* **AngularJS**

Angular JS is an open source JavaScript framework by Google to build web applications. It can be freely used, changed and shared by anyone. 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. It is a library 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/1.6.4/angular.min.js"> </script>
* Angular JS is an open source JavaScript framework that is used to build web applications. It can be freely used, changed and shared by anyone.
* Angular Js is developed by Google.
* It is an excellent framework for building single phase applications and line of business applications.

**Advantage of AngularJS**

There are a lot of JavaScript frameworks for building web applications. So, it is a genuine question, why to use Angular JS. Following are the advantages of AngularJS over other JavaScript frameworks:

* Dependency Injection: Dependency Injection specifies a design pattern in which components are given their dependencies instead of hard coding them within the component.
* Two way data binding: AngularJS creates a two way data-binding between the select element and the orderProp model. orderProp is then used as the input for the orderBy filter.
* Testing: Angular JS is designed in a way that we can test right from the start. So, it is very easy to test any of its components through unit testing and end-to-end testing.
* Model View Controller: In Angular JS, it is very easy to develop application in a clean MVC way. You just have to split your application code into MVC components i.e. Model, View and the Controller.
* Directives, filters, modules, routes etc.
* 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, textarea) to application data.
* The **ng-bind** directive binds application data to the HTML view.
* **Role of angularjs in platforms like mobile angular UI and IONIC**

As mobile apps have already transformed our work habits and browsing patterns, the ways of app development has also been through a lot fresh changes. Of late, there are a lot of cutting-edge tools, frameworks and platforms that have eased the task of developers. Designers can now create incredibly useful and high-end mobile apps easily. Consequently, it provides plenty of user-centric and helpful app to organizations as well as businesses to improve their overall business presence through their mobile app.

Ionic is a new entry in the range of hybrid mobile app development platforms. Basically, it is a front-end SDK that is built on top of the most useful AngularJS – a very popular JavaScript library. Whilst native apps still need separate codes written for individual platforms across mobiles like Windows, Android and iOS, hybrid applications offer an added advantage. It requires lesser efforts of development and reduces maintenance costs too. In addition, Ionic applications also provides companies and developers an array of benefits one single development code across several mobile platforms with the help of the most popular JavaScript framework called [AngularJS](http://www.mindinventory.com/angularjs-development.php).

There are several features of this winning combination of AngularJS and Ionic, which includes:

**1. One App Many Platforms**  
Technology has advanced and experts have come up with new ways to create interactive and customizable mobile apps for excellent user experience. Until recently, there were over 4 million developers worldwide having used Ionic for building more than 2 million apps. In fact, it is the most comprehensive SDK available for HTML5 mobile app development framework.

It offers plenty of advantages than most native apps. With built in Sass and AngularJS optimization, Ionic also offers an entire library of optimized HTML, JS and CSS tools and components for mobile. It reduces code writing for several platforms. So, one single app can actually run on all mobile platforms like Android, Windows and iOS, without any additional efforts or development time.

**2. Ionic Benefits and Features**  
You cannot just ignore Ionic! It is the best platform for debugging, UI, build and performance. In fact, Ionic is one platform with over a dozens of useful components and extensible options. In addition, the quality of each of these components is exceptional and offers seamless interaction.

With the latest version, application layout now depends on the type of device and the platform, Ionic, follows certain guidelines on receptive platforms for perfect element display. It also follows the intricacies of transition, platform-specific behavior and transition styling.

**3. Component Creation with Ionic**  
Most [app developers](http://www.mindinventory.com/hire-dedicated-developers.php) complain about the efforts needed to create splash screens and icons for most hybrid apps. It is vital to consider individual parameters like height, density, width, type of device, and host. Moreover, to complete the job successfully, icons with multiple splash screens should be created. Ionic makes this job relatively easier; you need to create 2 simple files with .png, .ai, or .psd extension within the main resource directory, state the template, and sorted!

**4. AngularJS and Ionic – The Deadly Combination**  
Experts opine that AngularJS and [Ionic](http://www.mindinventory.com/ionic-app-development.php) is one of the best matches that could have ever existed. It is a winning combo for developing one of the finest and robust SDK apt for building powerful and rich mobile apps for several app stores, which includes Google Play, Apple Store and Windows.

Ionic comprises a few dozens of components extending to different functionalities. JavaScript portions of the hybrid mobile apps can be built with the help of AngularJS, and Ionic is built using Angular 1.0. Likewise Ionic 2 works together with AngularJS 2.0.

**Major Advantages of Choosing Ionic and Angular JS Combo**  
Ionic, when combined with AngularJS, is a recipe for remarkable transition in the field of mobile app development. Using AngularJS front-end JavaScript framework makes it possible to build fast and scalable applications, offering various services like analytics and push notifications.

There are a plenty of advantages of Ionic and AngularJS as a combination as Ionic adds to the widely preferable Angular JS features.

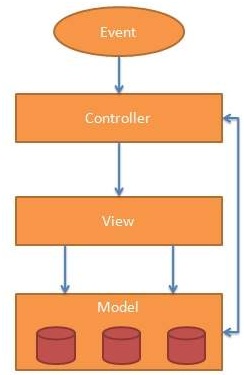
* Ionic provides one of the finest ready-made components for mobiles with extensible and beautiful themes with beautiful typography adapting to different platforms. There are plenty of JavaScript components and default CSS tools like Form Inputs, Buttons, Navigation, Sliding Menus, Tabs, Buttons, and Prompts etc. These styles are simple, functional and really stylish.
* Ionic has been perfectly modeled for standard development SDKs. It emphasizes more on the native apps running inside Cordova or PhoneGap to deploy apps natively. Likewise, AngularJS is easier to use while interacting with external data sources and backend services.
* One of the major advantages of using Ionic is the most potent CLI function that helps developers to build, create, test as well as deploy an app onto any platform using a single command. Some amazing features such as Live Reload and Logging also enhance experience.

In fact, one of the potential advantages of using Ionic is to assist the existing web developers to build mobile apps conveniently and easily, and with a lot of agility. In this way, it saves money and time for businesses owners as well as companies in identifying mobile-savvy developers.

# AngularJS MVC Architecture

MVC stands for Model View Controller. It is a software design pattern for developing web applications. It is very popular because it isolates the application logic from the user interface layer and supports separation of concerns. The MVC pattern is made up of the following three parts:

1. **Model:** It is responsible for managing application data. It responds to the requests from view and to the instructions from controller to update itself.
2. **View:** It is responsible for displaying all data or only a portion of data to the users. It also specifies the data in a particular format triggered by the controller's decision to present the data. They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology.
3. **Controller:** It is responsible to control the relation between models and views. It responds to user input and performs interactions on the data model objects. The controller receives input, validates it, and then performs business operations that modify the state of the data model.



**An AngularJS application consists of following three important parts −**

* **ng-app** − This directive defines and links an AngularJS application to HTML.
* **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
* **ng-bind** − This directive binds the AngularJS Application data to HTML tags.

## Steps to create AngularJS Application

### Step 1 − Load framework, Being a pure JavaScript framework, It can be added using <Script> tag.

### Step 2 − Define AngularJS Application using ng-app directive

### Step 3 − Define a model name using ng-model directive

### Step 4 − Bind the value of above model defined using ng-bind directive.

## Steps to run AngularJS Application,Use following mentioned three steps in an HTML page. And run the html file in browser.

