ngRepeat` directive, allows you to limit the number of items to be displayed

Example

This example only displays the first 3 employees.

```
<ul>
  <li ng-repeat="emp in employees | limitTo : 3">
    {{ emp.name }}
  </li>
</ul>
```

- The `limitTo` filter can also restrict the number of characters of a string that are displayed

Example

This example displays only the first 20 characters of name.

```
<tr ng-repeat="show in shows">
  <td>{{ show.name | limitTo : 20 }}</td>
  <td>{{ show.network }}</td>
  <td>{{ show.year }}</td>
</tr>
```

Do Not Duplicate

Example: Using limitTo

Example

scripts/controllers/TVShowController.js

```
var myApp = angular.module('myApp', []);

myApp.controller('TVShowController', function ($scope) {

    $scope.shows = [
        { name: 'The Big Bang Theory', network: 'CBS', year: 2007 },
        { name: 'Lost', network: 'ABC', year: 2004 },
        { name: 'M*A*S*H', network: 'CBS', year: 1972 },
        <!-- other shows not shown -->
    ];

});
```

limittoexampleview.html

```
<body ng-app="myApp" ng-controller="TVShowController">

    <h2>There are {{shows.length}} TV shows in the library</h2>

    <p>Number of shows to display:
        <select ng-model="numShows">
            <option>3</option>
            <option>5</option>
            <option>7</option>
            <option>10</option>
            <option>{{ shows.length }}</option>
        </select>
    </p>

    <table>
        <tr>
            <th>Name</th>
            <th>Network</th>
            <th>Year Debuted</th>
        </tr>
        <tr ng-repeat="show in shows | limitTo : numShows">
            <td>{{ show.name }}</td>
            <td>{{ show.network }}</td>
            <td>{{ show.year }}</td>
        </tr>
    </table>
```

Do Not Duplicate

Example: Using limitTo *cont'd*

```
<script src="scripts/angular.min.js"></script>
<script
  src="scripts/controllers/TVShowController.js"></script>
</body>
</html>
```

There are 23 TV shows in the library

Number of shows to display: 10 ▼

Name	Network	Year Debuted
The Big Bang Theory	CBS	2007
Lost	ABC	2004
M*A*S*H	CBS	1972
The Bob Newhart Show	CBS	1972
The Mary Tyler Moore Show	CBS	1970
The Carol Burnett Show	CBS	1967
The Dick Van Dyke Show	CBS	1961
I Love Lucy	CBS	1951
Gilligans Island	CBS	1964
Bewitched	ABC	1964

There are 23 TV shows in the library

Number of shows to display: 3 ▼

Name	Network	Year Debuted
The Big Bang Theory	CBS	2007
Lost	ABC	2004
M*A*S*H	CBS	1972

Do Not Duplicate

Restricting Data with Custom Text using `filter`

- The `filter` filter allows you to provide text that will be used to filter data
 - By default, the filter checks all property values

Example

This example restricts the return values to any objects that contain the substring 'ABC' in any of the properties.

```
<tr ng-repeat="show in shows | filter : 'ABC'" >
  <td>{{ show.name }}</td>
  <td>{{ show.network }}</td>
  <td>{{ show.year }}</td>
</tr>
```

- You can restrict the filter to a specific property

Example

This example restricts the return values to any objects that contain the substring 'C' in the `network` property.

```
<tr ng-repeat="show in shows | filter : {network:'C'}" >
  <td>{{ show.name }}</td>
  <td>{{ show.network }}</td>
  <td>{{ show.year }}</td>
</tr>
```

- If you want an exact match, you can pass `true` as another argument
 - The filter property normally matches on a substring

Do Not Duplicate

Restricting Data with Custom Text using `filter` *cont'd*

Example

This example restricts the return values to any objects that contain **exactly** the value 'ABC' in the `network` property.

```
<tr
  ng-repeat="show in shows | filter : {network:'ABC'} : true" >
  <td>{{ show.name }}</td>
  <td>{{ show.network }}</td>
  <td>{{ show.year }}</td>
</tr>
```

- **If the filter condition is complex, you can encapsulate the logic in a model method**
 - You can then use the method in the filter

Example

This method used by the filter returns `true` if the show's network is 'CBS' and it debuted during or after 1980.

```
var myApp = angular.module('myApp', []);

myApp.controller('TVShowController', function ($scope) {

  $scope.shows = [
    { name: 'The Big Bang Theory', network: 'CBS', year: 2007 },
    { name: 'Lost', network: 'ABC', year: 2004 },
    { name: 'M*A*S*H', network: 'CBS', year: 1972 },
    <!-- other shows not shown -->
  ];

  $scope.newShows = function(item) {
    return item.network === 'CBS' && item.year >= 1980;
  };

});
```

Do Not Duplicate

Restricting Data with Custom Text using `filter` *cont'd*

Example

```
<tr ng-repeat="show in shows | filter : newShows" >
  <td>{{ show.name }}</td>
  <td>{{ show.network }}</td>
  <td>{{ show.year }}</td>
</tr>
```

- You can even bind the filter to user-entered values associated with a model variable using `ngModel`

Example

filterexampleview.html

```
<p>
  Enter a search phrase:
  <input type="text" ng-model="searchFilter" style="width:100px">
</p>
<table>
  <tr>
    <th>Name</th>
    <th>Network</th>
    <th>Year Debuted</th>
  </tr>
  <tr ng-repeat="show in shows | filter : searchFilter" >
    <td>{{ show.name }}</td>
    <td>{{ show.network }}</td>
    <td>{{ show.year }}</td>
  </tr>
</table>
```

There are 23 TV shows in the library

Enter a search phrase: ×

Name	Network	Year Debuted
The Bob Newhart Show	CBS	1972
The Mary Tyler Moore Show	CBS	1970
The Carol Burnett Show	CBS	1967
The Dick Van Dyke Show	CBS	1961
The Ed Sullivan Show	CBS	1948
The Howdy Doody Show	NBC	1954

Chaining Filters

- You can chain filters by adding them one behind the other

Example

```
<tr
  ng-repeat="show in shows | filter : 'CBS' | orderBy : 'name'">
  <td>{{ show.name }}</td>
  <td>{{ show.network }}</td>
  <td>{{ show.year }}</td>
</tr>
```

Do Not Duplicate

Section 3–4

Displaying Data as JSON

Do Not Duplicate

JSON Data

- AngularJS also has a filter that displays JavaScript objects as JSON
 - Although interesting, this filter is mostly useful for debugging

Example

jsonexampleview.html

```
<ul>
  <li ng-repeat="show in shows | limitTo : 5">
    {{ show | json }}
  </li>
</ul>
```

