Module 3

Introduction to AngularJS and AngularJS Modules and Filters

Overview

Examples for this section are in the directory `Module3_Samples/angular0_samples`

AngularJS is an MVC (Model-View-Controller) framework for building rich client web applications. In MVC applications, there is a clear separation in the code between the three components. The application logic is handled by the Controller, the data managed by the Model, and the presentation is the View. In AngularJS applications, the controllers are the JavaScript classes, the view is the Document Object Model (DOM), and the model data is stored in object properties.

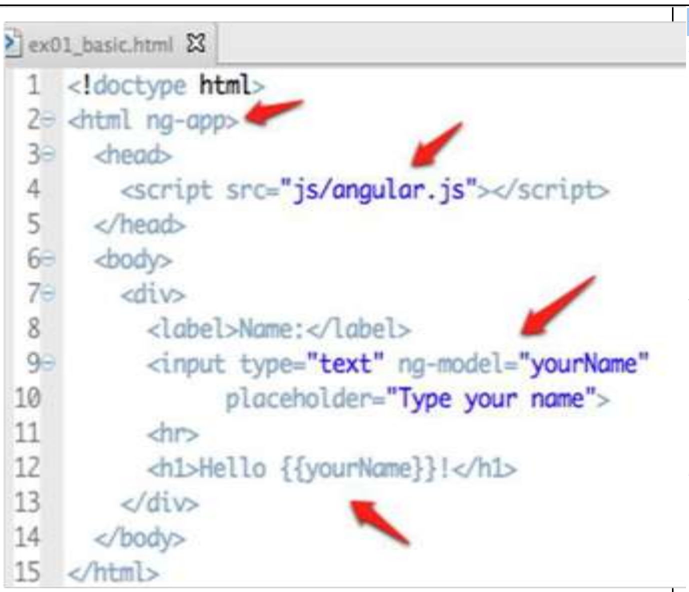
Anatomy of an AngularJS Application

The basic structure of an *AngularJS* application is shown below. The *JavaScript* library *angular.js* (or *angular.min.js*) is loaded for the web application. The ***ng-app*** directive informs Angular which part of the DOM it manages. If the directive ***ng-app*** is included as part of the <html> tag, the application becomes a 100% Angular application and all the DOM elements are managed by Angular.

Angular applications follow the Model-View-Controller pattern. The *Model* contains the data that represents the current state of the application. The *View* displays the application's data. The *Controller* manages the interaction between the model and the view.

In the following example, only the Model and the View are explicitly used. Angular implicitly controls the relationship between these two. The ***ng-model*** directive as part of the input element identifies the model elements. ***The directive provides a two-way data binding between the input element and the model attribute*****.** The ***ng-model*** directive is responsible for binding the view into the model.

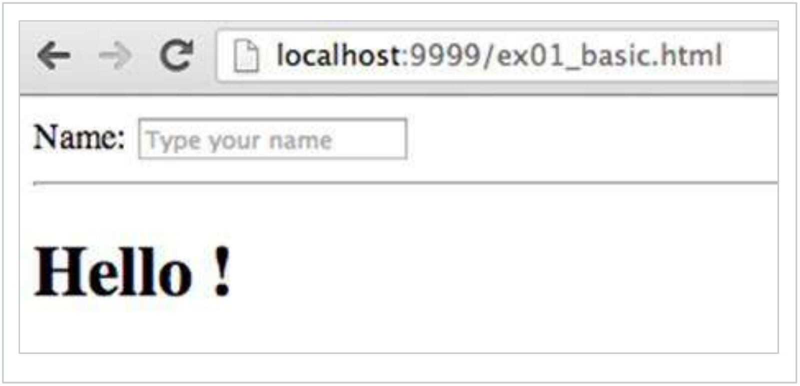
The ***model elements*** *are displayed in the HTML content using the double-curly bracket* (braces, {{…}}) notation as shown below.



```
<!doctype html>
<html ng-app>
  <head>
    <script src="js/angular.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName"
             placeholder="Type your name">
      <hr>
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```

```
<!doctype html>
<html ng-app>
  <head>
    <script src="js/angular10.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text"
ng-model="yourName"
            placeholder="Type your name">
      <hr>
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```

The initial output of the above application is shown below

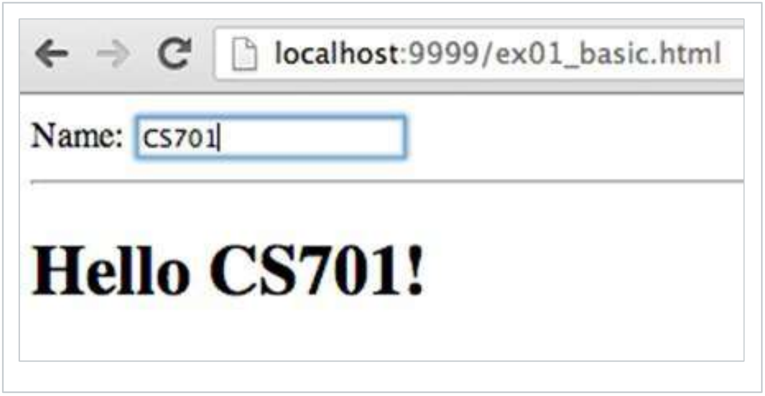


localhost:9999/ex01_basic.html

Name: Type your name

Hello !

As the user types data in the *input* element, the model element *yourName* is bound to the value as it is being typed. The view displays the typed data as shown below



localhost:9999/ex01_basic.html

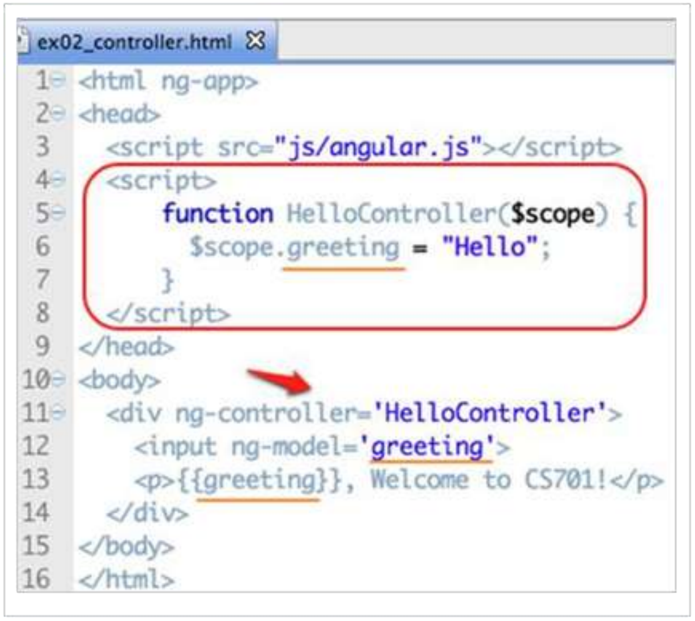
Name: CS701

Hello CS701!

Controller

The controllers represent the code behind the view. ***A controller manages the model, maintains the data in its scope, and publishes the data to the view***. The controllers are identified with the HTML body elements. Controllers can also be nested through specifying different controllers for different parts of the web application.

