# Angular

Capital One

developintelligence.com

# Topics for Today

- Upgraded Router
- Filters
- Looping
- Forms
- Error Messaging
- Directives
- Angular @ Startup
- Digest Loop
- Custom Directives

# ui-router

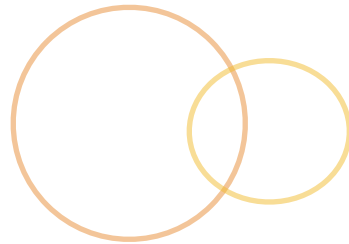
# A Better Router

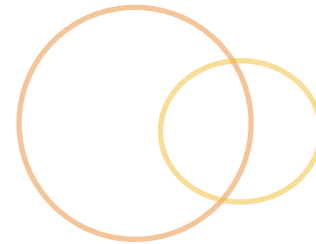
- **ui-router** gives more flexibility to routing within Angular
  - It is the second attempt at routing
- Based upon the idea of a state machine
  - Direct the application to states rather than just URLs
- Good for multiple views on the same page
- Helpful for nested views

# AngularUI

- A companion sweet of tools / modules for Angular
  - http://angular-ui.github.io/
  - ui-router is just one of the projects created by the angular-ui team
- AngularUI is modularized so to use one you don't need to include them all
  - Each component would need to be installed separately

# ui-router





- Download it:
  - https://github.com/angular-ui/ui-router
- Bower it:
  - **bower install angular-ui-router**
- Include it after angular:

```
<script src="angular.js"></script>
<script src="angular-ui-router.js"></script>
```

# ui-router [cont.]

- The ui-router comes with its own namespace
  - Doesn't utilize the **ng** namespace
  - Uses **ui** namespace
- We need to inject the ui-router into our application

```
var app = angular.module('demo', ['ui.router']);
```

# $stateProvider

- Our configuration of states is done via the **$stateProvider**
  - No longer use **$routeProvider**

```
app.config(['$stateProvider', function($stateProvider) {

  $stateProvider.state('start', {
    url: '/start',
    templateUrl: 'partials/start.html'
  });

});
```

# $urlRouterProvider

- ui-router also has a route provider not just a state provider
  - Allows us to specify what happens when a certain url is hit
  - Useful when we have user interaction that happens outside of our states
    - (i.e. redirection)
- It watches the $location service
  - When there is a change it looks through the configured rules and tries to find a match
  - Any state that has a URL automatically has a rule registered with the $urlRouterProvider

# $urlRouterProvider [cont.]

- $urlRouterProvider.**when**()
- Allows for redirection of application routes
- Takes 2 parameters:
  - first parameter is the path the user lands on
  - second parameter is the path the user will be taken to

```
app.config(['$urlRouterProvider', function($urlRouterProvider) {

  $urlRouterProvider.when('/route/user/lands/on',
       '/route/of/redirection');

});
```

# $urlRouterProvider

- $urlRouterProvider.**otherwise**()
- Allows for the application to redirect when a non-specified url is landed on

```
app.config(['$urlRouterProvider', function($urlRouterProvider) {

  $urlRouterProvider.otherwise('/index');

});
```

# ui-view Directive

- ui-view is the directive used by $state
  - Specifies where the templates should go
  - No longer use **ng-view**
  - ui-view has a 400 level priority and is terminal

```
<div ng-controller="DemoController">
  <div ui-view></div>
</div>
```

# $viewContentLoading

- Event fired before the DOM has rendered the view
  - (i.e. the view is beginning to load)
- Event is fired from the root scope

# $viewContentLoaded

- Event fired after the DOM has rendered the view
  - (i.e. the view is loaded)
- Event is fired from the ui-view directive scope

# ui-sref Directives

- ui-sref
  - Gives the ability to create links between states
  - Clicking the link creates a state transition
  - The path is relative to the state

Angular Code

```
<li>
  <a ui-sref="home">Home</a>
</li>
```

Compiled HTML

```
<li>
  <a href="#/home" ui-sref="home">Home</a>
</li>
```

# ui-sref Directives

- ui-sref-active
  - Works in tandem with **ui-sref**
  - Simplifies the updating of the user-experience by adding an active class to the current state
    - Easily add active classes to navigation menus

```
<li ui-sref-active="active">
  <a ui-sref="home">Home</a>
</li>
```

```
<li ui-sref-active="active" class="active">
  <a ui-sref="home" href="#/home">Home</a>
</li>
```

# $state Service

- Service used for state representation and transition
  - http://angular-ui.github.io/ui-router/site/#/api/ui.router.state.$state

# $state Service

- Gives us an additional way to navigate to a new state
  - $state.go()

```
<button ng-click="takeMeHome()">Waaahhh I wanna go home</button>
```

```
app.controller('Page2Controller',
  ['$scope', '$state', function($scope, $state) {

    $scope.takeMeHome = function() {
       $state.go('index');
    };

 }]);
```

# Nested Views [cont.]

- Sometimes we need views within views
  - For this to happen we can use ui-routers ability to nest views

```
$stateProvider.state('parent', {
  url: '/parent',
  templateUrl: 'parent.html,
  controller: 'ParentController'
});

$stateProvider.state('parent.nested', {
  url: '/nested',
  templateUrl: 'parent-nested.html,
  controller: 'NestedController'
});
```

# Nested Views [cont.]

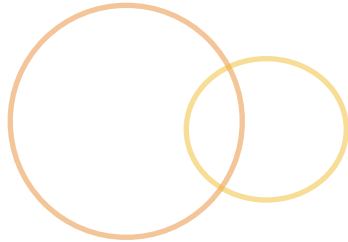
- Create links to the nested views
  - Use ui-sref with the appropriate state to get to the nested child

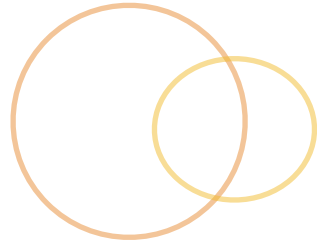
```
<a ui-sref=“.nested">Nested View</a>
```

# Nested Views [cont.]

- Navigating nested views
  - $state.go('.') : will take you to current state
  - $state.go('^') : will take you to the parent state if I am in a child
  - $state.go('^.sibling') : will take you to a sibling state if I am in a child
    - Same as saying $state.go('parent.sibling')

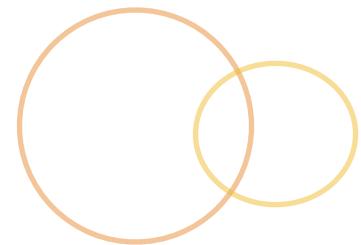
# Lab 3

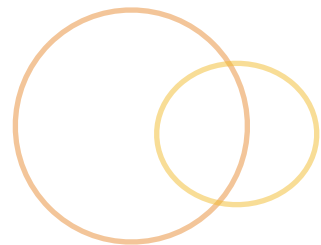




- Use the ui-router in your application instead of ngRoute
  - Bind the page title attribute to each template
  - Simplify the process of adding an active page class
- Sell page
  - Create a nested route for our products
  - Include a link to info about lemonade, healthy snacks and treats
- Update your footer to show the appropriate active state throughout all state changes

# Angular Expressions Behind the Scenes





# Expressions

- We have seen expressions between our handlebars
  - {{ expression }}
  - The scope is the context for {{ expression }}
  - {{ expression }} is meaningless without the scope's context
- JavaScript statements that are evaluated
  - Kind of like the POJSO eval() function
  - Angular automatically evaluates expressions
- Errors resulting from the expression are swallowed

# Expressions [cont.]

- Expressions are evaluated in their respective $scope
  - They have access to the $scope they are in
  - Including inherited parental scopes

# $parse Service

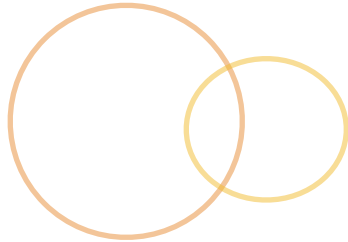
- **$parse** service evaluates our expressions
  - Happens automatically
- We can parse expressions manually
  - $parse will convert an expression to a function
  - We call that function with the context to evaluate our expression against (i.e. our $scope object)
- https://docs.angularjs.org/api/ng/service/$parse

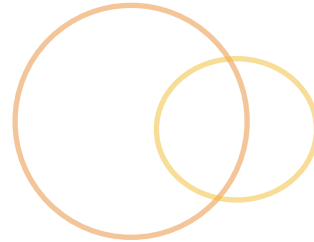
# $parse Service [cont.]

```
<body>
  <section ng-controller="DemoController">
    <input ng-model="expression" type="text" /><br>
    {{parsedExpression}}
  </section>
  <script src="main.js"></script>
</body>
```

```
app.controller('DemoController', function($scope, $parse) {
  //… Previous stuff :)
  $scope.parsedExpression;
  $scope.$watch('expression', function(newValue) {
      var parsedFunction = $parse(newValue);
      $scope.parsedExpression = parsedFunction($scope);
  });
});
```

# $watch





- **$watch** registers a handler function that will be called when the variable being watched changes
  - Takes 3 parameters
    - variable to watch
    - callback function when the variable changes
    - boolean for deep comparison

```
app.controller('DemoController', function($scope, $parse) {
  $scope.$watch('expression', function(newValue) {
      var parsedFunction = $parse(newValue);
      $scope.parsedExpression = parsedFunction($scope);
  });
});
```

# $interpolate Service

- The $interpolate service uses the $parse service to evaluate individual expressions within it
  - It will take individual expressions and convert the whole representation to a string
  - The $compile service uses the $interpolate service as one of its tools in setting up data binding
- Via interpolation we have our {{ … }} expressions translated into strings visible by the user

# $interpolate Service [cont.]

- We can create our own template engine for string interpolation
  - $interpolate will convert an expression to a function
  - We call that expression within the context of a specified scope
    - We call that function with the context to evaluate our expression against (i.e. specific template variables)

```
app.controller('DemoController', function($scope, $interpolate)
{
  //… Previous stuff :)
  $scope.$watch('sentenceTemplate', function(newValue) {
      var interpolatedFunction = $interpolate(newValue);
      $scope.output = interpolatedFunction({
        feeling: $scope.sentence.feeling,
        punctuation: $scope.sentence.punctuation,
        who: $scope.sentence.who
      });
  });
});
```

# $interpolate Service [cont.]

```
<body>
  <section ng-controller="DemoController">
    <!-- ... Previous feeling sentence ... --!>
    <section>
      <h2>Tell me how you are feeling?</h2>
      <input ng-model="sentenceTemplate" />
      <p>{{output}}</p>
    </section>
  </section>
  <script src="main.js"></script>
</body>
```

# $interpolateProvider Delimiters

- Let's say we don't like our delimiters (i.e. {{ }} )
  - $interpolate defaults to {{ }}
- $interpolateProvider allows us to configure how string interpolation will take place
  - $interpolate is a service (i.e. a singleton) so making changes will affect the whole application
  - Useful if dealing with another framework that utilizes the {{ }}
- https://docs.angularjs.org/api/ng/provider/$interpolateProvider

# $interpolateProvider Delimiters [cont.]

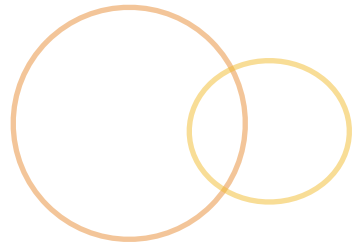
- We can switch up our template starting and ending symbols
  - Let's use %

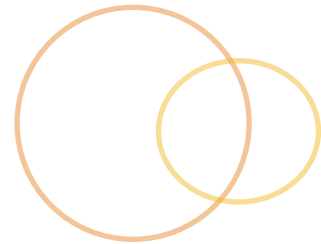
```
app.config(function($interpolateProvider) {
  $interpolateProvider.startSymbol('%');
  $interpolateProvider.endSymbol('%');
});
```

```
<section ng-controller="DemoController">
  <input ng-model="sentence.feeling" type="text" />
  <h1>%sentence.who% am feeling %sentence.feeling%
    %sentence.punctuation%
  </h1>
</section>
```

# Filters

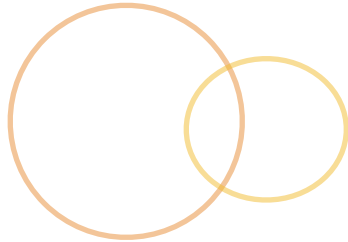
# Filters

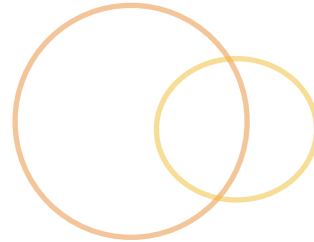




- Change the way information is displayed to users
  - Formatting the data
  - Filters don’t modify the original data in the scope
  - Filters can change the display in different parts of the app
- Filters change the data from scope to view

