Usage Guidelines

Do not forward this document to any non-Infosys mail ID. Forwarding this document to a non-Infosys mail ID may lead to disciplinary action against you, including termination of employment.

Contents of this material cannot be used in any other internal or external document without explicit permission from ETA@infosys.com.

Angular JS

Day 2

Education, Training and Assessment

We enable you to leverage knowledge anytime, anywhere!

Infosys® | Building Tomorrow's Enterprise

Copyright Guideline

Confidential Information

- This Document is confidential to Infosys Limited. This document contains information and data that Infosys considers confidential and proprietary ("Confidential Information").
- Confidential Information includes, but is not limited to, the following:
  - Corporate and Infrastructure information about Infosys
  - Infosys' project management and quality processes
  - Project experiences provided included as illustrative case studies
- Any disclosure of Confidential Information to, or use of it by a third party, will be damaging to Infosys.
- Ownership of all Infosys Confidential Information, no matter in what media it resides, remains with Infosys.
- Confidential information in this document shall not be disclosed, duplicated or used – in whole or in part – for any purpose other than reading without specific written permission of an authorized representative of Infosys.
- This document also contains third party confidential and proprietary information. Such third party information has been included by Infosys after receiving due written permissions and authorizations from the party/ies. Such third party confidential and proprietary information shall not be disclosed, duplicated or used – in whole or in part – for any purpose other than reading without specific written permission of an authorized representative of Infosys.

Course Information

Course Code : LA1211

Course Name : AngularJS

Version Number : 1.2

Day2 Objectives

- Scopes in Angular
  - Scope
  - Scope hierarchies and their evaluation
  - Scope events
  - Scope LifeCycle
- Angular Runtime
- Angular Expressions
- Directives
- Filters
  - Adding built-in filters

References

- Brad Green, Shyam Seshadri, AngularJS , O'Reilly Media, 2013.
- http://docs.Angularjs.org/guide/
- https://github.com/Angular

Scopes in Angular

Scope

- $scope is a simple JavaScript Object and glues the view and controller.
- Any property on $scope object can be used in the view expressions
- $scope is each instance per controller
- Following directives automatically create scope:
  - ng-app (Root Scope) - ng-controller - ng-repeat
- Scopes are usually connected via a hierarchy & the child scope has access to parent scope properties
- A model belongs to one and only one scope.
- Expressions are always evaluated in the local scope.
- $rootScope is one instance per an Angular app and shared globally for all the controllers of the application

Revisiting Checklist Case study

- Lets revisit the checklist case study implemented on Day1.

```
<!doctype html>

<html ng-app>

<head>

<!—code for including gular library,
controllers_casestudy ile and
style_casestud

</he

<bod

<center><he
Shopping</h2></

<div ng-controll mCtrl">

<h4>Please c k the items as and when you
finish buyi

<span
left</spa
```

```
<table>

<th>Status</th><th>Item</th>

<tr ng-repeat="item in items">

<td> <input type="checkbox"  ng-model="item.done"> </td>
<td> {{item.text}} </td>

</tr>

able> <br

<div class="ad

<form ng-submit="addItem()

<input type="text" ng-m l="itemText" class="txt"
placeholder="Add new it hecklist"> <br><br>

button  type="sub to
checklist</button

</form></div></div
```

Controller for the div

remaining() is a function defined in the controller and items is a model

The text input is set as a model with name itemText

A new row is created for every item in items array

The input is set as a model with name item.done. Hence whenever checkbox input changes, item.done value changes

ng-submit stops the default action of the form and executes the addItem() function

Revisiting Checklist Case study: controllers_casestudy.js

```
function ItemCtrl($scope) {
 $scope.items = [
   {text:'Wheat Flour', done:true},
   {text:'Toothpaste', done:false}
];
 $scope.remaining = function() {
   var count = 0;
                 angular.forEach($scope.items, function(item) {
                if(item.done==false)
                              count=count+1;
                });   return count;
  };
$scope.addItem = function() {
   $scope.items.push({text:$scope.itemText, done:false});
   $scope.itemText = '';    };
```

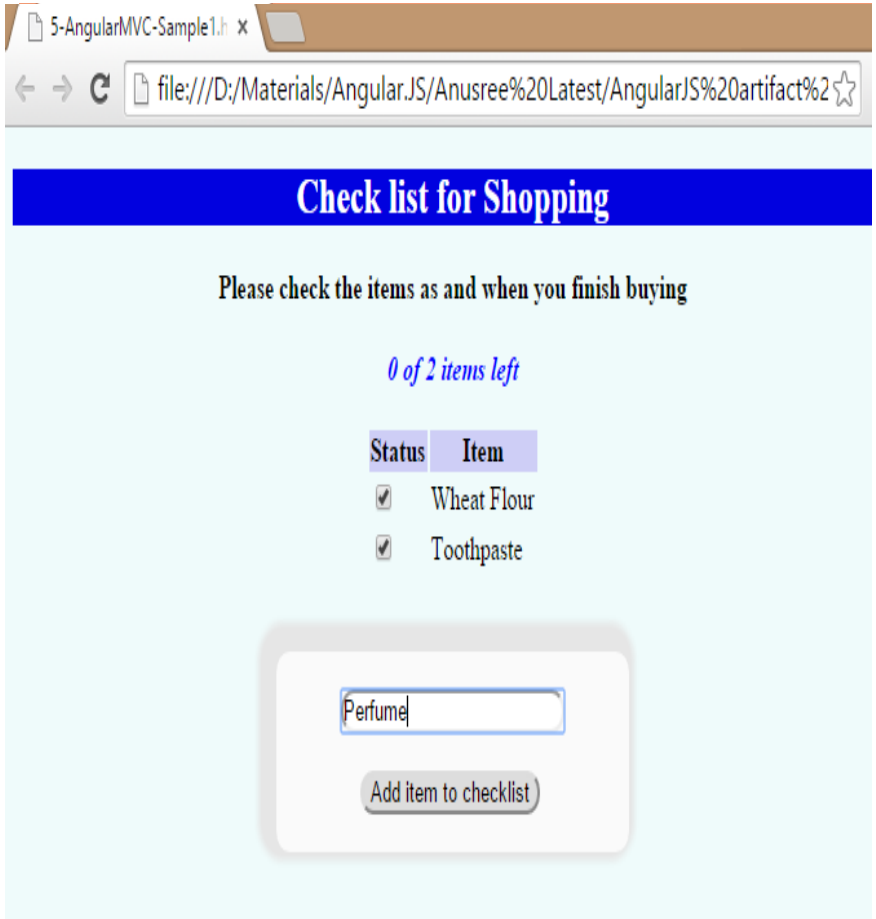
Revisiting Checklist Case study: styles_casestudy.css

```
body
{
background-color:#EFFBFB;
}
header
{
background-color:#0101DF;
 color: white;
}
span
{
font-style:italic;
color:blue;
font-weight:bolder;
}
```

```
.button
{
border-radius:10px;
}
.txt
{
border-radius:10px;
}
th
{
    background-color: #CECEF6;
}
```

```
.add
{
    Border-radius: 10px;
    background-color: #FAFAFA;
    width:40%;
    box-shadow: -1px -1px 2px 15px #FBFBEF;
    -webkit-box-shadow:-5px -5px 5px 9px #E6E6E6;
    -moz-box-shadow:0px 0px 3px 3px #FBFBEF;
}
```