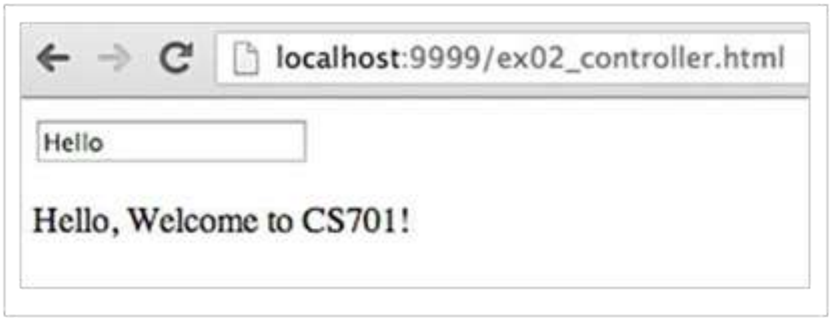
The following sample shows the controller specified for the *div* element of the HTML *body*. ***The scope for the controller is available as its argument, $scope***. The model elements are manipulated using the *$scope* object. When the controller is first initialized, the *greeting* property of the model is set to the initial value *Hello*. The *input* element is associated with the *greeting* model property. As the user changes the content in the *input* element, the corresponding value is set in the model.



```
<html ng-app>
<head>
  <script src="js/angular.js"></script>
  <script>
      function HelloController($scope) {
        $scope.greeting = "Hello";
      }
  </script>
</head>
<body>
  <div ng-controller='HelloController'>
    <input ng-model='greeting'>
    <p>{{greeting}}, Welcome to CS701!</p>
  </div>
</body>
</html>
```

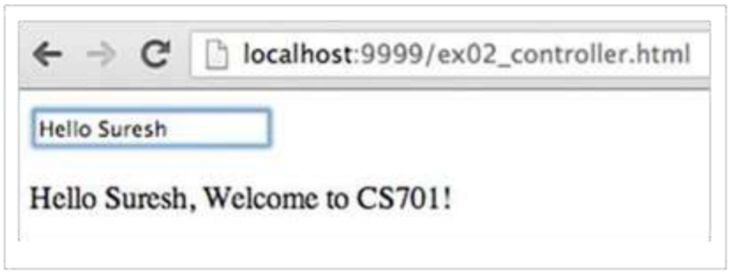
```
<html ng-app>
<head>
  <script
src="js/angular10.js"></script>
  <script>
      function HelloController($scope)
{
        $scope.greeting = "Hello";
      }
  </script>
</head>
<body>
  <div ng-controller='HelloController'>
    <input ng-model='greeting'>
    <p>{{greeting}}, Welcome to
CS701!</p>
  </div>
</body>
</html>
```

The initial display of the application is shown below. The initial value, *Hello*, as initialized in the controller appears as the value in the input box



Hello, Welcome to CS701!

As the user changes the content in the input box, the model is updated and the value displayed as shown below.



Hello Suresh

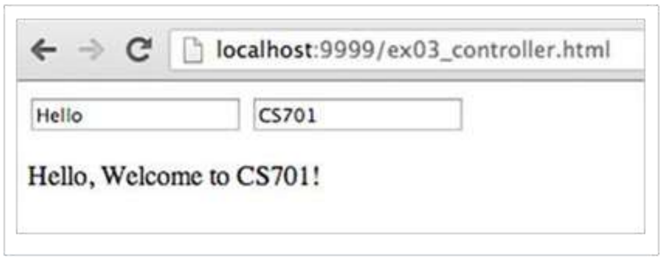
Hello Suresh, Welcome to CS701!

The following associates a JSON object with the model element using the *text* and *course* properties.

File: `ex03_controller.html`

```
<html ng-app>
<head>
  <script src="js/angular10.js"></script>
  <script>
      function HelloController($scope) {
        $scope.greeting =
            { text: 'Hello', course: 'CS701'};
      }
  </script>
</head>
<body>
  <div ng-controller='HelloController'>
    <input ng-model='greeting.text'>
    <input ng-model='greeting.course'>
    <p>{{greeting.text}}, Welcome to {{greeting.course}}!</p>
  </div>
</body>
</html>
```

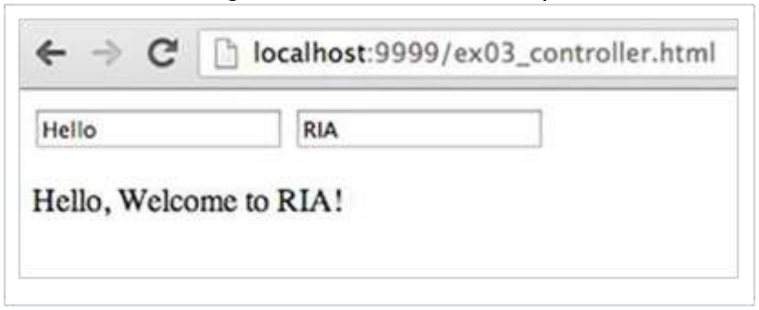
The initial display of the application with the values initialized in the controller is shown below.



Hello CS701

Hello, Welcome to CS701!

As the user changes the content in the input box, the model is updated and the value displayed:



Hello, Welcome to RIA!

Nested Scopes

When different controllers are used with nested levels of the HTML body elements, the scopes associated with the controllers are nested as well. The referenced property is first searched in the current scope. If the property is not defined in the current scope, its parent scope is then searched, and so on.

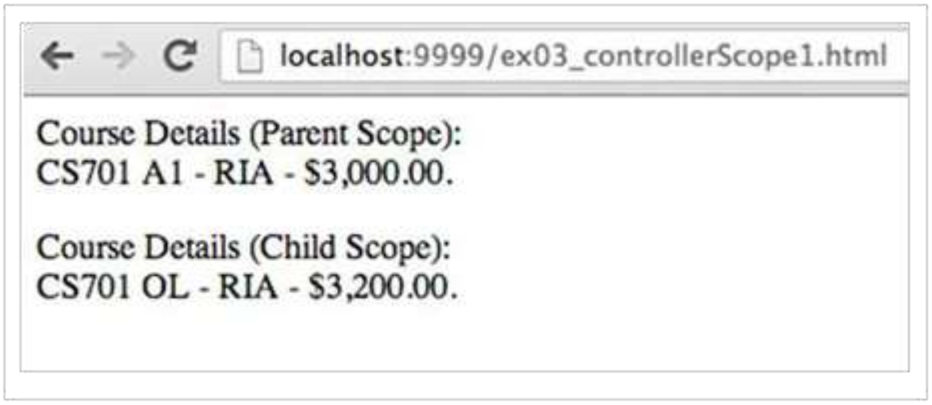
The following example defines two controllers, *ParentController*, and *ChildController*. The *ChildController*'s scope is nested within the scope of the *ParentController*. In the parent scope, the properties *courseName, courseId,* and *tuition* are defined. In the nested scope, the property *courseId* is overridden with a new value. The *tuition* property is overridden making use of the value defined in the parent scope:

File: `ex03_controllerScope1.html`

```
<html ng-app>
<head>
  <script src="js/angular10.js"></script>
 <script>
    function ParentController($scope) {
      $scope.courseName = 'RIA';
      $scope.courseId = 'CS701 A1';
      $scope.tuition = 3000;
    }
    function ChildController($scope) {
        console.log($scope);
      $scope.courseId = 'CS701 OL';
      $scope.tuition = $scope.tuition + 200;
    }
    </script>
</head>
<body>
  <div ng-controller="ParentController">
      <div>
        Course Details (Parent Scope): <br>
            {{courseId}}-{{courseName}}-{{tuition|currency}}.
      </div>
      <p></p>
      <div ng-controller="ChildController">
        Course Details (Child  Scope): <br>
            {{courseId}}-{{courseName}}-{{tuition|currency}}.
      </div>
    </div>
```