# Filters [cont.]





- Filtering can occur within the view
- Filter can occur within the controller
  - $filter service
- Angular has built-in filters we can utilize

# Filtering in View

- Filtering can occur within the view
  - Via “|” pipe
  - Inside our template binding {{ … }}

```
<section ng-controller="DemoController">
  <input ng-model="sentence.feeling" type="text" />
  <h1>{{sentence.who}} am feeling
    {{sentence.feeling | uppercase}} {{sentence.punctuation}}
  </h1>
</section>
```

# Filtering in View [cont.]

- Filtering occurs continuously


# Filtering in Controller

- Filter can occur within the controller
  - $filter service injection

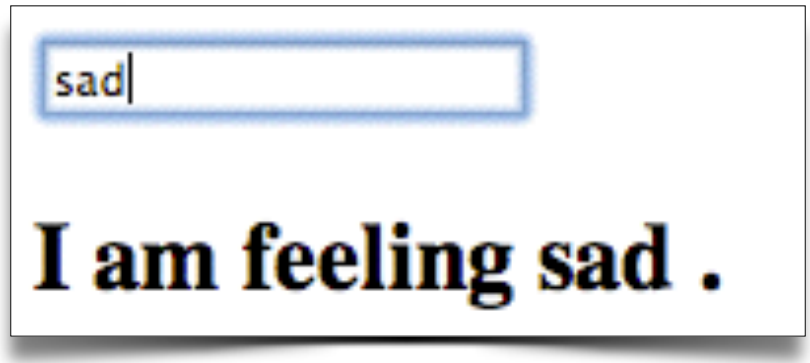
```
<section ng-controller="DemoController">
  <input ng-model="sentence.feeling" type="text" />
  <h1>{{sentence.who}} am feeling
    {{sentence.feeling}} {{sentence.punctuation}}
  </h1>
</section>
```

```
app.controller('DemoController', ['$scope', '$filter',
function($scope, $filter) {
  $scope.sentence = {
    who: 'I',
    feeling: $filter('uppercase')('happy'),
    punctuation: '.'
  };
}]);
```

# Filtering in Controller [cont.]

- This will make the default value (i.e. happy) uppercase


- It will not continuously filter data as our filter in the view did

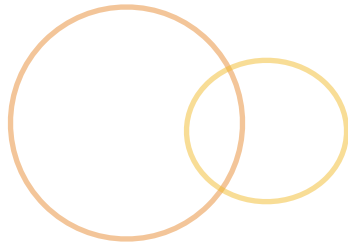


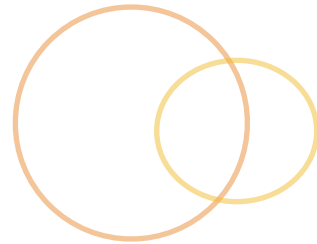
# Filtering in Controller [cont.]

- Should we be doing formatting in the controller?

- Best practice would be to keep formatting out of the controller if possible
- We aren't changing the actual data in the view, but we are in the controller
  - We want to use the data throughout the whole application
  - Preserve the data integrity

# Single Value Filters





# Initial Data

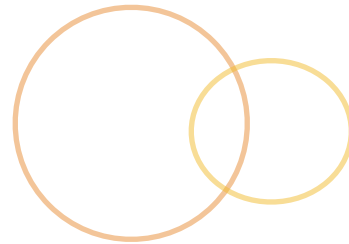
- Basic data we will work with

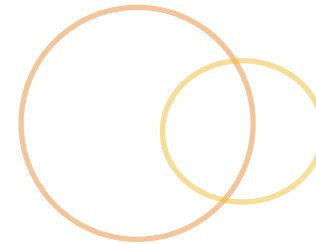
```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.car = {
    make: 'Tesla',
    model: 'S',
    price: '69000',
    quantity: '10',
    manufactureDate: Date.now()
  };
}];
```

**Cars**

Make: Tesla
Model: S
Price: 69000
Quantity: 10

# Currency





- Formats currency values

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
  </p>
</section>
```

**Cars**

Make: Tesla
Model: S
Price: $69,000.00
Quantity: 10

# Currency [cont.]

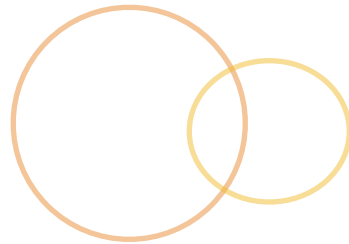
- Allows for the display of a currency symbol
  - It will default to the currency local
  - We can manually switch the currency indicator
  - https://docs.angularjs.org/api/ng/filter/currency

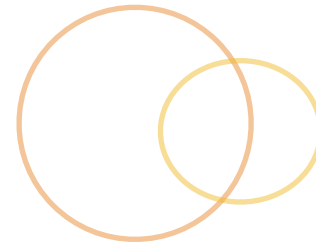
```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price | currency : "&pound"}}<br>
      Quantity: {{car.quantity}}
    </p>
</section>
```

**Cars**

Make: Tesla
Model: S
Price: £69,000.00
Quantity: 10

# Numbers





- Formats data to look like a number
  - It will insert commas in appropriate locations

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price | number}}<br>
      Quantity: {{car.quantity}}
    </p>
</section>
```

**Cars**

Make: Tesla
Model: S
Price: 69,000
Quantity: 10

# Numbers [cont.]

- Formats data fixed number of decimal places
  - https://docs.angularjs.org/api/ng/filter/number

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price | number:2}}<br>
      Quantity: {{car.quantity}}
    </p>
</section>
```

**Cars**

Make: Tesla
Model: S
Price: 69,000.00
Quantity: 10

# Numbers [cont.]

- If a string is defined instead of a number we have problems :(

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price | number:'a'}}<br>
      Quantity: {{car.quantity}}
    </p>
</section>
```

**Cars**

Make: Tesla
Model: S
Price: NaN.
Quantity: 10

# Date

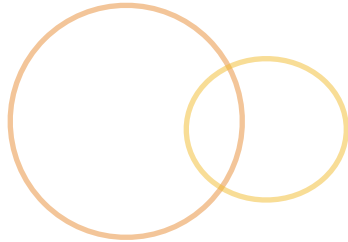
- Formats dates in a human readable way
  - Default behavior is **Month Day, Year**

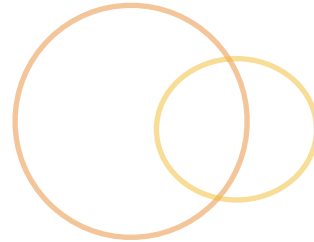
```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price | number:'a'}}<br>
      Quantity: {{car.quantity}}<br>
      Manufacture Date:
        {{car.manufactureDate | date}}
    </p>
</section>
```

**Cars**

Make: Tesla
Model: S
Price: 69,000.00
Quantity: 10
Manufacture Date: May 12, 2014

# Date [cont.]





- Format options
  - 4-digit year (‘**yyyy**’): 2014
  - 2-digit year (‘**yy**’): 14
  - Month as string (‘**MMMM**’): May
  - Month (‘**M**’): 5
  - Month padded (‘**MM**’): 05
  - Day (‘**d**’): 12
  - Day of week: (‘**EEEE**’): Monday
  - AM / PM: (‘**a**’): PM

# Date [cont.]

- Localizable formats
  - **medium**: May 12, 2014 3:30:00 PM
  - **fullDate**: Monday, May 12, 2014
  - **shortTime**: 3:30 PM

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price | number:'a'}}<br>
      Quantity: {{car.quantity}}<br>
      Manufacture Date:
        {{car.manufactureDate |
          date:'short'}}
    </p>
</section>
```

**Cars**

Make: Tesla
Model: S
Price: 69,000.00
Quantity: 10
Manufacture Date: 5/12/14 3:30 PM

# Date [cont.]

- Custom date formatting
  - https://docs.angularjs.org/api/ng/filter/date

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price | number:'a'}}<br>
      Quantity: {{car.quantity}}<br>
      Manufacture Date:
        {{car.manufactureDate |
          date:'d MMM yy'}}
    </p>
</section>
```

**Cars**

Make: Tesla
Model: S
Price: 69,000.00
Quantity: 10
Manufacture Date: 12 May 14

# Lowercase & Uppercase

- Formats strings complete contents to **UPPERCASE** or **lowercase**
  - https://docs.angularjs.org/api/ng/filter/uppercase
  - https://docs.angularjs.org/api/ng/filter/lowercase

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make | uppercase}}<br>
      Model: {{car.model | lowercase}}<br>
      Price: {{car.price}}<br>
      Quantity: {{car.quantity}}
    </p>
</section>
```

**Cars**

Make: TESLA
Model: s
Price: 69000
Quantity: 10

# JSON

- Filter the output as a JSON object
  - Probably not going to use this in production code
  - **Good for debugging**
  - https://docs.angularjs.org/api/ng/filter/json

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price}}<br>
      Quantity: {{car.quantity}}<br>
      Debug: {{car | json}}
    </p>
</section>
```

# JSON [cont.]

- Filter the output as a JSON object

**Cars**

Make: Tesla
Model: S
Price: 69000
Quantity: 10
Debug: { "make": "Tesla", "model": "S", "price": "69000", "quantity": "10", "manufactureDate": 1406675150690 }

# Filter Localization

- Angular has support for localization
  - It is a useful start
  - It is not the be all and end all of localization
    - Supports dates
    - Supports currency
    - Supports numbers
- Has support via extra download

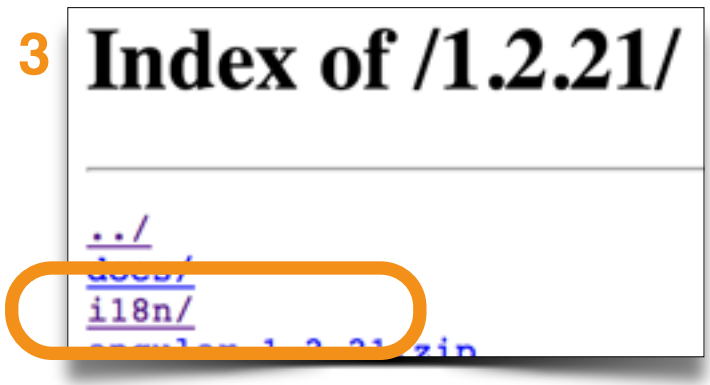
# Filter Localization [cont.]

- The download: Let's go British!

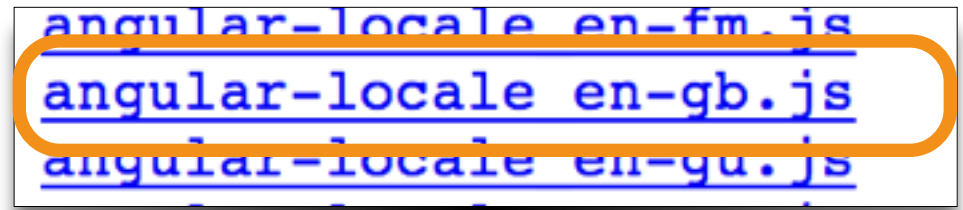
1


2


3




4



# Filter Localization [cont.]

- Angular has support for localization
  - Include the en-gb localization JavaScript file
    - i.e. English - Great Britain
  - **No need to inject** this into your application

```
<html ng-app="demo">
  <head>
    <script src="angular.js"></script>
    <script src="angular-locale_en-gb.js"></script>
  </head>
```

# Filter Localization [cont.]

- Great Britain localization (i.e. en-gb)

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
    <p>
      Make: {{car.make}}<br>
      Model: {{car.model}}<br>
      Price: {{car.price | currency}}<br>
      Quantity: {{car.quantity}}
      Manufacture Date:
        {{car.manufactureDate | date:'short'}}
    </p>
</section>
```

Make: Tesla
Model: S
Price: £69,000.00
Quantity: 10
Manufacture Date: 12/05/2014 15:30

# Custom Filters

- We can make our custom filters reusable
  - Instead of creating them inside our controller we can create a filter module
- Module.filter method used to create a custom filter
  - Takes 2 parameters
    - Name of the filter
    - Factory function that contains a function to implement the filtering

# Custom Filters [cont.]

- Let's create an abbreviate filter
  - Grab the instance of our

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

angular.module('app.filters', [])
  .filter('abbreviate', function() {
    return function(item) {
      if (angular.isString(item)) {
       return item.slice(0,3).toUpperCase() + '.';
       }
    };
  });
```

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

angular.module('demo')
  .filter('abbreviate', function() {
    //Same as above…
  });
```

# Custom Filters [cont.]

- Using our filter

```
<section>
  <h1>Cars</h1>
  <p ng-repeat="car in cars">
    Car: {{$index + 1}}<br>
    Make: {{car.make | abbreviate}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}<br>
    Release Date: {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# Custom Filters [cont.]

