**AngularJS: -** AngularJS is a **JavaScript framework**. It can be added to an HTML page with a <script> tag. AngularJS extends HTML attributes with **Directives** and binds data to HTML with **Expressions**.

**AngularJS is a JavaScript Framework: -** AngularJS is a JavaScript framework written in JavaScript. AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag: <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

**AngularJS Extends HTML: -** AngularJS extends HTML with **ng-directives**.

* The **ng-app** directive defines an AngularJS application.
* The **ng-model** directive binds the value of HTML controls (input, select, text area) to application data.
* The **ng-bind** directive binds application data to the HTML view.

<div ng-app="">  <p>Name: <input type="text" ng-model="name"></p>  <p ng-bind="name"></p></div>  
**AngularJS Directives:-** As you have already seen, AngularJS directives are HTML attributes with an **ng** prefix.The **ng-init** directive initializes AngularJS application variables.

<div ng-app="" ng-init="firstName='John'"><p>The name is <span ng-bind="firstName"></span></p></div>

**Alternatively, with valid HTML:**

<div data-ng-app="" data-ng-init="firstName='John'">

<p>The name is <span data-ng-bind="firstName"></span></p></div>

You can use **data-ng-**, instead of **ng-**, if you want to make your page HTML valid.

**AngularJS Expressions: -** AngularJS expressions are written inside double braces: **{{expression}}. AngularJS** will "output" data exactly where the expression is written:

<div ng-app=""> <p>My first expression: {{5 + 5}} </p></div>

AngularJS expressions bind AngularJS data to HTML the same way as the **ng-bind** directive.

<div ng-app=""> <p>Name: <input type="text" ng-model="name"></p> <p>{{name}} </p></div>  
**AngularJS Applications:-**AngularJS **modules** define AngularJS applications. AngularJS **controllers** control AngularJS applications. The **ng-app** directive defines the application, the **ng-controller** directive defines the controller.

<div ng-app="**myApp**" ng-controller="**myCtrl**">  
 First Name: <input type="text" ng-model="firstName"><br>  
 Last Name: <input type="text" ng-model="lastName"><br>  
 Full Name: {{firstName + " " + lastName}}  
</div>  
<script>  
 var app = angular.module('**myApp**', []);  
 app.controller('**myCtrl**', function($scope) {  
   $scope.firstName= "John";  
   $scope.lastName= "Doe";  
 });  
</script>

**AngularJS Module**

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

AngularJS controllers control applications:

**AngularJS Controller**

app.controller('myCtrl', function($scope) {  
   $scope.firstName= "John";  
  $scope.lastName= "Doe";  
 });

**AngularJS Expressions: -** AngularJS expressions can be written inside double braces: {{expression }}.AngularJS expressions can also be written inside a directive: ng-bind="expression". AngularJS will resolve the expression and return the result exactly where the expression is written.

**AngularJS expressions** contain literals, operators, and variables.

<div ng-app="">   <p>My first expression: {{5 + 5}} </p></div>

**If you remove the ng-app directive, HTML will display the expression as it is, without solving it:**

<div> <p>My first expression: {{5 + 5}} </p></div>

**Change the color of the input box below, by changing its value:**

<div ng-app="" ng-init="myCol='lightblue'"> <input style="background-color:{{myCol}}" ng-model="myCol"></div>

**AngularJS numbers**: -

<div ng-app="" ng-init="quantity=1; cost=5"> <p>Total in dollar: {{ quantity \* cost }}</p> </div>

Using ng-init is not very common. You will learn a better way to initialize data in the chapter about controllers.

<div ng-app="" ng-init="quantity=1;cost=5"> <p>Total in dollar: <span ng-bind="quantity \* cost"></span></p> </div>

**AngularJS strings**

<div ng-app="" ng-init="firstName='John';lastName='Doe'">  
 <p>The name is {{ firstName + " " + lastName }}</p>  
 </div>

**using ng-bind:**

<div ng-app="" ng-init="firstName='John';lastName='Doe'">  
 <p>The name is <span ng-bind="firstName + ' ' + lastName"></span></p>  
 </div>

**AngularJS objects**

<div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">  
 <p>The name is {{ person.lastName }}</p>  
 </div>

**using ng-bind:**

<div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">  
 <p>The name is <span ng-bind="person.lastName"></span></p>  
 </div>

**AngularJS arrays**

<div ng-app="" ng-init="points=[1,15,19,2,40]"> <p>The third result is {{ points[2] }}</p> </div>

**using ng-bind:**

<div ng-app="" ng-init="points=[1,15,19,2,40]"> <p>The third result is <span ng-bind="points[2]"></span></p></div>

**AngularJS Expressions vs. JavaScript Expressions**

* Like JavaScript expressions, AngularJS expressions can contain literals, operators, and variables.
* Unlike JavaScript expressions, AngularJS expressions can be written inside HTML.
* AngularJS expressions do not support conditionals, loops, and exceptions, while JavaScript expressions do.
* AngularJS expressions support filters, while JavaScript expressions do not.

**AngularJS Modules**

* An AngularJS module defines an application.
* The module is a container for the different parts of an application.
* The module is a container for the application controllers.
* Controllers always belong to a module.

**Creating a Module: -** A module is created by using the AngularJS function angular.module

<div ng-app="myApp">...</div>  
 <script> var app = angular.module("myApp", []);  </script>The **"myApp"** parameter refers to an HTML element in which the application will run.

**Adding a Controller: -** ng-controller directive:

<div ng-app="**myApp**" ng-controller=**"myCtrl"**>{{ firstName + " " + lastName }} </div>  
 <script>  
 var app = angular.module(**"myApp"**, []);  
 app.controller(**"myCtrl"**, function($scope) {  
   $scope.firstName = "John";  
   $scope.lastName = "Doe";  
 });  
</script>

**Adding a Directive:-**AngularJS has a set of built-in directives which you can use to add functionality to your application.