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

The above HTML shows the nested div elements controlled by the above two controllers. The `courseName` is not explicitly defined in the child controller and hence inherits the value from the parent controller. The sample output of the above application is shown below



```
localhost:9999/ex03_controllerScope1.html

Course Details (Parent Scope):
CS701 A1 - RIA - $3,000.00.

Course Details (Child Scope):
CS701 OL - RIA - $3,200.00.
```

Modifying Parent Scope

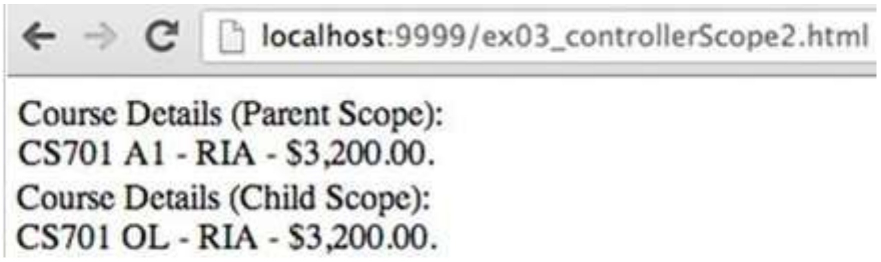
The values defined in the parent scope can be modified in the nested scope through the *$parent* property of the current scope. In the following example, the *tuition* property is defined in the parent scope. However, the child controller modifies the value in the parent scope by adding the value 200.

FILE: `ex03_controllerScope2.html`

```
<html ng-app>
<head>
  <script src="js/angular10.js"></script>
 <script>
    function ParentController($scope) {
      $scope.courseName = 'RIA';
      $scope.courseId = 'CS701 A1';
      $scope.tuition = 3000;
    }

    function ChildController($scope) {
      $scope.courseId = 'CS701 OL';
      $scope.$parent.tuition = $scope.tuition + 200;
      console.log($scope);
    }
    </script>
</head>
<body>
   <div ng-controller="ParentController">
      <div>
        Course Details (Parent Scope): <br>
            {{courseId}}-{{courseName}}-{{tuition|currency}}.
      </div>
      <p></p>
      <div ng-controller="ChildController">
        Course Details (Child  Scope): <br>
            {{courseId}}-{{courseName}}-{{tuition|currency}}.
      </div>
   </div>
</body>
</html>
```

The sample output of the above application is shown below. The *tuition* value is now $3200 in the parent scope as well as the child scope



Course Details (Parent Scope):
CS701 A1 - RIA - $3,200.00.
Course Details (Child Scope):
CS701 OL - RIA - $3,200.00.

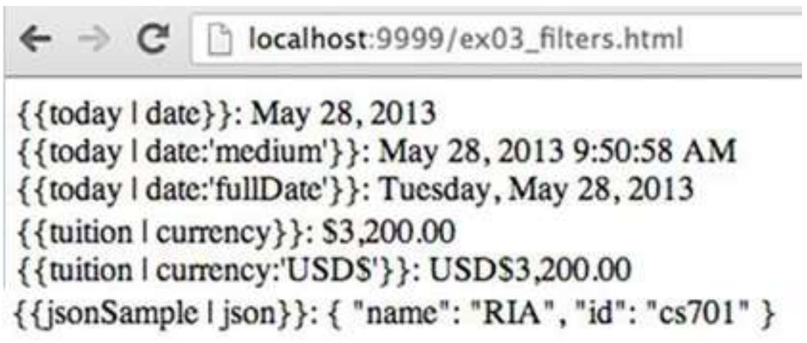
Filters

AngularJS filters allow data to be formatted based on the specified options. The following example shows some of the predefined filters

File: `ex03_filters.html`

```
<html ng-app>
<head>
  <script src="js/angular10.js"></script>
 <script>
    function TestController($scope) {
        var jsonData =
          {name : 'RIA', id : 'cs701'};
      $scope.jsonSample = jsonData;
      $scope.tuition = 3200;
      $scope.today = new Date();
    }
  </script>
</head>
<body>
  <div ng-controller="TestController">
    <span ng-non-bindable>{{today | date}}</span>:
        {{today | date}}<br>
    <span ng-non-bindable>{{today | date:'medium'}}</span>:
        {{today | date:'medium'}}<br>
    <span ng-non-bindable>{{today | date:'fullDate'}}</span>:
        {{today | date:'fullDate'}}<br>
    <p></p>
    <span ng-non-bindable>{{tuition | currency}}</span>:
        {{tuition | currency}}<br>
    <span ng-non-bindable>{{tuition | currency:'USD$'}}</span>:
        {{tuition | currency:'USD$'}}<br>
    <p></p>
    <span ng-non-bindable>{{jsonSample | json}}</span>:
        {{jsonSample | json}}<br>
    <p></p>
   </div>
</body>
</html>
```

The output is as below:



```
{{today | date}}: May 28, 2013
{{today | date:'medium'}}: May 28, 2013 9:50:58 AM
{{today | date:'fullDate'}}: Tuesday, May 28, 2013
{{tuition | currency}}: $3,200.00
{{tuition | currency:'USD$'}}: USD$3,200.00
{{jsonSample | json}}: { "name": "RIA", "id": "cs701" }
```

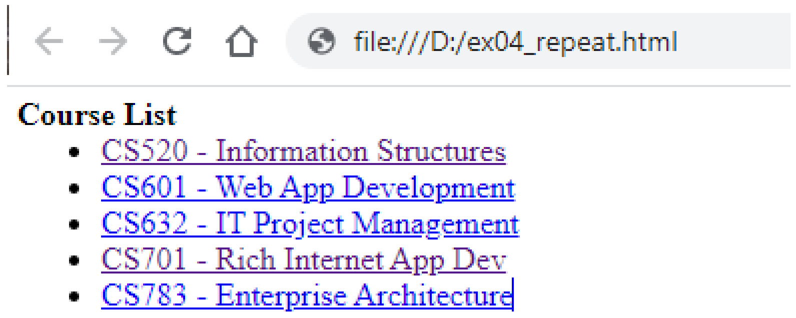
Repeat Directive

The *ngRepeat* directive is useful to iterate over a collection of items to generate the corresponding *HTML*. For each item that exists in the collection, the directive instantiates a template that has its own scope. The loop variable is set to the current item in the collection. The directive is illustrated with the following example (`ex04_repeate.html`) that displays the list of courses.

```
<!doctype html>
<html ng-app>
  <head>
    <script src="js/angular10.js"></script>
    <script>
      var courseList =
            [{name:'Information Structures',  id:'cs520'},
             {name:'Web App Development',     id:'cs601'},
             {name:'IT Project Management',   id:'cs632'},
             {name:'Rich Internet App Dev',   id:'cs701'},
             {name:'Enterprise Architecture', id:'cs783'}];
      function CourseListController($scope) {
        $scope.courses = courseList;
      }
    </script>
  </head>
  <body>
    <b>Course List</b>
    <ul ng-controller='CourseListController'>
      <li ng-repeat='course in courses'>
        <a href='http://www.bu.edu/csmet/{{course.id}}'
                    target=_blank>
            {{course.id | uppercase}} - {{course.name}}
        </a>
      </li>
    </ul>
  </body>
</html>
```

The data for the application is initialized as array `courseList` The controller for the application, *`CourseListController`*, initializes the *`courses`* variable in the scope. Each item in the list has the *name* and *id* for each course.
The ***ng-repeat*** directive is shown above. The controller is specified for the `<ul>` tag. Within the scope of the controller, a `<li>` item needs to be generated for each course. The repeat expression is thus specified for the `<li>` tag. The repeat expression *`course in courses`* controls the behavior of the loop.