Revisiting Checklist Case study



Understanding Scope

- Lets modify controllers file to include two controller functions having similar code.

```
function ItemCtrl1($scope) {
$scope.items = [
  {text:'Wheat Flour', done:true},
  {text:'Toothpaste', done:false} ];
 $scope.remaining = function() {
var count = 0;
angular.forEach($scope.items, function(item) {
if(item.done==false)
            count=count+1;
});
return count;
  };
}
```

```
function ItemCtrl2($scope) {
 $scope.items = [
  {text:'Pen', done:true},
  {text:'Pencil', done:false},
  {text:'Book', done:false},
  {text:'Crayon', done:false}];
 $scope.remaining = function() {
var count = 0;
angular.forEach($scope.items, function(item) {
if(item.done==false)
            count=count+1;
});
return count;  };
}
```

Both controller functions have the

Values assigned to *items* model is different in the controllers

Understanding scope

- Lets update the body of the html to include two separate which uses the controller functions

```
<body><center><header><h2>Check list for Shopping</h2></header>

<div ng-controller="ItemCtrl1">

<h4>List1:Please check the items as and when you finish buying</h4>

<span>{{remaining()}} of {{items.length}} items left</span><br>

<br>

<table>

<th>Status</th><th>Item</th>

<tr ng-repeat="item in items">

<td> <input type="checkbox"

 ng-model="item.done"> </td>

<td> {{item.text}} </td>

</tr></table> </div><br><br>
```

```
<div ng-controller="ItemCtrl2">

<h4>List2:Please check the items as and when you finish buying</h4>

<span>{{remaining()}} of {{items.length}} items left</span><br>

<br>

<table>

<th>Status</th><th>Item</th>

<tr ng-repeat="item in items">

<td> <input type="checkbox"  ng-model="item.done"> </td>

<td> {{item.text}} </td>

</tr>

</table>

</div></center></body>
```

Understanding scope

Scope1.html

file:///D:/Materials/Angular.JS/Anusree%20Latest/AngularJS%20artifact%2

Check list for Shopping

List1:Please check the items as and when you finish buying

1 of 2 items left

Status	Item
☑	Wheat Flour
☐	Toothpaste

List2:Please check the items as and when you finish buying

3 of 4 items left

Status	Item
☑	Pen
☐	Pencil
☐	Book
☐	Crayon

Scope Hierarchies and their evaluation

- Every Angular application has only one root scope.
- There can be several child scopes, however.
- Creation of multiple scopes is also supported by Angular as some directives create new child scopes.
- New scopes automatically become children of their parent scope. This will create a tree structure. Which will be parallel to the DOM.
- On encountering {{ emailId }}, the scope associated with the given element is checked.
- If such a property is not found, Angular searches the parent scope and moves on until the root scope is reached.

Understanding Scope Hierarchies

- Lets update the controller functions inside controllers file as below.

```
function ItemCtrl1($scope) {

$scope.title="List1:Please check the items as
and when you finish buying";

 $scope.items = [

 {text:'Wheat Flour', done:

 {text:'Toothpaste', done:false}]

 $scope.remaining = function(

var count = 0;

angular.forEach(items, function(item) {

 if(item.done==false)

            count=count+1;

});

return count;  };
```

```
function ItemCtrl2($scope) {


$scope.title="List2:Please check the items as
and when you finish buying";


.items = [

', done:true},

cil', done:false},

{text:'Book', done:false},

{text:'Crayon', done:false}

];
```

New *title* model has been added. *title* model contains the respective titles

remaining function is defined only inside ItemCtrl1

Understanding Scope Hierarchies

- Lets update the body of the html to include two nested which uses the controller functions

```
<body><center><header><h2>Check list for Shopping</h2></header>

<div ng-controller="ItemCtrl1">

<h4>{{title}}</h4>

<span>{{remaining()}} of {{items.length}} items left</span><br>

<br>

<table>

<th>Status</th><th>Item</th>

<tr ng-repeat="item in items">

<td> <input type="checkbox"

 ng-model="item.done"> </td>

<td> {{item.text}} </td>

</tr></table> <br><br>
```

Outer div

```
<div ng-controller="ItemCtrl2">

<h4>{{title}}</h4>

<span>{{remaining()}} of {{items.length}} items left</span><br>

<br>

<table>

<th>Status</th><th>Item</th>

<tr ng-repeat="item in items">

<td> <input type="checkbox"  ng-model="item.done"> </td>

<td> {{item.text}} </td>

</tr></table>

</div>

</div>

</center></body>
```

Inner div

Respective Controller functions

Understanding Scope Hierarchies

Scope2.html

file:///D:/Materials/Angular.JS/Anusree%20Latest/AngularJS%20art

Check list for Shopping

List1:Please check the items as and when you finish buying

1 of 2 items left

Status	Item
☑	Wheat Flour
☐	Toothpaste

List2:Please check the items as and when you finish buying

0 of 4 items left

Status	Item
☑	Pen
☑	Pencil
☑	Book
☑	Crayon

Remaining function is made available in the inner div via scope inheritance.