- { "name": "The Big Bang Theory", "network": "CBS1", "year": 2007 }
- { "name": "Lost", "network": "ABC", "year": 2004 }
- { "name": "M*A*S*H", "network": "CBS", "year": 1972 }
- { "name": "The Bob Newhart Show", "network": "CBS", "year": 1972 }
- { "name": "The Mary Tyler Moore Show", "network": "CBS", "year": 1970 }

Do Not Duplicate

Section 3–5

Using Filters with JavaScript

Do Not Duplicate

Using Filters with JavaScript

- You can also work with filters in your JavaScript code
 - For example, your controller methods can use the filters to format data in its methods
- The **\$filter** service accepts the name of a filter and returns a filter function
 - You then use the filter function

Syntax

`$filter(filterName)(value, parameter(s))`

- In order for the **\$filter** service to work within a controller, the **\$filter** service has to be passed to the controller function

Example

```
var myApp = angular.module('myApp', []);

myApp.controller('DemoController', function ($scope, $filter) {

    // ...

});
```

Do Not Duplicate

Using Filters with JavaScript *cont'd*

- If there are no parameters to the function, you simply pass the object to which the filter is applied

Example

In this example, the `price` is returned from a function formatted as currency

```
$scope.getPriceAsCurrency = function () {  
    return $filter('currency') ($scope.price);  
};
```

- If the filter expects arguments, they are passed in after the object to which the filter is applied

Example

In this example, the `endSaleDate` is returned from a function formatted as "MMMM d, yyyy".

```
$scope.getEndSaleAsLongDate = function () {  
    return $filter('date') ($scope.endSaleDate, 'MMMM d, yyyy');  
};
```

In this example, it creates the property named `commentsByStars` that contains `comments` objects that are ordered by the property `stars` (in reverse).

```
$scope.commentsByStars =  
    $filter('orderBy') ($scope.comments, '-stars');
```

In this example, it creates a property named `bestComments` that contains only the comment objects whose `stars` property is 5.

```
$scope.bestComments =  
    $filter('filter') ($scope.comments, {stars : 5});
```

Do Not Duplicate

Using Filters with JavaScript *cont'd*

- You can use the results of one filter in another
 - This is like chaining filters when data binding

Example

```
$scope.bestCommentsByName =  
    $filter('orderBy')(  
        $filter('filter')($scope.comments, { stars: 5 } ),  
        'name'  
    );
```

Do Not Duplicate

Example: Using the \$filter Service

Example

scripts/controllers/CamperSalesController.js

```
var myApp = angular.module('myApp', []);

myApp.controller('CamperSalesController', function ($scope, $filter)
{
    $scope.product = '2011 rPod 179 Camper';
    $scope.price = 12135.65;
    $scope.endDate = new Date(2015, 11, 24);

    $scope.comments = [
        { stars: 5, name: 'Kathy W.',
          comment: 'I love our rPod. Its cozy and functional.' },
        { stars: 4, name: 'G.K.',
          comment: "We've downsized from the larger campers.
                    We miss the couch, but enjoy everything else." },
        { stars: 5, name: 'Deb E.',
          comment: 'The rPod is so easy to pull.' },
        { stars: 5, name: 'D.H.',
          comment: 'The rPod is wonderful. Its fully functional. It
                    lets us commune with nature.' },
        { stars: 4, name: 'Mark W.',
          comment: 'Its so fun to belong to the rPod community.' },
        { stars: 1, name: 'Karla R.',
          comment: 'The rPod is too small to be functional. I wish
                    we had won Powerball so we could have a bigger camper.' }
    ];

    $scope.getProductAsUppercase = function () {
        return $filter('uppercase')($scope.product);
    };

    $scope.getProductAsLowercase = function () {
        return $filter('lowercase')($scope.product);
    };

    $scope.getPriceAsCurrency = function () {
        return $filter('currency')($scope.price);
    };

    $scope.getPriceAsNumber = function () {
        return $filter('number')($scope.price, 2);
    };
};
```

Do Not Duplicate

Example: Using the `$filter` Service

cont'd

```
$scope.getEndSaleAsLongDate = function () {
    return $filter('date')($scope.endSaleDate, 'MMMM d, yyyy');
};

$scope.getEndSaleAsAmericanDate = function () {
    return $filter('date')($scope.endSaleDate, 'MM-dd-yyyy');
};

$scope.getEndSaleAsUKDate = function () {
    return $filter('date')($scope.endSaleDate, 'dd-MM-yyyy');
};

$scope.commentsByStars =
    $filter('orderBy')($scope.comments, '-stars');

$scope.bestComments =
    $filter('filter')($scope.comments, {stars : 5});

$scope.bestCommentsByName = $filter('orderBy')(
    $filter('filter')($scope.comments, { stars: 5 }), 'name');
});
```

campersalesview.html

```
<body ng-app="myApp">
  <div ng-controller="CamperSalesController">

    <p>Product: {{ product }}</p>
    <p>Product in lowercase: {{ getProductAsLowercase() }}</p>
    <p>Product in uppercase: {{ getProductAsUppercase() }}</p>

    <hr />

    <p>Price: {{ price }}</p>
    <p>Price using currency: {{ getPriceAsCurrency() }}</p>
    <p>Price using number: {{ getPriceAsNumber() }}</p>

    <hr />

    <p>On sale until (month day, year):
        {{ getEndSaleAsLongDate() }}</p>
    <p>On sale until (monthNum-day-year):
        {{ getEndSaleAsAmericanDate() }}</p>
    <p>On sale until (day-monthNum-year):
        {{ getEndSaleAsUKDate() }}</p>

    <hr />
```

Do Not Duplicate

Example: Using the \$filter Service

cont'd

```
<p>Comments:</p>
<ul>
  <li ng-repeat="comment in bestCommentsByName">
    {{ comment }}
  </li>
</ul>
</div>

<script src="scripts/angular.min.js"></script>
<script src="scripts/controllers/CamperSalesController.js">
</script>
</body>
```

Product: 2011 rPod 179 Camper

Product in lowercase: 2011 rpod 179 camper

Product in uppercase: 2011 RPOD 179 CAMPER

Price: 12135.65

Price using currency: \$12,135.65

Price using number: 12,135.65

On sale until (month day, year): December 24, 2015

On sale until (monthNum-day-year): 12-24-2015

On sale until (day-monthNum-year): 24-12-2015

Comments:

- {"stars":5,"name":"D.H.,"comment":"The rPod is wonderful. Its fully functional. It lets us commune with nature."}
- {"stars":5,"name":"Deb E.,"comment":"The rPod is so easy to pull."}
- {"stars":5,"name":"Kathy W.,"comment":"I love our rPod. Its cozy and functional."}

Do Not Duplicate

Injecting Filters into the Controller

- A filter can be injected into a controller when it is registered
 - However, the name of the filter must have the word "Filter" appended

Example

```
app.controller('DemoController', function($scope, orderByFilter) {  
    // injects the orderBy filter into the controller  
});
```