The output of the sample application using the above properties is shown below.


file:///D:/ex04_repeat.html

Course List

- CS520 - Information Structures
- CS601 - Web App Development
- CS632 - IT Project Management
- CS701 - Rich Internet App Dev
- CS783 - Enterprise Architecture

The following properties are available for each instance in the loop:

- $index – zero-based index for the loop
- $first – true if the current item is the first item for the loop, false otherwise
- $middle – true if the current item is not the first item or the last item, false otherwise
  $last– true if the current item is the last item for the loop, false otherwise

We can modify the above HTML script `(now:ex_repeate2.html)` using those variables as follows

`ex_repeate2.html`

```
<!doctype html>
<html ng-app>
  <head>
    <script src="js/angular10.js"></script>
    <script>
      var courseList =
            [{name:'Information Structures',  id:'cs520'},
             {name:'Web App Development',     id:'cs601'},
             {name:'IT Project Management',   id:'cs632'},
             {name:'Rich Internet App Dev',   id:'cs701'},
             {name:'Enterprise Architecture', id:'cs783'}];

      function CourseListController($scope) {
        $scope.courses = courseList;
      }
    </script>
  </head>
  <body>
    <b>Course List</b>
    <ul ng-controller='CourseListController'>
      <li ng-repeat='course in courses'>
        <a href='http://www.bu.edu/csmet/{{course.id}}'
                    target=_blank>
            [{{$index + 1}}] -
            {{course.id | uppercase}} - {{course.name}}
            ({{$first}}, {{$middle}}, {{$last}})
        </a>
      </li>
    </ul>
  </body>
</html>
```

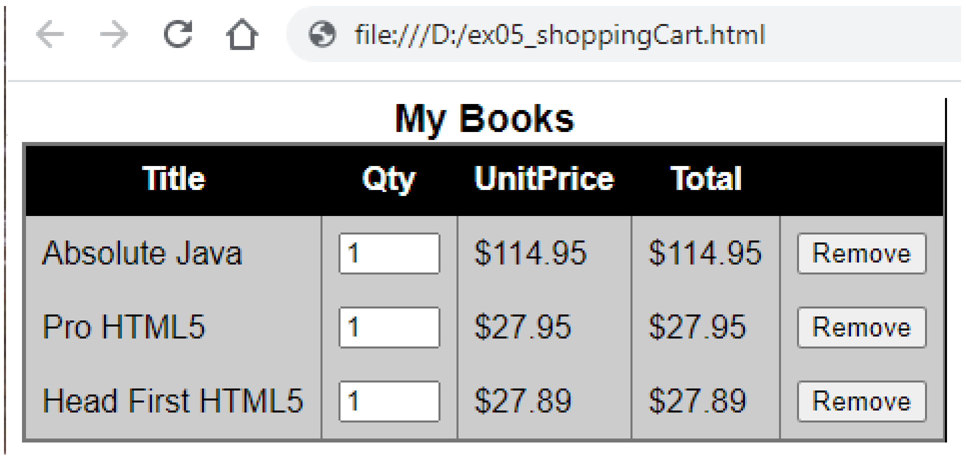
The output of modified HTML would now display:

Course List

- [1] - CS520 - Information Structures (true, false, false)
- [2] - CS601 - Web App Development (false, true, false)
- [3] - CS632 - IT Project Management (false, true, false)
- [4] - CS701 - Rich Internet App Dev (false, true, false)
- [5] - CS783 - Enterprise Architecture (false, false, true)

Case Study - Shopping Cart

A sample shopping cart application for ordering textbooks is shown below.



My Books

Title	Qty	UnitPrice	Total	
Absolute Java	1	$114.95	$114.95	Remove
Pro HTML5	1	$27.95	$27.95	Remove
Head First HTML5	1	$27.89	$27.89	Remove

The controller for the shopping cart application is shown below. The `books` variable in the controller's scope maintains the current selection of books along with the quantity for each book. When the book is removed from the shopping cart, the corresponding entry is removed from the `books` model element. Changes in the model are automatically reflected in the view.

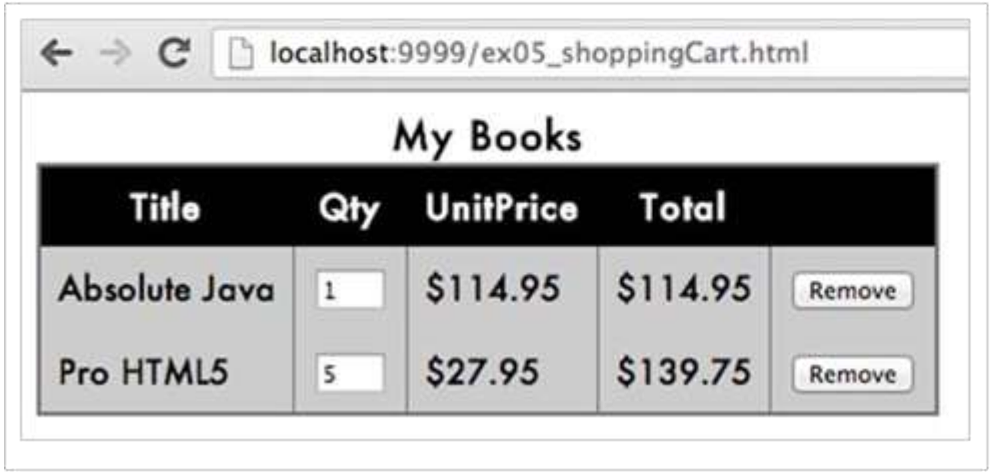
File: `ex05_shoppingCart.html`

```
<!doctype html>
<html lang='en' ng-app>
  <head>
   <title>Book Shopping Cart</title>
  <script src="js/angular10.js"></script>
  <script>
    function CartControler($scope) {
      $scope.books = [
        {title: 'Absolute Java',
            qty: 1, price: 114.95},
        {title: 'Pro HTML5',
            qty: 1, price: 27.95},
        {title: 'Head First HTML5',
            qty: 1, price: 27.89}
      ];
      $scope.removeBook = function(index) {
        $scope.books.splice(index, 1);
      }
    }
  </script>
  <link rel="stylesheet" href="css/ex05.css">
  </head>
```

The *body* of the web application controlled by the `CartController` is shown below. In the table, a table row element is created for each `book` in the `books` list. Note that the user has the option to specify the quantity for each book. Hence, the `input` element binds the model for the corresponding book's quantity. When the user types a new value, the `quantity` of the `book` is automatically updated and hence the view is updated to reflect the new total price for that `book`.

```
<body ng-controller="CartControler">
    <table>
      <caption><b>My Books</b></caption>
      <thead>
        <tr>
            <th>Title</th>
            <th>Qty</th>
            <th>UnitPrice</th>
            <th>Total</th>
            <th></th>
        </tr>
      </thead>
      <tbody >
        <tr ng-repeat="book in books">
            <td>{{book.title}}</td>
            <td>
                <input ng-model="book.qty" size="2">
            </td>
            <td>{{book.price | currency}}</td>
            <td>{{book.price * book.qty | currency}}</td>
            <td>
                <button ng-click="removeBook($index)">
                    Remove
                </button>
            </td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

After updating the quantity of the second book and removing the third book, the shopping cart appears as shown below.



My Books

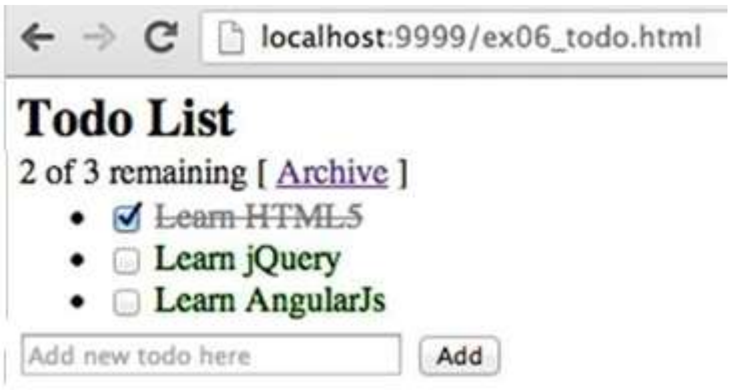
Title	Qty	UnitPrice	Total	
Absolute Java	1	$114.95	$114.95	Remove
Pro HTML5	5	$27.95	$139.75	Remove

Case Study – Todo List (`ex06_todo.html`)

A sample application for keeping track of *todo* tasks is shown below. The *todo* line items are displayed with a checkbox indicating whether the task is complete or not. The number of remaining tasks is also shown. The completed tasks can also be archived and removed from the list. New tasks are added at the end of the existing list.The HTML application is shown bellow. The *`TodoController`* is the controller of the application. The controller is defined in a separate JavaScript file, `ex06_todo.js` in directory *`js`*