Understanding rootScope

- Lets update the controller functions inside controllers file as below.

```
function ItemCtrl1($scope,$rootScope) {

$rootScope.title="Please check the items as a
when you finish buying";

 $scope.items = [

   {text:'Wheat Flour', done:true},

   {text:'Toothpaste', done:false}];

 $scope.remaining = function(items) {

   var count = 0;

angular.forEach(items, function(item) {

                if(item.done==false)

                                count=count+1;

                });

  return count;

 };  }
```

rootScope passed to ItemCtrl1

title set in rootScope

```
Ctrl2($scope) {

$scope.items = [

xt:'Pen', done:true},

   {text:'Pencil', done:false},

                {text:'Book', done:false},

                {text:'Crayon', done:false}

];


}
```

Understanding rootScope

RootScope.html

file:///D:/Materials/Angular.JS/Anusree%20Latest/AngularJS%20art

Check list for Shopping

Please check the items as and when you finish buying

1 of 2 items left

Status	Item
☑	Wheat Flour
☐	Toothpaste

Please check the items as and when you finish buying

2 of 4 items left

Status	Item
☑	Pen
☑	Pencil
☐	Book
☐	Crayon

title is retrieved from rootScope inside both the controllers

Scope LifeCycle

1. Creation
   - During application bootstrapping, the root scope is created by the $injector.
   - New child scopes are created by some directives during template linking.

2. Watcher registration
   - Watches are registered during template linking on the scope.
   - These watches are used to send model values to the DOM.

3. Model mutation
   - Mutations work properly only when applied within the scope.$apply(), which is by default.

Scope LifeCycle

4. Mutation observation
   - Angular runs $digest cycle on the root scope, after $apply.
   - During the $digest cycle, model mutations are verified for all $watched expressions.
   - If a mutation is found, the $watch listener is called.

5. Scope destruction
   - All child scopes which are not needed any longer, are destroyed via scope.$destroy() API.
   - On destroying scope, the propagation of $digest calls to the child scope can be avoided.
   - This ensures that memory used by these child scopes can be garbage collected.

$watch

- $scope.$watch() allows us to watch any expression/function for a change.

```
$scope.$watch( expression/function, listener, [objectEquality])
```

- If the value of the expression/function changes the listener method is invoked

```
$scope.$watch("name",function(newVal,oldVal){
        console.log('newVal = '+newVal+' | oldVal = '+oldVal);

    });
```

$watch-demonstration

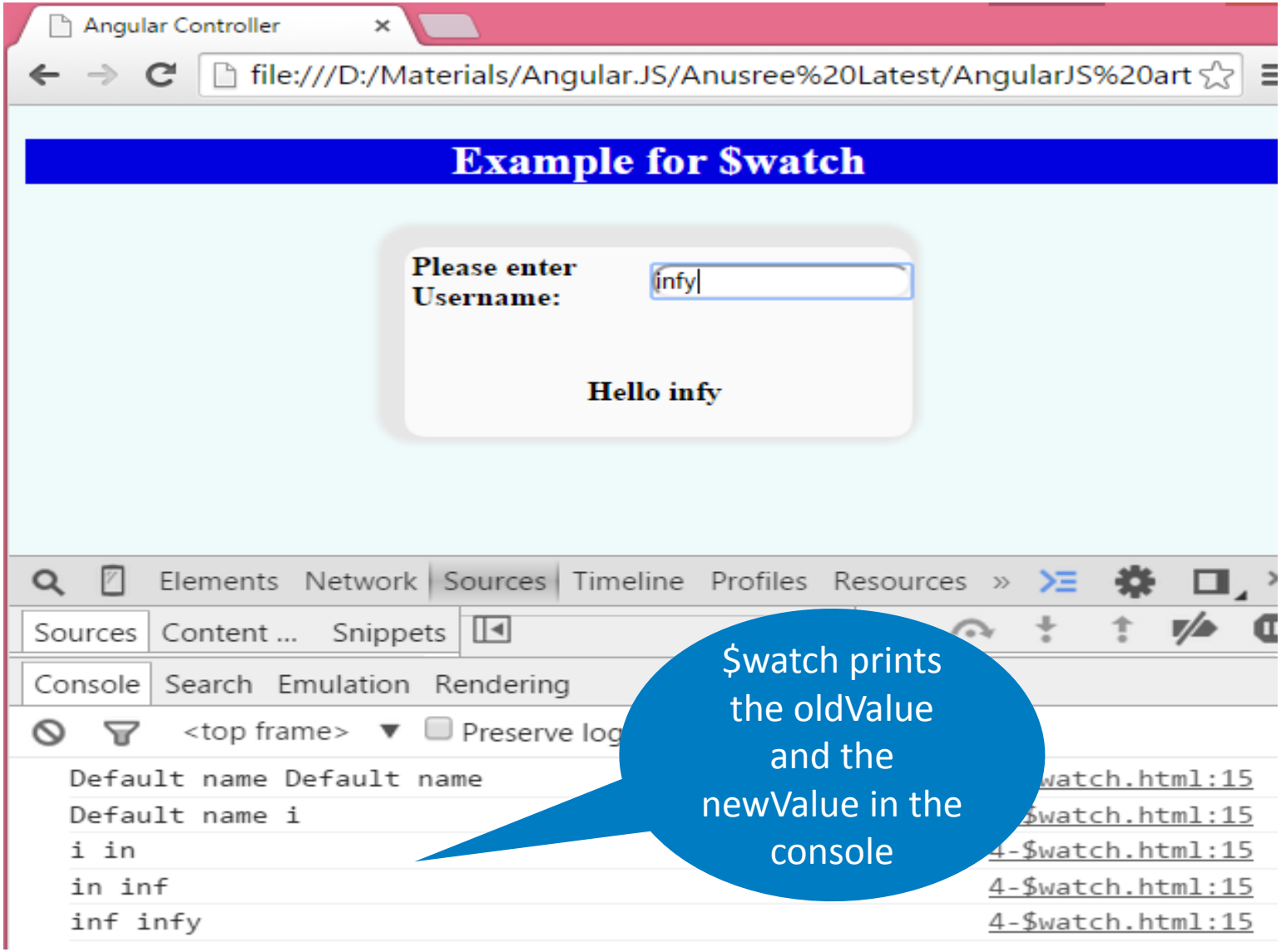
- Lets modify the first example on Angular controllers to watch for the changes in *username* model.

```
<!doctype html>

<html ng-app>

<head>  <title>Angular Controller</title>

<!—code to include stylesheet and Angular lib →

  <script type="text/javascript">

function MainController ($scope)

{

$scope.username="Default name";

$scope.$watch(function(scope) { return scope.username },

            function(newValue, oldValue) {

                console.log(oldValue + " "+newValue );

            }

  );          }              </script> </head>
```