- abbreviate Custom Filter: Output

**Cars**

Car: 1
Make: TES.
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Car: 2
Make: TOY.
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

Car: 3
Make: NIS.
Model: LEAF
Price: $27,620.00
Quantity: 0
Release Date: 12/1/10

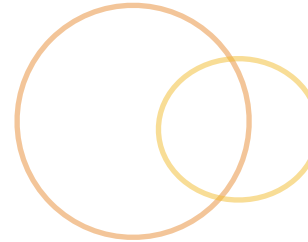
Car: 4
Make: CHE.
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

# Custom Filters [cont.]

- Best Practices:
  - Don’t do DOM manipulation in a filter
  - Don’t do heavy processing
  - Remember: These will be called multiple times in an application

# ngRepeat Directive

# Initial Data



- Basic data we will work with

```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.cars = [{
    make: 'Tesla', model: 'S',
    price: '69000', quantity: '1000',
    releaseDate: new Date('June, 2012')
  }, {
    make: 'Toyota', model: 'Prius',
    price: '34720', quantity: '800',
    releaseDate: new Date('1-1-1997')
  }, {
    make: 'Nissan', model: 'LEAF',
    price: '27620', quantity: '600',
    releaseDate: new Date('Dec, 2010')
  }, {
    make: 'Chevy', model: 'Volt',
    price: '34170', quantity: '900',
    releaseDate: new Date('Dec, 12 2010')
  }];
}];
```

# ngRepeat Directive

- Directive used to loop through a collection of data
  - Each iteration will display a template
  - Each iteration has its own $scope
  - https://docs.angularjs.org/api/ng/directive/ngRepeat

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars">
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# ngRepeat Directive [cont.]

- ng-repeat: Output

**Cars**

Make: Tesla
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Make: Toyota
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

Make: Nissan
Model: LEAF
Price: $27,620.00
Quantity: 600
Release Date: 12/1/10

Make: Chevy
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

# Special Properties

- $index
  - Numeric
  - Gives the index of where the loop is currently
  - Zero based

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars”>
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
</section>
```

# Special Properties

- $index: Output

**Cars**

Car: 1
Make: Tesla
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Car: 2
Make: Toyota
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

Car: 3
Make: Nissan
Model: LEAF
Price: $27,620.00
Quantity: 600
Release Date: 12/1/10

Car: 4
Make: Chevy
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

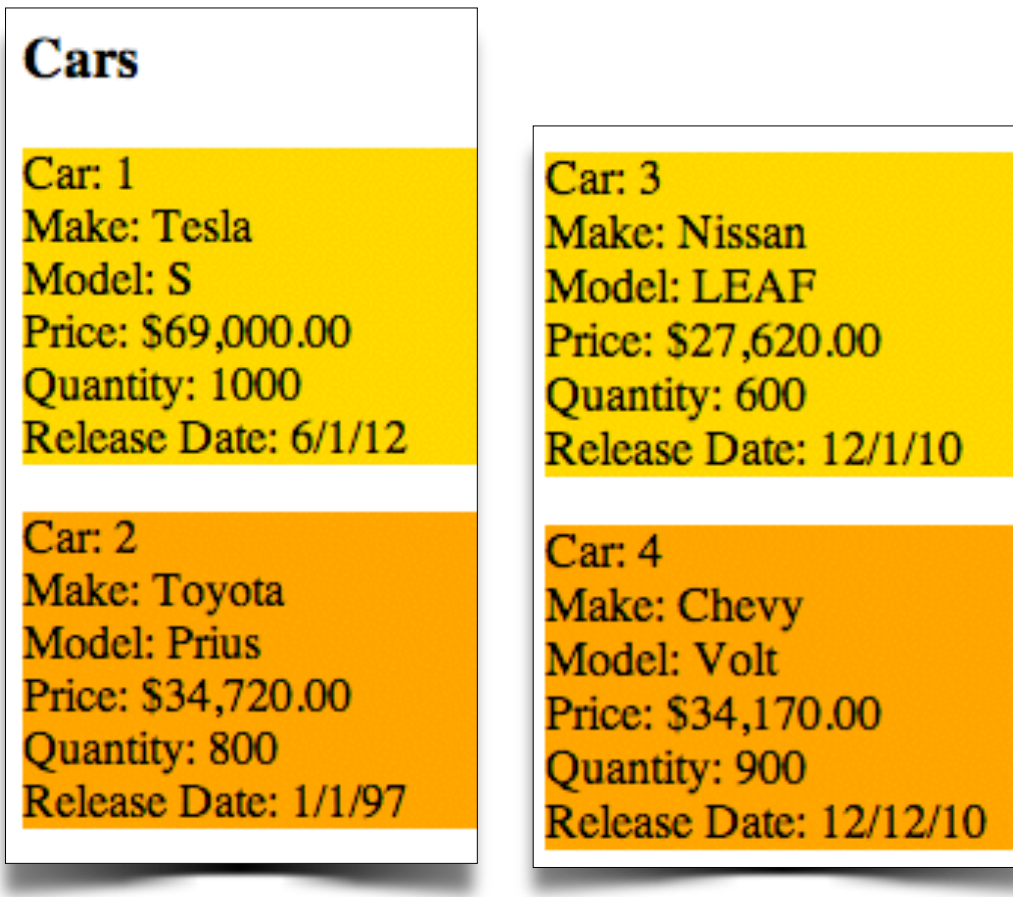
# Special Properties [cont.]

- $odd & $even
  - Boolean
  - Allows for easy class addition of odd/even rows

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars”
    ng-class="{even: !$even, odd: !$odd}">
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# Special Properties [cont.]

- $odd & $even: CSS & Output




```
<style>
   .even {
     background-color: orange;
   }
   .odd {
     background-color: gold;
   }
</style>
```

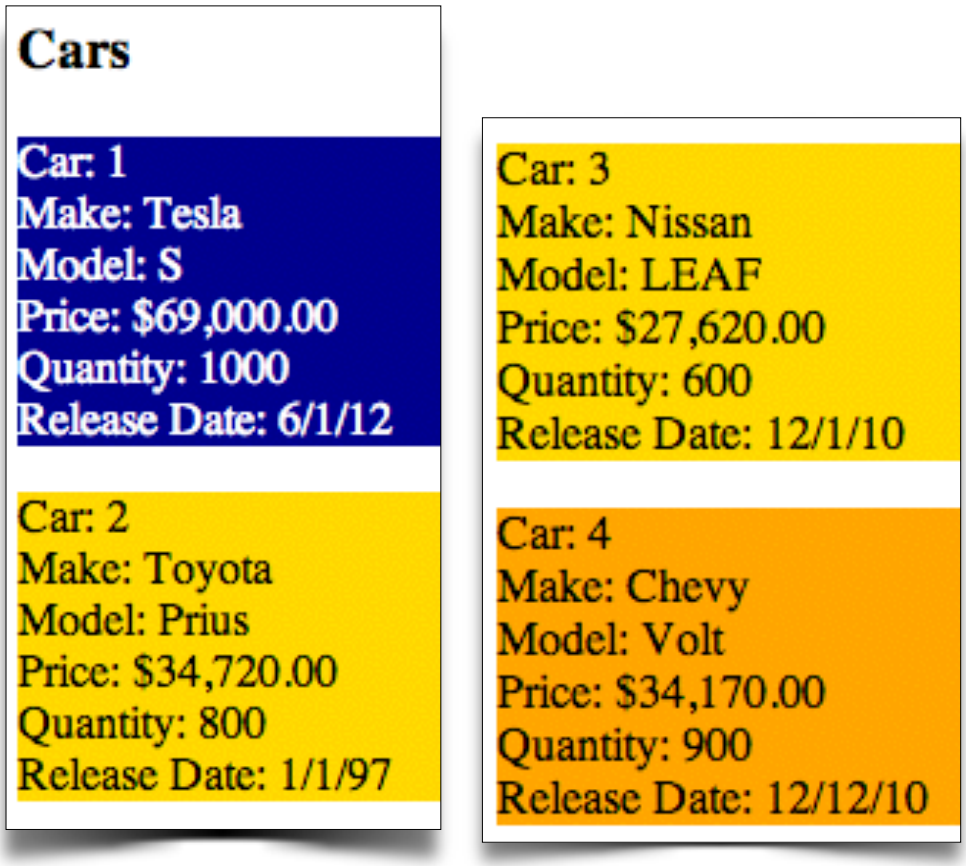
# Special Properties [cont.]

- $first, $middle, $last
  - Boolean
  - Allows for easy class addition of odd/even rows

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars"
    ng-class="{first: $first, middle: $middle, last: $last}">
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# Special Properties [cont.]

- $first, $middle, $last: CSS & Output




```
<style>
  .first {
    background-color: darkblue;
    color: white;
  }
  .middle {
    background-color: gold;
  }
  .last {
    background-color: orange;
  }
</style>
```

# More Repetition

- We can repeat a series of elements
  - We have only looked at repeating 1 element
  - ng-repeat-start: The element to begin our loop iteration
  - ng-repeat-end: The element to end our loop iteration

```
<section ng-controller="DemoController">
  <h2 ng-repeat-start="car in cars">Car: {{$index + 1}}</h2>
  <p>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}
  </p>
  <p ng-repeat-end>
    Quantity: {{car.quantity}}<br>
    Release Date: {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# More Repetition [cont.]

- Output

**Car: 1**

Make: Tesla
Model: S
Price: $69,000.00

Quantity: 1000
Release Date: 6/1/12

**Car: 2**

Make: Toyota
Model: Prius
Price: $34,720.00

Quantity: 800
Release Date: 1/1/97

**Car: 3**

Make: Nissan
Model: LEAF
Price: $27,620.00

Quantity: 600
Release Date: 12/1/10

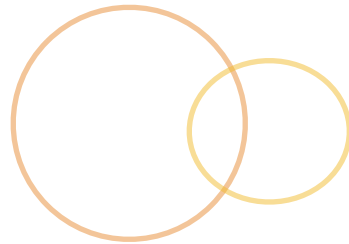
**Car: 4**

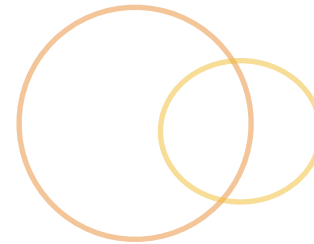
Make: Chevy
Model: Volt
Price: $34,170.00

Quantity: 900
Release Date: 12/12/10

# Collection Filters

# Initial Data

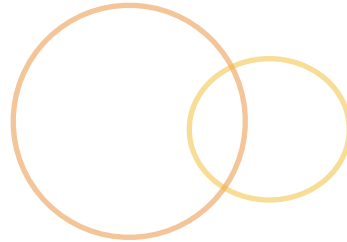


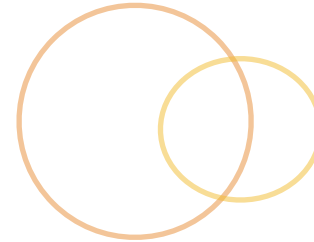


- Basic data we will work with

```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.cars = [{
    make: 'Tesla', model: 'S',
    price: '69000', quantity: '1000',
    releaseDate: new Date('June, 2012')
  }, {
    make: 'Toyota', model: 'Prius',
    price: '34720', quantity: '800',
    releaseDate: new Date('1-1-1997')
  }, {
    make: 'Nissan', model: 'LEAF',
    price: '27620', quantity: '600',
    releaseDate: new Date('Dec, 2010')
  }, {
    make: 'Chevy', model: 'Volt',
    price: '34170', quantity: '900',
    releaseDate: new Date('Dec, 12 2010')
  }];
}];
```

# limitTo

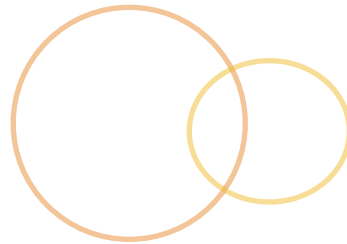


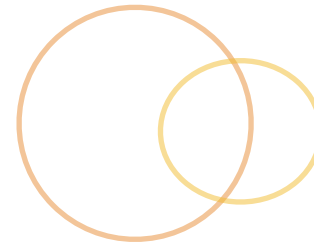


- Allows for the limiting of results show to users
  - limitTo will not change the data only the results shown

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars | limitTo:2”>
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# limitTo [cont.]