- You can then invoke the filter directly without having to use the `$filter` service

Example

In this example, the items in `comments` have a property called `name`

```
$scope.orderedComments = orderByFilter($scope.comments, 'name');
```

Do Not Duplicate

More Examples

- There's a great discussion on complex filters at dzone.com/articles/angularjs-different-ways-using

Do Not Duplicate

Module Summary

- **In AngularJS, filters provide a convenient mechanism for transforming data**
- **There are several types of filters, including:**
 - filters that format data for output
 - filters that sort data
 - filters that restrict the data in a collection
- **Filters are added to an expression using the pipe (|) character and the filter**
 - Formatting filters include uppercase, lowercase, number, currency and date
 - The `orderBy` filter is used to sort data
 - The `limitTo` and `filter` filters restrict the data being returned
- **Filters can also be used in JavaScript using the `$filter` service**
 - Alternatively, single filters can be injected into a controller

Do Not Duplicate

Do Not Duplicate

Module 4

Directives

Do Not Duplicate

Module Goals and Objectives

Major Goals

Understand the purpose of a directive

Use common AngularJS built-in directives used for attaching event handlers

Modify the DOM using built-in directives

Discuss the differences between `ngIf` and `ngShow/ngHide`

Specific Objectives

On completion of this module, students will be able to:

Bootstrap an AngularJS application using `ngApp`

Use `ngSrc` to load an image using an AngularJS expression

Show and hide DOM elements using `ngShow` and `ngHide`

Change page content with `ngSwitch`

Use `ngClick` to attach a click event handler

Do Not Duplicate

Section 4–1

Custom HTML Elements and Attributes

Do Not Duplicate

HTML Elements and Attributes

- **HTML is a markup language that contains several predefined elements (or tags) that inform the browser how to display some information**
 - Text can be displayed in a paragraph form, or as a heading
 - Image files can be displayed in a particular location
- **Various predefined attributes can be added to HTML elements to define the behavior or characteristics of an element**
 - For example, the `style` attribute can be used to define an inline style and the `src` attribute is used to specify the image file to display in an `` element
- **HTML does not allow you to create your own elements**
 - If you were to define your own HTML element in a file, the browser would not format the content in any special manner
- **You do, however, have the ability to define custom attributes within an HTML tag**
 - The browser will not recognize custom attributes, but the attribute will be available to the DOM
- **If you run your code through an HTML validator, the page will not be considered valid**

Do Not Duplicate

Custom Attributes in HTML5

- **With the release of HTML5, you can now define custom attributes as long as they are prefixed with data-**
 - You can provide a string value to the attribute
 - Although many text editors may not consider the custom attributes to be valid, the code will validate in the HTML validator

Example

```
<div data-location="Dallas"></div>
```

- **AngularJS defines several custom attributes that are used in creating an AngularJS application**
 - These custom attributes provide the ability to bootstrap an AngularJS application, define event handlers and manipulate the DOM
- **The custom attributes that are provided by AngularJS are known as directives**
 - They are typically prefixed with `ng-`
 - * According to the AngularJS documentation, it is short and sounds like "angular"

Do Not Duplicate

Section 4–2

Overview of AngularJS Directives

Do Not Duplicate

Overview – Directives

- **Directives are one of the most important components of an AngularJS application**
- **Directives are used to extend the behavior of HTML and teach the browser new syntax**
 - Can be used to create custom HTML elements, attributes and classes
- **AngularJS comes with a number of built-in directives**
 - The majority of the AngularJS library are pre-defined directives
- **If a built-in directive does not suit your needs, you can define your own**
 - More on this later

Do Not Duplicate

What are Directives?

- **Directives are markers on a DOM element which attach a special behavior to it**
 - AngularJS directives can be used as elements, attributes, classes or comments
 - The majority of directives are defined as elements and attributes
- **We have already used several directives**
 - The `ngApp` directive bootstraps an AngularJS application
 - The `ngModel` directive binds a form field to a property of the scope
 - The `ngController` directive attaches a controller object to a view
 - The `ngRepeat` directive causes an element to be repeated for each item in a collection
- **AngularJS defines over 50 different directives which are available to an AngularJS application by default**
 - You can also define your own directives, use directives available in an external AngularJS module or use third-party directives

Do Not Duplicate

Why Use Directives?

- Directives are used to expand browser features using simple syntax
- For example, HTML does not have an accordion element
 - To use an accordion, you need the features of a 3rd party library such as jQuery or jQuery UI
 - * You can also build your own using JavaScript

Example

If we wanted to create an accordion using jQuery UI, we would need to select an element from the DOM and then call the `accordion()` method.

```
$(someElement).accordion();
```

- You could create an AngularJS directive to do this instead

Do Not Duplicate

Why Use Directives? *cont'd*

- **AngularJS directives are attached to DOM elements in HTML code**
 - AngularJS's approach to creating UI components is intuitive
 - In HTML code, we might see a directive used as an element or as an attribute

Example

The following code uses a directive named `hottAccordion` as an element.

```
<hott-accordion></hott-accordion>
```

The following code uses a directive named `hottAccordion` as an attribute.

```
<div hott-accordion></div>
```

Do Not Duplicate

AngularJS Directives

- **Directives are typically attached to the DOM as attributes**
 - They can also be attached using elements, classes or comments
 - * It depends on how a directive is defined
- **The built-in AngularJS directives are named with the `ng-` prefix**
 - You can also use a `data-` or `x-` prefix on the attribute to make it HTML5 compatible
 - Although less common, you can also use a colon (`:`) or underscore (`_`) in place of the dash (`-`) when attaching a directive

Example

The following example uses several valid naming conventions for the AngularJS `ngBind` directive.

ngbind alts.html

```
<label for="city">Enter city name:</label>
<input type="text" ng-model="city"><br />

<p>You entered: <br />
  <span ng-bind="city"></span><br />
  <span data-ng-bind="city"></span><br />
  <span ng:bind="city"></span><br />
  <span ng_bind="city"></span><br />
  <span x-ng-bind="city"></span><br />
</p>
```

Do Not Duplicate

AngularJS Directives *cont'd*

- We will use the **ng-** prefix to attach all built-in AngularJS directives
 - In this module, we will discuss built-in directives; in a future module, we will discuss how to build custom directives

Do Not Duplicate

Directives and the HTML Compiler

- **When an AngularJS web page is loaded into a browser, it is loaded in a raw state containing both HTML and template code**
 - The initial DOM is created based on that code
- **When the AngularJS application is bootstrapped, the HTML compiler traverses the DOM and compiles all directives**
 - Then, it links the directives by combining a directive with a scope to create a new live view
 - The live view contains the DOM elements and functionality defined in the directive
 - * Including any attributes values or event bindings
- **During compilation, HTML tags and attributes are normalized to support the fact that AngularJS is case sensitive (whereas HTML is not)**
 - The x- and data- prefixes are stripped from in front of elements and attributes
 - Names containing :, - or _ are converted to camelCase