Associating $watch to listen to value changes

```
<body><center><header>

 <h2> {{ 'Example for '+' $watch' }}
</h2></header><br>

 <div class="LoginFormDiv"

ng-controller="MainController">

<table >

<td><b>Please enter Username:</b></td>

<td><input type="text" ng-model="username" ></td>

</tr> </table>  <br><br>

<b> Hello {{username}} </b><br><br>

  </div>

</center></body></html>
```

$watch-demonstration



Scope events-$emit & $broadcast

- An event can be sent to all the descendant scopes using $broadcast

```
$scope.$broadcast('event-name',data);
```

- Similarly an event can be sent to all the ancestor scopes by using $emit

```
$scope.$emit('event-name',data);
```

- Events can be captured by using $on

```
$scope.$on(' event-name ',function(event, data){
    ...
  });
```

Scope events


```
<!doctype html>

<html ng-app>

<head>  <title>Angular Controller</title>

  <style type="text/css">

        @import "styles/style.css";

      </style>

<script src="lib/Angular/angular.js"></script>

<script type="text/javascript">

function EventController($scope) {

$scope.username="Default name";

$scope.$on('Event', function() {

   $scope.username="New name";

});              }  </script></head>

<body><center><header>

<h2> {{ 'Example for '+' Scope Events' }}
</h2> </header><br>
```

Parent div

Child div

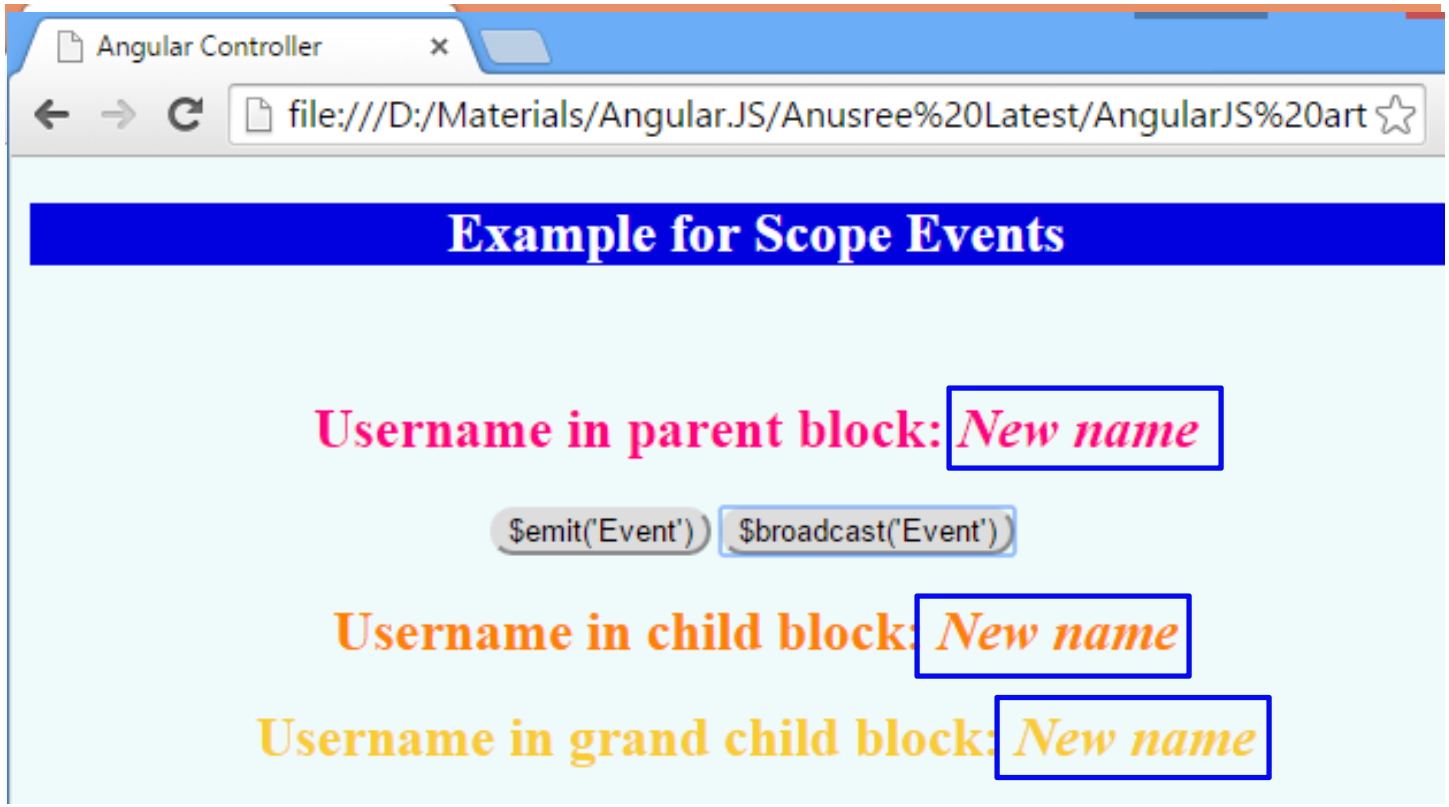
Grand Child div

```
<div ng-controller="EventController">

  <span style="color:#FF0080">

<h2>Username in parent block:
<i>{{username}}</i></h2></span>

  <div  ng-controller="EventController">

  <button ng-click="$emit('Event')"
  class="button">$emit('Event')</button>

  <button ng-click="$broadcast('Event')"
  class="button">$broadcast('Event')</button>

  <br><span style="color:#FF8000"><h2>Username in child
  block: <i>{{username}}</i></h2></span>

     <div ng-controller="EventController">

          <span style="color:#FACC2E"><h2>Username in
          grand child block: <i>{{username}}</i></h2></span>

     </div>

  </div>

</div>

</center></body></html>
```

Emits to all ancestor scopes

Broadcats to all the descendant scopes

Scope events



Angular Runtime

How AngularJS works-Runtime

- We can bind data to the web page very easily using the structure {{ my data }}.
- The $scope service takes care of detecting changes which happen in the model and update the HTML expressions accordingly in the view, with the help of controllers.
- According to changes which happen for the view, Angular ensures that the changes are also updated in the model. This means we need not write the majority of data-centric DOM manipulations.