- Output

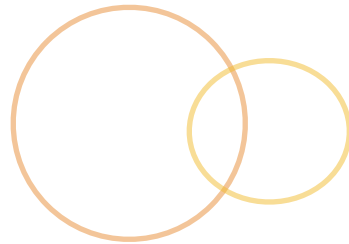
**Cars**

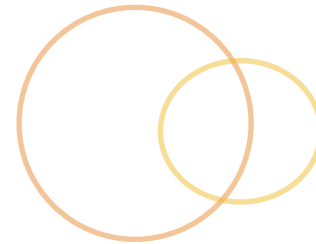
Car: 1
Make: Tesla
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Car: 2
Make: Toyota
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97


[ { "make": "Tesla", "model": "S", "price": "69000", "quantity": "1000", "releaseDate": "2012-06-01T06:00:00.000Z" }, { "make": "Toyota", "model": "Prius", "price": "34720", "quantity": "800", "releaseDate": "1997-01-01T07:00:00.000Z" }, { "make": "Nissan", "model": "LEAF", "price": "27620", "quantity": "600", "releaseDate": "2010-12-01T07:00:00.000Z" }, { "make": "Chevy", "model": "Volt", "price": "34170", "quantity": "900", "releaseDate": "2010-12-12T07:00:00.000Z" } ]

# filter Filter





- Angular gives us the ability to specify our own filtering criteria

```
//Add another Toyota to our line-up
{
  make: 'Toyota',
  model: 'RAV EV',
  price: '49800',
  quantity: '1500',
  releaseDate: new Date('Sep, 2012')
}
```

# filter Filter [cont.]

- Angular gives us the ability to specify our own filtering criteria

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars | filter:{make:'Toyota'}">
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# filter Filter [cont.]

- filter: Output

**Cars**

Car: 1
Make: Toyota
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

Car: 2
Make: Toyota
Model: RAV EV
Price: $49,800.00
Quantity: 1500
Release Date: 9/1/12

# filter Filter [cont.]

- We can make our filter more complex by moving the criteria to the controller

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars | filter:filterCriteria">
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# filter Filter [cont.]

- We can make our filter more complex by moving the criteria to the controller
  - Create a function defining the filter criteria
  - Pass a object (i.e. car) for comparison
  - Return a boolean out of the function for loop iteration comparison

```
app.controller('DemoController', ['$scope', function($scope) {

  $scope.filterCriteria = function(car) {
    //Gives Toyota and Tesla in filter results
    return car.make === 'Toyota' || car.make === 'Tesla';
  };

  //. . .
};
```

# filter Filter [cont.]

- filter: Output

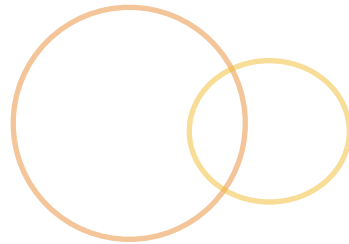
**Cars**

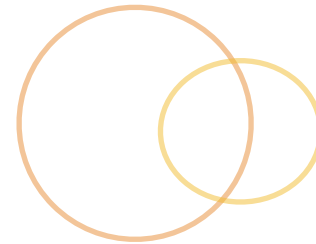
Car: 1
Make: Tesla
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Car: 2
Make: Toyota
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

Car: 3
Make: Toyota
Model: RAV EV
Price: $49,800.00
Quantity: 1500
Release Date: 9/1/12

# orderBy





- Sorting your data for user interaction in a logical manner
  - By default **orderBy** will sort in ascending order

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars | orderBy:'price'”>
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# orderBy [cont.]

- orderBy ascending: Output

**Cars**

Car: 1
Make: Nissan
Model: LEAF
Price: $27,620.00
Quantity: 600
Release Date: 12/1/10

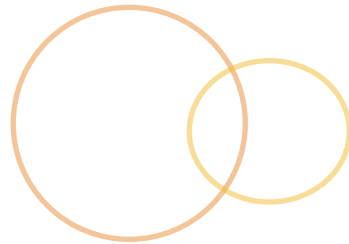
Car: 2
Make: Chevy
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

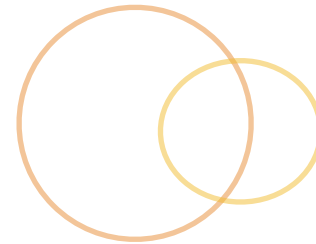
Car: 3
Make: Toyota
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

Car: 4
Make: Toyota
Model: RAV EV
Price: $49,800.00
Quantity: 1500
Release Date: 9/1/12

Car: 5
Make: Tesla
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

# orderBy





- Descending orderBy is done by adding a ‘-‘ in front of the property
  - Same as **orderBy:’-price':true**

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars | orderBy:'-price'”>
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# orderBy [cont.]

- orderBy descending: Output

**Cars**

Car: 1
Make: Tesla
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Car: 2
Make: Toyota
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

Car: 3
Make: Toyota
Model: RAV EV
Price: $49,800.00
Quantity: 1500
Release Date: 9/1/12

Car: 4
Make: Nissan
Model: LEAF
Price: $27,620.00
Quantity: 600
Release Date: 12/1/10

Car: 5
Make: Chevy
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

# orderBy [cont.]

- We can make our filter more complex by moving the criteria to the controller

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars | orderBy:orderByCriteria:true”>
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# orderBy [cont.]

- We can make our filter more complex by moving the criteria to the controller
  - Create a function defining the orderBy criteria
  - Pass a object (i.e. car) for comparison
  - Return a number out of the function for loop iteration comparison

```
app.controller('DemoController', ['$scope', function($scope)
{

  $scope.orderByCriteria = function(car) {
    //Sort by car price unless there is no quantity of the
    // car available
    return car.quantity > 0 ? car.price : car.quantity;
  };

  //. . .
};
```

# orderBy [cont.]

- orderBy: Output
  - Changed quantity of Toyota RAV EV & Nissan LEAF to 0

**Cars**

Car: 1
Make: Tesla
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Car: 2
Make: Toyota
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

Car: 3
Make: Chevy
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

Car: 4
Make: Toyota
Model: RAV EV
Price: $49,800.00
Quantity: 0
Release Date: 9/1/12

Car: 5
Make: Nissan
Model: LEAF
Price: $27,620.00
Quantity: 0
Release Date: 12/1/10

# orderBy [cont.]

- Using multiple predicates

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars | orderBy:['make', '-price']”>
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# orderBy [cont.]

- Multiple predicate: Output

**Cars**

Car: 1
Make: CHE.
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

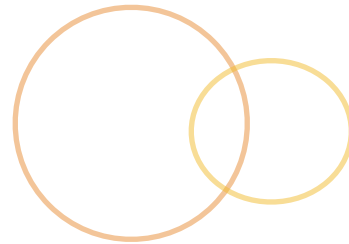
Car: 2
Make: NIS.
Model: ZLEAF
Price: $27,620.00
Quantity: 0
Release Date: 12/1/10

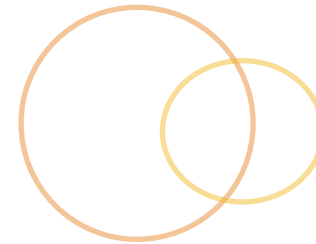
Car: 3
Make: TES.
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Car: 4
Make: TOY.
Model: RAV EV
Price: $49,800.00
Quantity: 0
Release Date: 9/1/12

Car: 5
Make: TOY.
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

# Chaining





- Chaining the filters together

```
<section ng-controller="DemoController">
  <h1>Cars</h1>
  <p ng-repeat="car in cars | orderBy:'make' | limitTo:2”>
    Car: {{$index + 1}}<br>
    Make: {{car.make}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}
    Release Date:
    {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# Chaining [cont.]

- Output

**Cars**

Car: 1
Make: CHE.
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

Car: 2
Make: NIS.
Model: ZLEAF
Price: $27,620.00
Quantity: 0
Release Date: 12/1/10

# Custom Collection Filters

- Let's create an **available quantity** filter
  - The quantity of the car needs to be above 0
  - Loop through the collection and filter

```
.filter('showAvailable', function() {
  return function(data) {
    var holder = [];
    if (angular.isArray(data)) {
      for(var i=0; i<data.length; i++) {
        if(data[i].quantity) {
          holder.push(data[i]);
        }
      }
    }
    return holder;
  };
});
```

# Custom Collection Filters [cont.]

- Using our filter

```
<section>
  <h1>Cars</h1>
  <p ng-repeat="car in cars | showAvailable">
    Car: {{$index + 1}}<br>
    Make: {{car.make | abbreviate}}<br>
    Model: {{car.model}}<br>
    Price: {{car.price | currency}}<br>
    Quantity: {{car.quantity}}<br>
    Release Date: {{car.releaseDate | date:'shortDate'}}
  </p>
  <p>
    {{cars | json}}
  </p>
</section>
```

# Custom Collection Filters

- Output

**Cars**

Car: 1
Make: TES.
Model: S
Price: $69,000.00
Quantity: 1000
Release Date: 6/1/12

Car: 2
Make: TOY.
Model: Prius
Price: $34,720.00
Quantity: 800
Release Date: 1/1/97

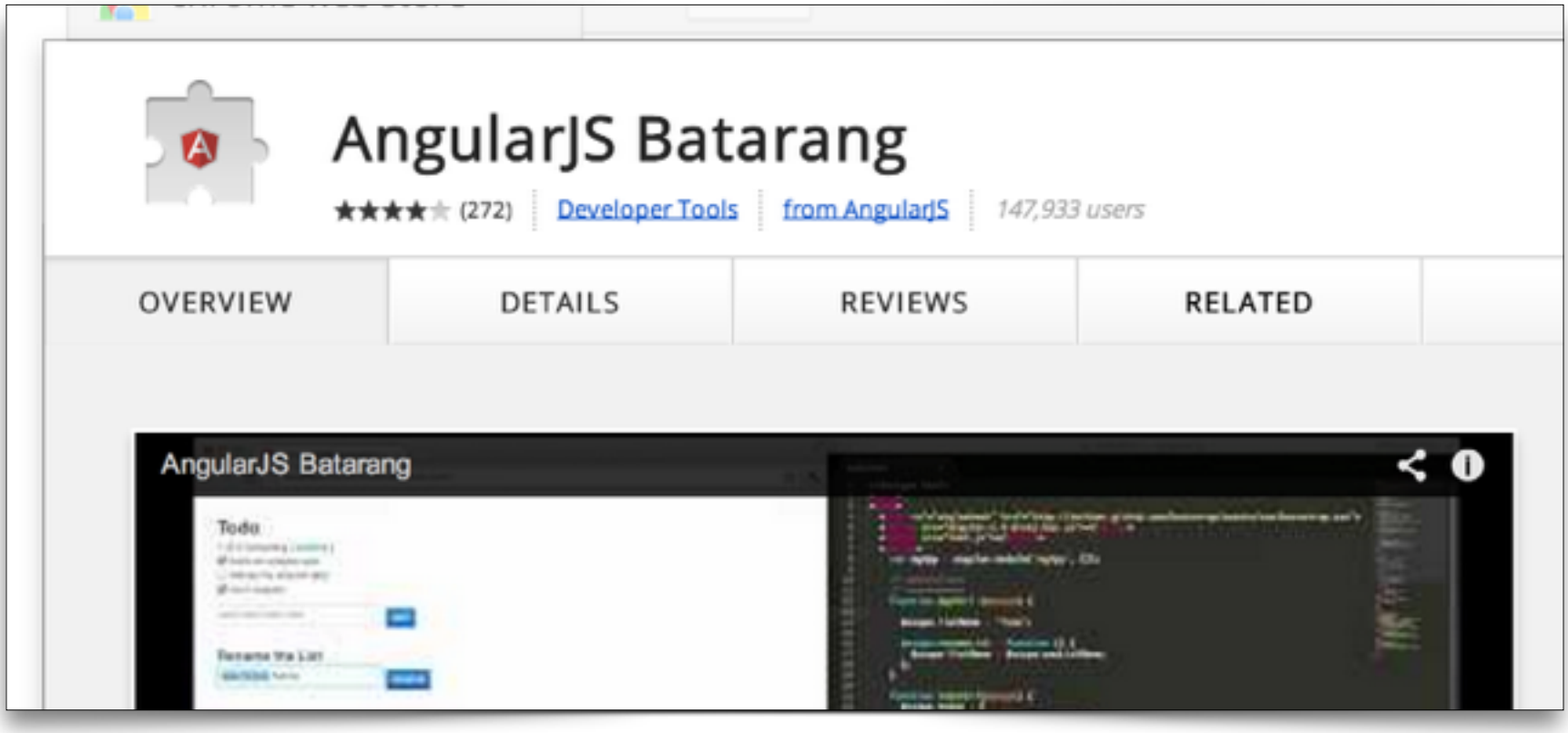
Car: 3
Make: CHE.
Model: Volt
Price: $34,170.00
Quantity: 900
Release Date: 12/12/10

# Lab 4

- Create a sales history page
  - Create object literal with 10 sales
  - Display the tabular data on the screen
    - Make the odd/even rows look different
  - Give the ability for your user to pick how many rows to see
    - Let the user know how many total records there are
- Filter the data
  - Make the numbers look nice
  - Make the date show like this: **JAN 1, 2012**
  - Limit the report to 5 rows
- Break the controllers up into separate files
- Set today's date in the header
  - <time datetime="1970-1-1">January 1, 1970</time>

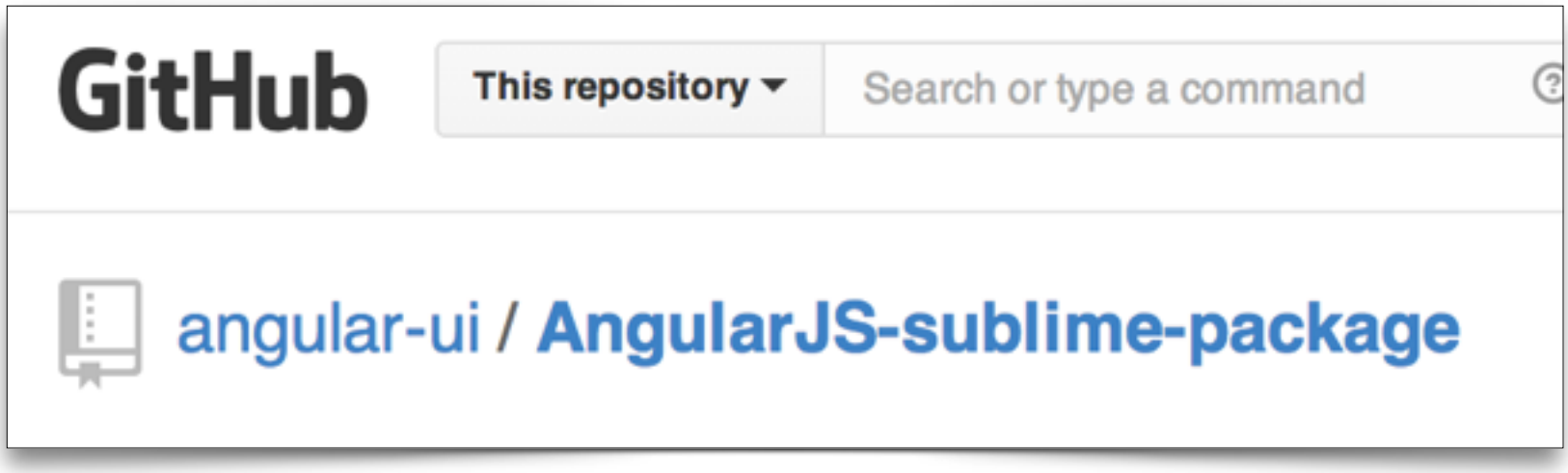
# Batarang

- Angular has a nifty Chrome plug-in to help with debugging your application
  - Let's take a look



# Sublime AngularJS

- By now it is okay to tell you about the AngularJS plugin for sublime
  - It will help with code completion :)



# Form Directives

# ng-disabled

- Binds the **disabled** attribute to the tag
  - When it is true the attribute **disabled="disabled"** is added to the <input> tag
  - Boolean attribute