Do Not Duplicate

AngularJS Built-in Directives

- **Built-in directives provide the majority of features for an AngularJS application**
 - These features include:
 - * Providing general AngularJS functionality
 - * Binding page elements to values in the `$scope`
 - * Binding page events to controllers
- **These directives are defined in the core AngularJS library and are available when the `angular.js` file is loaded by the browser**
- **The behavior of built-in AngularJS directives cannot be overridden**
 - However, you can create a custom directive with the same name and it is executed along with the built-in directive
 - A better approach would be to create a custom directive with a different name

Do Not Duplicate

Section 4–3

Core Functionality Directives

Do Not Duplicate

Directives for Core Functionality

- **Several built-in directives provide support for core AngularJS functionality**
 - They are used to bootstrap an AngularJS application or to ensure that expressions that are required by AngularJS are preserved in the DOM and include:

Directive	Description
ngApp	Used to auto-bootstrap an AngularJS application. This directive designates the root element of the application and is typically placed near the root element of the page, such as the <code><html></code> or <code><body></code> elements.
ngCloak	Used to prevent the AngularJS html template from being briefly displayed by the browser in its uncompiled form while the application is loading
ngController	Used to attach a controller class to the view
ngInclude	Fetches, compiles and includes an external HTML fragment.
ngList	Used to convert an <code>Array</code> object in the scope into a delimiter-separated string (defaults to a comma).
ngHref	Using AngularJS markup like <code>{{hash}}</code> in an <code>href</code> attribute will cause the link to visit the wrong URL is the user clicks it before AngularJS has a chance to replace the <code>{{hash}}</code> markup with its value. Until AngularJS replaces the markup, the link will be broken and will most likely return a 404 error. This directive solves this problem.
ngSrc	Using AngularJS markup like <code>{{hash}}</code> in a <code>src</code> attribute doesn't work correctly because the browser returns from the URL with the literal text <code>{{hash}}</code> until AngularJS replaces the expression inside <code>{{hash}}</code> . This directive solves this problem.
ngSrcset	Using AngularJS markup like <code>{{hash}}</code> in a <code>srcset</code> attribute doesn't work correctly because the browser returns from the URL with the literal text <code>{{hash}}</code> until AngularJS replaces the expression inside <code>{{hash}}</code> . This directive solves this problem.
ngDisabled	Sets the <code>disabled</code> attribute on the element is the expressions within the directive evaluates to truthy.

Do Not Duplicate

Directives for Core Functionality *cont'd*

<code>ngChecked</code>	Sets the <code>checked</code> attribute on the element if the expression within the directive evaluates to truthy.
<code>ngReadonly</code>	The HTML specification does not require browsers to preserve the values of Boolean attributes such as <code>readonly</code> , <code>selected</code> and <code>required</code> . (Their presence means true and their absence means false). If we place an AngularJS interpolation expression into one of these attributes, the binding information would be lost when the browser removes the attribute. These directives solve this problem for the appropriate attribute.
<code>ngSelected</code>	
<code>ngRequired</code>	
<code>ngPluralize</code>	Enables you to display messages according to the en-US localization rules bundled with AngularJS. This directive can be configured by adding <code>count</code> and <code>when</code> attributes.
<code>ngView</code>	Includes a rendered template of the current route into the main shell page

- We have used several of these directives (such as `ngApp`, `ngCloak` and `ngController`) to this point
 - We will demonstrate some of the others in the following pages

Do Not Duplicate

Example: Using ngSrc

Example

The following code uses the `ngSrc` directive to correctly load an image file. Although AngularJS does evaluate the expression in the `src` attribute eventually, you can see in the screenshot that the browser originally tried to load `{{img}}` (evaluates to `%7B%7Bimg%7D%7D`) and then correctly loads `london.jpg`. By using the `ngSrc` directive, the name of the file is interpolated before the request is made for the file.

scripts/ngSrc.js

```
var app = angular.module('imgApp', []);
app.controller('ImageController', function($scope) {
    $scope.caption = 'Photo of London';
    $scope.img = 'london.jpg';
});
```

ngsrc.html

```
<!DOCTYPE html>
<html ng-app="imgApp">
<head>
    <title>Using ngSrc</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
    <div class="container" ng-controller="ImageController">
        <div class="page-header text-center">
            <h1>Using ngSrc</h1>
        </div>
        <div class="row">
            <div class="col-xs-6">
                <h2>Using src Attribute</h2>
                
                <h2>Using ngSrc Directive</h2>
                
            </div>
        </div>
    </div>
    <script src="scripts/angular.min.js"></script>
    <script src="scripts/ngSrc.js"></script>
</body>
</html>
```

Do Not Duplicate

Example: Using ngSrc *cont'd*

Using src Attribute



Using ngSrc Directive



Network - Using ngSrc

6 requests, 307.75 KB, 0.35 s

Status	Method	File	Domain	Type	Trans...	Size	0 ms	160 ms	320 ms
304	GET	ngSrc.html	127.0.0.1:53549	html	0.77 KB	0.77 KB	→ 1 ms		
304	GET	bootstrap.min.css	127.0.0.1:53549	css	106.95 KB	106.95 KB	→ 2 ms		
404	GET	{{img}}	127.0.0.1:53549	plain	0.05 KB	0 KB	→ 4 ms		
304	GET	angular.min.js	127.0.0.1:53549	js	156.30 KB	156.30 KB	→ 2 ms		
304	GET	ngSrc.js	127.0.0.1:53549	js	0.17 KB	0.17 KB	→ 2 ms		
304	GET	london.jpg	127.0.0.1:53549	jpeg	—	43.57 KB	→ 2 ms		

Do Not Duplicate

Example: Using ngPluralize

Example

The following code uses ngPluralize to display a message based on the count of a value in the model. The directive's when attribute is used to specify the output based on the number of meeting attendees.

scripts/ngPluralize.js

```
var app = angular.module('meetingApp', []);
app.controller('MeetingController', function ($scope) {
    $scope.attendees = [];
});
```

ngpluralize.html

```
<!DOCTYPE html>
<html ng-app="meetingApp">
<head>
    <title>Using ngPluralize</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
    <div class="container" ng-controller="MeetingController">
        <div class="col-md-3">
            <p>Type in a list of comma ( , ) separated names.</p>
            <textarea class="form-control" ng-model="attendees"
ng-list=","></textarea>
            <pre>{{attendees|json}}</pre>
            <ng-pluralize count="attendees.length" offset="2"
when="{
    '0': 'There are currently no attendees.',
    '1': '{{attendees[0]}} is attending.',
    '2': '{{attendees[0]}} and {{attendees[1]}} are
        attending.',
    'one': '{{attendees[0]}}, {{attendees[1]}} and one other
        person are attending.',
    'other': '{{attendees[0]}}, {{attendees[1]}} and {}
        other people are attending.'}">
        </ng-pluralize>
        </div>
    </div>
    <script src="scripts/angular.min.js"></script>
    <script src="scripts/ngPluralize.js"></script>
</body>
</html>
```