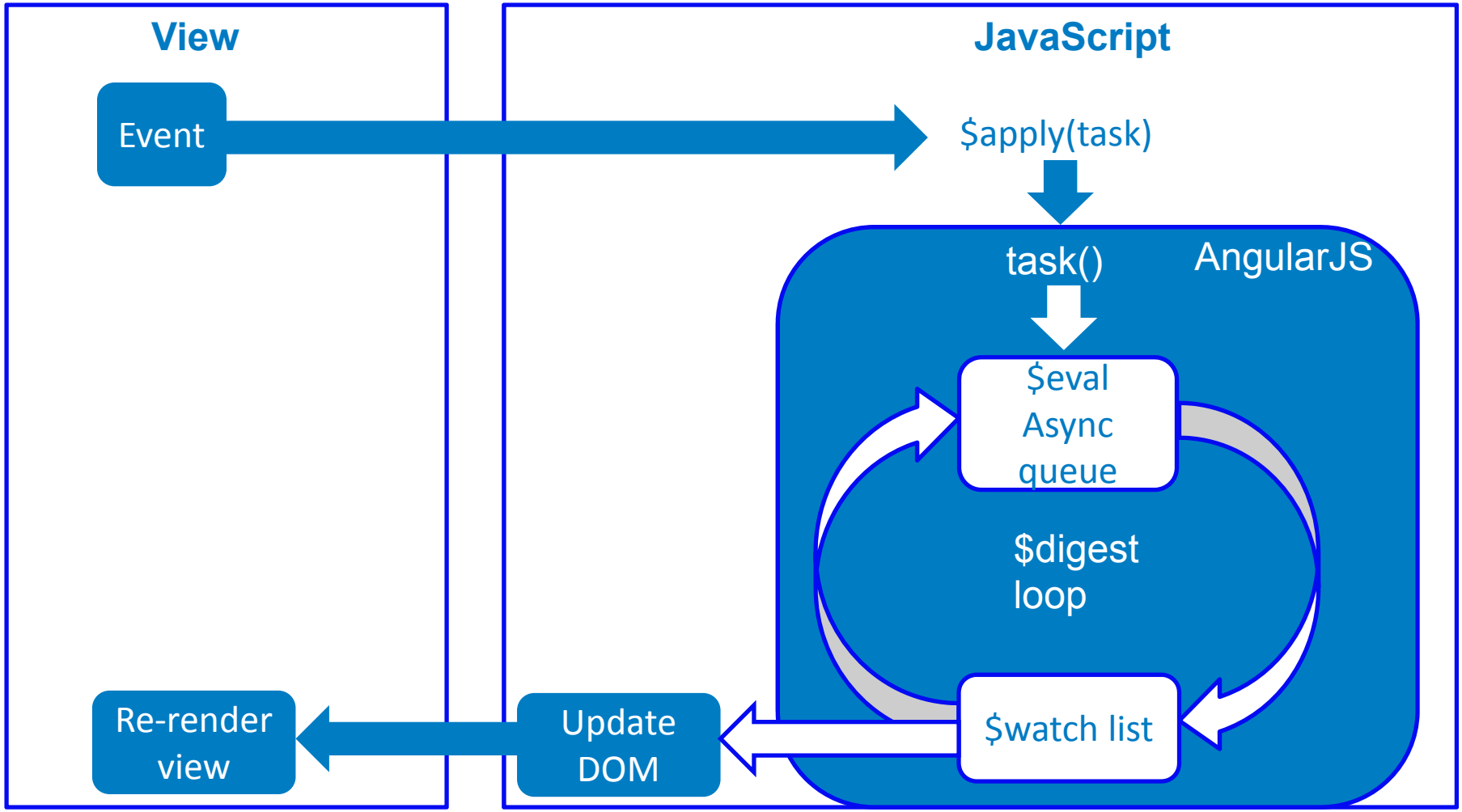
How AngularJS works-Runtime

- The below steps are followed when a browser deals with event handling.
  - In normal scenarios, when an event occurs, the corresponding callback method is called.
  - On invoking the callback method, we enter the JavaScript context. Using the callback method, we can modify the DOM.
  - Once the callback execution is completed, the browser comes out of the JavaScript context and re-renders the view based on changes in the DOM.
- Angular deals with event handling in slightly different manner.
- Angular has its own event processing loop.

How AngularJS works-Runtime

- Angular alters the usual JavaScript flow by providing a separate event processing loop.
- Angular splits the JavaScript into two contexts: the classical JS context and the Angular execution context.
- Places where Angular expression is used will will benefit from Angular's features like databinding, property watching, etc.
- *$apply()* can be used to enter to Angular's execution context from JavaScript.
- $apply() is usually applied by default by the directive itself. $apply() has to be called explicitly only when binding custom events or while working with third-party libraries.

How AngularJS works-Runtime



How AngularJS works-Runtime

- Call scope.$apply(fnc) to enter the angular execution context.
- fnc will be the work which needs to be done in the Angular execution context.
- On executing fnc, the application's state changes.
- Angular enters $digest loop. $digest has 2 smaller loops.
  - The smaller loops process $evalAsync queue and $watch list.
  - $digest repeats until model stabilizes, i.e., $evalAsync queue should be empty and $watch doesn't detect a change
  - $evalAsync queue contains the work to be done before updating view.
  - $watch list contains a set of expressions which have some changes with respect to previous iterations. On encountering a change, $watch is called and updates the view with the new value.
  - Once the execution of $digest loop finishes, it leaves the Angular and JavaScript context. This will prompt the browser to re-render the DOM to reflect the changes.

Revisiting $watch-demonstration

- Lets revisit the example which demonstrated $watch.

```
<!doctype html>

<html ng-app>

<head>  <title>Angular Controller</title>

<!—code to include stylesheet and Angular lib →

  <script type="text/javascript">

function MainController ($scope)

{

$scope.username="Default name";

$scope.$watch(function(scope) { return scope.username },

            function(newValue, oldValue) {

                console.log(oldValue + " "+newValue );

            }

  );          }              </script> </head>
```

Associating $watch to listen to value changes

```
<body><center><header>

 <h2> {{ 'Example for '+' AngularJS Controller' }}
</h2></header><br>

 <div class="LoginFormDiv"

ng-controller="MainController">

<table >

<td><b>Please enter Username:</b></td>

<td><input type="text" ng-model="username" ></td>

</tr> </table>  <br><br>

<b> Hello {{username}} </b><br><br>

  </div>

</center></body></html>
```

Revisiting $watch-demonstration

Angular Controller

file:///D:/Materials/Angular.JS/Anusree%20Latest/AngularJS%20art

Example for $watch

Please enter Username: infy

Hello infy

Elements Network Sources Timeline Profiles Resources »

