

## Angularjs

**AngularJS version 1.0 was released in 2012.**

Miško Hevery, a Google employee, started to work with AngularJS in 2009.

The idea turned out very well, and the project is now officially supported by Google.

**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 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, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

```
<!DOCTYPE html>
```

```
<html>
```

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/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>
```

## Example explained:

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 content of the <p> element to the application variable **name**.

## 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>
```

OR

```
<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 are written inside double braces: **{{ expression }}**.

## 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>
<br>
Full Name: {{ firstName + " " + lastName }}
</div>
<script>
var app = angular.module('myApp', []); //AngularJS Module
//AngularJS Controller
app.controller('myCtrl', function($scope) {
    $scope.firstName= "John";
    $scope.lastName= "Doe";
});
</script>
```

## 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. Example **{{ 5 + 5 }}** or **{{ firstName + " " + lastName }}** If you remove the **ng-app** directive, HTML will display the expression as it is, without solving it.

## 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.

A module is created by using the AngularJS function **angular.module**

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

**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

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. AngularJS will invoke the controller with a **\$scope** object. In AngularJS, **\$scope** is the application object (the owner of application variables and functions). Add a controller to your application, and refer to the controller with the **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>
```

```
    };  
});  
</script>
```

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.

## 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.

The ng-app directive also tells AngularJS that the <div> element is the "owner" of the AngularJS application.

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.

## 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
- Attribute
- Class
- Comment

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

With the ng-model directive you can bind the value of an input field to a variable created in AngularJS. The binding goes both ways. If the user changes the value inside the input field, the AngularJS property will also change its value.

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 *model* is updated as well. This happens immediately and automatically, which makes sure that the model and the view is updated at all times.

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {

    $scope.name = "John Doe";

});
```

## 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>
```

In the example above, the span will be displayed only if the expression in the ng-show attribute returns true.

## 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>
```

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

### 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.

When you make a controller in AngularJS, you pass the \$scope object as an argument.

```
app.controller('myCtrl', function($scope) {
    $scope.carname = "Volvo";
});
```

### 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.

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.

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>

```

**o/p:** The rootScope's favorite color: **blue**  
The scope of the controller's favorite color: **red**

The rootScope's favorite color is still: **blue**

Notice that controller's color variable does not overwrite the `rootScope`'s color value.

## Filter:



Filters can be added to expressions by using the pipe character |, followed by a filter.

```
<p>The name is {{ lastName | uppercase }}</p>
```

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

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

### 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.

```
<li ng-repeat="x in names | filter : 'i'"> {{ x }} </li>
```

This example displays only the names containing the letter "i".

### Custom Filters

You can make your own filters by registering a new filter factory function with your module:

Make a custom filter called "myFormat":

```
<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>
```

The myFormat filter will format every other character to uppercase.

## 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();
});
```

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.

## 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.

Ex: 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);
});
```

## 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 the custom made service named hexafy to convert a number into a hexadecimal number:

```
app.controller('myCtrl', function($scope, hexafy) {
  $scope.hex = hexafy.myFunc(255);
});
```

## 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:

Example: The service hexafy used in the filter myFormat:

```
app.filter('myFormat',['hexafy', function(hexafy) {
  return function(x) {
    return hexafy.myFunc(x);
  };
}]);
```

## AngularJS AJAX - \$http:

**\$http** is an AngularJS service for reading data from remote servers.

The AngularJS \$http service makes a request to the server, and returns a response.

Example: 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

The example above uses the .get method of the \$http service.

The .get method is a shortcut method of the \$http service. There are several shortcut methods:

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

The methods above are all shortcuts of calling the \$http service:

```
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;
});
});
```

The example above executes the \$http service with an object as an argument. The object is specifying the HTTP method, the url, what to do on success, and what to do on failure.

### 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.

```
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;
  });
});
```

### 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>
```

## AngularJS Select Boxes:

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

Example:

```
$scope.names = ["Emil", "Tobias", "Linus"];
```

```
<select ng-model="selectedName" ng-options="x for x in names">
```

OR

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

Because the ng-repeat directive repeats a block of HTML code for each item in an array, it can be used to create options in a dropdown list, but the ng-options directive was made especially for filling a dropdown list with options, and has at least one important advantage:

Dropdowns made with ng-options allows the selected value to be an **object**, while dropdowns made from ng-repeat has to be a string.