**ckkajs1.html**

<!DOCTYPE html>  
<html>  
<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>  
<div ng-app="">  
  <p>Name: <input type="text" ng-model="name"></p>  
  <p ng-bind="name"></p>  
</div>  
</body>  
</html>

AngularJS starts automatically when the web page has loaded. The **ng-app** directive tells AngularJS that the <div> element is the "owner" of an AngularJS **application**. The **ng-model** directive binds the value of the input field to the application variable **name**. The **ng-bind** directive binds the **innerHTML** of the <p> element to the application variable **name**.

## How AngularJS integrates with HTML

* ng-app directive indicates the start of AngularJS application.
* ng-model directive then creates a model variable named "name" which can be used with the html page and within the div having ng-app directive.
* ng-bind then uses the name model to be displayed in the html span tag whenever user input something in the text box.
* Closing</div> tag indicates the end of AngularJS application.

## AngularJS Directives

AngularJS directives are HTML attributes with an **ng** prefix. The **ng-init** directive initializes AngularJS application variables. You can use data-ng instead of ng-.

**ckkajs2.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>

<div ng-app="" ng-init="firstName='God is great'">

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

</div>

</body>

</html>

## 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** are much like **JavaScript expressions:** They can contain literals, operators, and variables.
* If you remove the ng-app directive, HTML will display the expression as it is, without solving it:

Example {{ 5 + 5 }} or {{ firstName + " " + lastName }}

**ckkajs3.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

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

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

**ckkajs4.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>  
<div ng-app="">  
  <p>Name: <input type="text" ng-model="name"></p>  
  <p>{{name}}</p>  
</div>  
</body>  
</html>

**AngularJS change the value of CSS properties.**

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

**ckkajs5.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

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

</body>  
</html>

## AngularJS Numbers

AngularJS numbers are like JavaScript numbers:

**ckkajs6.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>  
<div ng-app="" ng-init="quantity=2;cost=5">  
<p>Total in amount: {{ quantity \* cost }}</p>  
</div>

</body>  
</html>

Same example using ng-bind:

**ckkajs7.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>  
<div ng-app="" ng-init="quantity=2;cost=5">  
<p>Total in amount: <span ng-bind="quantity \* cost"></span></p>  
</div>

</body>  
</html>

## AngularJS Strings

AngularJS strings are like JavaScript strings:

**ckkajs8.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

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

</body>  
</html>

Same example using ng-bind:

**ckkajs9.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

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

</body>  
</html>

## AngularJS Objects

AngularJS objects are like JavaScript objects:

**ckkajs10.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

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

</body>  
</html>

Same example using ng-bind:

**ckkajs11.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

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

</body>  
</html>

## AngularJS Arrays

AngularJS arrays are like JavaScript arrays:

**ckkajs12.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

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

</body>  
</html>

Same example using ng-bind:

**ckkajs13.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

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

</body>  
</html>

## 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.
* Learn about JavaScript in our [JavaScript Tutorial](https://www.w3schools.com/js/default.asp).
* **AngularJS Directives**
* AngularJS lets you extend HTML with new attributes called **Directives**.
* AngularJS has a set of built-in directives which offers functionality to your applications.
* AngularJS also lets you define your own directives.
* These are special attributes starting with ng- prefix.
* **ng-app directive**

ng-app directive starts an AngularJS Application. It defines the root element. It automatically initializes or bootstraps the application when web page containing AngularJS Application is loaded. It is also used to load various AngularJS modules in AngularJS Application. In following example, we've defined a default AngularJS application using ng-app attribute of a div element. You can only have one ng-app directive in your HTML document. If you have more than one ng-app directive; the first appeared directive will be used.

Syntax:

**<element** ng-app="modulename"**>**

...

  content inside the ng-app root element can contain AngularJS code

...

**</element>**

**modulename:** It is an optional parameter. It specifies the name of a module that you want to add with the application.

**ckkajs14.html**

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

The ng-app directive also tells AngularJS that the <div> element is the "owner" of the AngularJS application. 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. ng-app is the simplest, easiest and most common way to bootstrap an application.

**ckkajs15.html**

<!DOCTYPE html**>**

<**html**>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<**body**>

<**div** ng-app="">

<**p**>The calculated value is : {{ 5 + 5 \* 8 / 2}}</**p**>

</**div**>

</**body**>

</**html**>

* **ng-init directive**

ng-init directive initializes an AngularJS Application data. It is used to put values to the variables to be used in the application. In following example, we'll initialize an array of countries. Normally, you will not use ng-init. You will use a controller or module instead.

**<div** ng-app = "" ng-init = "countries = [{locale:'en-IND',name:'India'}, {locale:'en-PAK',name:'Pakistan'}, {locale:'en-AUS',name:'Australia'}]"**>**

   ...

**</div>**

* **ng-model directive**

This directive binds the values of AngularJS application data to HTML input controls. In following example, we've defined a model named "name". 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.

<div ng-app = "">

...

<p>Enter your Name: <input type = "text" ng-model = "name"></p>

</div>

## 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>

<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 = "God is great";  
});  
</script>

* **ng-repeat directive**

ng-repeat directive repeats html elements for each item in a collection. In following example, we've iterated over array of countries. The ng-repeat directive repeats an HTML element:

**ckkajs16.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<div ng-app="" ng-init="names=['Suneel','Prashant','Ankita']">

<ul>

<li ng-repeat="x in names">

{{ x }}

</li>

</ul>

</div>

</div>

</body>

</html>

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:

**ckkajs17.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<div ng-app="" ng-init="names=[  
{name:'SRD',country:'Jamangar'},

{name:'PMD',country:'Bhavnagar'},

{name:'CKK',country:'Rajkot'}]">

<ul>  
  <li ng-repeat="x in names">  
    {{ x.name + ', ' + x.country }}  
  </li>  
</ul>  
</div>

</div>

</body>

</html>

ng-repeat directive repeats html elements for each item in a collection. In following example, we've iterated over array of countries.

<div ng-app = "">

...

<p>List of Countries with locale:</p>

<ol>

<li ng-repeat = "country in countries">

{{ 'Country: ' + country.name + ', Locale: ' + country.locale }}

</li>

</ol>

</div>

**ckkajs18.html**

Following example will showcase all the above mentioned directives.

<html>

<head>

<title>AngularJS Directives</title>

</head>

<body>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<h1>Sample Application</h1>

<div ng-app = "" ng-init = "countries = [{locale:'Rajkot',name:'India-Rajkot'}, {locale:'Jamnagar',name:'India-Jamnagar'}, {locale:'bhavnagar',name:'India-Bhavnagar'}]"> <p>Enter your Name: <input type = "text" ng-model = "name"></p>

<p>Hello <span ng-bind = "name"></span>!</p>

<p>List of Countries with locale:</p>

<ol>

<li ng-repeat = "country in countries">

{{ 'Country: ' + country.name + ', Locale: ' + country.locale }}

</li>

</ol>

</div>

</body>

</html>

# 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.

**Ckkajs19.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<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= "God";

$scope.lastName= " is great";

});

</script>

</body>

</html>

# 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.
* 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.

Now you can add controllers, directives, filters, and more, to your AngularJS application.