Sources Content ... Snippets

Console Search Emulation Rendering

<top frame> Preserve log

```
Default name Default name                    watch.html:15
Default name i                               $watch.html:15
i in                                         4-$watch.html:15
in inf                                       4-$watch.html:15
inf infy                                     4-$watch.html:15
```

$watch prints the oldValue and the newValue in the console

Runtime

- In the previous example, data-binding effect is shown. As and when the user enters the value in the textbox, the same is rendered in the view.
- The below gives an insight into how $watch is internally working.
- Explanation:
  - Compilation phase:
    - the ng-model and input directive sets up a keydown listener on the <input> control
    - the {{username}} interpolation sets up a $watch to be notified whenever username changes

Runtime

- Runtime phase:
  - Pressing a key will lead to keydown event
  - The input directive captures update to the <input> tag's value attriubute and calls $apply("username = 'i';") to update the application model data.
  - Angular applies username='i'
  - $digest loop is started
  - $watch detects an update to the username property and triggers {{username}} interpolation. This updates the DOM.
  - Angular exits the execution context, thereby exiting keydown event.
  - Browser finally re-renders the view with the updated text.

Angular Expressions

Angular Expressions

- Expressions are bindings placed inside {{ expression }}.
- Expressions in Angular are processed by $parse service.
- Examples of valid expressions:
  - 2+3
  - 2*10 | currency
  - user.name
- Angular does not use eval() to evaluate expressions and hence Angular expressions are not the same as JavaScript expressions.

Angular Expressions

- An Angular expression is a JS snippet bound by
  - {{ expression }}, or
  - ng-bind = “expression”
- Angular expressions are different from JS Expressions:
  - No error if a part of expression evaluates to undefined or null
  - Do not support if-else, loops, switch-case or exceptions
  - Evaluated within the current scope
  - Support Angular filters

Expressions

- Comparison between Angular Expressions and JS Expressions

	JavaScript	Angular
Evaluation of attributes	Done against *window* object	Done against *scope*
Forgiving	Evaluating *undefined* properties results in error	Expression evaluation is forgiving to *undefined* and *null*
Control flow statements	Available in JS	We cannot have *conditional, loops* or *throw* statements
Filters	We need to call method separately.	Result of expression evaluation can be passed through filter chains.

Expressions - $ prefix

- $ prefix is used in Angular to differentiate the API of Angular from others.
- We can avoid any conflicts that might come up. For eg: if the developer adds a user defined length method, it might lead to collision with the length property already defined in Angular. Prefixing $ helps retain the Angular namespace thereby avoiding namespace conflicts.

Directives

Directives

- Directives help to teach new syntax to the browser.
- Angular has in-built directives which can be used during application development.
- Directives usually have snake cased names. Eg: ng-repeat
- The directives can be placed in element names, attributes, class names, as well as comments.
- Directives are behaviors which should be triggered on encountering a specific HTML construct during the compilation phase.
- Directive's behaves based on its attributes and scope.
- The designer is also given the privilege to create his own directives which are specific to his app.
- All the compilation required happens at the browser side and no pre-compilation from the server is needed.

Directives

- ng-app
  - Describes the root of the application.
- ng-controller
  - Specifies a JavaScript controller class that evaluates HTML expressions.
- ng-model
  - Helps in creation of a model.
  - Allows two-way data binding between the view and the scope
- ng-bind
  - Changes the inner content of an HTML element to the value of a given expression.

Directives

- ng-include
  - Includes and compiles external html
- ng-view
  - This directives handles the different routes encountered and dynamically renders the templates. Templates rendering is driven by specified controllers.
- ng-repeat
  - Used to iterate over a set of values like arrays.
  - ng-repeat also provides $index for each item