In addition you can use the module to add your own directives to your applications:

<div ng-app="myApp" w3-test-directive></div>  
<script>   
var app = angular.module("myApp", []);  
app.directive("w3TestDirective", function() {  
  return {  
    template : "I was made in a directive constructor!"  
  };  
});  
</script>

**Modules and Controllers in Files:-**It is common in AngularJS applications to put the module and the controllers in JavaScript files.In this example, "myApp.js" contains an application module definition, while "myCtrl.js" contains the controller:

<div ng-app="**myApp**" ng-controller="**myCtrl**">  
{{ firstName + " " + lastName }}  
</div>  
<script src="**myApp.js**"></script>  
<script src="**myCtrl.js**"></script>  
**myApp.js**

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

The [] parameter in the module definition can be used to define dependent modules.

Without the [] parameter, you are not creating a new module, but retrieving an existing one.

**myCtrl.js**

app.controller(**"myCtrl"**, function($scope) {  
  $scope.firstName = "John";  
  $scope.lastName= "Doe";  
});

**Functions can Pollute the Global Namespace**

Global functions should be avoided in JavaScript. They can easily be overwritten or destroyed by other scripts.

AngularJS modules reduces this problem, by keeping all functions local to the module.

**When to Load the Library:-**While it is common in HTML applications to place scripts at the end of the <body> element, it is recommended that you load the AngularJS library either in the <head> or at the start of the <body>.

This is because calls to angular.module can only be compiled after the library has been loaded.

<!DOCTYPE html>  
<html>  
<body>  
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>  
<div ng-app="myApp" ng-controller="myCtrl">  
{{ firstName + " " + lastName }}  
</div>  
<script>  
var app = angular.module("myApp", []);  
app.controller("myCtrl", function($scope) {  
  $scope.firstName = "John";  
  $scope.lastName = "Doe";  
});  
</script>  
</body>  
</html>

**AngularJS Directives:-**AngularJS directives are extended HTML attributes with the prefix ng-.

The ng-app directive initializes an AngularJS application.

The ng-init directive initializes application data.

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

<div ng-app="" ng-init="firstName='John'">  
<p>Name: <input type="text" ng-model="firstName"></p>  
<p>You wrote: {{ firstName }}</p>  
</div>

**Data Binding:-**The {{ firstName }} expression, in the example above, is an AngularJS data binding expression.

Data binding in AngularJS binds AngularJS expressions with AngularJS data.{{ firstName }} is bound with ng-model="firstName".In the next example two text fields are bound together with two ng-model directives:

<div ng-app="" ng-init="quantity=1;price=5">  
Quantity: <input type="number" ng-model="quantity">  
Costs:    <input type="number" ng-model="price">  
Total in dollar: {{ quantity \* price }}  
</div>

**Repeating HTML Elements:-**The ng-repeat directive repeats an HTML element:

<div ng-app="" ng-init="names=['Jani','Hege','Kai']">  
  <ul>  
    <li ng-repeat="x in names">  
      {{ x }}  
    </li>  
  </ul>  
</div>

The ng-repeat directive actually **clones HTML elements** once for each item in a collection.

The ng-repeat directive used on an array of objects:

<div ng-app="" ng-init="names=[  
{name:'Jani',country:'Norway'},  
{name:'Hege',country:'Sweden'},  
{name:'Kai',country:'Denmark'}]">  
<ul>  
  <li ng-repeat="x in names">  
    {{ x.name + ', ' + x.country }}  
  </li>  
</ul>  
</div>

**The ng-app Directive:-**The ng-app directive defines the **root element** of an AngularJS application.The ng-app directive will **auto-bootstrap** (automatically initialize) the application when a web page is loaded.

**The ng-init Directive:-**The ng-init directive defines **initial values** for an AngularJS application.Normally, you will not use ng-init. You will use a controller or module instead.You will learn more about controllers and modules later.

**The ng-model Directive:-**The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.The ng-model directive can also:Provide type validation for application data (number, email, required).

Provide status for application data (invalid, dirty, touched, error).Provide CSS classes for HTML elements.

Bind HTML elements to HTML forms.

**Create New Directives:-**In addition to all the built-in AngularJS directives, you can create your own directives.

New directives are created by using the .directive function.To invoke the new directive, make an HTML element with the same tag name as the new directive.When naming a directive, you must use a camel case name, w3TestDirective, but when invoking it, you must use - separated name, w3-test-directive:

<body ng-app="myApp">  
<w3-test-directive></w3-test-directive>  
<script>  
var app = angular.module("myApp", []);  
app.directive("w3TestDirective", function() {  
  return {  
    template : "<h1>Made by a directive!</h1>"  
  };  
});  
</script>  
</body>

You can invoke a directive by using:

Element name:-<w3-test-directive></w3-test-directive>

Attribute:-<div w3-test-directive></div>

Class:-<div class="w3-test-directive"></div>

Comment:- <!-- directive: w3-test-directive -->

**Restrictions:-**You can restrict your directives to only be invoked by some of the methods.By adding a restrict property with the value "A", the directive can only be invoked by attributes:

var app = angular.module("myApp", []);  
app.directive("w3TestDirective", function() {  
  return {  
    restrict : "A",  
    template : "<h1>Made by a directive!</h1>"  
  };  
});  
The legal restrict values are:

E for Element name

A for Attribute

C for Class

M for Comment

By default the value is EA, meaning that both Element names and attribute names can invoke the directive.

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

**The ng-model Directive:-**With the ng-model directive you can bind the value of an input field to a variable created in AngularJS.