## Adding a Controller

Add a controller to your application, and refer to the controller with the ng-controller directive:

### Example

**Ckkajs20.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\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 = "God ";

$scope.lastName = "is great";

});

</script>

</body>

</html>

## 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:

**Ckkajs21.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<div ng-app="myApp" testdirective>

{{ firstName + " " + lastName }}

</div>

<script>

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

app.directive("testdirective ", function() {

tem=return {

"I was made in a directive constructor!"

};

});

</script>

</body>

</html>

## 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:

**Ckkajs22.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

{{ firstName + " " + lastName }}

</div>

<script src="myApp.js"></script>

<script src="myCtrl.js"></script>

</body>

</html>

### 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 = "CKK";  
    $scope.lastName= "God is great";  
});

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.

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.

### Example

<!DOCTYPE html>  
<html>  
<body>  
<script src="c:\angular-1.6.6\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 = "ckk";  
    $scope.lastName = " god is great";  
});  
</script>  
</body>  
</html>

## 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:

### Example

**Ckkajs23.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>

<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 Rs: {{ quantity \* price }}

</div>

</html>

# AngularJS Controllers

AngularJS controllers **control the data** of AngularJS applications. AngularJS controllers are regular **JavaScript Objects**. 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**. OR

AngularJS controllers are used to control the flow of data of AngularJS application. A controller is defined using ng-controller directive. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control.

**Ckkajs24.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>

<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 = "CKK";

$scope.lastName = " God is great";

});

</script>

</html>

* The AngularJS application is defined by  **ng-app="myApp"**. The application runs inside the <div>.
* The **ng-controller="myCtrl"** attribute is an AngularJS directive. It defines a controller.
* The **myCtrl** function is a JavaScript function.
* AngularJS will invoke the controller with a **$scope** object.
* In AngularJS, $scope is the application object (the owner of application variables and functions).
* The controller creates two properties (variables) in the scope (**firstName** and **lastName**).
* The **ng-model** directives bind the input fields to the controller properties (firstName and lastName).

## Controller Methods

The example above demonstrated a controller object with two properties: lastName and firstName. A controller can also have methods (variables as functions):

**Ckkajs25.html**

## <!DOCTYPE html>

## <html>

## <script src="c:\angular-1.6.6\angular.min.js"></script>

## <body>

## <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 = "CKK";

## $scope.lastName = " God is great";

## $scope.fullName = function() {

## return $scope.firstName + " " + $scope.lastName;

## };

## });

## </script>

## </html>

## 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 [perctrl.js](https://www.w3schools.com/angular/personController.js):

**Ckkajs26.html**

## <!DOCTYPE html>

## <html>

## <script src="c:\angular-1.6.6\angular.min.js"></script>

## <body>

## <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="perctrl.js"></script>

## </html>

**Perctrl.js**

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

app.controller('personCtrl', function($scope) {

$scope.firstName = "CKK";

$scope.lastName = " God is great";

$scope.fullName = function() {

return $scope.firstName + " " + $scope.lastName;

};

})

**nc.js**

angular.module('myApp', []).controller('namesCtrl', function($scope) {  
    $scope.names = [  
        {name: 'ckk',country:'Rajkot'},  
        {name:'SRD',country:'Jamnagar'},  
        {name:'PMD',country:'Bhavnagar'}  
    ];  
});

Save the file as  [nc.js](https://www.w3schools.com/angular/namesController.js): And then use the controller file in an application:

**Ckkajs27.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="namesCtrl">

<ul>

<li ng-repeat="x in names">

{{ x.name + ', ' + x.country }}

</li>

</ul>

</div>

<script src="nc.js"></script>

</html>

# ng-bind

The AngularJS ng-bind directive replaces the content of an HTML element with the value of a given variable, or expression. If you change the value of the given variable or expression, AngularJS changes the content of the specified HTML element as well as. It is an alternative to the interpolation directive.

**Syntax:**

**As attribute:**

<element ng-bind="expression"></element>

**As CSS class:**

<element **class**="ng-bind: expression"></element>

**expression:** It specifies a variable, or an expression to evaluate.

**Ckkajs28.html**

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>ng-bind example</title>

<script src="c:\angular-1.6.6\angular.min.js"></script>

</head>

<body ng-app="bindExample">

<script>

angular.module('bindExample', [])

.controller('ExampleController', ['$scope', function($scope) {

$scope.name = 'Kem chho';

}]);

</script>

<div ng-controller="ExampleController">

<label>Enter name: <input type="text" ng-model="name"></label><br>

Hello <span ng-bind="name"></span>!

</div>

</body>

</html>

# ng-click

The AngularJS ng-click directive facilitates you to specify custom behavior when an element is clicked. So, it is responsible for the result what you get after clicking. It is supported by all HTML elements.

**Syntax:**

**<element** ng-click="expression"**></element>**

**expression:** It specifies an expression that is executed when an element is clicked.

Let's take an example to demonstrate the ng-click directive.

**Ckkajs29.html**

## <!DOCTYPE html>

## <html>

## <script src="c:\angular-1.6.6\angular.min.js"></script>

## <body ng-app="">

## <p>Click the button:</p>

## <button ng-click="count = count + 1" ng-init="count=0">Ram nam bhaj</button>

## <p>You have pressed ram nam bhaj button <strong>{{count}} </strong>times.</p>

## <p><strong>Note:</strong>This example counts a value every time you click on the button and increase the value of the variable.</p>

## </body>

## </html>

## ng-click directive example using function

**ckkajs30.html**

# <!DOCTYPE html>

# <html>

# <script src="c:\angular-1.6.6\angular.min.js"></script>

# <body ng-app="myApp">

# <div ng-controller="myCtrl">

# <p>Click the button to execute a function:</p>

# <button ng-click="myFunc()">OK</button>

# <p>The button has been clicked <strong>{{count}}</strong> times.</p>

# </div>

# <script>

# angular.module('myApp', [])

# .controller('myCtrl', ['$scope', function($scope) {

# $scope.count = 0;

# $scope.myFunc = function() {

# $scope.count++;

# };

# }]);

# </script>

# </body>

# </html>

# ng-show

The AngularJS ng-show directive is used to show or hide the given HTML element according to the expression given to the ng-show attribute. It shows the specified HTML element if the given expression is true, otherwise it hides the HTML element. It is supported by all HTML elements.

**Syntax:**

**<element** ng-show="expression"**></element>**

**expression:** It specifies an expression that will show the element only if the expression returns true.

**Ckkajs31.html**

# <!DOCTYPE html>

# <html>

# <script src="c:\angular-1.6.6\angular.min.js"></script>

# <body ng-app="">

# Show HTML element: <input type="checkbox" ng-model="myVar">

# <div ng-show="myVar">

# <h1>Welcome to Department of Computer science</h1>

# <p>A solution of all technology.</p>

# </div>

# </body>

# </html>

# ng-hide

The AngularJS ng-hide directive is used to hide the HTML element if the expression is set to true. The element is shown if you remove the ng-hide CSS class and hidden, if you add the ng-hide CSS class onto the element. The ng-hide CSS class is predefined in AngularJS and sets the element's display to none.