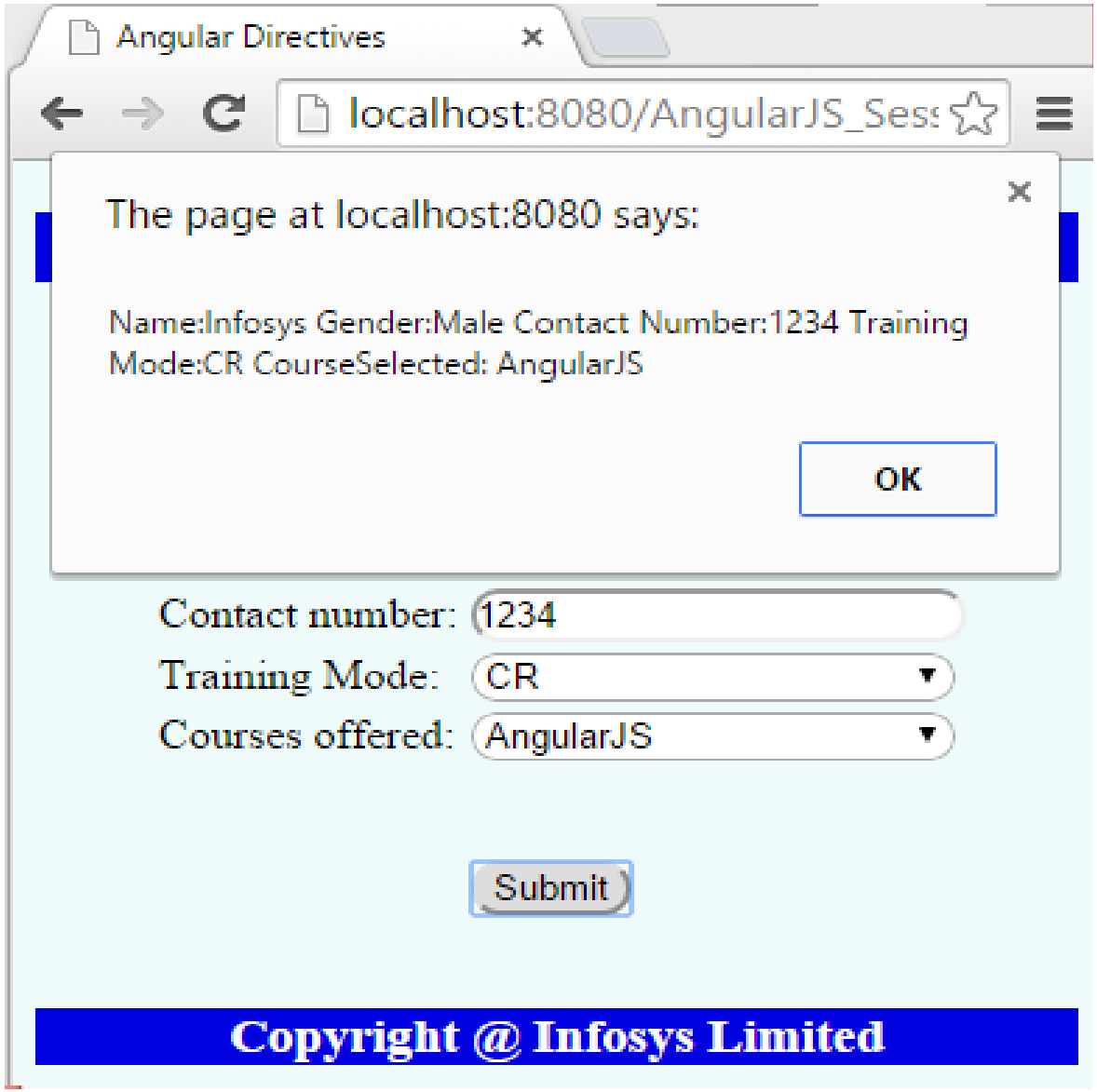
Directives

- The **ng-style** and **ng-class** directives help us in providing class and style values to elements
- ng-class
  - Allows to dynamically load class attributes.
- ng-show & ng-hide
  - Show or hide an element conditionally , depending on the value of a Boolean expression.
- ng-if
  - Basic if statement directive. Allow to show the element if the conditions are true.
- ng-switch
  - Conditionally instantiate one template from a set of choices, depending on the value of a selection expression.

Example for Directives-Scenario

- Lets consider a scenario where the user would like to register for online training of few courses.
- Trainings are offered in two modes: VCR (Virtual Classroom) and CR (live classroom).
- Courses offered in VCR mode: HTML and CSS
- Courses offered in CR mode: AngularJS and BackboneJS
- User needs to enter name, gender, training mode and course. If Male gender is selected, then additionally the form needs to capture the Contact number of the nominee.
- Courses will be listed in the menu only after selecting the appropriate mode of training.
- On providing all details, and on click of Submit button, the entered details are displayed to the user in a JS alert.

Example for directives



Example for Directives


```
<!doctype html>
<html ng-app="app">
<head>
<title>Angular Directives</title>
<style type="text/css">
@import "styles/style.css";
</style>
<script src="lib/Angular/angular.js"></script>
<script src="js/controllers_directives.js"></script>
</head>
<body>
<center>
<div ng-include="'Header.html'"></div>
```

ng-app directive to bootstrap Angular

Including the controller js file

Including the content of Header.html

```
<h2 ng-class="{strike: false, red: true}"
ng-style="{'font-style': 'italic'}"
ng-bind="'Course Registration'"  </h2>
<div ng-controller="Ctrl">
<!– Directives Example→
</div>
<div ng-include="'Footer.html'" ng-controller="Ctrl"></div>
</center>
</body>
</html>
```

ng-class applies the CSS classes whichever is true

ng-bind sets the content of <h2> with the text given

ng-style applies an inline style of italics

Mentioning the controller to be used here using ng-controller

Including the content of Footer.html

Mentioning the controller to be used here

Example for Directives

Header.html

```
<header>
 <h2> {{ 'Example for '+' Directives' }} </h2>
</header>
```

Footer.html

```
<footer>
        <center>
            <h3 ng-bind="footer"></h3>
        </center>
</footer>
```

Using ng-bind to set the content of <h3> with the text given

Example for Directives

Code for Example on Directives

```
<table>
<tr>
<td>Name:</td>
<td><input type="text" ng-model="name"
class="input"/></td></tr>
<tr>
<td>Gender:</td>
<td><span>
<input type="radio" name="g" ng-model="gender"
value="Male"/>Male
<input type="radio" name="g" ng-model="gender"
value="Female"/>Female</span>  </td></tr>
<tr ng-show="gender=='Male'">
<td>Contact number:</td>
<td><input type="text" ng-model="contact"
class="input"/></td></tr>
<tr>  <td>Training Mode:</td>
<td> <select ng-model="mode"
ng-change="fun()" class="select">
<option value="VCR">VCR
<option value="CR">CR
</select></td>
</tr>
<td>Courses offered:</td>
<td>  <select ng-model="courseselected"
class="select">
<option ng-repeat="course in courses">
{{course.coursename}}
</select></td>
</tr></table>  <br><br>
<button type="button" ng-click="save()"
class="button">Submit</button>
<br><br>
```

Invoked for change event of menu

ng-model creates model and stores the entered value in the model

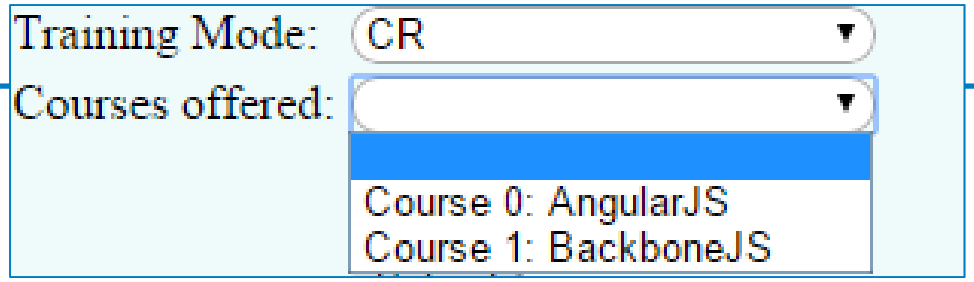
Invoked for click event of button

ng-repeat repeats <option> for every course

ng-repeat-in detail

- ng-repeat directive can be used to iterate over arrays.
- Directives like ng-repeat clone the DOM elements for every element in the collection.
- Since we have a separate compile and link phase, the performance will be better. This is because the cloned templates need to compiled only once whereas the same can be linked once for each clone instance.
- ng-repeat also provides $index for each item
- Example:

```
<select ng-model="courseselected" class="select">
  <option ng-repeat="course in courses"> Course {{$index}}: {{course.coursename}}
</select>
```



Filters

Filters

- Filters are used for data transformation.
- We can use the filters to format the data in locale specific output.