**Example**

<div ng-app="myApp" ng-controller="myCtrl">  
  Name: <input ng-model="name">  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.name = "John Doe";  
});  
</script>

**Two-Way Binding:-**The binding goes both ways. If the user changes the value inside the input field, the AngularJS property will also change its value:

<div ng-app="myApp" ng-controller="myCtrl">  
  Name: <input ng-model="name">  
  <h1>You entered: {{name}}</h1>  
</div>

**Validate User Input:-**The ng-model directive can provide type validation for application data (number, e-mail, required):

<form ng-app="" name="myForm">  
  Email:  <input type="email" name="myAddress" ng-model="text">  
  <span ng-show="myForm.myAddress.$error.email">Not a valid e-mail address</span>  
</form>

**Application Status:-**The ng-model directive can provide status for application data (valid, dirty, touched, error):

<form ng-app="" name="myForm" ng-init="myText = 'post@myweb.com'">  
  Email:  <input type="email" name="myAddress" ng-model="myText" required>  
  <h1>Status</h1>  
  {{myForm.myAddress.$valid}}  
  {{myForm.myAddress.$dirty}}  
  {{myForm.myAddress.$touched}}  
</form>

**CSS Classes:-**The ng-model directive provides CSS classes for HTML elements, depending on their status:

<style>

input.ng-invalid {  
  background-color: lightblue;  
}

</style><body>  
<form ng-app="" name="myForm">  
  Enter your name:  <input name="myName" ng-model="myText" required>  
</form>

The ng-model directive adds/removes the following classes, according to the status of the form field:

* ng-empty
* ng-not-empty
* ng-touched
* ng-untouched
* ng-valid
* ng-invalid
* ng-dirty
* ng-pending
* ng-pristine

Data binding in AngularJS is the synchronization between the model and the view.

**Data Model**

AngularJS applications usually have a data model. The data model is a collection of data available for the application.

**Example**

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.firstname = "John";  
  $scope.lastname = "Doe";  
});

**HTML View:-**The HTML container where the AngularJS application is displayed, is called the view.The view has access to the model, and there are several ways of displaying model data in the view.You can use the ng-bind directive, which will bind the innerHTML of the element to the specified model property:

<p ng-bind="firstname"></p>

<p>First name: {{firstname}}</p>

**The ng-model Directive:-**Use the ng-model directive to bind data from the model to the view on HTML controls (input, select, textarea)

<input ng-model="firstname">The ng-model directive provides a two-way binding between the model and the view.

**Two-way Binding:-**Data binding in AngularJS is the synchronization between the model and the view.When data in the model changes, the view reflects the change, and when data in the view changes, the modelis updated as well. This happens immediately and automatically, which makes sure that the model and the view is updated at all times.

<div ng-app="myApp" ng-controller="myCtrl">  
  Name: <input ng-model="firstname">  
  <h1>{{firstname}}</h1>  
</div>  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.firstname = "John";  
  $scope.lastname = "Doe";  
});

</script>

**AngularJS Controller: -**Applications in AngularJS are controlled by controllers. Because of the immediate synchronization of the model and the view, the controller can be completely separated from the view, and simply concentrate on the model data. Thanks to the data binding in AngularJS, the view will reflect any changes made in the controller.

<div ng-app="myApp" ng-controller="myCtrl">  
  <h1 ng-click="changeName()">{{firstname}}</h1>  
</div>  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.firstname = "John";  
  $scope.changeName = function() {  
    $scope.firstname = "Nelly";  
  }  
});

</script>

**AngularJS Controllers:-**AngularJS applications are controlled by controllers.The **ng-controller** directive defines the application controller.A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

<div ng-app="myApp" ng-controller="myCtrl">  
First Name: <input type="text" ng-model="firstName"><br>  
Last Name: <input type="text" ng-model="lastName"><br>  
<br>Full Name: {{firstName + " " + lastName}}  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.firstName = "John";  
  $scope.lastName = "Doe";  
});  
</script>

**Controller Methods:-**The example above demonstrated a controller object with two properties: lastName and firstName.

A controller can also have methods (variables as functions):

<div ng-app="myApp" ng-controller="personCtrl">  
First Name: <input type="text" ng-model="firstName"><br>  
Last Name: <input type="text" ng-model="lastName"><br>  
<br>Full Name: {{fullName()}}  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('personCtrl', function($scope) {  
  $scope.firstName = "John";  
  $scope.lastName = "Doe";  
  $scope.fullName = function() {  
    return $scope.firstName + " " + $scope.lastName;  
  };  
});  
</script>

**Controllers In External Files:-**In larger applications, it is common to store controllers in external files.

Just copy the code between the <script> tags into an external file named [personController.js](https://www.w3schools.com/angular/personController.js):

<div ng-app="myApp" ng-controller="personCtrl">  
First Name: <input type="text" ng-model="firstName"><br>  
Last Name: <input type="text" ng-model="lastName"><br>  
<br>Full Name: {{fullName()}}</div>  
<script src="personController.js"></script>

**How to Use the Scope?** When you make a controller in AngularJS, you pass the $scope object as an argument:Properties made in the controller, can be referred to in the view:

<div ng-app="myApp" ng-controller="myCtrl"><h1>{{carname}}</h1></div>  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.carname = "Volvo";  
});  
</script>

**Understanding the Scope:-**If we consider an AngularJS application to consist of:

* View, which is the HTML.
* Model, which is the data available for the current view.
* Controller, which is the JavaScript function that makes/changes/removes/controls the data.
* Then the scope is the Model.
* The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.

<div ng-app="myApp" ng-controller="myCtrl"><input ng-model="name"><h1>My name is {{name}}</h1></div>  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.name = "John Doe";  
});

</script>

**Know Your Scope:-**It is important to know which scope you are dealing with, at any time.In the two examples above there is only one scope, so knowing your scope is not an issue, but for larger applications there can be sections in the HTML DOM which can only access certain scopes.

<div ng-app="myApp" ng-controller="myCtrl"><ul>  <li ng-repeat="x in names">{{x}}</li></ul></div>  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.names = ["Emil", "Tobias", "Linus"];  
});

</script>

**Root Scope:-**All applications have a $rootScope which is the scope created on the HTML element that contains the ng-app directive.The rootScope is available in the entire application.If a variable has the same name in both the current scope and in the rootScope, the application uses the one in the current scope.