```
<section ng-controller="DemoController">
  <!-- stuff & things --!>
  <input ng-model="sentence.feeling"
    ng-disabled="form.isDisabled" type="text" />
  <!-- stuff & things --!>
</section>
```

```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.form = {
    isDisabled: true
  };
}]);
```

# ng-readonly

- Binds the **readonly** attribute to the tag
  - When it is true the attribute **readonly="readonly"** is added to the <input> tag
  - Boolean attribute

```
<section ng-controller="DemoController">
  <!-- stuff & things --!>
  <input ng-model="sentence.feeling"
    ng-readonly="form.isReadonly" type="text" />
  <!-- stuff & things --!>
</section>
```

```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.form = {
    isReadonly: true
  };
}]);
```

# ng-checked

- Binds the **checked** attribute to the tag
  - When it is true the attribute **checked="checked"** is added to the <input> tag
  - Boolean attribute

```
<section ng-controller="DemoController">
  <!-- stuff & things --!>
  <input type="checkbox" ng-checked="form.isChecked">
  <!-- stuff & things --!>
</section>
```

```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.form = {
    isChecked: true
  };
}]);
```

# ng-selected

- Binds the **selected** attribute to the tag
  - When it is true the attribute **selected =“selected”** is added to the <input> tag
  - Boolean attribute

```
<section ng-controller="DemoController">
  <select  id="foo">
    <option>A</option>
    <option ng-selected="form.isSelected">B</option>
    <option>C</option>
  </select>
</section>
```

```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.form = {
    isSelected: true
  };
}]);
```

# select Directive

- HTML <select> element with data binding
  - Utilizes ngOptions and ngModel
  - ng-options iterates through a model and builds the child <option> elements of the <select> element

```
<p>
  Why don't you choose a car
  <select ng-options="car.model for car in cars"
        ng-model="currentlySelected">
  </select>
  {{currentlySelected.make}}
</p>
```

```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.cars = [{ //… }];
  //Setting the default option
  $scope.currentlySelected = $scope.cars[1];
}]);
```

# ng-change

- Binds input change event to an action/expression
  - When the user input changes the action will be called

```
<section ng-controller="DemoController">
  <input type="text" ng-model="sentence.speak"
    ng-change="amplify()"/>
  <h1>{{sentence.yell}}</h1>
</section>
```

```
app.controller('DemoController', ['$scope', function($scope) {
  $scope.amplify = function() {
    $scope.sentence.yell = $scope.sentence.speak.toUpperCase();
  };
}]);
```

# More Events

- ng Events allow us to specify custom behavior for browser events
- ng Events correspond to our normal browser events
  - ng-focus: Customer behavior when an element gains focus
  - ng-keydown / ng-keyup / ng-keypress
    - Custom behavior for key events
  - ng-mousedown, ng-mouseenter, ng-mouseleave, ng-mousemove, ng-mouseover, ng-mouseup
    - Custom behavior for mouse events

# ng-submit

- Binds form submit event to an action/expression
  - When a form is submitted the action is called
  - It will prevent the default form submission if no **action** attribute is present on the <form> element

```
<section ng-controller="DemoController">
  <form ng-submit="submit()"
    ng-controller="CarFormController">
    Why don't you choose a car
    <select ng-options="car.model for car in cars"
      ng-model="currentlySelected">
    </select>
    <p>
      {{currentlySelected.make}}
      {{buyer.purchase.price}}
    </p>
    <input type="submit" value="go" />
  </form>
</section>
```

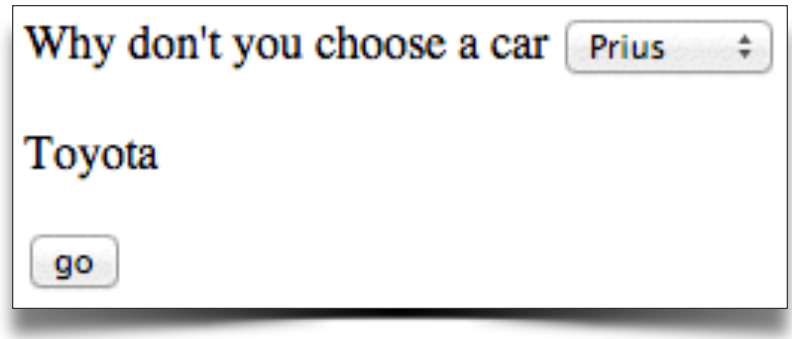
# ng-submit [cont.]

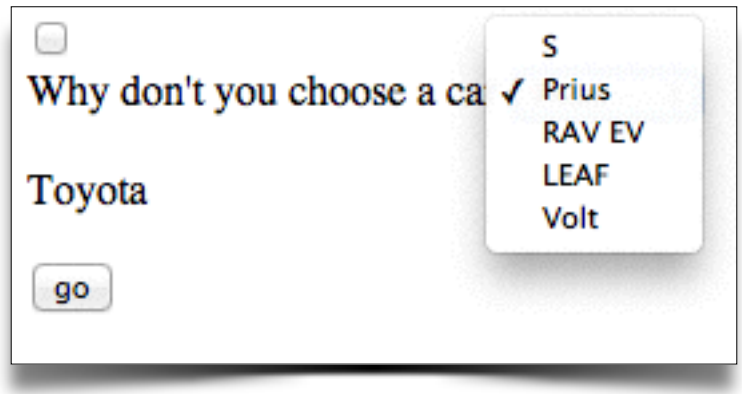
- Create a separate controller for the car form
  - Keeps our controllers slim
  - We still have access to its parent scope with all the car information

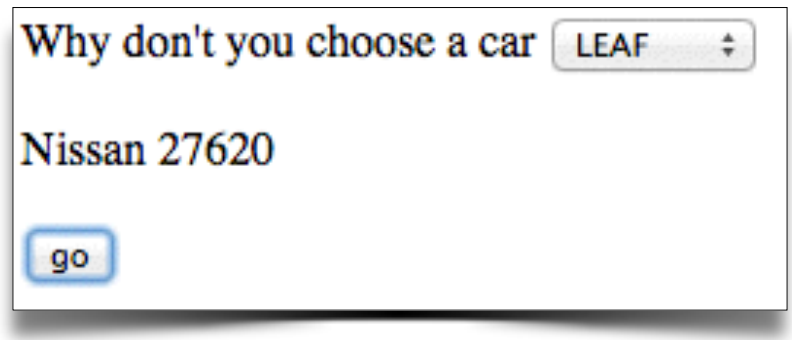
```
app.controller('CarFormController', ['$scope',
function($scope) {

  $scope.buyer = {
    purchase: ''
  };

  $scope.submit = function() {
    $scope.buyer.purchase = $scope.currentlySelected;
  };

}]);
```

# ng-submit [cont.]

- Output



# Form Validation

# Form Validation

- We need to give client validation to our users for good usability and a pleasant user experience
  - Don’t think of client validation as a way to secure your application
  - It can be circumvented easily
- Client-side validation gives users that instantaneous feedback!

# Angular vs HTML5 validation

- Angular is going to give us a mix of HTML5 validation attributes and its own directives
- You need to place **novalidate** attribute on the <form> element itself
  - This will ensure native form validation is turned off
  - We want Angular to do the validation

```
<form novalidate>
  <!-- ... --!>
</form>
```

# Angular Validation Basics

- Angular wants us to
  - Put a **name** attribute on form element
  - Put a **name** attribute on input elements
  - Put an **ng-model** on the input elements
    - If you forget the **ng-model** the CSS class bindings won't take place

```
<form name="buyerInformation" novalidate>
  <!-- ... --!>
</form>
```

# ng-required Directive

- **required**
  - HTML5 attribute that is placed on an <input> element to specify the user needs to fill it out before submitting the form
    - It is simply placed as a flag
- **ng-required**
  - Allows us to set the required attribute
  - **ng-required="true"** adds the required html attribute
  - **ng-required="false"** removes it

# ng-required Directive [cont.]

- Angular keeps **ng-invalid** and **ng-invalid-required** classes on <input> until something has been filled in
  - If you just use **require** and not **ng-require** you will also get the above classes added
- Once the field has input then the classes change to **ng-valid** and **ng-valid-required**

**In our HTML code**

```
<input type="text" ng-required="true" />
```

**Generated by angular**

```
<input type="text" ng-required="true" required />
```

# ng-minlength Directive

- There is no specific HTML5 attribute for minimum length in an <input> element
  - This would have to be done via the pattern attribute
- Angular gives us the **ng-minlength** attribute
  - Angular keeps **ng-invalid** and **ng-invalid-min-length** classes on <input> until at least 3 characters have been reached