**<element** ng-hide="expression"**></element>**

As a CSS class:

**<element** class="ng-hide"**></element>**

**expression:** It specifies an expression that will hide the element if the expression returns true.

Let's take an example to demonstrate ng-hide directive.

**Ckkajs32.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body ng-app="">

Hide HTML: <input type="checkbox" ng-model="myVar">

<div ng-hide="myVar">

<h1>Welcome to Department of Compute Science</h1>

<p>A solution of all technologies.</p>

</div>

</body>

</html>

# ng-submit

The AngularJS ng-submit directive specifies a function to be executed when the form is submitted. If you don't use the ng-submit directive in the form then it will not be submitted. It is supported by <form> element.

**Syntax:**

**<form** ng-submit="expression"**></form>**

**expression:** It specifies a function that is called when the form is being submitted, or an expression to be evaluated, which should return a function call.

**Ckkajs33.js**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body ng-app="myApp" ng-controller="myCtrl">

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

<input type="text">

<input type="submit">

</form>

<p>{{cmyTxt}}</p>

<p>In this example you can see, how an AngularJS function is executed when a form is submitted.</p>

<script>

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

app.controller("myCtrl", function($scope) {

$scope.cmyTxt = "Nothing is submitted.";

$scope.myFunc = function () {

$scope.cmyTxt = "Submit button clicked successfully.";

}

});

</script>

</body>

</html>

**Ckkajs34.html**

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<script src="c:\angular-1.6.6\angular.min.js"></script>

</head>

<body ng-app="submitExample">

<script>

angular.module('submitExample', [])

.controller('ExampleController', ['$scope', function($scope) {

$scope.list = [];

$scope.text = 'Welcome to JavaTpoint!';

$scope.submit = function() {

if ($scope.text) {

$scope.list.push(this.text);

$scope.text = '';

}

};

}]);

</script>

<form ng-submit="submit()" ng-controller="ExampleController">

Enter text and hit enter:

<input type="text" ng-model="text" name="text" />

<input type="submit" id="submit" value="Submit" />

<pre>list={{list}}</pre>

</form>

</body>

</html>

## 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:

### Example

<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
* Attribute
* Class
* Comment

The examples below will all produce the same result:

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.

### Example

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.

## How to add directives

**See this example:**

<!DOCTYPE html**>**

**<html>**

**<script** src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"**></script>**

**<body>**

**<div** ng-app="myApp" w3-test-directive**></div>**

**<script>**

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

app.directive("w3TestDirective", function() {

    return {

        template : "This is a directive constructor. "

    };

});

**</script>**

**</body>**

**</html>**

Following is a list of AngularJS directives:

|  |  |
| --- | --- |
| **Directive** | **Description** |
| [ng-app](https://www.javatpoint.com/angularjs-ng-app-directive) | It defines the root element of an application. |
| [ng-bind](https://www.javatpoint.com/angularjs-ng-bind-directive) | It binds the content of an html element to application data. |
| [ng-bind-html](https://www.javatpoint.com/angularjs-ng-bind-html-directive) | It binds the inner HTML of an HTML element to application data, and also removes dangerous code from the html string. |
| [ng-bind-template](https://www.javatpoint.com/angularjs-ng-bind-template-driective) | It specifies that the text content should be replaced with a template. |
| [ng-blur](https://www.javatpoint.com/angularjs-ng-blur-directive) | It specifies a behavior on blur events. |
| [ng-change](https://www.javatpoint.com/angularjs-ng-change-directive) | It specifies an expression to evaluate when content is being changed by the user. |
| [ng-checked](https://www.javatpoint.com/angularjs-ng-checked-directive) | It specifies if an element is checked or not. |
| [ng-class](https://www.javatpoint.com/angularjs-ng-class-directive) | It specifies css classes on html elements. |
| [ng-class-even](https://www.javatpoint.com/angularjs-ng-class-even-directive) | It is same as ng-class, but will only take effect on even rows. |
| [ng-class-odd](https://www.javatpoint.com/angularjs-ng-class-odd-directive) | It is same as ng-class, but will only take effect on odd rows. |
| [ng-click](https://www.javatpoint.com/angularjs-ng-click-directive) | It specifies an expression to evaluate when an element is being clicked. |
| [ng-cloak](https://www.javatpoint.com/angularjs-ng-cloak-directive) | It prevents flickering when your application is being loaded. |
| [ng-controller](https://www.javatpoint.com/angularjs-ng-controller-directive) | It defines the controller object for an application. |
| [ng-copy](https://www.javatpoint.com/angularjs-ng-copy-directive) | It specifies a behavior on copy events. |
| [ng-csp](https://www.javatpoint.com/angularjs-ng-csp-directive) | It changes the content security policy. |
| [ng-cut](https://www.javatpoint.com/angularjs-ng-cut-directive) | It specifies a behavior on cut events. |
| [ng-dblclick](https://www.javatpoint.com/angularjs-ng-dblclick-directive) | It specifies a behavior on double-click events. |
| [ng-focus](https://www.javatpoint.com/angularjs-ng-focus-directive) | It specifies a behavior on focus events. |
| [ng-hide](https://www.javatpoint.com/angularjs-ng-hide-directive) | It hides or shows html elements. |
| [ng-href](https://www.javatpoint.com/angularjs-ng-href-directive) | It specifies a URL for the <a> element. |
| [ng-if](https://www.javatpoint.com/angularjs-ng-if-directive) | It removes the html element if a condition is false. |
| [ng-include](https://www.javatpoint.com/angularjs-ng-include-directive) | It includes html in an application. |
| [ng-init](https://www.javatpoint.com/angularjs-ng-init-directive) | It defines initial values for an application. |
| [ng-jq](https://www.javatpoint.com/angularjs-ng-jq-directive) | It specifies that the application must use a library, like jQuery. |
| [ng-keydown](https://www.javatpoint.com/angularjs-ng-keydown-directive) | It specifies a behavior on keydown events. |
| [ng-keypress](https://www.javatpoint.com/angularjs-ng-keypress-directive) | It specifies a behavior on keypress events. |
| [ng-keyup](https://www.javatpoint.com/angularjs-ng-keyup-directive) | It specifies a behavior on keyup events. |
| [ng-list](https://www.javatpoint.com/angularjs-ng-list-directive) | It converts text into a list (array). |
| [ng-open](https://www.javatpoint.com/angularjs-ng-open-directive) | It specifies the open attribute of an element. |
| [ng-options](https://www.javatpoint.com/angularjs-ng-options-directive) | It specifies <options> in a <select> list. |
| [ng-paste](https://www.javatpoint.com/angularjs-ng-paste-directive) | It specifies a behavior on paste events. |
| [ng-pluralize](https://www.javatpoint.com/angularjs-ng-pluralize-directive) | It specifies a message to display according to en-us localization rules. |
| [ng-readonly](https://www.javatpoint.com/angularjs-ng-readonly-directive) | It specifies the readonly attribute of an element. |
| [ng-required](https://www.javatpoint.com/angularjs-ng-required-directive) | It specifies the required attribute of an element. |
| [ng-selected](https://www.javatpoint.com/angularjs-ng-selected-directive) | It specifies the selected attribute of an element. |
| [ng-show](https://www.javatpoint.com/angularjs-ng-show-directive) | It shows or hides html elements. |
| [ng-src](https://www.javatpoint.com/angularjs-ng-src-directive) | It specifies the src attribute for the <img> element. |
| [ng-srcset](https://www.javatpoint.com/angularjs-ng-srcset-directive) | It specifies the srcset attribute for the <img> element. |
| [ng-style](https://www.javatpoint.com/angularjs-ng-style-directive) | It specifies the style attribute for an element. |
| [ng-submit](https://www.javatpoint.com/angularjs-ng-submit-directive) | It specifies expressions to run on onsubmit events. |
| [ng-switch](https://www.javatpoint.com/angularjs-ng-switch-directive) | It specifies a condition that will be used to show/hide child elements. |
| [ng-transclude](https://www.javatpoint.com/angularjs-ng-transclude-directive) | It specifies a point to insert transcluded elements. |
| [ng-value](https://www.javatpoint.com/angularjs-ng-value-directive) | It specifies the value of an input element. |
| [ng-disabled](https://www.javatpoint.com/angularjs-directives) | It specifies if an element is disabled or not. |
| [ng-form](https://www.javatpoint.com/angularjs-directives) | It specifies an html form to inherit controls from. |
| [ng-model](https://www.javatpoint.com/angularjs-directives) | It binds the value of html controls to application data. |
| [ng-model-options](https://www.javatpoint.com/angularjs-directives) | It specifies how updates in the model are done. |
| [ng-mousedown](https://www.javatpoint.com/angularjs-directives) | It specifies a behavior on mousedown events. |
| [ng-mouseenter](https://www.javatpoint.com/angularjs-directives) | It specifies a behavior on mouseenter events. |
| [ng-mouseleave](https://www.javatpoint.com/angularjs-directives) | It specifies a behavior on mouseleave events. |
| [ng-mousemove](https://www.javatpoint.com/angularjs-directives) | It specifies a behavior on mousemove events. |
| [ng-mouseover](https://www.javatpoint.com/angularjs-directives) | It specifies a behavior on mouseover events. |
| [ng-mouseup](https://www.javatpoint.com/angularjs-directives) | It specifies a behavior on mouseup events. |
| [ng-non-bindable](https://www.javatpoint.com/angularjs-directives) | It specifies that no data binding can happen in this element, or it's children. |
| [ng-repeat](https://www.javatpoint.com/angularjs-directives) | It defines a template for each data in a collection. |