<body ng-app="myApp">  
<p>The rootScope's favorite color:</p>  
<h1>{{color}}</h1>  
<div ng-controller="myCtrl">  <p>The scope of the controller's favorite color:</p>  <h1>{{color}}</h1></div>  
<p>The rootScope's favorite color is still:</p>  
<h1>{{color}}</h1>  
<script>

var app = angular.module('myApp', []);  
app.run(function($rootScope) {  
  $rootScope.color = 'blue';  
});  
app.controller('myCtrl', function($scope) {  
  $scope.color = "red";  
});  
</script>  
</body>

**AngularJS Filters**

* currency Format a number to a currency format.
* date Format a date to a specified format.
* filter Select a subset of items from an array.
* json Format an object to a JSON string.
* limitTo Limits an array/string, into a specified number of elements/characters.
* lowercase Format a string to lower case.
* number Format a number to a string.
* orderBy Orders an array by an expression.
* uppercase Format a string to upper case.

**Adding Filters to Expressions:-**Filters can be added to expressions by using the pipe character |, followed by a filter.

The uppercase filter format strings to upper case:

<div ng-app="myApp" ng-controller="personCtrl">  
<p>The name is {{ lastName | uppercase }}</p>  
</div>

**Adding Filters to Directives:-**Filters are added to directives, like ng-repeat, by using the pipe character |, followed by a filter:

<div ng-app="myApp" ng-controller="namesCtrl">  
<ul>  
  <li ng-repeat="x in names | orderBy:'country'">  
    {{ x.name + ', ' + x.country }}  
  </li>  
</ul>  
</div>

**The currency Filter:-**The currency filter formats a number as currency:

<div ng-app="myApp" ng-controller="costCtrl"><h1>Price: {{ price | currency }}</h1></div>

**The filter Filter:-**The filter filter selects a subset of an array.The filter filter can only be used on arrays, and it returns an array containing only the matching items.

<div ng-app="myApp" ng-controller="namesCtrl">  
<ul>  
  <li ng-repeat="x in names | filter : 'i'">  
    {{ x }}  
  </li>  
</ul>  
</div>

**Filter an Array Based on User Input:-**By setting the ng-model directive on an input field, we can use the value of the input field as an expression in a filter.

<div ng-app="myApp" ng-controller="namesCtrl">  
<p><input type="text" ng-model="test"></p>  
<ul>  
  <li ng-repeat="x in names | filter : test">  
    {{ x }}  
  </li>  
</ul>  
</div>

**Sort an Array Based on User Input**

|  |  |
| --- | --- |
| **Name** | **Country** |
| Jani | Norway |
| Carl | Sweden |
| Margareth | England |
| Hege | Norway |
| Joe | Denmark |
| Gustav | Sweden |
| Birgit | Denmark |
| Mary | England |
| Kai | Norway |

**Custom Filters**

<ul ng-app="myApp" ng-controller="namesCtrl">  
  <li ng-repeat="x in names">  
    {{x | **myFormat**}}  
  </li>  
</ul>  
<script>

var app = angular.module('myApp', []);  
app.filter('**myFormat**', function() {  
  return function(x) {  
    var i, c, txt = "";  
    for (i = 0; i < x.length; i++) {  
      c = x[i];  
      if (i % 2 == 0) {  
        c = c.toUpperCase();  
      }  
      txt += c;  
    }  
    return txt;  
  };  
});  
app.controller('namesCtrl', function($scope) {  
  $scope.names = ['Jani', 'Carl', 'Margareth', 'Hege', 'Joe', 'Gustav', 'Birgit', 'Mary', 'Kai'];  
});

</script>

**What is a Service?**In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.AngularJS has about 30 built-in services. One of them is the $location service.The $location service has methods which return information about the location of the current web page:

Use the $location service in a controller:

var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $location) {  
    $scope.myUrl = $location.absUrl();  
});

**Why use Services?**For many services, like the $location service, it seems like you could use objects that are already in the DOM, like the window.location object, and you could, but it would have some limitations, at least for your AngularJS application.AngularJS constantly supervises your application, and for it to handle changes and events properly, AngularJS prefers that you use the $location service instead of the window.location object.

**The $http Service:-**The $http service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

Use the $http service to request data from the server:

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $http) {  
  $http.get("welcome.htm").then(function (response) {  
    $scope.myWelcome = response.data;  
  });  
});

**The $timeout Service:-**The $timeout service is AngularJS' version of the window.setTimeout function.

Display a new message after two seconds:

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $timeout) {  
  $scope.myHeader = "Hello World!";  
  $timeout(function () {  
    $scope.myHeader = "How are you today?";  
  }, 2000);  
});  
**The $interval Service:-**The $interval service is AngularJS' version of the window.setInterval function.

Display the time every second:

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $interval) {  
  $scope.theTime = new Date().toLocaleTimeString();  
  $interval(function () {  
    $scope.theTime = new Date().toLocaleTimeString();  
  }, 1000);  
});  
**Create Your Own Service:-**To create your own service, connect your service to the module:Create a service named hexafy:

app.service('hexafy', function() {  
  this.myFunc = function (x) {  
    return x.toString(16);  
  }  
});

To use your custom made service, add it as a dependency when defining the controller:

**Use a Custom Service Inside a Filter:-**Once you have created a service, and connected it to your application, you can use the service in any controller, directive, filter, or even inside other services.To use the service inside a filter, add it as a dependency when defining the filter:

The service hexafy used in the filter myFormat:

app.filter('myFormat',[**'hexafy'**, function(**hexafy**) {  
  return function(x) {  
    return **hexafy**.myFunc(x);  
  };  
}]);  
**AngularJS $http**

Make a simple request to the server, and display the result in a header:

<div ng-app="myApp" ng-controller="myCtrl">   
<p>Today's welcome message is:</p>  
<h1>{{myWelcome}}</h1>  
</div>  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $http) {  
  $http.get("welcome.htm")  
  .then(function(response) {  
    $scope.myWelcome = response.data;  
  });  
});

</script>

**Methods**

* .delete()
* .get()
* .head()
* .jsonp()
* .patch()
* .post()
* .put()

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $http) {  
  $http({  
    method : "GET",  
      url : "welcome.htm"  
  }).then(function mySuccess(response) {  
    $scope.myWelcome = response.data;  
  }, function myError(response) {  
    $scope.myWelcome = response.statusText;  
  });  
});

**Properties:-**The response from the server is an object with these properties:

.config the object used to generate the request.

