**Exercise04\_02\_01 – Step 1**



1. Create a new folder for the exercise called ***Exercise04\_02\_01***. Copy all of the files into it from the previous ***Exercise04\_01\_01***.
2. Go to the ***builds/angular*** folder and open up the ***index.html*** file. First we can use an AngularJS ***conditional*** directive to show the expression after ***search*** only if we have data in the model for it. We must first enclose the expression in an HTML tag to use the ***ng-show*** directive, which is an attribute. So we can enclose it in a ***<span>*** tag and use the directive as follows:  
    ***search <span ng-show="query">{{ 'for: ' + query}}</span>***Let’s give this a test in the browser, and our expression should only appear when data is typed in the ***<input>*** box. Open up the ***Developer Tools*** in the browser and go to the ***Elements*** tab to examine the DOM. Expand the various elements until the new ***<span>*** element is visible. Notice that an AngularJS ***class*** attribute has been added. If there is data in the ***<input>*** field, the class is just ***ng-binding***. If there is no data, the class ***ng-hide*** is added. This shows the ***injection*** process.
3. Now let’s try to accomplish the same objective with an ***ng-if*** directive, and see in what different ways this affects the DOM. Let’s replace the ***ng-show*** with an ***ng-if***:  
    ***search <span ng-if="query">{{ 'for: ' + query}}</span>***In the ***Developer Tools***, the ***Elements*** tab does not even show the ***<span>*** element if there is no data. If there is data in the ***<input>*** the field, an AngularJS ***class*** attribute has been injected. The class is both ***ng-binding*** and ***ng-scope***.
4. We can now try a loop which will be designed to display all of the artists except just one. Go back to the **data.json** file and copy the entire JSON array with all of the artist objects in it. Go back into ***app.js*** and replace the ***$scope.artist*** assignment with ***$scope.artists***, and assign the entire new array:  
    ***$scope.artist = [  
    {  
    "name":"Barot Bellingham",  
    "shortname":"Barot\_Bellingham",  
    "reknown":"Royal Academy of Painting and Sculpture",  
    "bio":"Barot has just finished his final year at The Royal   
    Academy of Painting and Sculpture, where he excelled in glass   
    etching paintings and portraiture. Hailed as one of the most   
    diverse artists of his generation, Barot is equally as skilled with   
    watercolors as he is with oils, and is just as well-balanced in   
    different subject areas. Barot's collection entitled \"The Un-  
    Collection\" will adorn the walls of Gilbert Hall, depicting his   
    range of skills and sensibilities - all of them, uniquely Barot, yet   
    undeniably different"  
    },  
    …  
    ]***  
   When we look at this in the browser, we can see that the artists’ display is no longer working. Instead of one artist, we have no given the model an array of artists, so we have some more modifications to make.
5. Open up the ***index.html*** file and, in the last major ***<div>***, find the ***<li>*** element with class: **list-group-item**. Enter the ***ng-repeat*** directive as an attribute. Set it equal to a syntax that is similar to a JavaScript ***for..in*** construct as follows:  
    ***<li class="list-group-item" ng-repeat="artist in artists">***  
   Let’s give this a test in the browser. We can see that we are now looping through all of the artists in the array with an absolute minimum of coding.
6. Now let’s set the stage for the ability to ***search*** artists. We do not want the artists to show unless there is something typed in our search ***<input>*** field. So let’s go to the line above, which is the ***<ul>*** element with class: ***list-group d-flex***. We can again use an ***ng-if*** directive depending on the ***query*** model to display the list or not:  
   ***<ul class=" list-group d-flex" ng-if="query">***  
   It is not yet actually doing a search, but artists will not show now without data in the ***<input>*** field.

**Exercise04\_02\_01 – Step 2**



1. Go to the ***builds/angular/js*** folder and open up the ***app.js*** file. First we can make it a dependency of the Controller. To do this, we add it as a parameter of the function which is passed to the controller constructor as follows:  
   ***myApp.controller('MyController', function MyController($scope,   
    $http) {***
2. Next, as the first command in the function, let’s call an AngularJS style ***get()*** method from the ***$http*** service and target the ***data.json*** file with our artist data:  
   myApp.controller('MyController', function MyController($scope, $http) { ***$http.get('js/data.json')***
3. The ***$http.get()*** call returns a ***Promise***, so let’s build out our Promise code as follows:  
    $http.get('js/data.json')  
    ***.then(function(response) {  
      
    });***
4. Now we can really clean up our ***$scope.artists*** assignment by doing a one-liner inside the ***.then()*** method, then deleting the entire original assignment:  
    .then(function(response) {  
    ***$scope.artists = response.data;*** });  
   Let’s give this a test in the browser. Everything should be working with some really clean code.