```
<!doctype html>
<html ng-app>
  <head>
    <script src="js/angular10.js"></script>
    <script src="js/ex06_todo.js"></script>
    <link rel="stylesheet" href="css/ex06_todo.css">
  </head>
  <body>
    <h2>Todo List</h2>
    <div ng-controller="TodoController">
      <span>{{remaining()}} of {{todos.length}} remaining</span>
      [ <a href="" ng-click="archive()">Archive</a> ]
      <ul>
        <li ng-repeat="todo in todos">
          <input type="checkbox" ng-model="todo.done">
          <span class="done-{{todo.done}}">{{todo.text}}</span>
        </li>
      </ul>
      <form ng-submit="addTodo()">
        <input type="text" ng-model="todoText"  size="30"
               placeholder="Add new todo here">
        <input type="submit" value="Add">
      </form>
    </div>
  </body>
</html>
```

The initial display of the application is shown below. The first task is shown checked as the *done* property is true in the initial data. Two out of the three tasks are still remaining.


localhost:9999/ex06_todo.html

Todo List

2 of 3 remaining [Archive]

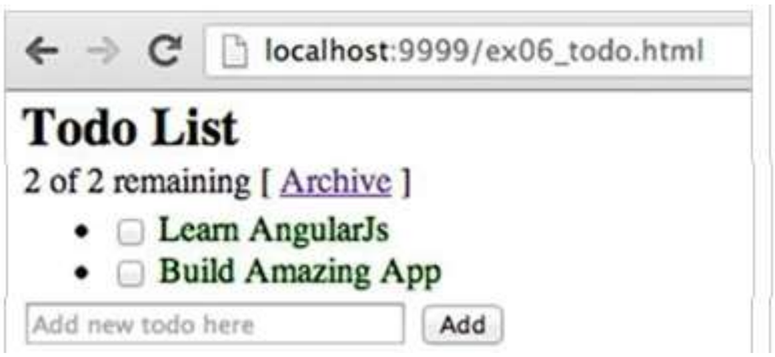
- ✓ ~~Learn HTML5~~
- ☐ Learn jQuery
- ☐ Learn AngularJs

The controller is defined in a separate JavaScript file, `ex06_todo.js` in directory `js`. The code defining the controller is shown below. The *`todos`* variable in the controller's scope keeps track of the current tasks and their status. The *`addTodo`* function adds the non-empty *`todoText`* to the *`todos`* list. The *`todoText`* is bound to the *input* element in the HTML page. The *remaining* function counts and returns the number of tasks that are yet to be done. The *`archive`* function clears the *`todos`* list and adds back the tasks that are not yet done.

File: `ex06_todo.js`

```
function TodoController($scope) {
  $scope.todos = [
    {text:'Learn HTML5', done:true},
    {text:'Learn jQuery', done:false},
    {text:'Learn AngularJs', done:false}];
  $scope.todoText = '';
  $scope.addTodo = function() {
    if ($scope.todoText.trim() != '') {
        $scope.todos.push(
            {text:$scope.todoText, done:false});
        $scope.todoText = '';
     }
  };
  $scope.remaining = function() {
    var count = 0;
    angular.forEach($scope.todos, function(todo) {
      count += todo.done ? 0 : 1;
    });
    return count;
  };
  $scope.archive = function() {
    var oldTodos = $scope.todos;
    $scope.todos = [];
    angular.forEach(oldTodos, function(todo) {
      if (!todo.done) $scope.todos.push(todo);
    });
  };
}
```

Clicking on the *Archive* link deletes the *done* tasks as shown below:



localhost:9999/ex06_todo.html

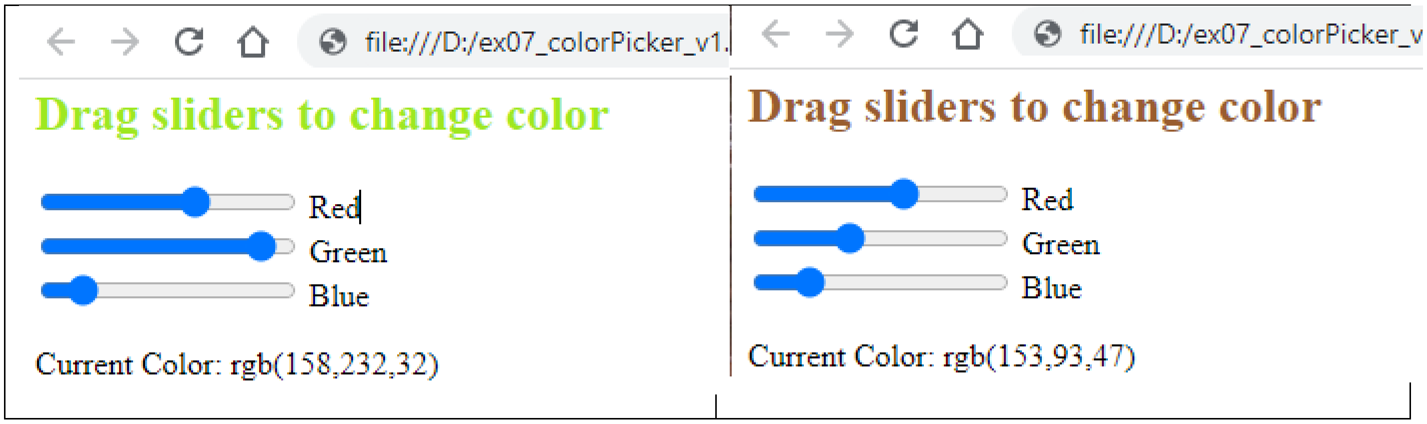
Todo List

2 of 2 remaining [Archive]

- ☐ Learn AngularJs
- ☐ Build Amazing App

Comparing Coding Styles

The following example shows different styles for solving the same problem. The application illustrated here provides three sliders for selecting the *red*, *green*, and *blue* color values in the range 0–255.


file:///D:/ex07_colorPicker_v1.
file:///D:/ex07_colorPicker_v

Drag sliders to change color

Red
Green
Blue

Current Color: rgb(158,232,32)

Drag sliders to change color

Red
Green
Blue

Current Color: rgb(153,93,47)

The associated *html* is shown below. The model variables `r, g`, and `b` capture the user selections and change the color of the *target*. The `setColor` function is called when any of these values change

File: `ex07_colorPicker_v1.html`