.data a string, or an object, carrying the response from the server.

.headers a function to use to get header information.

.status a number defining the HTTP status.

.statusText a string defining the HTTP status.

**Example**

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope, $http) {  
  $http.get("welcome.htm")  
  .then(function(response) {  
    $scope.content = response.data;  
    $scope.statuscode = response.status;  
    $scope.statustext = response.statusText;   
  });  
});

**JSON: -** The data you get from the response is expected to be in JSON format. SON is a great way of transporting data, and it is easy to use within AngularJS, or any other JavaScript. Example: On the server we have a file that returns a JSON object containing 15 customers, all wrapped in array called records.

The ng-repeat directive is perfect for looping through an array:

<div ng-app="myApp" ng-controller="customersCtrl">   
<ul>  
  <li ng-repeat="x in myData">  
    {{ x.Name + ', ' + x.Country }}  
  </li>  
</ul>  
</div>  
<script>

var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $http) {  
  $http.get("customers.php").then(function(response) {  
    $scope.myData = response.data.records;  
  });  
});

</script>

**Displaying Data in a Table**

<div ng-app="myApp" ng-controller="customersCtrl">   
<table>  
  <tr ng-repeat="x in names">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $http) {  
  $http.get("customers.php")  
  .then(function (response) {$scope.names = response.data.records;});  
});  
</script>

**Displaying with CSS Style**

<style>  
table, th , td {  
  border: 1px solid grey;  
  border-collapse: collapse;  
  padding: 5px;  
}  
table tr:nth-child(odd) {  
  background-color: #f1f1f1;  
}  
table tr:nth-child(even) {  
  background-color: #ffffff;  
}  
</style>

**Display with orderBy Filter**

<table>  
  <tr ng-repeat="x in names | orderBy : 'Country'">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>

**Display with uppercase Filter**

<table>  
  <tr ng-repeat="x in names">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country | uppercase }}</td>  
  </tr>  
</table>

**Display the Table Index ($index):-**To display the table index, add a <td> with **$index**:

<table>  
  <tr ng-repeat="x in names">  
    <td>{{ $index + 1 }}</td>  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>

**Using $even and $odd**

<table>  
  <tr ng-repeat="x in names">  
    <td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Name }}</td>  
    <td ng-if="$even">{{ x.Name }}</td>  
    <td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Country }}</td>  
    <td ng-if="$even">{{ x.Country }}</td>  
  </tr>  
</table>

**Creating a Select Box Using ng-options:-**If you want to create a dropdown list, based on an object or an array in AngularJS, you should use the ng-options directive:

<div ng-app="myApp" ng-controller="myCtrl">  
<select ng-model="selectedName" ng-options="x for x in names">  
</select>  
</div>  
<script>

var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.names = ["Emil", "Tobias", "Linus"];  
});

</script>

[Try it Yourself »](https://www.w3schools.com/angular/tryit.asp?filename=try_ng_select)

**ng-options vs ng-repeat:-** You can also use the ng-repeat directive to make the same dropdown list:

<select>  
  <option ng-repeat="x in names">{{x}}</option>  
</select>

**Server Code Examples: -** The following section is a listing of the server code used to fetch SQL data.

* Using PHP and MySQL. Returning JSON.
* Using PHP and MS Access. Returning JSON.
* Using ASP.NET, VB, and MS Access. Returning JSON.
* Using ASP.NET, Razor, and SQL Lite. Returning JSON.

**Cross-Site HTTP Requests: -** A request for data from a different server (other than the requesting page), are called **cross-site** HTTP requests. Cross-site requests are common on the web. Many pages load CSS, images, and scripts from different servers. In modern browsers, cross-site HTTP requests **from scripts** are restricted to **same site** for security reasons.

The following line, in our PHP examples, has been added to allow cross-site access.

header ("Access-Control-Allow-Origin: \*");

**The ng-disabled Directive: -**The **ng-disabled** directive binds AngularJS application data to the disabled attribute of HTML elements.

<div ng-app="" ng-init="mySwitch=true"><p><button ng-disabled="mySwitch">Click Me!</button></p>  
<p><input type="checkbox" ng-model="mySwitch">Button</p>  
<p>{{ mySwitch }}</p>  
</div>

**The ng-show Directive:-** The **ng-show** directive shows or hides an HTML element.

<div ng-app=""><p ng-show="true">I am visible.</p><p ng-show="false">I am not visible.</p></div>

**The ng-hide Directive:-** The **ng-hide** directive hides or shows an HTML element.

<div ng-app=""><p ng-hide="true">I am not visible.</p><p ng-hide="false">I am visible.</p></div>

**AngularJS Events:-** You can add AngularJS event listeners to your HTML elements by using one or more of these directives:

* ng-blur
* ng-change
* ng-click
* ng-copy
* ng-cut
* ng-dblclick
* ng-focus
* ng-keydown
* ng-keypress
* ng-keyup
* ng-mousedown
* ng-mouseenter
* ng-mouseleave
* ng-mousemove
* ng-mouseover
* ng-mouseup
* ng-paste

**Mouse Events:-** Mouse events occur when the cursor moves over an element, in this order:

* ng-mouseover
* ng-mouseenter
* ng-mousemove
* ng-mouseleave
* Or when a mouse button is clicked on an element, in this order:
* ng-mousedown
* ng-mouseup
* ng-click

<div ng-app="myApp" ng-controller="myCtrl">  
<h1 ng-mousemove="count = count + 1">Mouse over me!</h1>  
<h2>{{ count }}</h2>  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.count = 0;  
});  
</script>

**The ng-click Directive:-**The ng-click directive defines AngularJS code that will be executed when the element is being clicked.