# AngularJS Filters

Filters can be added in AngularJS to format data.

AngularJS provides filters to transform data:

* 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:

**Ckkajs35.html**

 <!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body>**

**<div** ng-app="myApp" ng-controller="personCtrl"**>**

**<p>**The name is {{ firstName | uppercase }}**</p>**

**<p>**The name is {{ lastName | lowercase }}**</p>**

**</div>**

**<script>**

angular.module('myApp', []).controller('personCtrl', function($scope) {

    $scope.firstName = "Prof. CK Kumbharana",

    $scope.lastName = "God is great"

});

**</script>**

**</body>**

**</html>**

## Adding Filters to Directives

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

**Ckkajs36.html**

 <!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body>**

**<div** ng-app="myApp" ng-controller="namesCtrl"**>**

**<p>**Looping with objects:**</p>**

**<ul>**

**<li** ng-repeat="x in names | orderBy:'country'"**>**

    {{ x.name + ', ' + x.country }}

**</li>**

**</ul>**

**</div>**

**<script>**

angular.module('myApp', []).controller('namesCtrl', function($scope) {

    $scope.names = [

        {name:'Ramesh',country:'India'},

        {name:'Alex',country:'USA'},

        {name:'Pooja',country:'India'},

        {name:'Maitri',country:'India'},

        {name:'Ixita',country:'India'},

        {name:'Ramanujam',country:'India'},

        {name:'Osama',country:'Iraq'},

        {name:'Johnson',country:'UK'},

        {name:'Karl',country:'Russia'}

        ];

});

**</script>**

**</body>**

**</html>**

## The currency Filter

The currency filter formats a number as currency:

**Ckkajs37.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="personCtrl">

<p>The price is {{ price | currency }}</p>

</div>

<script>

angular.module('myApp', []).controller('personCtrl', function($scope) {

$scope.price="205";

});

</script>

</body>

</html>

## 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.

**Ckkajs38.html**

<!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body>**

**<div** ng-app="myApp" ng-controller="namesCtrl"**>**

**<ul>**

**<li** ng-repeat="x in names | filter : 'o'"**>**

    {{ x }}

**</li>**

**</ul>**

**</div>**

**<script>**

angular.module('myApp', []).controller('namesCtrl', function($scope) {

    $scope.names = [

'Ramesh',

'Pooja',

'Mahesh',

'Ramanujam',

'Osama',

'Iqbal',

'Karl',

'Johnson',

'Alex'

    ];

});

**</script>**

**<p>**This example displays only the names containing the letter "o".**</p>**

**</body>**

**</html>**

## Filter an array based on user input

You can use the value of the input field as an expression in a filter by setting the ng-model directive on an input field.

**Ckkajs39.html**

<!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body>**

**<div** ng-app="myApp" ng-controller="namesCtrl"**>**

**<p>**Type a letter in the input field:**</p>**

**<p><input** type="text" ng-model="test"**></p>**

**<ul>**

**<li** ng-repeat="x in names | filter:test"**>**

    {{ x }}

**</li>**

**</ul>**

**</div>**

**<script>**

angular.module('myApp', []).controller('namesCtrl', function($scope) {

    $scope.names = [

        'Ramesh',

'Pooja',

'Mahesh',

'Ramanujam',

'Osama',

'Iqbal',

'Karl',

'Johnson',

'Alex'

   ];

});

**</script>**

**<p>**The list will only contain the names matching the filter.**</p>**

**</body>**

**</html>**

## Sort an array based on user input

You can sort an array according to the table columns.

**Ckkajs40.html**

<!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body>**

**<p>**Click the table headers to change the sorting order:**</p>**

**<div** ng-app="myApp" ng-controller="namesCtrl"**>**

**<table** border="1" width="100%"**>**

**<tr>**

**<th** ng-click="orderByMe('name')"**>**Name**</th>**

**<th** ng-click="orderByMe('country')"**>**Country**</th>**

**</tr>**

**<tr** ng-repeat="x in names | orderBy:myOrderBy"**>**

**<td>**{{x.name}}**</td>**

**<td>**{{x.country}}**</td>**

**</tr>**

**</table>**

**</div>**

**<script>**

angular.module('myApp', []).controller('namesCtrl', function($scope) {

       $scope.names = [

        {name:'Ramesh',country:'India'},

        {name:'Alex',country:'USA'},

        {name:'Pooja',country:'India'},

        {name:'Maitri',country:'India'},

        {name:'Ixita',country:'India'},

        {name:'Ramanujam',country:'India'},

        {name:'Osama',country:'Iraq'},

        {name:'Johnson',country:'UK'},

        {name:'Karl',country:'Russia'}

        ];

    $scope.orderByMe = function(x) {

        $scope.myOrderBy = x;

    }

});

**</script>**

**</body>**

**</html>**

## AngularJS Custom Filters

You can create your own filters by register a new filter factory function with your module. Make a custom filter called "myFormat":

**Ckkajs41.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body>

<p>It is a custom filter, to change the alternate charaters to uppercase:</p>

<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>

</body>

</html>

The myFormat filter will format every other character to uppercase.