```
<!DOCTYPE html>
<html ng-app>
  <head>
     <title>Color Sample Version1</title>
    <script src='js/angular10.js'></script>
    <script>
            function ColorController($scope) {
                $scope.setColor = function() {
                    var colors = document.getElementsByTagName('input');
                    document.getElementById('target').style.color =
                       'rgb(' + colors[0].value + ',' +
                         colors[1].value + ',' + colors[2].value + ')';
                }
            }
        </script>
  </head>
  <body ng-controller='ColorController'>
    <h1 id="target">
      Drag sliders to change color
    </h1>
    <input type='range' min='0' max='255'
          ng-model='r' ng-change='setColor();'> Red<br/>
    <input type='range' min='0' max='255'
          ng-model='g' ng-change='setColor();'> Green<br/>
    <input type='range' min='0' max='255'
          ng-model='b' ng-change='setColor();'> Blue<br/>
    <p></p>
    Current Color: rgb({{r}},{{g}},{{b}})
  </body>
</html>
```

The current values of the *input* elements are used to set the color of the *target* element.

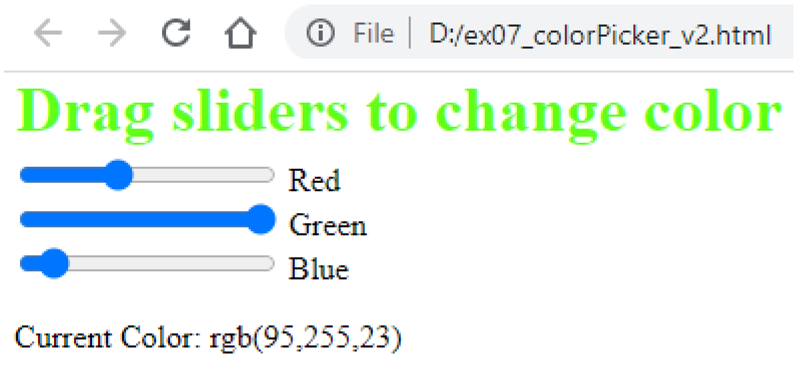
The above example could be modified without using any controller *JavaScript* code as shown below. The `ng-init` directive initializes the values for the model variables `r, g,` and `b`. The *h1* element's *color* style picks the current values from the `scope`. Whenever the slider values change, the color is automatically updated.

File: `ex07_colorPicker_v2.html`

```
<!DOCTYPE html>
<html ng-app>
  <head>
    <title>Color Picker Version2</title>
    <script src='js/angular10.js'></script>
  </head>
  <body ng-init='r=128; b=128; g=128;'>

    <h1 style='color: rgb({{r}},{{g}},{{b}});'>
      Drag sliders to change color
    </h1>
    <input type='range' min='0' max='255'
          ng-model='r'>  Red   <br/>
    <input type='range' min='0' max='255'
          ng-model='g'>  Green <br/>
    <input type='range' min='0' max='255'
          ng-model='b'>  Blue  <br/>
    <p></p>
    Current Color: rgb({{r}},{{g}},{{b}})
  </body>
</html>
```

HTML file when open gives the same display as before. The initial rendering starts with initial values (128,128,128). As we move the sliders, color of the text changes



Drag sliders to change color

Red

Green

Blue

Current Color: rgb(95,255,23)

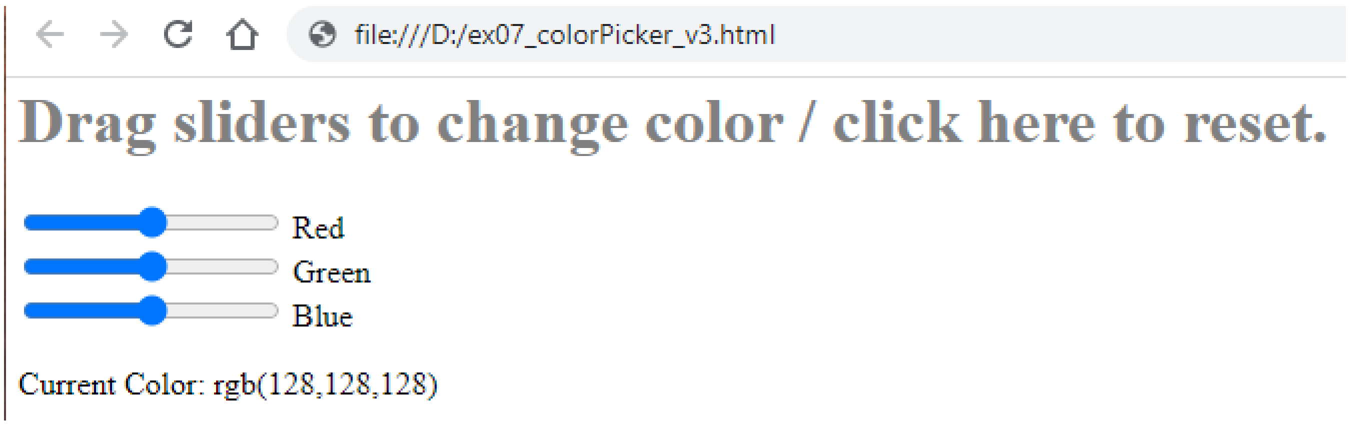
The above example can be enhanced further using the *ng-click* directive. When the user clicks on the *h1* element, the *r*, *g*, and *b* values are reset to (128, 128, 128) as shown below.

File: `ex07_colorPicker_v3.html`

```
<!DOCTYPE html>
<html ng-app>
  <head>
    <title>Color Picker Version3</title>
    <script src='js/angular10.js'></script>
  </head>
  <body ng-init='r=128; b=128; g=128;'>

    <h1 ng-click='r=128; b=128; g=128;'
        style='color: rgb({{r}},{{g}},{{b}});'>
      Drag sliders to change color / click here to reset.
    </h1>
    <input type='range' min='0' max='255'
          ng-model='r'> Red <br/>
    <input type='range' min='0' max='255'
          ng-model='g'> Green <br/>
    <input type='range' min='0' max='255'
          ng-model='b'> Blue <br/>
    <p></p>
    Current Color: rgb({{r}},{{g}},{{b}})
  </body>
</html>
```

The page renders as below:


file:///D:/ex07_colorPicker_v3.html

Drag sliders to change color / click here to reset.

Red
Green
Blue

Current Color: rgb(128,128,128)

Mapping Input Selections to JSON

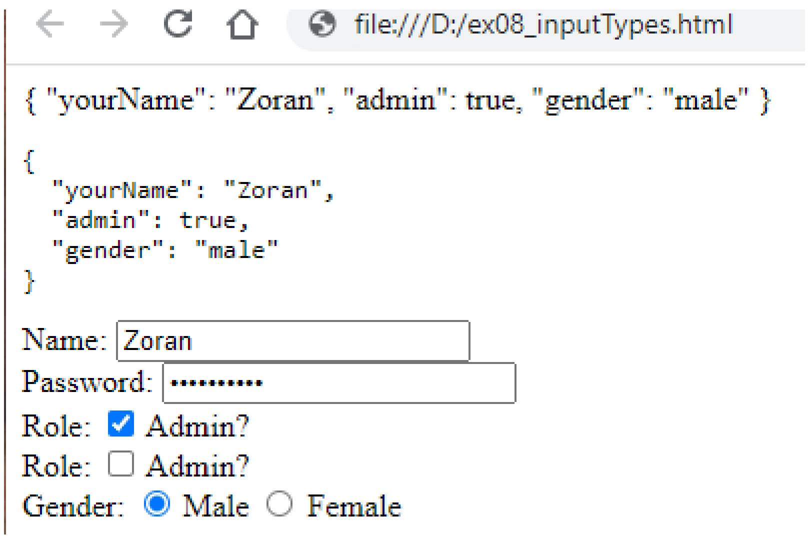
The following example maps the model properties to JSON. The *user* object is output using the *json* filter. Various *input* types are bound to the corresponding *user* model properties in the following example.

File: `ex08_inputType.html`