<div ng-app="myApp" ng-controller="myCtrl">  
<button ng-click="count = count + 1">Click me!</button>  
<p>{{ count }}</p>  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.count = 0;  
});  
</script>

**Toggle, True/False:-** If you want to show a section of HTML code when a button is clicked, and hide when the button is clicked again, like a dropdown menu, make the button behave like a toggle switch:

Click Me

<div ng-app="myApp" ng-controller="myCtrl">  
<button ng-click="myFunc()">Click Me!</button>  
<div ng-show="showMe">  
  <h1>Menu:</h1>  
  <div>Pizza</div>  
  <div>Pasta</div>  
  <div>Pesce</div>  
</div>  
  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.showMe = false;  
  $scope.myFunc = function() {  
    $scope.showMe = !$scope.showMe;  
  }  
});  
</script>

**$event Object:-** You can pass the $event object as an argument when calling the function.The $event object contains the browser's event object:

<div ng-app="myApp" ng-controller="myCtrl">  
<h1 ng-mousemove="myFunc($event)">Mouse Over Me!</h1>  
<p>Coordinates: {{x + ', ' + y}}</p>  
</div>  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
  $scope.myFunc = function(myE) {  
    $scope.x = myE.clientX;  
    $scope.y = myE.clientY;  
  }  
});  
</script>

**Input Controls**

* input elements
* select elements
* button elements
* textarea elements

**Data-Binding:-** Input controls provides data-binding by using the ng-model directive.

<input type="text" ng-model="firstname">

<script>  
var app = angular.module('myApp', []);  
app.controller('formCtrl', function($scope) {  
  $scope.firstname = "John";  
});  
</script>

**Checkbox:-** A checkbox has the value true or false. Apply the ng-model directive to a checkbox, and use its value in your application.

 <form>  
  Check to show a header:  
  <input type="checkbox" ng-model="myVar">  
</form>  
<h1 ng-show="myVar">My Header</h1>

**Radiobuttons:-** Bind radio buttons to your application with the ng-model directive. Radio buttons with the same ng-model can have different values, but only the selected one will be used.

<form>  
  Pick a topic:  
  <input type="radio" ng-model="myVar" value="dogs">Dogs  
  <input type="radio" ng-model="myVar" value="tuts">Tutorials  
  <input type="radio" ng-model="myVar" value="cars">Cars  
</form>

**Selectbox:-** Bind select boxes to your application with the ng-model directive. The property defined in the ng-model attribute will have the value of the selected option in the selectbox.

<form>  
  Select a topic:  
   <select ng-model="myVar">  
     <option value="">  
     <option value="dogs">Dogs  
     <option value="tuts">Tutorials  
     <option value="cars">Cars  
   </select>  
 </form>

**An AngularJS Form Example**

<div ng-app="myApp" ng-controller="formCtrl">  
   <form novalidate>  
    First Name:<br>   <input type="text" ng-model="user.firstName"><br>  
     Last Name:<br>    <input type="text" ng-model="user.lastName">  <br><br>  
     <button ng-click="reset()">RESET</button>  
   </form>  
   <p>form = {{user}}</p>  
   <p>master = {{master}}</p>  
</div>  
<script>  
 var app = angular.module('myApp', []);  
 app.controller('formCtrl', function($scope) {  
   $scope.master = {firstName: "John", lastName: "Doe"};  
   $scope.reset = function() {  
    $scope.user = angular.copy($scope.master);  
   };  
   $scope.reset();  
 });  
</script>

**Form Validation: -**

* AngularJS offers client-side form validation.
* AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.
* AngularJS also holds information about whether they have been touched, or modified, or not.
* You can use standard HTML5 attributes to validate input, or you can make your own validation functions.
* Client-side validation cannot alone secure user input. Server side validation is also necessary.

**Required**

<form name="myForm">  
   <input name="myInput" ng-model="myInput" required>  
 </form>  
 <p>The input's valid state is:</p>  
 <h1>{{myForm.myInput.$valid}}</h1>

**E-mail**

<form name="myForm">  
   <input name="myInput" ng-model="myInput" type="email">  
 </form>  
 <p>The input's valid state is:</p>  
 <h1>{{myForm.myInput.$valid}}</h1>

**Form State and Input State: -** AngularJS is constantly updating the state of both the form and the input fields.Input fields have the following states:

* $untouched The field has not been touched yet
* $touched The field has been touched
* $pristine No fields have been modified yet
* $dirty One or more have been modified
* $invalid The form content is not valid
* $valid The form content is valid
* $submitted The form is submitted
* They are all properties of the form, and are either true or false.

You can use these states to show meaningful messages to the user. Example, if a field is required, and the user leaves it blank, you should give the user a warning:

**Example**

Show an error message if the field has been touched AND is empty:

<input name="myName" ng-model="myName" required>  
<span ng-show="myForm.myName.$touched && myForm.myName.$invalid">The name is required.</span>

**CSS Classes: -** AngularJS adds CSS classes to forms and input fields depending on their states.The following classes are added to, or removed from, input fields:

* ng-untouched The field has not been touched yet
* ng-touched The field has been touched
* ng-valid-key One key for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated
* The following classes are added to, or removed from, forms:
* ng-pristine No fields has not been modified yet
* ng-dirty One or more fields has been modified
* ng-valid The form content is valid
* ng-invalid The form content is not valid
* The classes are removed if the value they represent is false.