# AngularJS Filters

In AngularJS, filters are used to format data. Following is a list of filters used for transforming data.

|  |  |
| --- | --- |
| **Filter** | **Description** |
| [Currency](https://www.javatpoint.com/angularjs-currency-filter) | It formats a number to a currency format. |
| [Date](https://www.javatpoint.com/angularjs-date-filter) | It formats a date to a specified format. |
| [Filter](https://www.javatpoint.com/angularjs-filter-filters) | It select a subset of items from an array. |
| [Json](https://www.javatpoint.com/angularjs-json-filter) | It formats an object to a Json string. |
| [Limit](https://www.javatpoint.com/angularjs-limit-to-filter) | It is used to limit an array/string, into a specified number of elements/characters. |
| [Lowercase](https://www.javatpoint.com/angularjs-lower-case-filter) | It formats a string to lower case. |
| [Number](https://www.javatpoint.com/angularjs-number-filter) | It formats a number to a string. |
| [Orderby](https://www.javatpoint.com/angularjs-order-by-filter) | It orders an array by an expression. |
| [Uppercase](https://www.javatpoint.com/angularjs-uppercase-filter) | It formats a string to upper case. |

# AngularJS Scope

* The scope is the binding part between the HTML (view) and the JavaScript (controller).
* The scope is an object with the available properties and methods.
* The scope is available for both the view and the controller.

## 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>

When adding properties to the $scope object in the controller, the view (HTML) gets access to these properties. In the view, you do not use the prefix $scope, you just refer to a propertyname, like {{carname}}.

## 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.

### Example

If you make changes in the view, the model and the controller will be updated:

<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.

### Example

When dealing with the ng-repeat directive, each repetition has access to the current repetition object:

<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>

Each <li> element has access to the current repetition object, in this case a string, which is referred to by using x.

## 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.

### Example

A variable named "color" exists in both the controller's scope and in the rootScope:

<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 Forms

AngularJS facilitates you to create a form enriches with data binding and validation of input controls. Input controls are ways for a user to enter data. A form is a collection of controls for the purpose of grouping related controls together. Following are the input controls used in AngularJS forms:

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

AngularJS provides multiple events that can be associated with the HTML controls. These events are associated with the different HTML input elements. Following is a list of events supported in AngularJS:

* ng-click
* ng-dbl-click
* ng-mousedown
* ng-mouseup
* ng-mouseenter
* ng-mouseleave
* ng-mousemove
* ng-mouseover
* ng-keydown
* ng-keyup
* ng-keypress
* ng-change

## Data Binding: ng-model directive is used to provide data binding.Let's take an example where ng-model directive binds the input controller to the rest of your application

**Ckkajs42.html**

<!DOCTYPE html**>**

**<html** lang="en"**>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body>**

**<div** ng-app="myApp" ng-controller="formCtrl"**>**

**<form>**

    First Name: **<input** type="text" ng-model="firstname"**>**

**</form>**

**</div>**

**<script>**

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

app.controller('formCtrl', function($scope) {

    $scope.firstname = "God is graet";

});

**</script>**

**</body>**

**</html>**

You can also change the example in the following way:

**Ckkajs43.html**

<!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body>**

**<div** ng-app=""**>**

**<form>**

    First Name: **<input** type="text" ng-model="firstname"**>**

**</form>**

**<h2>**You entered: {{firstname}}**</h2>**

**</div>**

**<p>**Change the name inside the input field, and you will see the name in the header changes accordingly.**</p>**

**</body>**

**</html>**

## AngularJS Checkbox

A checkbox has a value true or false. The ng-model directive is used for a checkbox.

**Ckkajs44.html**

<!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body>**

**<div** ng-app=""**>**

**<form>**

    Check to show this:

**<input** type="checkbox" ng-model="myVar"**>**

**</form>**

**<h1** ng-show="myVar"**>**Checked**</h1>**

**</div>**

**<p>**The ng-show attribute is set to true when the checkbox is checked.**</p>**

**</body>**

**</html>**

## AngularJS Radio Buttons

ng-model directive is used to bind radio buttons in your applications. Let's take an example to display some text, based on the value of the selected radio buttons. In this example, we are also using ng-switch directive to hide and show HTML sections depending on the value of the radio buttons.

**Ckkajs45.html**

<!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body** ng-app=""**>**

**<form>**

  Pick a topic:

**<input** type="radio" ng-model="myVar" value="tuts"**>**Tutorials

**<input** type="radio" ng-model="myVar" value="fest"**>**Festivals

**<input** type="radio" ng-model="myVar" value="news"**>**News

**</form>**

**<div** ng-switch="myVar"**>**

**<div** ng-switch-when="tuts"**>**

**<h1>**Tutorials**</h1>**

**<p>**Welcome to the best tutorials over the net**</p>**

**</div>**

**<div** ng-switch-when="fest"**>**

**<h1>**Festivals**</h1>**

**<p>**Most famous festivals**</p>**

**</div>**

**<div** ng-switch-when="news"**>**

**<h1>**News**</h1>**

**<p>**Welcome to the news portal.**</p>**

**</div>**

**</div>**

**<p>**The ng-switch directive hides and shows HTML sections depending on the value of the radio buttons.**</p>**

**</body>**

**</html>**

## AngularJS Selectbox

ng-model directive is used to bind select boxes to your application.

**Ckkajs46.html**

<!DOCTYPE html**>**

**<html>**

<script src="c:\angular-1.6.6\angular.min.js"></script>

**<body** ng-app=""**>**

**<form>**

  Select a topic:

**<select** ng-model="myVar"**>**

**<option** value=""**>**

**<option** value="tuts"**>**Tutorials

**<option** value="fest"**>**Festivals

**<option** value="news"**>**News

**</select>**

**</form>**

**<div** ng-switch="myVar"**>**

**<div** ng-switch-when="tuts"**>**

**<h1>**Tutorials**</h1>**

**<p>**Welcome to the best tutorials over the net.**</p>**

**</div>**

**<div** ng-switch-when="fest"**>**

**<h1>**Festivals**</h1>**

**<p>**Most famous festivals.**</p>**

**</div>**

**<div** ng-switch-when="news"**>**

**<h1>**News**</h1>**

**<p>**Welcome to the news portal.**</p>**

**</div>**

**</div>**

**<p>**The ng-switch directive hides and shows HTML sections depending on the value of the radio buttons.**</p>**

**</body>**

**</html>**

# AngularJS Form Validation

AngularJS provides client-side form validation. It checks the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state. It also holds the information about whether the input fields have been touched, or modified, or not. Following directives are generally used to track errors in an AngularJS form:

* **$dirty** - states that value has been changed.
* **$invalid** - states that value entered is invalid.
* **$error** - states the exact error.