Do Not Duplicate

Example: Using ngReadOnly and ngDisabled

Example

The following code uses the `ngReadOnly` and `ngDisabled` directives to allow the user to interact with a pair of form controls. The `ngReadOnly` directive is attached to a text field and is toggled using a button click. The `ngDisabled` directive is attached to a button and is toggled using the state of a checkbox.

scripts/ngReadOnly.js

```
var app = angular.module('disableApp', []);
app.controller('MainController', function($scope) {
    $scope.isReadOnly = false;
    $scope.status = 'write';
    $scope.toggle = function() {
        $scope.isReadOnly = !$scope.isReadOnly;
        if ($scope.status === 'readonly') {
            $scope.status = 'write';
        }
        else {
            $scope.status = 'readonly';
        }
    };
});
```

ngreadonly.html

```
<!DOCTYPE html>
<html ng-app="disableApp">
<head>
    <title>ngReadOnly and ngDisabled Directives</title>
    <style>
        .readonly {color:red;}
        .write {color:green;}
    </style>
</head>
<body ng-controller="MainController">
    <h3>ngReadOnly</h3>
    <div>
        <input type="text" ng-readonly="isReadOnly">
        <button ng-class="{true:'readonly',false:'write'}[isReadOnly]"
            ng-click="toggle()">{{status}}</button><br />
    </div>
```

Example: Using ngReadOnly and ngDisabled *cont'd*

```
<br/>
<h3>ngDisabled</h3>
<p>
  <button ng-disabled="dis">Click Me!</button>
</p>
<p>
  <label><input type="checkbox" ng-model="dis">
    Click to Disable the Button
  </label>
</p>
<script src="scripts/angular.min.js"></script>
<script src="scripts/ngReadOnly.js"></script>
</body>
</html>
```

ngReadOnly

ngDisabled

☐ Click to Disable the Button

ngReadOnly

ngDisabled

☒ Click to Disable the Button

Do Not Duplicate

Section 4–4

Data Binding Directives

Do Not Duplicate

Directives that Bind to the Model

- **AngularJS contains several directives that enable you to bind data in the scope to DOM elements**
- **Data can be bound in several ways, including:**
 - Using AngularJS expressions in an attribute definition
 - * Expressions might evaluate to a string, object or an array
 - Specifying expressions that evaluate to a truthy or false value
 - * For example, the `ngShow` and `ngHide` directives
- **Directives that are defined within the `ng` module include:**

Directive	Description
<code>ngBind</code>	Tells AngularJS to replace the text content of the specified HTML element with the value of a given expression, and to update the text content when the value of that expression changes.
<code>ngBindTemplate</code>	Specifies that the element text content should be replaced with the interpolation of the template in the <code>ngBindTemplate</code> attribute. Unlike <code>ngBind</code> , this directive can contain multiple <code>{{ }}</code> expressions. Required because some HTML elements cannot contain <code></code> elements.
<code>ngBindHtml</code>	Evaluates the expression and inserts the resulting HTML into the element in a secure way. By default, the resulting HTML content will be sanitized using the <code>\$sanitize</code> service. To utilize this functionality, you must be sure to include the <code>ngSanitize</code> module.
<code>ngClass</code>	Allows you to dynamically set CSS classes on an HTML element by databinding an expression that represents all classes to be added.
<code>ngClassOdd</code>	These directives work exactly like <code>ngClass</code> , but they work in conjunction with <code>ngRepeat</code> and take effect only on odd or even rows.
<code>ngClassEven</code>	
<code>ngIf</code>	Removes or recreates a portion of the DOM tree based on an expression. If the expression assigned to the directive evaluates to a false value, then the element is removed from the DOM; otherwise, a clone of the element is reinserted into the DOM.

ng Module Directives *cont'd*

Directive	Description
ngInit	Allows you to evaluate an expression in the current scope.
ngList	Text input that converts between a delimited string and an array of strings. The default delimiter is a comma, followed by a space.
ngModel	Binds an input, select, textarea or custom form control to a property on the scope using <code>NgModelController</code> .
ngShow	Shows or hides the given HTML element based on the provided expression. The element is shown or hidden by removing or adding the <code>ng-hide</code> CSS class onto the element. The <code>ng-hide</code> class is defined by AngularJS sets the <code>display</code> style to <code>none</code> (using an <code>!important</code> flag).
ngHide	
ngRepeat	Instantitates a template once per item from a collection. Each collection instance gets its own scope, where the loop variable is set to the current collection item and <code>\$index</code> is set to the item index or key.
ngStyle	Allows you to set CSS style on an HTML element conditionally.
ngSwitch	Used to conditionally swap DOM structure on your template based on a scope expression. Elements within <code>ngSwitch</code> but without <code>ngSwitchWhen</code> or <code>ngSwitchDefault</code> directives will be preserved at the location as specified in the template.
ngValue	Binds the given expression to the value of <code><option></code> or <code>input[radio]</code> , so that when the element is selected, the <code>ngModel</code> of that element is set to the bound values/

Example

The following code uses the `ngShow` and `ngClass` directives to display messages to the user when the entered username doesn't meet the requirements.

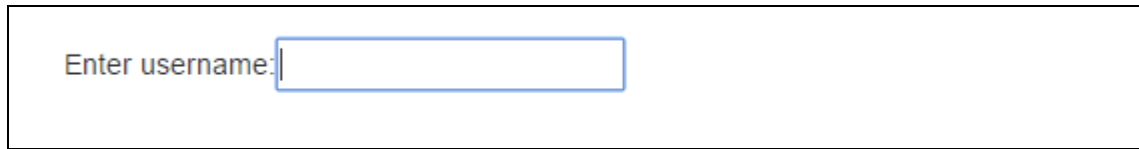
warnings.html

```
<form>
  Enter username:<input type="text" ng-model="username" />
</form>
<p>
  <div class="alert" ng-show="username.length === 0">
    Username is required!
  </div>
  <div class="alert" ng-class="{ 'alert-warning':username.length<5,
    'alert-danger':username.length>7}" ng-if="username.length < 5
    || username.length > 7">
    Username must be 5-7 characters
  </div>
</p>
```

Do Not Duplicate

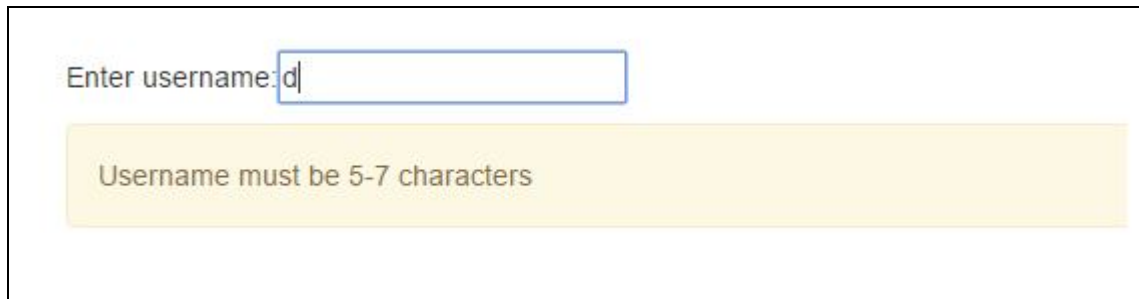
ng Module Directives *cont'd*

When the page first displays, no message is displayed.