**Exercise04\_02\_01 – Step 3**



1. Go to the ***builds/angular*** folder and open up the ***index.html*** file. First we can use a basic AngularJS ***filter*** to convert the artist’s names to ***uppercase***. To do this, find the line with the ***<h5>*** element that uses the expression for ***artist.name***. Let’s apply the filter as follows:  
    ***<h5 class="my-0 text-dark">{{artist.name | uppercase }}</h5>***Let’s give this a test in the browser.
2. Next let’s use a basic AngularJS ***filter*** to convert some data to ***lowercase***. To do this, find the line with the ***<div>*** element that uses the expression for ***artist.reknown***. Let’s apply the filter as follows:  
   ***<div class="text-secondary font-italic">  
    {{artist.reknown | lowercase}}</div>***Let’s give this a test in the browser.

**Exercise04\_02\_01 – Step 4**



1. Go to the ***builds/angular*** folder and open up the ***index.html*** file. Right now we have to type something in the ***search*** box to show anything from the list. Let’s get rid of that by removing the ***ng-if*** directive from the ***<ul>*** element:  
    ***<ul class=" list-group d-flex">***Let’s give this a test in the browser and we should have the whole list.
2. Let’s also get rid of the ***uppercase*** and ***lowercase*** transformations as follows:  
    ***{{artist.name | uppercase }}  
    {{artist.reknown}***Let’s give this a test in the browser.
3. Now let’s just give a test to our ability to limit the number of records. Let’s place a filter into the ng-repeat directive’s value as follows:  
    ***ng-repeat="artist in artists | limitTo: 4"***Let’s give this a test in the browser.
4. Now let’s do our real objective which is to implement a search for artists. Let’s remove the previous ***limitTo*** filter. We will use the actual ***filter*** keyword. We are already getting data from our ***ng-model*** named ***query***, so let’s filter by that as follows:  
    ***ng-repeat="artist in artists | filter: query"***  
   Let’s give this a test in the browser. Fool around with a number of different character combinations and see how it works on a combination of the data fields, looking for single letter matches.
5. Now let’s add a sort to this. We can do this by using more than one filter as follows:  
    ***ng-repeat="artist in artists | filter: query | orderBy: ’name’"***  
   Let’s give this a test in the browser. Fool around with a number of different character combinations and see how it works on a combination of the data fields, looking for single letter matches that are now sorted.

**Exercise04\_02\_01 – Step 5**



1. Let’s extend the user interface of the app to enable sorting choices. Open up the ***sort.html*** file and copy its entire contents. Go to the ***index.html*** file and copy the contents in, below the comment ***<!--form-row -->***:  
   ***<div class="form-row">  
    . . .   
   </div>  
   <!-- form-row -->***Let’s give this a test in the browser. We can see the new parts of the user interface, but we have not yet wired them to be operational.
2. Go down to the first ***<select>*** element in the new ***<div>*** with class: ***form-row***. Let’s add an attribute to the tag using an ***ng-model*** directive, which we will set to ***artistOrder*** as follows:  
    <select class="form-control form-control-inline mr-3 w-100"   
    ***ng-model="artistOrder"***>
3. We now will modify our ***ng-repeat*** directive to have its ***orderBy*** filter use the new model as follows:  
    <li class="list-group-item"   
    ng-repeat="artist in artists | filter: query |   
    ***orderBy:*** ***artistOrder***">  
   Let’s give this a test in the browser, using the ***<select>*** element to pick our sort field.
4. We can now implement the sort direction to be ascending or descending. Find the first radio button ***<input>*** field with name: ***direction***. Let’s add an attribute to the tag using an ***ng-model*** directive, which we will set to ***direction*** as follows:  
   <input class="form-check-input" type="radio"   
    name="direction" checked   
    ***ng-model="direction"***>
5. Find the next radio button ***<input>*** field below it with name: ***direction***. Let’s do the same thing and add an attribute to the tag using an ***ng-model*** directive, which we will set to ***direction*** as follows:  
   <input class="form-check-input" type="radio"   
    name="direction"   
    ***ng-model="direction"***>
6. We now will modify our ***ng-repeat*** directive to have its ***orderBy*** filter add the new model as follows:  
    <li class="list-group-item"   
    ng-repeat="artist in artists | filter: query |   
    ***orderBy:*** ***artistOrder:direction***">  
   Let’s give this a test in the browser, using the ***<select>*** element and the ***<input>*** radio buttons to pick check our sorts.
7. We have one slight problem with our sorts. When the form first comes up there is no value in the ***<select>*** element. Because of this, the array is not initially sorted. We can fix this in our ***controller***, so bring up the ***app.js*** file. We can add another property to the ***$scope*** global to set the default value for our ***artistOrder*** model:  
    $scope.artists = response.data; ***$scope.artistOrder = 'name';***Let’s give this a test in the browser, and our sort order should come up correctly from the start. This is an excellent example of ***two way data binding***.