Example2:

```
$scope.cars = [  
  {model : "Ford Mustang", color : "red"},  
  {model : "Fiat 500", color : "white"},  
  {model : "Volvo XC90", color : "black"}  
];
```

**The ng-repeat directive has its limitations, the selected value must be a string.**

```
<select ng-model="selectedCar">  
<option ng-repeat="x in cars" value="{{x.model}}">{{x.model}}</option>  
</select>
```

When using the ng-options directive, the selected value can be an object:

```
<select ng-model="selectedCar" ng-options="x.model for x in cars">  
</select>
```

```
<h1>You selected: {{selectedCar.model}}</h1>
```

```
<p>Its color is: {{selectedCar.color}}</p>
```

When the selected value can be an object, it can hold more information, and your application can be more flexible.

Example3:

```
$scope.cars = {  
  car01 : "Ford",
```

```
car02 : "Fiat",  
car03 : "Volvo"  
};
```

Using an object as the data source, x represents the key, and y represents the value:

```
<select ng-model="selectedCar" ng-options="x for (x, y) in cars">  
</select>  
<h1>You selected: {{selectedCar}}</h1>
```

The selected value will always be the **value** in a key-**value** pair.

The **value** in a key-**value** pair can also be an object:

Example4:

The selected value will still be the **value** in a key-**value** pair, only this time it is an object:

```
$scope.cars = {  
car01 : {brand : "Ford", model : "Mustang", color : "red"},  
car02 : {brand : "Fiat", model : "500", color : "white"},  
car03 : {brand : "Volvo", model : "XC90", color : "black"}  
};
```

The options in the dropdown list does not have to be the **key** in a **key**-value pair, it can also be the value, or a property of the value object:

```
<select ng-model="selectedCar" ng-options="y.model for (x, y) in cars">  
</select>  
<h1>You selected: {{selectedCar.brand}}</h1>  
<h2>Model: {{selectedCar.model}}</h2>  
<h3>Color: {{selectedCar.color}}</h3>
```

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

The **ng-show** directive shows or hides an HTML element. The ng-show directive shows (or hides) an HTML element based on the **value** of ng-show.

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

## 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:

```
$scope.showMe = false;
$scope.myFunc = function() {
  $scope.showMe = !$scope.showMe;
}
```

## **\$event Object**

You can pass the \$event object as an argument when calling the function.

The \$event object contains the browser's event object:

```
<h1 ng-mousemove="myFunc($event)">Mouse Over Me!</h1>
event contains all events details:
altKey:false,bubbles:false,button:0,cancelBubble:false,cancelable:false,clientX:209,clientY:57,currentT
arget:null,defaultPrevented:false etc
```

## **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.

Forms have the following states:

- \$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-pristine` The field has not been modified yet
- `ng-dirty` The field has been modified
- `ng-valid` The field content is valid
- `ng-invalid` The field content is not valid
- `ng-valid-key` One *key* for each validation. Example: `ng-valid-required`, useful when there are more than one thing that must be validated
- `ng-invalid-key` Example: `ng-invalid-required`

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
- `ng-valid-key` One *key* for each validation. Example: `ng-valid-required`, useful when there are more than one thing that must be validated
- `ng-invalid-key` Example: `ng-invalid-required`

## 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>

```

complete example for validation:

```

<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

```

```

<h2>Validation Example</h2>

```

```

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

```

```

<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 = 'John Doe';
$scope.email = 'john.doe@gmail.com';

```

```
});  
</script>  
</body>  
</html>
```

**The HTML form attribute novalidate is used to disable default browser validation.**

With AngularJS, you can include HTML from an external file.

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

```
<div ng-include="myFile.htm"></div>
```

Pass parameter to Angular ng-include:You don't need that. all ng-include's sources have the same controller. So each view sees the same data.

AngularJS provides animated transitions, with help from CSS.

### **What is an Animation?**

An animation is when the transformation of an HTML element gives you an illusion of motion.

Add ngAnimate as a dependency in your application module:

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

### **AngularJS Routing**

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.

```
<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>
```

### **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>
```

OR

```
<ng-view></ng-view>
```

OR

```
<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.

Define the \$routeProvider using the config method of your application. Work registered in the config method will be performed when the application is loading.

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";
});