Enter username:

When the user begins to enter the username, a message is displayed when the requirements are not met.



Enter username:

Username must be 5-7 characters

Do Not Duplicate

Example: Using ngBindHtml

Example

The following code demonstrates how content that includes HTML markup is rendered in a DOM element by default.

scripts/bindDemo.js

```
var app = angular.module('bindApp', ['ngSanitize']);
app.controller('MainController', function($scope) {
  $scope.cityImage = '';
});
```

ngbindhtml1.html

```
<!DOCTYPE html>
<html ng-app="bindApp">
<head>
<meta charset="utf-8">
<title>Default Rendering of HTML Markup</title>
<link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body ng-controller="MainController">
  <div class="container">
    <p style="margin:15px">{{cityImage}}</p>
  </div>
<script src="scripts/angular.min.js"></script>
<script src="scripts/bindDemo.js"></script>
</body>
</html>
```

``

Do Not Duplicate

Example: Using ngBindHtml *cont'd*

Example

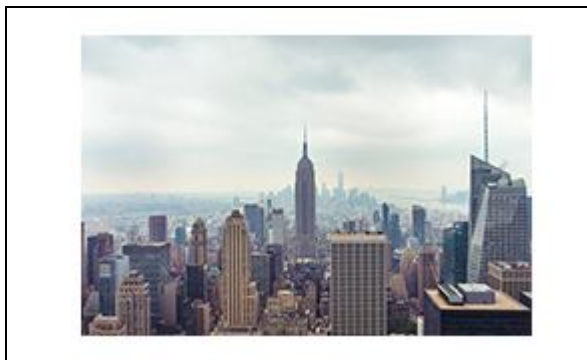
The following code demonstrates the use of the `ngBindHtml` directive to render HTML markup within a DOM element. The `ngSanitize` module must be included as a dependency to the module. Without the inclusion of `ngSanitize`, you will receive an exception because AngularJS will view the content as unsafe.

scripts/sanitizeDemo.js

```
var app = angular.module('bindApp', ['ngSanitize']);
app.controller('MainController', function($scope) {
    $scope.cityImage = '';
});
```

ngbindhtml2.html

```
<!DOCTYPE html>
<html ng-app="bindApp">
<head>
<meta charset="utf-8">
<title>Using ngBindHtml</title>
<link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body ng-controller="MainController">
    <div class="container">
        <p style="margin:15px" ng-bind-html="cityImage">
        </p>
    </div>
<script src="scripts/angular.min.js"></script>
<script src="scripts/angular-sanitize.min.js"></script>
<script src="scripts/sanitizeDemo.js"></script>
</body>
</html>
```



Do Not Duplicate

Example: Using ngShow

Example

The following code uses ngShow to display an image when the value entered into a text box matches the name of a city.

scripts/Cities.js

```
var app = angular.module('cityApp', []);
app.controller('CityController', function($scope){
    $scope.cities = ['london', 'new york', 'tokyo', 'paris', 'rio'];
});
```

ngshow.html

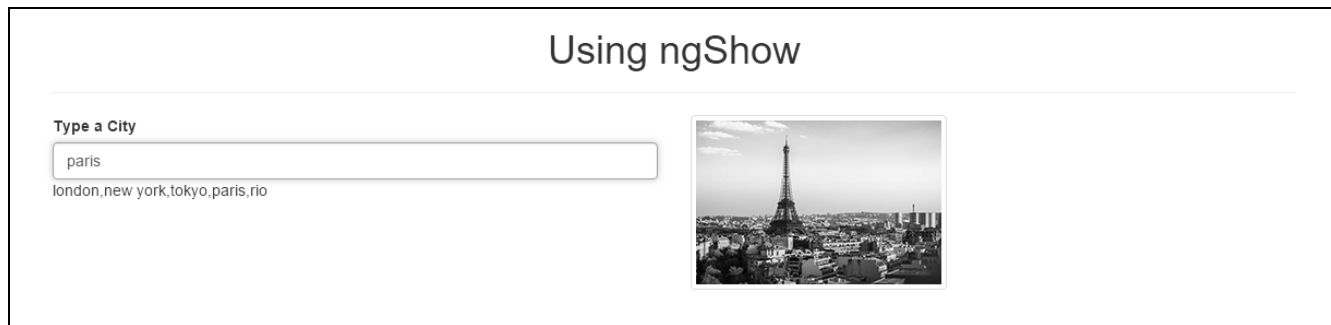
```
<!DOCTYPE html>
<html ng-app="cityApp">
<head>
    <title>Using ngShow</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
    <div class="container" ng-controller="CityController">
        <div class="page-header text-center">
            <h1>Using ngShow</h1>
        </div>

        <!-- Cities -->
        <div class="row">
            <div class="col-xs-6">
                <form>
                    <div class="form-group">
                        <label>Type a City</label>
                        <input type="text" class="form-control" ng-model="city">
                        <span ng-bind="cities"></span>
                    </div>
                </form>
            </div>
            <div class="col-xs-6">
                
                
                
```

Example: Using ngShow *cont'd*

```


</div>
</div>
</div>
<script src="scripts/angular.min.js"></script>
<script src="scripts/Cities.js"></script>
</body>
</html>
```



Do Not Duplicate

Using ngIf

- The **ngIf** directive removes or recreates a portion of the DOM tree based on an AngularJS expression
 - Used as an attribute
- If the expression assigned to **ngIf** evaluates to a false value, then the element is removed from the DOM; otherwise, a clone of the element is reinserted into the DOM

Example

In the following code, only one of the buttons will be present in the DOM at any one time. **ngIf** adds and removes the button based on the value of **mute**.

```
<button type="button" class="btn btn-default btn-lg" ng-if="!mute"
ng-click="muteToggle()" >
  <span class="glyphicon glyphicon-volume-off"></span> Muted
</button>
```

```
<button type="button" class="btn btn-default btn-lg" ng-if="mute"
ng-click="muteToggle()" >
  <span class="glyphicon glyphicon-volume-up"></span> Full Volume!
</button>
```