```
<!doctype html>
<html ng-app>
  <head>
    <script src="js/angular10.js"></script>
  </head>
  <body>
    <div>
    {{user | json}}
     <pre ng-bind="user | json"></pre>
      <label>Name:</label>
      <input type="text"
          ng-model="user.yourName"
          placeholder="Type your name"> <br/>
      <label>Password:</label>
      <input type="password"
          ng-model="user.password"
          ng-minlength="6" ng-maxlength="10"
          ng-pattern="/^.*(?=.*\d)(?=.*[a-zA-Z]).*$/"> <br/>
      <label>Role:</label>
      <input type="checkbox"
          ng-model="user.admin"> Admin? <br/>
      <label>Role:</label>
      <input type="checkbox"
          ng-model="user.role"  ng-true-value="admin"
            ng-false-value="basic"> Admin? <br/>
      <label>Gender:</label>
      <input type="radio"
          ng-model="user.gender" value="male"> Male
      <input type="radio"
          ng-model="user.gender" value="female"> Female <br/>
    </div>
  </body>
</html>
```

The *user* object is rendered as shown below. The password pattern makes sure that there is at least one digit and has a minimum length of 6 and a maximum length of 10.

The *checkbox* input maps, by default, the model value to either *true* or *false*. The *true* value and the *false* value may also be explicitly specified using the *ng-true-value* and *ng-false-value* attributes


file:///D:/ex08_inputTypes.html

{ "yourName": "Zoran", "admin": true, "gender": "male" }

```
{
  "yourName": "Zoran",
  "admin": true,
  "gender": "male"
}
```

Name: Zoran
Password: ••••••••••
Role: [✓] Admin?
Role: [ ] Admin?
Gender: (●) Male ( ) Female

Options for *Select* Element

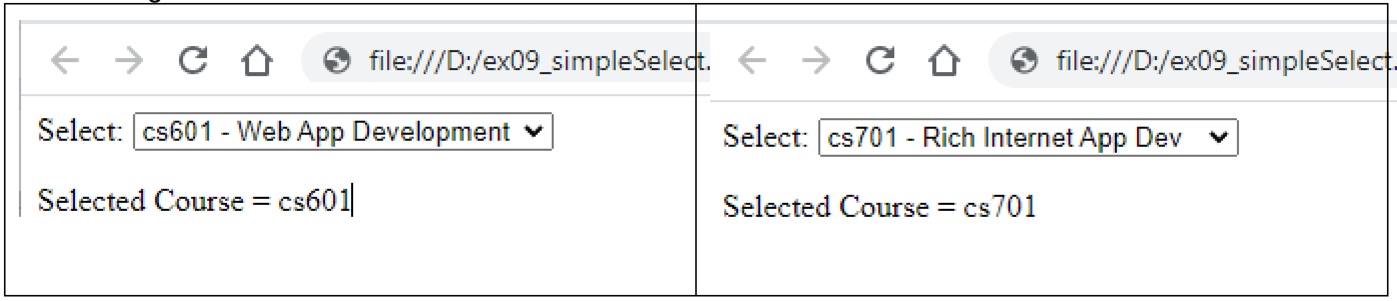
The following example shows different approaches for building the options for the *select* element. The controller initializes the model values `courses` and the current selection as `myChoice`. The `getDisplayName` function is used for formatting the display values :

File: `ex09_simpleSelect.html`

```
<!doctype html>
<html ng-app>
  <head>
    <script src="js/angular10.js"></script>
    <script>
      var courseList =
            [{name:'Information Structures',  id:'cs520'},
             {name:'Web App Development',     id:'cs601'},
             {name:'IT Project Management',   id:'cs632'},
             {name:'Rich Internet App Dev',   id:'cs701'},
             {name:'Enterprise Architecture', id:'cs783'}];

      function CourseListController($scope) {
        $scope.courses = courseList;
        $scope.getDisplayName = function(course) {
            return course.id + ' - ' + course.name;
        };
        $scope.myChoice = 'cs701';
      }
    </script>
  </head>
  <body ng-controller='CourseListController'>
    <div>
      <label>Select:</label>
       <select ng-model="myChoice">
          <option ng-repeat="course in courses"
              value="{{course.id}}"
              ng-selected="course.id == myChoice">
            {{getDisplayName(course)}}
          </option>
       </select>
       <p></p>
      <div>Selected Course = {{myChoice}}</div>
    </div>
  </body>
</html>
```

The page renders as bellow. When you select a course from the drop-down list, Selected Course display value changes:


file:///D:/ex09_simpleSelect.
file:///D:/ex09_simpleSelect.html

Select: cs601 - Web App Development

Selected Course = cs601

Select: cs701 - Rich Internet App Dev

Selected Course = cs701

The above example could be enhanced by including the `type` property for each course. Also, the selected choice is manipulated as the course object itself rather than as a string property.

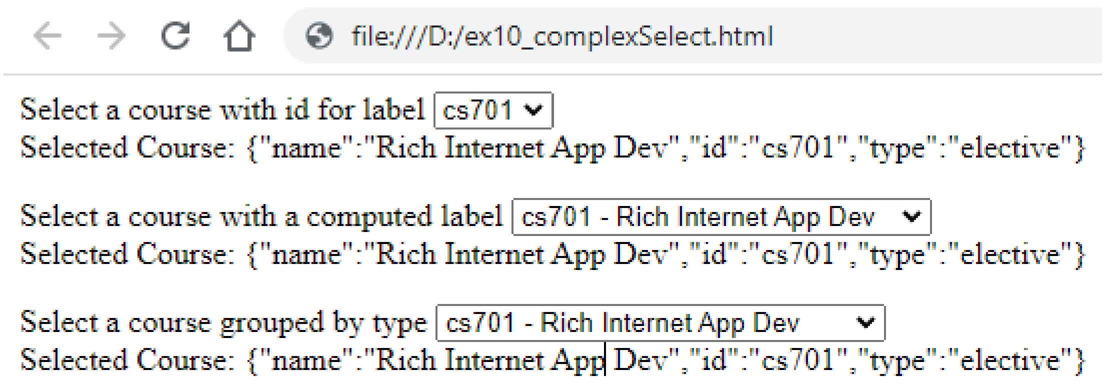
File: `ex10_complexSelect.html`

```
<!doctype html>
<html ng-app>
  <head>
    <script src="js/angular10.js"></script>
    <script>
      var courseList =
        [{name:'Information Structures',  id:'cs520', type:'core'},
         {name:'Web App Development',     id:'cs601', type:'core'},
         {name:'IT Project Management',   id:'cs632', type:'elective'},
         {name:'Rich Internet App Dev',   id:'cs701', type:'elective'},
         {name:'Enterprise Architecture', id:'cs783', type:'elective'}];
      function CourseListController($scope) {
        $scope.courses = courseList;
        $scope.getDisplayName = function(course) {
            return course.id + ' - ' + course.name;
        };
        $scope.myChoice = $scope.courses[3];
      }
    </script>
  </head>
  <body ng-controller='CourseListController'>
        <div>
            <label>Select a course with id for label</label>
            <select ng-model="myChoice"
               ng-options="course.id for course in courses"></select>
            <br/>
            Selected Course: {{ myChoice }}
            <p></p>
            <label>Select a course with a computed label</label>
            <select ng-model="myChoice"
               ng-options="getDisplayName(course)
                   for course in courses"></select>
            <br/>
            Selected Course: {{ myChoice }}
            <p></p>
            <label>Select a course grouped by type</label>
            <select ng-model="myChoice"
               ng-options="getDisplayName(course)
                   group by course.type for course in courses"></select>
            <br/>
            Selected Course: {{ myChoice }}
        </div>
    </body>
</html>
```

The display for the `select` element is built using the `ng-options` rather than an explicit loop. The option value will always be the index of the object in the collection. In the following snippet, the `course id` is used for the display values