```
<input type="text" ng-minlength=3 />
```

# ng-maxlength Directive

- There is no specific HTML5 attribute for maximum length in an <input> element
  - This would have to be done via the pattern attribute
- Angular gives us the **ng-maxlength** attribute
  - Angular keeps **ng-valid** and **ng-valid-max-length** classes on <input> until at least characters have been reached
  - After 20 characters Angular changes to **ng-invalid** and ng-**invalid-max-length**

```
<input type="text" ng-maxlength=5 />
```

# ng-pattern Directive

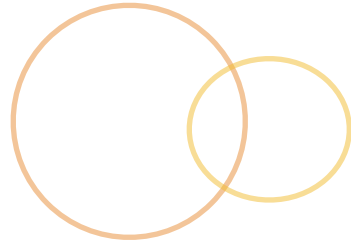
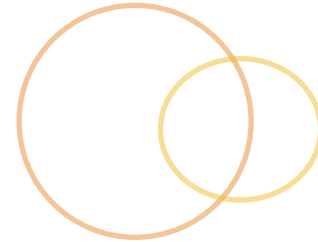
- Allows us to specify a regular expression to use for validating the form
  - Angular keeps **ng-valid** and **ng-valid-pattern** classes on <input> until at the pattern is not met
  - If the pattern is not met the classes are changed to **ng-invalid** and **ng-invalid-pattern**

```
<input type="text" ng-pattern="/[a-zA-Z]/" />
```

# HTML5 Element Validation

- If we want to validate new HTML5 elements we simply need to add an **ng-model** to the input and we will get the added class identifiers

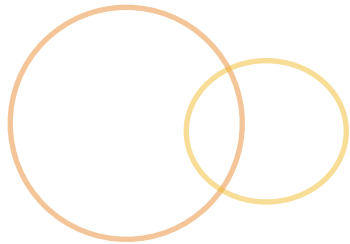
# Email

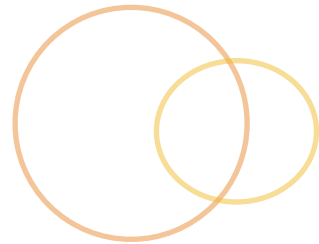





- HTML5 element used to validate email addresses
  - Checks the input has an @ symbol
  - Also will bring a specialized keyboard up on phones/tablets
- Adds **ng-valid** and **ng-valid-email** if the input is valid
  - Changes to **ng-invalid** and **ng-invalid-email** if the input is invalid

```
<input type="email" name="email" ng-model="buyer.email" />
```

# Number

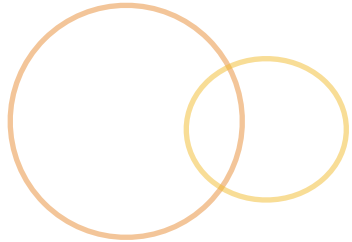
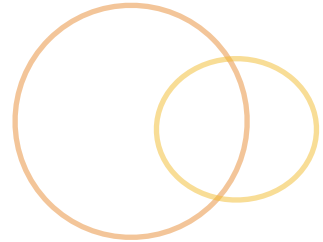




- HTML5 element used to validate numbers
  - Makes sure all input is numeric
  - Also will bring a specialized keyboard up on phones/tablets
  - Depending on browser it will bring up a rocker to go up and down in value
- Adds **ng-valid** and **ng-valid-number** if the input is valid
  - Changes to **ng-invalid** and **ng-invalid-number** if the input is invalid

```
<input type="number" name="number" ng-model="buyer.age" />
```

# URL


- HTML5 element used to validate a URL
  - Checks the input begins with http:// or https://
  - Also will bring a specialized keyboard up on phones/tablets
- Adds **ng-valid** and **ng-valid-url** if the input is valid
  - Changes to **ng-invalid** and **ng-invalid-url** if the input is invalid

```
<input type="url" name="url" ng-model="person.bankURL" />
```

# Form Styling

- We have seen some classes that are added for validation to our <input> element
- We also have a couple others
  - ng-valid: Lets us know when a form/input element is valid
  - ng-invalid: Lets us know when a form/input element is invalid

```
input.ng-invalid{
        border: 1px solid red;
}
```

# Control Variables

- Just like we have classes in the DOM for styling we also have access to the properties within the form controller
- Our form properties are available in the $scope the form is in
  - We can check individual input fields or the form as a whole

# Control Variables [cont.]

- Good idea to check control variables on a form submission or on the user leaving an input element (i.e. the onblur event)

```
<form name="buyerInformation" novalidate ng-submit="submitInfo()"
  ng-controller="BuyerFormController">
    <input ng-model=“person.name" ng-blur="leavingName()"
      name="name" ng-pattern="/[a-zA-Z]/" type=“text”/>
    <input type="submit" value="go" />
</form>
```

# Control Variables [cont.]

- **$pristine**
  - **true** if the form/input property has not been modified
- **$dirty**
  - **true** if the form/input has been modified

```
app.controller('BuyerFormController', ['$scope', function($scope) {
  $scope.submitInfo = function() {
    //Form as a whole
    console.log('form $pristine:' +
     $scope.buyerInformation.$pristine);
    console.log('form $dirty:' +
     $scope.buyerInformation.$dirty);
    //Individual input
    console.log('Buyer name $pristine:' +
      $scope.buyerInformation.name.$pristine);
    console.log('Buyer name $dirty:' +
      $scope.buyerInformation.name.$dirty);
  };
}]);
```

# Control Variables [cont.]

- **$valid**
  - **true** if the form/input field is valid
- **$invalid**
  - **true** if the form/input field is not valid

```
app.controller('BuyerFormController', ['$scope', function($scope) {
  $scope.submitInfo = function() {
    //Form as a whole
    console.log('form $valid:' +
     $scope.buyerInformation.$valid);
    console.log('form $invalid:' +
     $scope.buyerInformation.$invalid);
    //Individual input
    console.log('Buyer Name $valid:' +
      $scope.buyerInformation.name.$valid);
    console.log('Buyer Name $invalid:' +
      $scope.buyerInformation.name.$invalid);
  };
}]);
```

# Control Variables [cont.]

- **$error** object
  - Contains information about the input elements that have an Angular Validation Directive on them
  - If the input element has no error then it will only have a false as the value

```
> $scope.buyerInformation.$error
<· ▶ Object {pattern: Array[2], url: false}
```

# ng-show Directive

- Allows us to show and hide DOM elements
- ng-show="false"
  - Adds **ng-hide** class to the DOM element
- ng-show="true"
  - Adds no class to the element because it is just showing :)

**Our HTML code**

```
<div ng-show="false">Hello World</div>
```

**Angular generated code**

```
<div ng-show="false" class="ng-hide">Hello World</div>
```

**Angular CSS**

```
ng-hide {display: none !important;}
```

# ng-hide Directive

- Allows us to hide and show DOM elements
- ng-hide="true"
  - Adds **ng-hide** class to the DOM element
- ng-hide="false"
  - Adds no class to the element because it is just showing :)

**Our HTML code**

```
<div ng-hide="true">Hello World</div>
```

**Angular generated code**

```
<div ng-hide="true" class="ng-hide">Hello World</div>
```

**Angular CSS**

```
ng-hide {display: none !important;}
```

# Showing Errors

- Angular pre 1.3
- Need to use an extra HTML element that will be hidden/shown if there are validation errors

```
<input name="name" type="text" ng-model="person.name"
  ng-pattern="/[a-zA-Z]/" />
<span ng-show="buyerInformation.name.$dirty &&
  buyerInformation.name.$invalid">
  Please contain a letter
</span>
```

# Showing Multiple Errors

- If one input has multiple errors you can show only the relevant errors as follows using AngularJS 1.2

```
<input name="name" type="text" ng-model="person.name"
  ng-pattern=“/[a-zA-Z]/" ng-minlength=“3” />

<div ng-show="buyerInformation.name.$dirty &&
  buyerInformation.name.$invalid">
  <div ng-show=“buyerInformation.name.$error.pattern”>
       Please enter only letters.
  </div>
  <div ng-show=“!buyerInformation.name.$error.pattern &&
buyerInformation.name.$error.minlength”>
       Please enter at least 3 characters.
  </div>
</div>
```

# Disabling the submit

- **Check validity of form and enable submission**

```
<button type="submit" ng-disabled="myFormName.$invalid">
Save
</button>
```

# ng-messages Directive

- Angular 1.3 +
- ngMessages
  - 1.3 Upgrade needs to be brought in and utilized via dependency injection like ng-routes
  - It is an angular module
    - Needs to be injected into our app

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

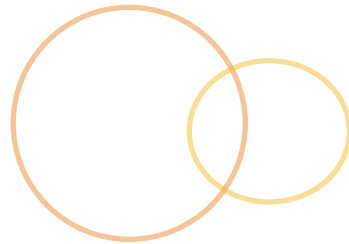
  - Can show one error or multiple errors
    - ng-messages
    - ng-messages-multiple

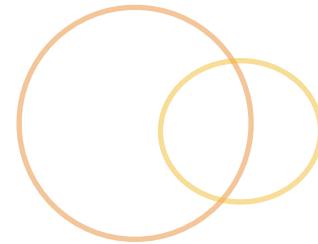
# ng-messages Directive [cont.]

- Errors are shown in the order listed
  - required, pattern, minlength

```
<input name="name" type="text"
  ng-model="person.name"
  ng-pattern="/[a-zA-Z]/"
  ng-minlength=3
  />
<div class="error"
  ng-messages="buyerInformation.name.$error"
  ng-messages-multiple>
  <div ng-message="pattern">Please conain a letters</div>
  <div ng-message="minlength">Longer than 3 please</div>
</div>
```

# Lab 5





- Create a give page with information about the charity we are giving to
- Build a form for user submission to donate to Alex's Lemonade Stand
  - Get person's name (min length, required)
  - Get a person's phone number (valid format)
  - Get their mailing address
  - Get their zip-code (pattern, required)
  - Get their email address (valid format)
  - Have them select the best way to contact them
  - On submission display a thank you message
  - If the form isn't valid display error(s)

# Directives

# Directives

- We have seen some directives so far
- Directives focus on DOM element manipulation
  - Angular's way of creating new HTML elements
  - Extending HTML's functionality
- DOM manipulation belongs in the Directives
  - No DOM manipulation in controllers or services
- Let's take a look at a few more

# URL Directive

- ng-href attribute
  - Boolean like directive
  - The **ng-href** will make sure the user's interaction with the application goes smoothly and the link is correct
  - https://docs.angularjs.org/api/ng/directive/ngHref
- Best practice for dealing with dynamic URLs
  - Use this over the simple **href** attribute
  - The **href** does not guarantee that the link will be correct
  - A user could click on the link before the link variable has been filled (i.e. they will get a 404)
  - **ng-href** does not allow link interaction until the URL has been resolved

# Image Directive

- ng-src attribute
  - Boolean like directive
  - Forces the browser to wait to load the image until the **src** has been resolved
  - https://docs.angularjs.org/api/ng/directive/ngSrc
- Best practice when dealing with dynamically loaded images
  - Use this over the simple **src** attribute
  - It can be buggy with loading if you use **src**
  - The browser may try to fetch the URL “{{image_source}}” until the variable is replaced :(

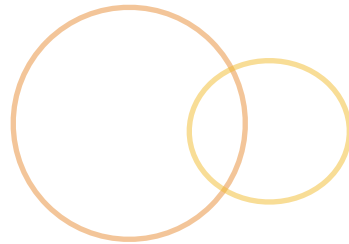
# Child Scope Creation

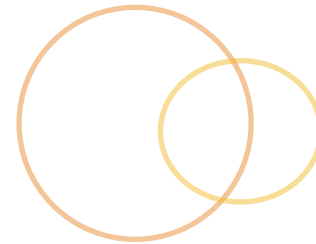
- Some directives create a child scope within them
  - We have seen ng-app, ng-controller and ng-view so far

# ng-if

- Allows us to handle **if** logic in out templates
  - If something is true do it
- Completely removes or recreates and element
  - ng-show / ng-hide just shows and hides via CSS
  - When an element is removed its scope is destroyed
  - When it comes back it has a newly created scope

# ng-if [cont.]