- **ngIf** is different from the **ngShow** and **ngHide** directives because it completely removes and recreates the element in the DOM
 - **ngShow** and **ngHide** set the `display` CSS property
 - This can be important when dealing with some of the CSS selectors that select elements based on their position in the DOM
 - * For example, `:first-child` and `:last-child`

Do Not Duplicate

Using `ngIf` *cont'd*

- When an element is removed using `ngIf`, its scope is destroyed and a new scope is created when the element is restored
 - The scope created within `ngIf` inherits from its parent scope
 - However, if `ngModel` is used within `ngIf` to bind to a JavaScript primitive defined in the parent scope, modifications to the variable within the child scope will override the value in the parent scope
- The `ngIf` directive recreates elements using their compiled state
 - For example, if an element's `class` attribute is directly modified after it has been compiled (possibly using jQuery's `addClass()` method) and then the element is removed
 - When `ngIf` recreates the element, the added class will be lost because the element is recreated using the original compiled state

Do Not Duplicate

Example: Using ngIf

Example

The following code demonstrates the use of the `ngIf` directive to add or remove an element from the DOM. When the value of `ngIf` evaluates to a truthy value, the element is added to the DOM; when it evaluates to a false, the element is removed.

scripts/ngIf.js

```
var app = angular.module('ifApp', []);
app.controller('MainController', function($scope) {
  $scope.mute = true;
  // Toggles the button
  $scope.muteToggle = function() {
    $scope.mute = !$scope.mute;
  };
});
```

ngif.html

```
<!DOCTYPE html>
<html ng-app="ifApp">
  <head>
    <meta charset="utf-8" />
    <title>Using ngIf</title>
    <link rel="stylesheet" href="css/bootstrap.min.css" />
  </head>
  <body ng-controller="MainController">
    <div class="container">
      <button type="button" class="btn btn-default btn-lg"
        ng-if="!mute" ng-click="muteToggle()" >
        <span class="glyphicon glyphicon-volume-off"></span> Muted
      </button>
      <button type="button" class="btn btn-default btn-lg"
        ng-if="mute" ng-click="muteToggle()" >
        <span class="glyphicon glyphicon-volume-up"></span>
        Full Volume!
      </button>
      <p>{{scope.mute}}</p>
    </div>
    <script src="scripts/angular.min.js"></script>
    <script src="scripts/ngIf.js"></script>
  </body>
</html>
```

Do Not Duplicate

Using ngSwitch

- The `ngSwitch` directive can be used to add or remove HTML elements from the DOM conditionally based on a scope expression
 - Used as an attribute
- There are three directives involved in the use of `ngSwitch`: `ngSwitch`, `ngSwitchWhen` and `ngSwitchDefault`
 - The `ngSwitch` directive should be added to a container element and set the expression that acts as a selection condition
 - Then, `ngSwitchWhen` directives should be added for each child element that should be added or removed from the DOM specifying the condition
 - * If the condition is matched, the element is included in the DOM
 - * If the same match appears multiple times, all elements will be displayed
 - The `ngSwitchDefault` directive is used on the element that will be included in the DOM if there is no match among the `ngSwitchWhen` options
- **WARNING: When matching against attribute values, they cannot be expressions**
 - They are interpreted as literal string values
- Elements within `ngSwitch`, but without `ngSwitchWhen` or `ngSwitchDefault` directives will be preserved at the location specified in the template

Example: Using ngSwitch

Example

The following uses ngSwitch to render a div based on the value of showNumber.

ngswitch.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Using ngSwitch</title>
  <link rel="stylesheet" href="css/bootstrap.min.css" />
</head>
<body ng-app>
  <div class="container">
    <br>
    <p>
      <label>Type the number you want to show (1 to 5):
        <input type="text" ng-model="showNumber" />
      </label>
    </p>
    <div ng-switch="showNumber">
      <div ng-switch-when="1" style="width: 50px;
        background-color: red; text-align: center;">1
      </div>
      <div ng-switch-when="2" style="width: 50px;
        background-color: green; text-align: center;">2
      </div>
      <div ng-switch-when="3" style="width: 50px;
        background-color: yellow; text-align: center;">3
      </div>
      <div ng-switch-when="4" style="width: 50px;
        background-color: fuchsia; text-align: center;">4
      </div>
      <div ng-switch-when="5" style="width: 50px;
        background-color: orange; text-align: center;">5
      </div>
      <div ng-switch-default style="width: 50px;
        background-color: lightgray; text-align: center;">None
      </div>
    </div>
  </div>
  <script src="scripts/angular.min.js"></script>
</body>
</html>
```

Do Not Duplicate

Using ngClass

- The **ngClass** directive allows you to dynamically set CSS classes on an HTML element by databinding an expression that represents all classes to be added
 - Used as an attribute
 - Duplicate classes are not added
- The **ngClass** directive operates differently depending on the type to which the expression evaluates:
 - If the expression evaluates to a string, the string should be one or more space-delimited class names
 - If the expression evaluates to an object, then each key/value pair of the object is evaluated
 - * Each key that has a truthy value is then used as a class name
 - If the expression evaluates to an array, each element of the array should either be a string or an object
 - * Strings and objects can be mixed together in an array to provide more control
- When the expression changes, the previously added classes are removed and only then are the new classes added
- To view several examples of using **ngClass**, visit <https://scotch.io/tutorials/the-many-ways-to-use-ngclass>

Do Not Duplicate

Example: Using ngClass

Example

The following code uses the `ngClass` directive to apply a CSS class if a checkbox is checked. If the checkbox is checked, the `hasError` value is assigned a `true` or `false`. The `ngClass` evaluates the value of `hasError` and applies the appropriate class based on the `true` or `false` value.

ngclass.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Using ngClass</title>
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <style type="text/css">
    .hasError {
      background-color: yellow;
    }
    .hasSuccess {
      background-color: white;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="well col-md-3">
      <p>Has Error <input type="checkbox" ng-model="hasError" /></p>
      <label>First Name</label>
      <input type="text" class="form-control"
        ng-class="hasError ? 'hasError': 'hasSuccess'" />
      <br />
      <label>Last Name</label>
      <input type="text" class="form-control"
        ng-class="hasError ? 'hasError': 'hasSuccess'" />
    </div>
  </div>
  <script src="scripts/angular.min.js"></script>
</body>
</html>
```

Do Not Duplicate

Example: Using ngClassEven and ngClassOdd

Example

The `ngClassOdd` directive operates similarly to `ngClass`, but takes effect only on odd rows. The following demonstrates the use of the `ngClassOdd` directive to apply a CSS class to only the odd-numbered rows.