## Ckkajs47.html

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body ng-app="">

<p>Try writing in the input field:</p>

<form name="myForm">

<input name="myInput" ng-model="myInput" required>

</form>

<p>The input's valid state is:</p>

<h1>{{myForm.myInput.$valid}}</h1>

</body>

</html>

**Ckkajs48.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body ng-app="">

<p>Input your valid mail:</p>

<form name="myForm">

<input name="myInput" type="email" ng-model="myInput" required>

</form>

<p>The input's valid state is:</p>

<h1>{{myForm.myInput.$valid}}</h1>

</body>

</html>

## 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 The field has not been modified yet
* $dirty The field has been modified
* $invalid The field content is not valid
* $valid The field content is valid

They are all properties of the input field, and are either true or false.

## Ckkajs49.html

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<style>

input.ng-invalid {

background-color:pink;

}

input.ng-valid {

background-color:lightgreen;

}

</style>

<body ng-app="">

<p>Try writing in the input field:</p>

<form name="myForm">

<input name="myName" ng-model="myName" required>

</form>

<p>The input field requires content, and will therefore become green when you write in it.</p>

</body>

</html>

Ckkajs50.html

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<style>

form.ng-pristine {

background-color:lightblue;

}

form.ng-dirty {

background-color:pink;

}

</style>

<body ng-app="">

<form name="myForm">

<p>Try writing in the input field:</p>

<input name="myName" ng-model="myName" required>

<p>The form gets a "ng-dirty" class when the form has been modified, and will therefore turn pink.</p>

</form>

</body>

</html>

**Ckkajs51.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js">

</script>

<body ng-app="">

<p>Try leaving the first input field blank:</p>

<form name="myForm">

<p>Name:

<input name="myName" ng-model="myName" required>

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

</p>

<p>Address:

<input name="myAddress" ng-model="myAddress" required>

</p>

</form>

<p>We use the ng-show directive to only show the error message if the field has been touched AND is empty.</p>

</body>

</html>

**Ckkajs52.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js">

</script>

<body ng-app="">

<p>Try leaving the first input field blank:</p>

<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>

<p>We use the ng-show directive to only show the error message if the field has been touched AND is empty.</p>

</body>

</html>

**Ckkajs53.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js">

</script>

<body ng-app="">

<p>Try leaving the first input field blank:</p>

<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>

<p>We use the ng-show directive to only show the error message if the field has been touched AND is empty.</p>

</body>

</html>

**Ckkajs54.html**

# <!DOCTYPE html>

# <html>

# <script src="c:\angular-1.6.6\angular.min.js">

# </script>

# <body>

# <form ng-app="myApp" ng-controller="validateCtrl" name="myForm" novalidate>

# <h2>Validation Example</h2>

# <p>Username:<br>

# <input type="text" name="user" ng-model="user" required>

# <span style="color:red" ng-show="myForm.user.$dirty && myForm.user.$invalid">

# <span ng-show="myForm.user.$error.required">Username is required.</span>

# </span>

# </p>

# <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 = 'Prof. CK Kumbharana';

# $scope.email = 'ckkumbharana@yahoo.com';

# });

# </script>

# </body>

# </html>

**Ckkajs55.html**

<!DOCTYPE html>

<html>

<head>

<title>Angular JS Forms</title>

# <script src="c:\angular-1.6.6\angular.min.js">

</script>

<style>

table, th , td {

border: 1px solid grey;

border-collapse: collapse;

padding: 5px;

}

table tr:nth-child(odd) {

background-color: lightpink;

}

table tr:nth-child(even) {

background-color: lightyellow;

}

</style>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app = "mainApp" ng-controller = "studentController">

<form name = "studentForm" novalidate>

<table border = "0">

<tr>

<td>Enter first name:</td>

<td><input name = "firstname" type = "text" ng-model = "firstName" required>

<span style = "color:red" ng-show = "studentForm.firstname.$dirty && studentForm.firstname.$invalid">

<span ng-show = "studentForm.firstname.$error.required">First Name is required.</span>

</span>

</td>

</tr>

<tr>

<td>Enter last name: </td>

<td><input name = "lastname" type = "text" ng-model = "lastName" required>

<span style = "color:red" ng-show = "studentForm.lastname.$dirty && studentForm.lastname.$invalid">

<span ng-show = "studentForm.lastname.$error.required">Last Name is required.</span>

</span>

</td>

</tr>

<tr>

<td>Email: </td><td><input name = "email" type = "email" ng-model = "email" length = "100" required>

<span style = "color:red" ng-show = "studentForm.email.$dirty && studentForm.email.$invalid">

<span ng-show = "studentForm.email.$error.required">Email is required.</span>

<span ng-show = "studentForm.email.$error.email">Invalid email address.</span>

</span>

</td>

</tr>

<tr>

<td>

<button ng-click = "reset()">Reset</button>

</td>

<td>

<button ng-disabled = "studentForm.firstname.$dirty &&

studentForm.firstname.$invalid || studentForm.lastname.$dirty &&

studentForm.lastname.$invalid || studentForm.email.$dirty &&

studentForm.email.$invalid" ng-click="submit()">Submit</button>

</td>

</tr>

</table>

</form>

</div>

<script>

var mainApp = angular.module("mainApp", []);

mainApp.controller('studentController', function($scope) {

$scope.reset = function(){

$scope.firstName = "Prof. CK";

$scope.lastName = "Kumbharana";

$scope.email = "ckkumbharana@yahoo.com";

}

$scope.reset();

});

</script>

</body>

</html>

# AngularJS Services

In AngularJS you can make your own service, or use one of the many built-in services. 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:

**Ckkajs56.html**

<!DOCTYPE html>

<html>

# <script src="c:\angular-1.6.6\angular.min.js">

</script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>The url of this page is:</p>

<h3>{{myUrl}}</h3>

</div>

<p>This example uses the built-in $location service to get the absolute url of the page.</p>

<script>

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

app.controller('myCtrl', function($scope, $location) {

$scope.myUrl = $location.absUrl();

});

</script>

</body>

</html>

Note that the $location service is passed in to the controller as an argument. In order to use the service in the controller, it must be defined as a dependency.

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.

**Ckkajs57.html**

<!DOCTYPE html>

<html>

# <script src="c:\angular-1.6.6\angular.min.js">

</script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>Today's welcome message is:</p>

<h1>{{myWelcome}}</h1>

</div>

<p>The $http service requests a page on the server, and the response is set as the value of the "myWelcome" variable.</p>

<script>

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

app.controller('myCtrl', function($scope, $http) {

$http.get("welcome.htm").then(function (response) {

$scope.myWelcome = response.data;

});

});

</script>

</body>

</html>

## The $timeout Service

The $timeout service is AngularJS' version of the window.setTimeout function.

**Ckkajs58.html**

<!DOCTYPE html>

<html>

# <script src="c:\angular-1.6.6\angular.min.js">

</script>

## <body>

## <div ng-app="myApp" ng-controller="myCtrl">

## <p>This header will change after two seconds:</p>

## <h1>{{myHeader}}</h1>

## </div>

## <p>The $timeout service runs a function after a specified number of milliseconds.</p>

## <script>

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

## app.controller('myCtrl', function($scope, $timeout) {

## $scope.myHeader = "Hello World!";

## $timeout(function () {

## $scope.myHeader = "How are you today?";

## }, 2000);

## });

## </script>

## </body>

## </html>

## The $interval Service

The $interval service is AngularJS' version of the window.setInterval function.