```
<div ng-if="'2' === 2">
  Not gonna show
</div>
```

```
<div ng-if="2 === 2">
  Gonna show up
</div>
```

```
<!-- ngIf: '2' === 2 -->
<!-- ngIf: 2 === 2 -->
<div ng-if="2 === 2" class="ng-scope">
      Gonna show up
    </div>
<!-- end ngIf: 2 === 2 -->
```

# ng-switch

- Allows use to use a switch statement in our templates
- **ng-switch**: Sets up the switch statement
- **on**: The variable to switch on
- **ng-switch-default**: The default statement
- n**g-switch-when**: The cases for the switch statement

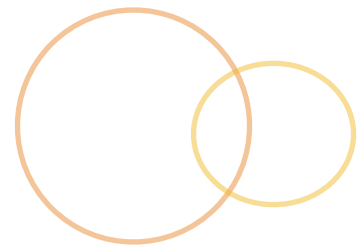
```
<div ng-switch on="buyer.name">
  <h2 ng-switch-default>Who wants a new car?</h2>
  <h3 ng-switch-when="Kamren">{{buyer.name}}</h3>
  <h3 ng-switch-when="Me">Me</h3>
  <h3 ng-switch-when="You">You</h3>
</div>
```

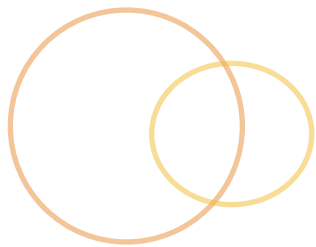
# A New Scope

- Did you just ask yourself… Why do ng-if and ng-switch create new child scopes?

- Memory management
  - When the DOM changes its structure Angular creates new scope
- Angular is a neat freak
  - It likes to clean up after itself
  - Under the covers Angular is managing bindings and listeners
  - Setup and tear down of the scope takes care of these bindings and listeners being handled correctly

# Custom Directives





# Directives

- Declarative
  - They describe objects
  - They don't tell them how to behave
- Data Driven
  - Data is connected to the view elements
  - Scope handles all changes to data for us
- Conversational
  - Directives can speak to each other through events and service interactions

# Directives

- Simply put they give extra functionality to “dumb” DOM elements
  - ng-click adds the ability to have a DOM element listen
  - The “extra” functionality is added via our directive factory function
  - The directive is defined via the **directive** method off a module

```
app.directive('diHome', function() {
  //Fun directive stuff :)
});
```

# Custom Directives

- We can make our own directive
- Let's start with something basic
- Angular will invoke this directive
  - Invoke at HTML compilation

```
<di-home></di-home>
```

# Namespaces

- Prefixes are a good thing
  - Our custom directive namespace
  - Hence “di”
  - Angular uses “ng”

```
<!-- Dasherized -->
<di-home></di-home>

<!-- Colon separated as XML namespace -->
<di:home></di:home>
```

# Directive Creation

- We could simply return a post linking function from our directive
  - This doesn't give us much flexibility

```
app.directive('diHome', function() {
  //Run after Angular does linking on page
  return function postLink(scope, instElement, instAttribute) {
       //We could write watches and custom behavior
       //  Would be set on the instance scope of directive in DOM
  }
});
```

# Directive Creation

- Use the directive factory function to create our directive definition object
  - This JavaScript will back our <di-home> directive
- We use camel casing for the directive name in JS
- We use dasherized casing for the directive in HTML

```
app.directive('diHome', function() {
  return {
    restrict: 'E',
    template: '<a href="http://www.developintelligence.com"
       target="_blank">Take me home!</a>'
  };
});
```

# Directive Rendering

- Our directive renders the href


- The source code shows our custom tag directly in the HTML

```
<di-home>
  <a href="http://www.developintelligence.com" target=
  "_blank">Take me home!</a>
</di-home>
```

# Replace Property

- We don’t need to have our custom tag wrap the template within
  - replace property allows us to swap our template with our custom tag

```
app.directive('diHome', function() {
  return {
    restrict: ‘E',
    replace: true,
    template: '<a href="http://www.developintelligence.com"
       target="_blank">Take me home!</a>'
  };
});
```

```
<a href="http://www.developintelligence.com" target=
"_blank">Take me home!</a>
```

# replace flag

- Angular 1.3 has deprecated the **replace** flag on the directive
- This means all directives going forward will have their custom tags surrounding the directive

# Directive Declaration

- Different ways to declare our directives

```
<!-- As element name … 'E' -->
<di-home></di-home>

<!-- As element attribute … 'A' -->
<div di-home></div>

<!-- As element class … 'C' -->
<div class="di-home"></div>

<!-- As a comment … 'M' -->
<!-- directive:di-home -->
```

# Directive Declaration [cont.]

- Different ways to declare our directives with expressions

```
<!-- As element name … 'E' -->
<di-home action="expression"></di-home>

<!-- As element attribute … 'A' -->
<div di-home="value"></div>

<!-- As element class … 'C' -->
<div class="di-home: expression"></div>

<!-- As a comment … 'M' -->
<!-- directive: di-home expression -->
```

# Directive Instantiation [cont.]

- We need to have our JavaScript directive definition support our directive declarations
  - restrict: ‘EACM’ will allow us create via element, attribute, class or comment

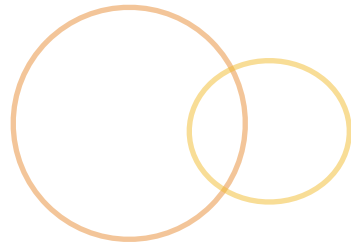
```
app.directive('diHome', function() {
  return {
    restrict: ‘EACM',
    template: '<a href="http://www.developintelligence.com"
       target="_blank">Take me home!</a>'
  };
});
```

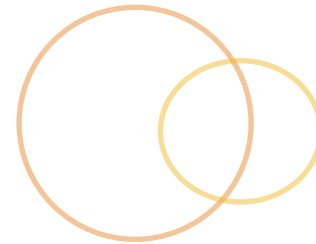
# Directive Instantiation [cont.]

- IE concerns
  - https://docs.angularjs.org/guide/ie
  - IE 8 and below will create our element in the wrong way
    - It will create <di-home> incorrectly
- Best to create directives via attribute

```
<!-- IE 8 and below approved -->
<div di-home></div>

<!-- IE 8 and below NOT going to work -->
<di-home></di-home>
```

# Scope

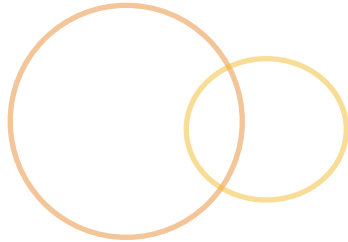


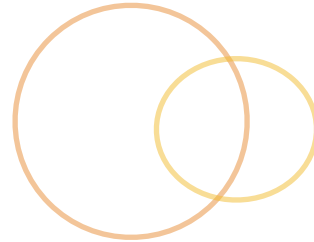


- Are we creating a separate scope for this directive?

```
app.directive('diHome', function() {
  return {
    restrict: 'EACM',
    template: '<a href="http://www.developintelligence.com"
       target="_blank">Take me home!</a>'
  };
});
```

# Scope [cont.]

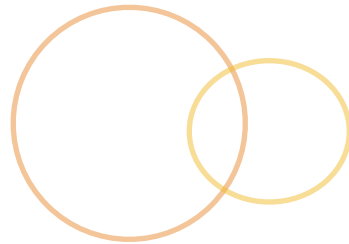


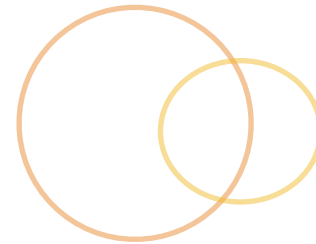


- Same as previous page
  - This directive is using the exact same scope object of the context in which the directive has been placed
  - Scope property defaults to **false**
    - Same scope as parent

```
app.directive('diHome', function() {
  return {
    restrict: 'EACM',
    scope: false,
    template: '<a href="http://www.developintelligence.com"
       target="_blank">Take me home!</a>'
  };
});
```

# Scope [cont.]





- Creating a new scope
  - Set **scope** property to **true**
  - A new scope is created via prototypal inheritance
  - Note: If there are multiple directives declared on the same element and they all request a new scope only 1 new scope will be created

```
app.directive('diHome', function() {
  return {
    restrict: 'EACM',
    scope: true,
    template: '<a href="{{diUrl}}"
      target="_blank">{{diLinkText}}</a>'
  };
});
```

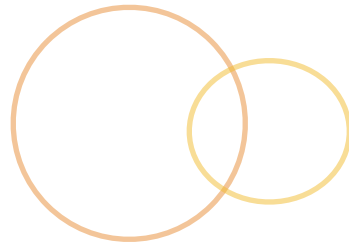
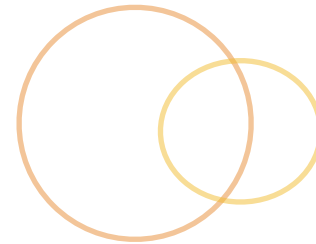
# Scope [cont.]

- How do we pass data into this directive?
  - We want to do what we have below, but it just won't work
  - We are missing a binding strategy
  - Below won't work

```
app.directive('diHome', function() {
  return {
    restrict: 'EACM',
    replace: true,
    template: '<a href="{{diUrl}}"
      target="_blank">{{diLinkText}}</a>'
  };
});
```

```
<div di-home
  di-url="http://www.developintelligence.com"
  di-link-text="Welcome home"></div>
```

# Scope [cont.]


- We could try prototypal inheritance
  - We would still have to have diUrl & diLinkText defined on the parent scope
  - Creating an inherited scope doesn't help

```
app.directive('diHome', function() {
  return {
    restrict: 'EACM',
    replace: true,
    scope: true,
    template: '<a href="{{diUrl}}"
       target="_blank">{{diLinkText}}</a>'
  };
});
```

```
<div di-home
  di-url="http://www.developintelligence.com"
  di-link-text="Welcome home"></div>
```

# Isolate Scope [cont.]

- Isolate scope is created via assigning an object literal to the **scope** property
  - **scope: {}**
  - Good for creating directives that can be reused without worrying about what context they are placed within
- Not created via prototypal inheritance
  - No access to the parent $scope
  - No access to the $rootScope
- A completely separate scope
  - Destroys the prototype chain

# Isolate Scope [cont.]

- Created with an object literal assigned to the scope property
  - Very useful for creating modular components
  - We won't manipulate parent scope

```
app.directive('diHome', function() {
  return {
    restrict: 'EA',
    replace: true,
    scope: {}
    template: '<a href="{{diUrl}}"
       target="_blank">{{diLinkText}}</a>'
  };
});
```

# Binding Strategies

- We have an isolate scope
  - Separate from the world :)
  - That is kind of useless …
- We are going to need a way to have our application interact with that directive/scope somehow
- Angular gives us mechanisms to bind data from that outside world

# Binding Strategies [cont.]

- Local scope property
- Bi-directional
- Parent execution

- Allow us to have the directive's isolate scope inherit from parent scope

# Local Scope Strategy

- Allows us to copy a value from an attribute on our DOM into our isolate scope
  - We are specifying only the variables that can come in
  - Kind of like the directive's API
- Value: '@'
- The binding strategy is setup per property
  - We don't set it up directive wide

# Local Scope Strategy [cont.]

- Finally we get a working solution

```
app.directive('diHome', function() {
  return {
    restrict: 'EA',
    replace: true,
    scope: {
      diUrl: '@',
      diLinkText: '@'
    },
    template: '<a href="{{diUrl}}"
       target="_blank">{{diLinkText}}</a>'
  };
});
```

```
<div di-home
  di-url="http://www.developintelligence.com"
  di-link-text="Welcome home"></div>
```

# Local Scope Strategy [cont.]

- We can decouple the values passed in from our API

```
app.directive('diHome', function() {
  return {
    restrict: 'EA',
    replace: true,
    scope: {
      diUrl: '@diChangeToUrl',
      diLinkText: '@'
    }
    template: '<a href="{{diUrl}}"
       target="_blank">{{diLinkText}}</a>'
  };
});
```

```
<div di-home
  di-change-to-url="http://www.developintelligence.com"
  di-link-text="Welcome home"></div>
```

# Bi-directional Binding Strategy

- We can setup a binding between the directive's local scope property and what would be the parent's scope property
  - (i.e. This is our normal Angular 2 way binding)
- Value: '='
- We can also decouple via: '=attributeName'
  - Same decoupling as '@'

# Bi-directionalScope Strategy [cont.]

- Utilizing bi-directional scope
  - Parent controller needs to contain a property $scope.diUrl

```
app.directive('diHome', function() {
  return {
    restrict: 'EA',
    replace: true,
    scope: {
      diUrl: '=',
      diLinkText: '='
    }
    template: '<a href="{{diUrl}}"
       target="_blank">{{diLinkText}}</a>'
  };
});
```

```
<div di-home
  di-url="diUrl"
  di-link-text="diLinkText"></div>
```

# Parent Execution Binding Strategy

- Allows for the execution of a method on what would be the parent scope
  - (i.e. we can treat the directive as a child that still has access to parent methods)
- Value: '&'

# Parent Execution Binding [cont.]

- Utilizing parent execution binding strategy
  - Parent controller needs to contain a method $setAReference

Directive JS