scripts/ngClassOdd.js

```
var app = angular.module('classOddApp', []);
app.controller('WeekdayController', function($scope) {
    $scope.weekdays = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
                        'Thursday', 'Friday', 'Saturday'];
});
```

ngclassodd.html

```
<!DOCTYPE html>
<html ng-app="classOddApp">
  <head>
    <meta charset="utf-8">
    <title>ngClassOdd Directive</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <style>
      .odd {
        background: #C32;
        color: #FFF;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h3>Using ngClassOdd in AngularJS</h3>
      <div ng-controller="WeekdayController">
        <div ng-repeat="wd in weekdays" ng-class-odd="'odd'">
          {{wd}}
        </div>
      </div>
    </div>
    <script src="scripts/angular.min.js"></script>
    <script src="scripts/ngClassOdd.js"></script>
  </body>
</html>
```

Do Not Duplicate

Using ngStyle

Example

The following code uses `ngStyle` to apply a background color to a heading tag. The color is initialized to silver using `ngInit`, but when a radio button is clicked, the color is changed.

ngstyle.html

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Using ngStyle</title>
  <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
  <div class="container" ng-init="color='silver'">
    <h1 ng-style="{background:color}">Tours</h1>
    <p>Pick a Season:</p>
    <form>
      <label><input type="radio" ng-model="color" name="color"
        value="#ff66ff">Spring</label><br>
      <label><input type="radio" ng-model="color" name="color"
        value="#00cc00">Summer</label><br>
      <label><input type="radio" ng-model="color" name="color"
        value="#ff9933">Fall</label><br>
      <label><input type="radio" ng-model="color" name="color"
        value="#99ccff">Winter</label><br>
    </form>
  </div>
  <script src="scripts/angular.min.js"></script>
</body>
</html>
```



Section 4–5

Event Handling Directives

Do Not Duplicate

Binding Page Events with Directives

- **AngularJS contains several directives that bind browser events to code in a controller**
 - This means that user interaction can be handled and linked to the scope context
- **The following directives can be used to bind page events to the AngularJS model**
 - Each of these directives enable you to specify an expression, such as a function defined in the scope

Directive	Description
ngBlur	Used to specify custom behavior on <code>blur</code> event
ngChange	Evaluates the given expression when the user changes input to a form element. The expression is evaluated immediately, unlike the JavaScript <code>onchange</code> event which only triggers at the end of a change (usually when the user shifts focus from a field).
ngChecked	Allows you to evaluate an expression when a checkbox or radio button is checked
ngClick	Allows you to specify custom behavior when an element is clicked
ngDbClick	Allows you to specify custom behavior on a <code>dblclick</code> event
ngFocus	Used to specify custom behavior on <code>focus</code> event
ngMouseDown	Allows you to specify custom behavior on <code>mousedown</code> event
ngMouseup	Used to specify custom behavior on <code>mouseup</code> event
ngMouseover	Used to specify custom behavior on <code>mouseover</code> event
ngMouseenter	Used to specify custom behavior on <code>mouseenter</code> event
ngMouseleave	Used to specify custom behavior on <code>mouseleave</code> event
ngMouseMove	Used to specify custom behavior on <code>mousemove</code> event
ngKeydown	Used to specify custom behavior on <code>keydown</code> event
ngKeyup	Used to specify custom behavior on <code>keyup</code> event
ngKeyPress	Used to specify custom behavior on <code>keypress</code> event

Do Not Duplicate

ng Module Directives *cont'd*

Directive	Description
ngCopy	Used to specify custom behavior on copy event
ngCut	Used to specify custom behavior on cut event
ngPaste	Used to specify custom behavior on paste event
ngSubmit	Enables binding AngularJS expressions to onsubmit events

- **Event handlers can also be attached using JavaScript and jqLite code**
- **We will cover many of these directives when forms are discussed**

Do Not Duplicate

Accessing Event Information

- You can pass the JavaScript event object into event expressions by using the **\$event** keyword
- **\$event** enables you to access information about the event

Example

The following code uses the `ngKeyPress` directive within a text field to capture the key that was pressed. When the user enters a non-numeric key, the keypress is ignored and a message is displayed in the console.

scripts/PreventDefault.js

```
var app = angular.module('eventApp', []);
app.controller('EventController', function($scope,$log){
    $scope.message = '';

    $scope.captureKey = function($event) {
        if(isNaN(String.fromCharCode($event.keyCode)))
        {$scope.message= 'Non-Numeric Key Pressed -
            Action Cancelled';
            $event.preventDefault();
        }
        else
        {
            $scope.message = '';
        }
    };
});
```

Do Not Duplicate

Accessing Event Information *cont'd*

preventdefault.html

```
<!DOCTYPE html>
<html ng-app="eventApp">
<head>
  <title>Canceling a Default Action with preventDefault()</title>
  <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
  <div class="container" ng-controller="EventController">
    <label>Enter a number into the text box</label>
    <input type="text" ng-keypress="captureKey($event)">
    <p>{{message}}</p>
  </div>

  <script src="scripts/angular.min.js"></script>
  <script src="scripts/PreventDefault.js"></script>
</body>
</html>
```

Do Not Duplicate

Using ngChange

- **The ngChange directive evaluates a given expression when the user changes the input**
 - Used as an attribute
- **The ngChange expression evaluates immediately**
 - Unlike the JavaScript `onchange` event which triggers at the end of a change (such as when the focus leaves a field)
- **The ngChange expression is only evaluated when a change in the input values causes a new value to be committed to the model**
- **The expression is not evaluated if:**
 - the value returned from the `$parsers` transformation pipeline has not changed
 - the input has continued to be invalid since the model will remain `null`
 - the model is changed programmatically and not by a change to the input value

Do Not Duplicate

Using ngClick

- The **ngClick** directive allows you to specify custom behavior when an element is clicked
 - Used as an attribute

Example

The following code uses the **ngClick** directive to increment a counter each time a button is clicked. The current value of the counter is then displayed in a `` element.

ngclick.html

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Using ngClick</title>
  <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <button ng-click="count = count + 1" ng-init="count=0">
      Increment
    </button>
    <span>
      count: {{count}}
    </span>
  </div>
  <script src="scripts/angular.min.js"></script>
</body>
</html>
```

Do Not Duplicate

Module Summary

- **Directives are among the most important features of AngularJS**
- **Directives are used to extend the behavior of HTML and teach the browser new syntax**
 - They can also be attached using elements, attributes, classes or comments
 - * It depends on how a directive is defined
- **AngularJS comes with a number of built-in directives**
 - Named using the `ng-` prefix
- **If necessary, you can also build your own custom directives**
- **Built-in directives provide the majority of features for an AngularJS application**
 - Directives such as `ngApp` and `ngController` provide general AngularJS functionality
 - Directives such as `ngModel` and `ngBind` bind page elements to values in the `$scope`
 - Directives such as `ngChange` and `ngClick` bind page events to controllers

Do Not Duplicate