Apply styles, using standard CSS:

<style>

input.ng-invalid {  
  background-color: pink;  
 }  
 input.ng-valid {  
   background-color: lightgreen;  
 }

</style>

Apply styles for unmodified (pristine) forms, and for modified forms:

<style>

form.ng-pristine {  
   background-color: lightblue;  
 }  
 form.ng-dirty {  
   background-color: pink;  
}

</style>

**Custom Validation: -** To create your own validation function is a bit more tricky; You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

Create your own directive, containing a custom validation function, and refer to it by using my-directive.

The field will only be valid if the value contains the character "e":

<form name="myForm">  
 <input name="myInput" ng-model="myInput" required my-directive>  
 </form>  
<script>

var app = angular.module('myApp', []);  
 app.directive('myDirective', function() {  
   return {  
     require: 'ngModel',  
    link: function(scope, element, attr, mCtrl) {  
       function myValidation(value) {  
         if (value.indexOf("e") > -1) {  
          mCtrl.$setValidity('charE', true);  
        } else {  
           mCtrl.$setValidity('charE', false);  
         }  
         return value;  
       }  
      mCtrl.$parsers.push(myValidation);  
     }  
  };  
 });

</script>

**Validation Example**

<h2>Validation Example</h2>  
 <form  ng-app="myApp"  ng-controller="validateCtrl" name="myForm" novalidate>  
 <p>Email:<br>  
   <input type="email" name="email" ng-model="email" required>  
   <span style="color:red" ng-show="myForm.email.$dirty && myForm.email.$invalid">  
   <span ng-show="myForm.email.$error.required">Email is required.</span>  
   <span ng-show="myForm.email.$error.email">Invalid email address.</span>  
   </span>  
 </p>  
 <p>  
  <input type="submit"   ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||  myForm.email.$dirty && myForm.email.$invalid">  
 </p>

</form>  
<script>  
 var app = angular.module('myApp', []);  
 app.controller('validateCtrl', function($scope) {  
   $scope.user = 'John Doe';  
   $scope.email = 'john.doe@gmail.com';  
 });  
</script>  
</body>  
**AngularJS Global API: -** The AngularJS Global API is a set of global JavaScript functions for performing common tasks like:

* Comparing objects
* Iterating objects
* Converting data

|  |  |
| --- | --- |
| **API** | **Description** |
| angular.lowercase() | Converts a string to lowercase |
| angular.uppercase() | Converts a string to uppercase |
| angular.isString() | Returns true if the reference is a string |
| angular.isNumber() | Returns true if the reference is a number |

**angular.lowercase()**

<div ng-app="myApp" ng-controller="myCtrl">  
   <p>{{ x1 }}</p>  <p>{{ x2 }}</p>

</div>  
 <script>  
 var app = angular.module('myApp', []);  
 app.controller('myCtrl', function($scope) {  
  $scope.x1 = "JOHN";  
  $scope.x2 = angular.lowercase($scope.x1);  
 });  
 </script>

**angular. uppercase ()**

<div ng-app="myApp" ng-controller="myCtrl">  
   <p>{{ x1 }}</p>  <p>{{ x2 }}</p>  
 </div>  
 <script>  
 var app = angular.module('myApp', []);  
 app.controller('myCtrl', function($scope) {  
  $scope.x1 = "John";  
  $scope.x2 = angular.uppercase($scope.x1);  
 });  
 </script>

**angular. isString ()**

<div ng-app="myApp" ng-controller="myCtrl">  
   <p>{{ x1 }}</p>  <p>{{ x2 }}</p>  
 </div>   
 <script>  
 var app = angular.module('myApp', []);  
 app.controller('myCtrl', function($scope) {  
  $scope.x1 = "JOHN";  
  $scope.x2 = angular.isString($scope.x1);  
 });  
</script>

**angular. isNumber ()**

<div ng-app="myApp" ng-controller="myCtrl">  
   <p>{{ x1 }}</p>  <p>{{ x2 }}</p>  
 </div>  
<script>  
 var app = angular.module('myApp', []);  
 app.controller('myCtrl', function($scope) {  
  $scope.x1 = "JOHN";  
  $scope.x2 = angular.isNumber($scope.x1);  
 });  
</script>

**Directives**

|  |  |
| --- | --- |
| **AngularJS Directive** | **Description** |
| <body ng-app | Defines an application for the <body> element |
| <body ng-controller | Defines a controller for the <body> element |
| <tr ng-repeat | Repeats the <tr> element for each user in users |
| <button ng-click | Invoke the function editUser() when the <button> element is clicked |
| <h3 ng-show | Show the <h3>s element if edit = true |
| <h3 ng-hide | Hide the form if hideform = true, and hide the <h3> element if edit = true |
| <input ng-model | Bind the <input> element to the application |
| <button ng-disabled | Disables the <button> element if error or incomplete = true |

**W3.CSS Classes Explained**

|  |  |  |
| --- | --- | --- |
| **Element** | **Class** | **Defines** |
| <div> | w3-container | A content container |
| <table> | w3-table | A table |
| <table> | w3-bordered | A bordered table |
| <table> | w3-striped | A striped table |
| <button> | w3-btn | A button |
| <button> | w3-green | A green button |
| <button> | w3-ripple | A ripple effect when you click the button |
| <input> | w3-input | An input field |
| <input> | w3-border | A border on the input field |

**JavaScript Code Explained**

|  |  |
| --- | --- |
| **Scope Properties** | **Used for** |
| $scope.fName | Model variable (user first name) |
| $scope.lName | Model variable (user last name) |
| $scope.passw1 | Model variable (user password 1) |
| $scope.passw2 | Model variable (user password 2) |
| $scope.users | Model variable (array of users) |
| $scope.edit | Set to true when user clicks on 'Create user'. |
| $scope.hideform | Set to false when user clicks on 'Edit' or 'Create user'. |
| $scope.error | Set to true if passw1 not equal passw2 |
| $scope.incomplete | Set to true if any field is empty (length = 0) |
| $scope.editUser | Sets model variables |
| $scope.$watch | Watches model variables |
| $scope.test | Tests model variables for errors and incompleteness |