```
<div>
            <label>Select a course with id for label</label>
            <select ng-model="myChoice"
               ng-options="course.id for course in courses"></select>
            <br/>
            Selected Course: {{ myChoice }}
```

The generated *html* for the above is shown below.



file:///D:/ex10_complexSelect.html

Select a course with id for label cs701

Selected Course: {"name":"Rich Internet App Dev","id":"cs701","type":"elective"}

Select a course with a computed label cs701 - Rich Internet App Dev

Selected Course: {"name":"Rich Internet App Dev","id":"cs701","type":"elective"}

Select a course grouped by type cs701 - Rich Internet App Dev

Selected Course: {"name":"Rich Internet App Dev","id":"cs701","type":"elective"}

The selected `choice` is mapped to the model *`myChoice`*

Introduction to AngularJS Modules

Examples for this section are in the directory `Module3_Samples/angular1_modules`

The AngularJS code consists of controllers, filters, directives, services, etc. Modules allow the AngularJS code to be packaged under a single entity. The components defined within a module are accessible throughout the module. A module may also depend on the code written in other modules. These other modules are known as dependencies for the given module. The `ng-app` attribute in the web page specifies the AngularJS module that will be bootstrapped when the web application is rendered.

The following example refers to the module *`myApp`* defined in the associated JavaScript file. The code associated with the specified module is bootstrapped by AngularJS.

File: `ex_basic.html`

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <script src="js/angular157.js"></script>
    <script src="js/ex01_basic.js"></script>
  </head>
  <body>
   Angular Application
  </body>
</html>
```

The associated JavaScript file shows the definition of the module, named *myApp*.

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

The first argument for the Angular *`module`* method is the name of the module. The second argument is an array of module names that this module depends on. If there are no dependencies, the second argument is simply the empty array. Omitting the second argument results in Angular looking for a reference to an already defined module with the specified first argument, rather than creating the module.

Defining Controllers within a Module

The Module's `controller` method is used for defining the controllers within the specified module. The first argument is the name of the controller. The second argument is the constructor function for creating the controller. The dependencies for the controller are specified as the arguments for the function. The `$scope` service is almost used by every controller that provides access to the data and logic for the view.

The model variable `greeting` initialized in the controller's scope is used in the view (HTML page) as shown below.

File: `ex02_controller.html`

```
<html ng-app="myApp">
<head>
  <script src="js/angular157.js"></script>
  <script src="js/ex02_controller.js"></script>
</head>
<body>
  <div ng-controller='HelloController'>
    <input ng-model='greeting'>
    <p>{{greeting}}, Welcome to CS701!</p>
  </div>
</body>
</html>
```

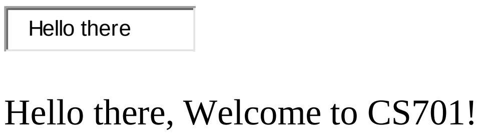
The controller is specified as:

File: `js/ex02_controller.js`

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

myModule.controller('HelloController', function ($scope) {
        $scope.greeting = "Hello";
    }
);
```

The page renders as



```
Hello there

Hello there, Welcome to CS701!
```

A common approach is to not use a separate variable to initialize the AngularJS module and instead define the controller along with the module definition as shown below.

File: `js/ex02_controller2.js`

```
angular.module('myApp', [])
  .controller('HelloController', function ($scope) {
        $scope.greeting = "Hello";
    }
   );
```

Controller `HelloController` from this script could be invoked from a practically identical HTML page

File: `ex02_controller2.html`

```
<html ng-app="myApp">
<head>
  <script src="js/angular157.js"></script>
  <script src="js/ex02_controller2.js"></script>
</head>
<body>
  <div ng-controller='HelloController'>
    <input ng-model='greeting'>
    <p>{{greeting}}, Welcome to CS701!</p>
  </div>
</body>
</html>
```

Defining Multiple Controllers within a Module

Multiple controllers can be defined within the same module.

You can have multiple controllers, but you cannot have multiple ng-app directives on the same page. This means you should only have a single ng-app directive in your html that points to a single module that will be used in your application.

You then define this module and define all your controllers in this module:

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

app.controller('TextController', function ($scope) {
    //Controller Code Here
});
```

```
app.controller('ItemController', function ($scope) {
    //Controller Code Here
});
```
If for some reason you want to have controllers in separate modules, you can still do that, and include those modules as dependencies of your main module:

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

text.controller('TextController', function ($scope) {
    //Controller Code Here
});

items.controller('ItemController', function ($scope) {
    //Controller Code Here
});
```

Generally you don't need to have a module for each controller. Modules are used to group related pieces of functionality together to make it easy to take that and re-use it in another application.

Example with Single Controller

File: `ex00_singleController.html`

```
<div ng-app='app'>
    <div ng-controller='ItemController'>
            <h1>Your Orders</h1>

        <div ng-repeat='item in items'> <span>{{item.title}}</span>
            <input type="text" ng-model="item.quantity"><span>{{item.price |
                currency}}</span>&nbsp;|&nbsp;<span>{{item.price *
                item.quantity | currency}}</span>
        </div>
    </div>
    <hr/>
    <div ng-controller='TextController'>Enter a text::
        <input ng-model='text.message'>
         <h2>{{text.message}}, Journey just started!</h2>
    </div>
</div>
```

File: `ex00_singleController.js`

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

app.controller('TextController', function ($scope) {
    $scope.text = {
        message: 'Welcome!!'
    };
});
app.controller('ItemController', function ($scope) {
    $scope.items = [{
        title: 'Pencil',
        quantity: 8,
        price: 4.2
    }, {
        title: 'Pen',
        quantity: 2,
        price: 5.2
    }, {
        title: 'Watch',
        quantity: 3,
        price: 10.2
    }];
});
```

Example with Multiple Controllers

File: `ex00_multipleControllers.html`

```
<div ng-app='app'>
    <div ng-controller='ItemController'>
            <h1>Your Orders</h1>
        <div ng-repeat='item in items'> <span>{{item.title}}</span>
            <input type="text" ng-model="item.quantity"><span>{{item.price |
                currency}}</span>&nbsp;|&nbsp;<span>{{item.price *
                item.quantity | currency}}</span>
        </div>
    </div>
    <hr/>
    <div ng-controller='TextController'>Enter a text::
        <input ng-model='text.message'>
         <h2>{{text.message}}, Journey just started!</h2>
    </div>
</div>
```

File: `ex00_multipleControllers.js`

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

text.controller('TextController', function ($scope) {
    $scope.text = {
        message: 'Welcome!!'
    };
});

items.controller('ItemController', function ($scope) {
    $scope.items = [{
        title: 'Pencil',
        quantity: 8,
        price: 4.2
    }, {
        title: 'Pen',
        quantity: 2,
        price: 5.2
    }, {
        title: 'Watch',
        quantity: 3,
        price: 10.2
    }];
});
```