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

- Filters transform the data to a new data type by formatting the data.
- Filters can be chained and can take optional arguments.
  - {{ expression | filter1 | filter2 }}
- To create a filter, a filter object can be injected into the module. A filter function is returned which has the input value as the first argument

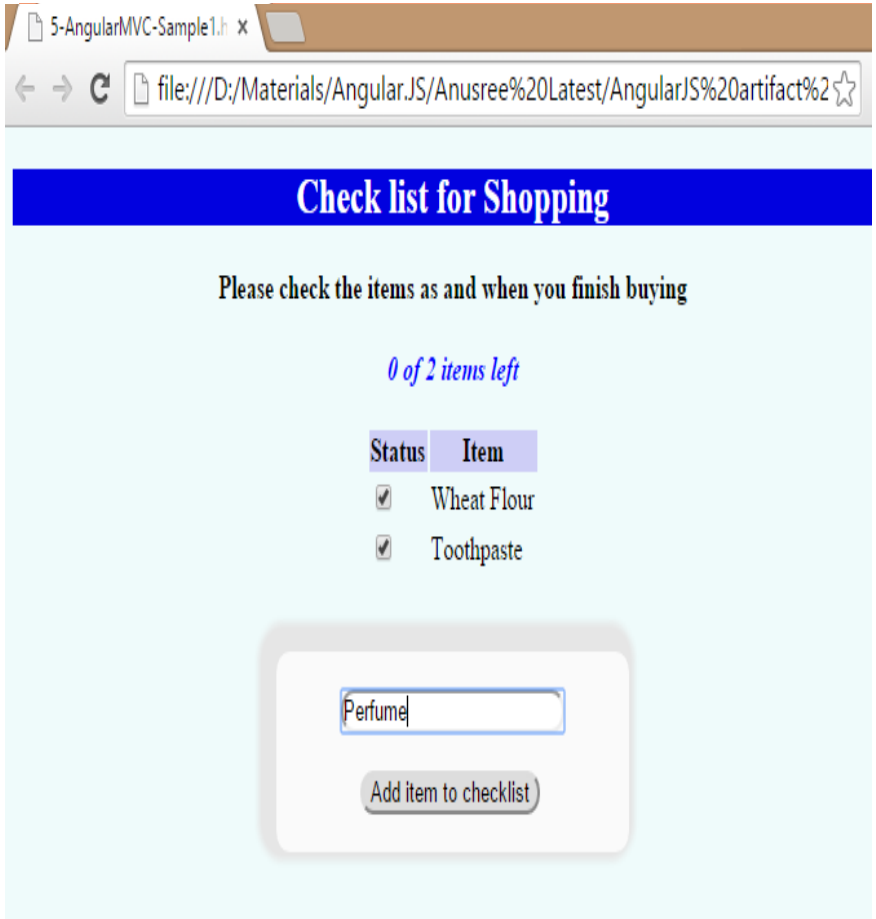
Filters

- When data is shown to the user, we might want to convert data to much more user friendly format.
- Filters in Angular help us here.
- Eg: name | uppercase
  - In this case, the name is passed to the uppercase filter for modification
- Filter chaining eg: value | filter1 | filter2
- We can also pass arguments to the filter.
  - Eg: 345 | number:2 → This tells Angular to display 345 to display with 2 decimal points.

Filters

Filter	Description
Currency	Formatting to currency format
Filter	Selecting subset of items from array
Lowercase	Formatting to lowercase
orderBy	Helps in ordering
uppercase	Formatting to uppercase

Revisiting Checklist Case study



Revisiting Checklist Case study

- Lets modify the inside the html to include filters.

```
<div ng-controller="ItemCtrl">

<h4>Please check the items as and when you
finish buying</h4>

<span>{{remaining()}} of {{items.length}} items
left</span><br>

<br>

<h4>Search in checklist:

<input type='text' ng-model="search_item"
class="txt" /></h4>

<h4>Order items by:

<select ng-model="order_item" class="select">

        <option value="done">Status</option>

        <option value="text">Item name</option>

    </select></h4>

<br><br>
```

Item to be searched is stored in search_item model

Order_item stores mode of ordering

```
<table>

<th>Status</th><th>Item</th>

<tr ng-repeat="item in items | filter:search_item |
orderBy:order_item" >

                <td><input type="checkbox"

ng-model="item.done"></td>

                <td> {{item.text}} </td>

</tr></table> <br><br>

<div class="add"><br>

<form ng-submit="addItem()">

<input type="text" ng-model="itemText" class="txt"
placeholder="Add new item to checklist"><br><br>

<button  type="submit" ng-click="" class="button">Add item
to checklist</button> <br><br>

</form>  </div>
```

List is filtered based on value of search_item

List is ordered based on value of order_item model

Revisiting Checklist Case study: controllers_filters.js

Adding few more items

```
function ItemCtrl($scope) {
 $scope.items = [
  {text:'Wheat Flour', done:true},
   {text:'Tooth Paste', done:false},
   {text:'Rice Flour', done:true},
   {text:'Talcum Powder', done:false},
   {text:'Ragi', done:true},
   {text:'Maida', done:false}
];
```

```
$scope.remaining = function() {
    var count = 0;
                angular.forEach($scope.items, function(item) {
               if(item.done==false)
                          count=count+1;
               });   return count;
  };
$scope.addItem = function() {
    $scope.items.push({text:$scope.itemText, done:false});
    $scope.itemText = '';    };

}
```

Modifying Checklist Case study

localhost:8080/AngularJS_

localhost:8080/AngularJS_Session_demos/Day2/7-Filters.html

Check list for Shopping

Please check the items as and when you finish buying

3 of 6 items left

Search in checklist:

Order items by:

Status	Item
☑	Wheat Flour
☐	Tooth Paste
☑	Rice Flour
☐	Talcum Powder
☑	Ragi
☐	Maida

Add new item to checklist

Add item to checklist

localhost:8080/AngularJS_

localhost:8080/AngularJS_Session_demos/Day2/7-Filters.html

Check list for Shopping

Please check the items as and when you finish buying

3 of 6 items left

Search in checklist: flo

Order items by:

Status	Item
☑	Wheat Flour
☑	Rice Flour

Add new item to checklist

Add item to checklist

localhost:8080/AngularJS_

localhost:8080/AngularJS_Session_demos/Day2/7-Filters.html

Check list for Shopping

Please check the items as and when you finish buying

3 of 6 items left

Search in checklist:

Order items by: Status

Status	Item
☐	Tooth Paste
☐	Talcum Powder
☐	Maida
☑	Wheat Flour
☑	Rice Flour
☑	Ragi

Add new item to checklist

Add item to checklist

Summary

- Scope
- Scope hierarchies and their evaluation
- Scope events
- Scope LifeCycle
- Angular Runtime
- Angular Expressions
- Directives
- Filters

Thank You

Infosys® | Building Tomorrow's Enterprise