**Ckkajs59.html**

<!DOCTYPE html>

<html>

# <script src="c:\angular-1.6.6\angular.min.js">

</script>

## <body>

## <div ng-app="myApp" ng-controller="myCtrl">

## <p>The time is:</p>

## <h1>{{theTime}}</h1>

## </div>

## <p>The $interval service runs a function every specified millisecond.</p>

## <script>

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

## app.controller('myCtrl', function($scope, $interval) {

## $scope.theTime = new Date().toLocaleTimeString();

## $interval(function () {

## $scope.theTime = new Date().toLocaleTimeString();

## }, 1000);

## });

## </script>

## </body>

## </html>

## Create Your Own Service

To create your own service, connect your service to the module:

**Ckkajs60.html**

<!DOCTYPE html>

<html>

# <script src="c:\angular-1.6.6\angular.min.js">

</script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>The hexadecimal value of 255 is:</p>

<h1>{{hex}}</h1>

</div>

<p>A custom service with a method that converts a given number into a hexadecimal number.</p>

<script>

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

app.service('hexafy', function() {

this.myFunc = function (x) {

return x.toString(16);

}

});

app.controller('myCtrl', function($scope, hexafy) {

$scope.hex = hexafy.myFunc(255);

});

</script>

</body>

</html>

AngularJS supports the concepts of "Separation of Concerns" using services architecture. Services are javascript functions and are responsible to do a specific tasks only. This makes them an individual entity which is maintainable and testable. Controllers, filters can call them as on requirement basis. Services are normally injected using dependency injection mechanism of AngularJS.

AngularJS provides many inbuilt services for example, $https:, $route, $window, $location etc. Each service is responsible for a specific task for example, $https: is used to make ajax call to get the server data. $route is used to define the routing information and so on. Inbuilt services are always prefixed with $ symbol.

There are two ways to create a service.

* factory
* service

**Ckkajs61.html**

<!DOCTYPE html>

<html>

<head>

<title>Angular JS Services</title>

<script src="c:\angular-1.6.6\angular.min.js"></script>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app = "mainApp" ng-controller = "CalcController">

<p>Enter a number: <input type = "number" ng-model = "number" /></p>

<button ng-click = "square()">X<sup>2</sup></button>

<p>Result: {{result}}</p>

</div>

<script>

var mainApp = angular.module("mainApp", []);

mainApp.factory('MathService', function() {

var factory = {};

factory.multiply = function(a, b) {

return a \* b

}

return factory;

});

mainApp.service('CalcService', function(MathService){

this.square = function(a) {

return MathService.multiply(a,a);

}

});

mainApp.controller('CalcController', function($scope, CalcService) {

$scope.square = function() {

$scope.result = CalcService.square($scope.number);

}

});

</script>

</body>

</html>

**Angular JS TO-DO applications**

**Ckkajs62.html**

<!DOCTYPE html>

<html>

<script src="c:\angular-1.6.6\angular.min.js"></script>

<body ng-app="myApp" ng-controller="todoCtrl">

<h2>My Todo List</h2>

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

<input type="text" ng-model="todoInput" size="50" placeholder="Add New">

<input type="submit" value="Add New">

</form>

<br>

<div ng-repeat="x in todoList">

<input type="checkbox" ng-model="x.done"> <span ng-bind="x.todoText"></span>

</div>

<p><button ng-click="remove()">Remove marked</button></p>

<script>

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

app.controller('todoCtrl', function($scope) {

$scope.todoList = [{todoText:'Clean House', done:false}];

$scope.todoAdd = function() {

$scope.todoList.push({todoText:$scope.todoInput, done:false});

$scope.todoInput = "";

};

$scope.remove = function() {

var oldList = $scope.todoList;

$scope.todoList = [];

angular.forEach(oldList, function(x) {

if (!x.done) $scope.todoList.push(x);

});

};

});

</script>

</body>

</html>

* **Config() and Run()**

**There are two important blocks in** **AngularJS**, which is not popular but we mostly use in our Angular Application. Both are used when we defined our ng-app with angular. module in our **javascript file**.

In **Angular application** most of **objects** **are** **instantiated** **and** **wired** **together automatically by the** **injector service**. **We can use five types of recipe with injector or we can say injector identify only five type of recipe to create any object**.  **This block is used to prevent accidental instantiation of service before they have been fully configured.**  
  
**We can use** **config()** **block to inject only** **providers** **and** **constants** **in ourAngularJS application**. This block executed in **AngularJS application** during the**provider registrations and configuration phase**. Now we have several types of providers in our **AngularJS application**. But if we want to use these provider first**we need to inject these** **providers** **in our application with the help of** **config()block.**

**The** **providers** **basically create new** **instances, but only once for each provider.**  
  
**Look at the following provider, which we can use in our application:**

**1.) $httpProvider**

**2.) $locationProvider**

**3.) $routeProvider**

**4.) $provider**

**5.) User defined Provider**

**6.) Constant**

In the above discussion we know the **importance of** **config()** **block and provider**but here we have an important question- "**how we use config() block in our application?**"  
  
**Look at the following code:**

angular.module('crazydeveloperModule', [])

.config(['$httpProvider','$locationProvider','$routeProvider','$provider','cookieProvider','constant',

function ($httpProvider,$locationProvider,$routeProvider,$provider,cookieProvider,constant) {

// provider-injector

//This is an example of config block.

// You can have as many of these as you want.

// You can only inject Providers (not instances)

// into config blocks.

}])

In the above code**we can see the way we can use config() block in our application.** Now move ahead to next point,**here we have run() block. We know that in config() block can't inject any instances. run() executed after theconfig() block and here we have facility with this block to inject any instance and constants in our application. This block is just like main() method in other language.**This block is a great place to**put any event handlers that we need to be executed at the root level for the application. For example, authentication handlers.**  
Again we have an important question- "**how we use run() block in our application?**"  
  
**Look at the following code:**

angular.module('myModule', [])

. run(['$rootscope','security','pageFactory','$httpBackend',

function ($rootscope,security,pageFactory,$httpBackend) {

// instance-injector

// This is an example of a run block.

// You can have as many of these as you want.

// You can only inject instances (not Providers)

// into run blocks

}]);

After all of the above discussion we can say **"An Angular Module useConfiguration and Run blocks to inject dependencies (like providers, services and constants) which get applied to the angular app during the bootstrap process."**

Configuration block – This block is executed during the provider registration and configuration phase. We can have these as many as we want in the application. Only providers and constantscan be injected into configuration blocks and not instances. This block is used to inject module wise configuration settings to prevent accidental instantiation of services before they have been fully configured. This block is created using config() method.

angular.module(myModule, []).   
config(function (injectables)   
{   
  
//.....  
  
}).

Run block – This is executed after the execution of configuration block. It inserts instances andconstants but not providers. This block is created using run() method. This is similar to main method in C# or Java

run(function (injectables)   
{   
  
//....  
  
});