```
scope: {
  action: '&'
},
link: function(scope) {
  scope.someAction = function() {
    scope.action({make: 'Toyota, model: 'Sienna'});
  }
```

Directive HTML

```
<li a-directive
  action='setAReference(model, make)'></li>
```

Parent Controller JS

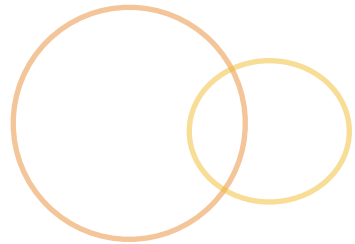
```
$scope.setAReference(carModel, carMake) {
  $scope.carModel = carModel;
  $scope.carMake = carMake;
}
```

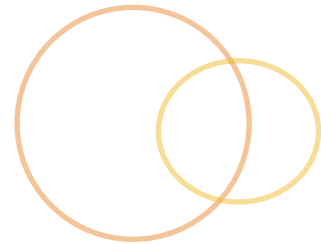
# replace flag

- Angular 1.3 has deprecated the **replace** flag on the directive
- This means all directives going forward will have their custom tags surrounding the directive

# Complete Angular Bootstrap Process

# The Boot





- What really goes on when Angular boots up?

- At first there are just a bunch directives in our HTML code
- Angular starts up and our application gets invoked
- A compile phase takes place
- A linking phase takes place
- Users interact with our application

# Application Startup … updated

**Client**

Downloads AngularJS

AngularJS registers DOMContentLoaded event listener with the browser

DOMContentLoaded event is fired AngularJS event handler callback is executed

**AngularJS**

AngularJS crawls the DOM for the ng-app directive

The module associated with the ng-app directive is loaded

The $injector is created. Which in turn creates the $rootScope and the $compile service

**Compilation with $compile**

# Application Startup … updated

**Compilation with $compile**

The $injector is created. Which in turn creates the $rootScope and the $compile service

$compile service links ng-app DOM element with the $rootScope

$compile crawls the DOM and finds all the directives.

$compile orders directives based on individual priorities

**Compile Phase**

$compile grabs each of the directive compile functions and executes them

Opportunity for the directive to modify the DOM. Directives aren't bound with scope data yet

Each individual directive compile returns the template function, as the link function of the directives

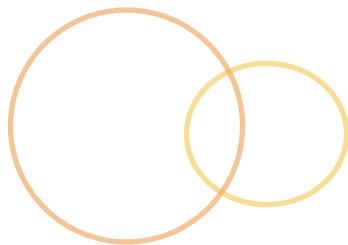
**Link Phase**

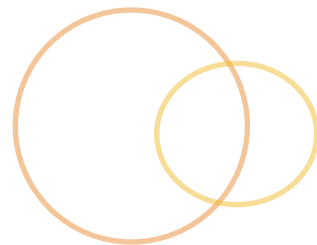
Starts with the main directive compile function and walks down through each child directive

link functions connect the compiled directives (i.e. the templates) to the $scope and to the $rootScope

The application is all setup

# Run Time / Execution: After the Compilation





# Browser Events

- Event listeners are registered with the browser
  - When an event is triggered the event handler callbacks are executed for the appropriate event listener
  - DOM manipulation and/or business logic ensues

# Angular Events

- After the Angular application is all bootstrapped a waiting game ensues for events
  - Could be user based, network based …
- Angular utilizes event listening registration
  - The registration takes place in the directives
  - Hence the ng-click directive registers an event listener
  - That's not the whole story

# Angular Event [cont.]

- Angular adds a layer of event processing on top of the way the browser handles events
- Angular's processing of events is handled within the $digest loop
  - The $digest loop runs the directives event handler
  - After the $digest loop is all finished the DOM is re-rendered

# Entering the $digest Loop

- An event fires
- The event handler is called within the directive context
  - That directive context is within the AngularJS execution context
- AngularJS calls the directive via $apply()
  - $apply() is invoked off of the encompassing scope
  - This kicks off the $digest cycle

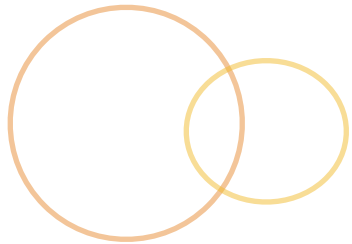
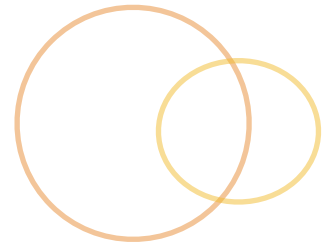
# $digest Loop

- Handles the synchronization of the the controller, scope and view
  - $digest loop executes on $rootScope
- Contains the evalAsync queue and the $watch list
  - These handle work that needs to be done and variables that have the possibility of changing

# $digest Loop [cont.]

- If a change to a variable is detected, $watch is called
  - If that causes another change then $watch is called over and over again until there is nothing to update
  - This is Angular's dirty checking of variables
  - Each watch function is checked to see if any changes have been made
  - If there are changes then the watch functions are called all over again
- Once the $watch list has stabilized and there is nothing scheduled in the evalAsync queue the DOM is rendered

# Recap


- Event listeners, $watch statements and $scope, along with their linking, are setup during compilation

- At runtime events occur and directives handle them
  - $apply() is called off of the directive $scope updating the application
  - The $digest loop is entered and the $watch list checks if updating needs to be done
  - The DOM is re-rendered

# Destruction

- As a note…

- When the $scope isn't needed it is destroyed
- Handled via garbage collection of the directive / controller that created it
  - Automatic via $destroy()
  - Nothing we will need to do ourself

# Custom Directives Continued

# What We Have Seen

- How to instantiate directives
- How to incorporate templates
- Different ways to set up scopes

# Setting Directive Priority

- Just as ng directives have priorities we can set priorities for our directives
  - If no priority is set then it will default to 0
  - Remember ng-repeat is at 1000 so it will get executed before any other directive on the same element

```
app.directive('diHome', function() {
  return {
    priority: 100
  };
});
```

# Terminal Property

- The terminal property will let AngularJS know if it should process any other directives with a lower priority than the current directive on the same element
- Terminal defaults to false
  - Other directives will be automatically compiled by Angular

```
app.directive('diHome', function() {
  return {
    priority: 100,
    terminal: true
  };
});
```

# Terminal Property [cont.]

- ng-repeat utilizes the terminal property: Why?
  - terminal: true
- It will handle all of the the processing of the child elements in its own **compile** method

# TemplateUrl Property

- Besides creating a template inline as a string, we could point to a template that was created in a separate HTML file
  - Template is fetched asynchronously

JS

```
app.directive('diHome', function() {
  return {
    priority: 100,
    terminal: true,
    templateUrl: '/templates/diHome.html'
  };
});
```

diHome.html

```
<a href="{{diUrl}}"
      target="_blank">{{diLinkText}}</a>
```

# Ajax for Templates

- Once a template has been fetched via an ajax call it is cached via $templateCache
- The compiling and linking of the template can't happen until the template is fetched
- Ajax is good thing overall
  - However, too many requests can slow down a system

# Pre-Cache Templates

- In production we might want to lower our number of Ajax calls
- We can pre-cache one or more templates
  - Happens before deploying the application
  - No Ajax call will need to be made for those templates
  - Allows our front-end give a better user-experience
  - Less calls reduces server load
- We will look at this more later with Grunt

# Transclusion Property

- When we modularize our directives we might want other developers to add information into the directive
- Inclusion of DOM elements into our directive
  - transclude: true

# Transclusion Property [cont.]

- We also need to specify in our HTML where the transcluded code should show up
  - ng-transclude

```
<div ng-transclude></div>
```

# Required Property [cont.]

- Allows us to specify another needed directive for the current directive to work
  - require: '^aDirective' : Require a parent directive called **aDirective**
  - That parent directive will have a controller for interaction with this nested directive otherwise there will be an error thrown

# Controller Property

- Allows us to specify a controller for our custom directive
  - We could add properties to the directive $scope
  - We could add methods to the directive $scope
  - Usually used to have nested directives interact with each other

```
app.directive('diHome', function() {
  return {
    controller: function($scope, $element, $attrs, $transclude) {
       //Place for us to control the directive
       //  Useful for interacting with parent scope
    }
  };
});
```

# Compile Function

- Allows us to work on our element before it is inserted in the DOM
  - It Doesn't care about a specific scope
  - Can't be used if we are trying to utilize a DOM related plugin
  - This will be rarely used in custom directives

```
app.directive('diHome', function() {
  return {
    compile: function(tempElement, tempAttributes) {
      //Not something we will use often
      // Used by internal directives like ng-view and ng-repeat
    }
  };
});
```

# Compile Phase [cont.]

- Before the compile phase ends we can interact with the compiled DOM
  - Great time to interact with the DOM
  - No scope/data binding has take place
  - Very low performance cost for manipulation
- ng-repeat goes to work here
  - It builds itself out before data binding occurs
  - It first creates the needed HTML
  - Then the new/manipulated DOM goes to the linking phase

# Link Function

- Allows us to work on our element after it is inserted in the DOM
  - Attaches the compiled DOM element to the appropriate scope
  - Where we will setup our watchers and DOM event listeners

```
app.directive('diHome', function() {
  return {
    link: function(scope, instElement, instAttrs,
       controller, transcludeFunction) {
       //Interact with the specific element after compilation
  };
});
```

# Compile & Link

- Our directives can't have a **compile** property and a **link** property defined
  - If we utilize the **compile** property then our **link** property will be ignored by Angular
- In order to get the compile and link functionality into our directive we will need to have the **compile** function return an object that contains a **pre-link** function and a **post-link** function
  - The **pre-link** would be our compile function
  - The **post-link** would be our link function

# Pre-link / Post-link Function

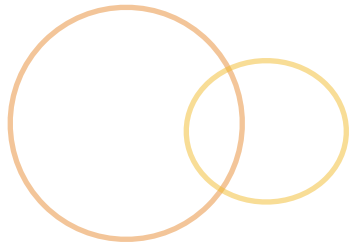
- Angular would execute our compile/pre-link function on our parent directive
  - It then crawls down the DOM tree of child directives
  - Then the compile/pre-link function on our child directive
- At the deepest child Angular will execute our compile/pre-link function and then it will execute our post-link function
- Angular will then walk back up the DOM tree executing the child post-link functions all the way up to and including the parent directive's post-link function

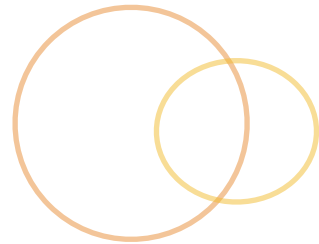
# Pre / Post Function [cont.]

- Allows us to manipulate the directive element in the compile phase and then in the link phase
  - Attaches the compiled DOM element to the scope

```
app.directive('diHome', function() {
  return {
    compile: function() {
      return {
        //Compilation
        pre: function ($scope, $element, $attributes) {},
        //Linking
        post: function link ($scope, $element, $attributes) {}
      }
    }
  };
});
```

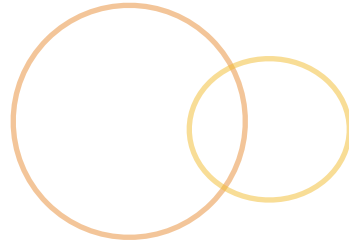
# Lab 6

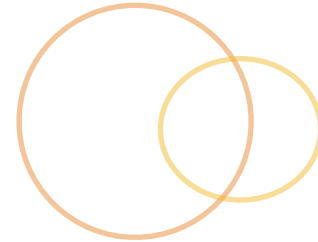




- Create a custom directive for the page title
  - Use the directive across all you pages
- Create a custom directive for your reports page
  - Change up your Sales History Report
    - Remove the table and give it a card based layout
  - When a user selects a card make it show as selected
    - Load your template via Ajax
    - Have the directive utilize your sales history object
    - Have the directive add a selected class to the card when clicked
    - Have the report controller write out wether the month selected was profitable (i.e. over $200 in gross profit)

# Lab 6





- continued…
- Bonus
  - Create a nested footer directive for the card
  - Set a gross profit to display in the footer of the card
  - Set up a random number generator to choose from 3 pictures for thumbs up (i.e. congratulations)
    - Show the picture if it is profitable
  - Whenever a new card is added (i.e. the number is increased) pick one of the 3 pictures randomly to display
    - You will need to specify a $watch function for this

# End