**AngularJS Includes: -** With AngularJS, you can include HTML content using the **ng-include** directive:

<body ng-app=""> <div ng-include="'myFile.htm'"></div> </body>

**Include AngularJS Code: -** The HTML files you include with the ng-include directive, can also contain AngularJS code:

**myTable.htm:**

<table>  
   <tr ng-repeat="x in names">  
     <td> {{x. Name}} </td>  
     <td> {{x. Country}} </td>  
   </tr>  
 </table>

Include the file "myTable.htm" in your web page, and all AngularJS code will be executed, even the code inside the included file:

<body>  
 <div ng-app="myApp" ng-controller="customersCtrl">   
   <div ng-include="'myTable.htm'"></div>  
 </div>  
 <script>

var app = angular.module('myApp', []);  
 app.controller('customersCtrl', function($scope, $http) {  
   $http.get("customers.php").then(function (response) {  
    $scope.names = response.data.records;  
   });  
 });

</script>

**Include Cross Domains: -** By default, the ng-include directive does not allow you to include files from other domains.

To include files from another domain, you can add a whitelist of legal files and/or domains in the config function of your application:

<body ng-app="myApp">  
 <div ng-include="'https://tryit.w3schools.com/angular\_include.php'"></div>  
 <script>

var app = angular. module ('myApp', [])  
 app.config(function($sceDelegateProvider) {  
  $sceDelegateProvider.resourceUrlWhitelist([  
     'https://tryit.w3schools.com/\*\*'  
   ]);  
 });

</script>  
</body>

Be sure that the server on the destination allows cross domain file access. The ngRoute module helps your application to become a Single Page Application.

**What is Routing in AngularJS?** If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the ngRoute module. The ngRoute module routes your application to different pages without reloading the entire application.

**Navigate to "red.htm", "green.htm", and "blue.htm":**

<body ng-app="myApp">  
 <p><a href="#/!">Main</a></p>  
 <a href="#!red">Red</a>  
 <a href="#!green">Green</a>  
 <a href="#!blue">Blue</a>  
 <div ng-view></div>  
 <script>  
 var app = angular.module("myApp", ["ngRoute"]);  
 app.config(function($routeProvider) {  
  $routeProvider .when("/", {  
    templateUrl : "main.htm"  
  })  .when("/red", {  
    templateUrl : "red.htm"  
  })  .when("/green", {  
    templateUrl : "green.htm"  
   })  .when("/blue", {  
    templateUrl : "blue.htm"  
   });

});  
 </script>  
</body>

**What do I Need?** To make your applications ready for routing, you must include the AngularJS Route module:

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>

Then you must add the ngRoute as a dependency in the application module:

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

Now your application has access to the route module, which provides the $routeProvider.

**Where Does it Go?** Your application needs a container to put the content provided by the routing. This container is the ng-view directive. There are three different ways to include the ng-view directive in your application:

<div ng-view></div>

<ng-view></ng-view>

<div class="ng-view"></div>

Applications can only have one ng-view directive, and this will be the placeholder for all views provided by the route.

**$routeProvider: -** with the $routeProvider you can define what page to display when a user clicks a link. Define the $routeProvider using the config method of your application. Work registered in the configmethod will be performed when the application is loading.

**Controllers: -**With the $routeProvider you can also define a controller for each "view".

var app = angular.module("myApp", ["ngRoute"]);  
 app.config(function($routeProvider) {  
  $routeProvider.when("/", {  
    templateUrl : "main.htm"  
  })  .when("/london", {  
    templateUrl : "london.htm",  
     controller : "londonCtrl"  
  })  .when("/paris", {  
    templateUrl : "paris.htm",  
     controller : "parisCtrl"  
   });  
 });  
 app.controller("londonCtrl", function ($scope) {  
   $scope.msg = "I love London";  
 });  
 app.controller("parisCtrl", function ($scope) {  
  $scope.msg = "I love Paris";  
});  
The files looks like this:

[london.htm](https://www.w3schools.com/angular/london.htm)

<h1>London</h1>  
<h3>London is the capital city of England.</h3>  
<p>It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>  
<p>{{msg}}</p>

[paris.htm](https://www.w3schools.com/angular/paris.htm)

<h1>Paris</h1>  
<h3>Paris is the capital city of France.</h3>  
<p>The Paris area is one of the largest population centers in Europe, with more than 12 million inhabitants.</p>  
<p>{{msg}}</p>

**Template: -** In the previous examples we have used the templateUrl property in the $routeProvider.when method.

You can also use the template property, which allows you to write HTML directly in the property value, and not refer to a page.

var app = angular.module("myApp", ["ngRoute"]);  
 app.config(function($routeProvider) {  
  $routeProvider .when("/", {  
    template : "<h1>Main</h1><p>Click on the links to change this content</p>"  
  })  .when("/banana", {

    template : "<h1>Banana</h1><p>Bananas contain around 75% water.</p>"  
  })  .when("/tomato", {  
     template : "<h1>Tomato</h1><p>Tomatoes contain around 95% water.</p>"  
  });  
});  
**The otherwise method:-** In the previous examples we have used the when method of the $routeProvider.

You can also use the otherwise method, which is the default route when none of the others get a match.

If neither the "Banana" nor the "Tomato" link has been clicked, let them know:

var app = angular.module("myApp", ["ngRoute"]);  
 app.config(function($routeProvider) {  
  $routeProvider .when("/banana", {  
     template : "<h1>Banana</h1><p>Bananas contain around 75% water.</p>"  
  })  .when("/tomato", {  
    template : "<h1>Tomato</h1><p>Tomatoes contain around 95% water.</p>"  
   })  .otherwise({  
    template : "<h1>None</h1><p>Nothing has been selected</p>"  
  });  
});