```

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

### **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.

```

var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
    $routeProvider

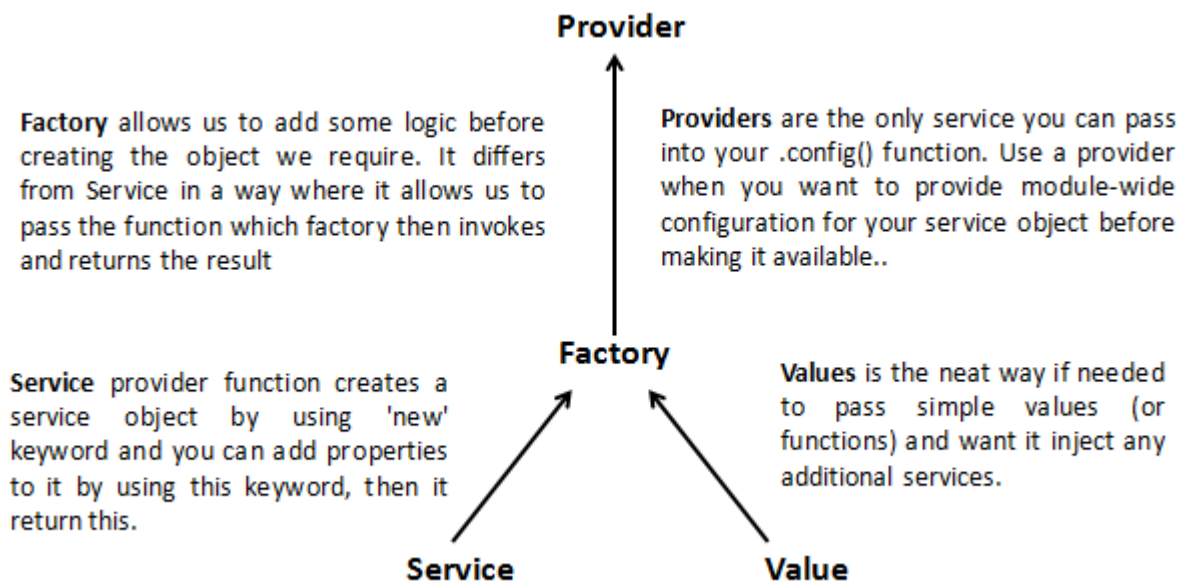
```

```

.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>Nothing</h1><p>Nothing has been selected</p>"
});
});

```

Service, factory and Providers:



he difference between factory and service is just like the difference between a function and an object

### Factory Provider

- Gives us the function's return value ie. You just create an object, add properties to it, then return that same object. When you pass this service into your controller, those properties on the object will now be available in that controller through your factory. (Hypothetical Scenario)
- Singleton and will only be created once

- Reusable components
- Factory are a great way for communicating between controllers like sharing data.
- Can use other dependencies
- Usually used when the service instance requires complex creation logic
- Cannot be injected in `.config()` function.
- Used for non configurable services
- If you're using an object, you could use the factory provider.
- Syntax: `module.factory('factoryName', function);`

### Service Provider

- Gives us the instance of a function (object)- You just instantiated with the 'new' keyword and you'll add properties to 'this' and the service will return 'this'. When you pass the service into your controller, those properties on 'this' will now be available on that controller through your service. (Hypothetical Scenario)
- Singleton and will only be created once
- Reusable components
- Services are used for communication between controllers to share data
- You can add properties and functions to a service object by using the `this` keyword
- Dependencies are injected as constructor arguments
- Used for simple creation logic
- Cannot be injected in `.config()` function.
- If you're using a class you could use the service provider
- Syntax: `module.service('serviceName', function);`

```
module.service('MyService', function() {

  this.method1 = function() {
    //..method1 logic
  }

  this.method2 = function() {
    //..method2 logic
  }

});
```

```

module.factory('MyFactory', function() {

    var factory = {};

    factory.method1 = function() {
        //..method1 logic
    }

    factory.method2 = function() {
        //..method2 logic
    }

    return factory;
});

```

## Services

Syntax: `module.service( 'serviceName', function );`

Result: When declaring `serviceName` as an injectable argument **you will be provided with an instance of the function. In other words** `new FunctionYouPassedToService()`.

## Factories

Syntax: `module.factory( 'factoryName', function );`

Result: When declaring `factoryName` as an injectable argument you will be provided with **the value that is returned by invoking the function reference passed to `module.factory`**.

## Providers

Syntax: `module.provider( 'providerName', function );`

Result: When declaring `providerName` as an injectable argument **you will be provided with** (`new ProviderFunction()`).`$get()`. The constructor function is instantiated before the `$get` method is called - `ProviderFunction` is the function reference passed to `module.provider`.

Providers have the advantage that they can be configured during the module configuration phase.

"How does this and `$scope` work in AngularJS controllers?"

## Short answer:

- `this`
  - When the controller constructor function is called, `this` is the controller.
  - When a function defined on a `$scope` object is called, `this` is the "scope in effect when the function was called". This may (or may not!) be the `$scope` that the function is defined on. So, inside the function, `this` and `$scope` may **not** be the same.



- \$scope
  - Every controller has an associated \$scope object.
  - A controller (constructor) function is responsible for setting model properties and functions/behaviour on its associated \$scope.
  - Only methods defined on this \$scope object (and parent scope objects, if prototypical inheritance is in play) are accessible from the HTML/view. E.g., from ng-click, filters, etc.

### What is the difference between angular-route and angular-ui-router?

[ui-router](#) is a 3rd-party module and is very powerful. It supports everything the normal ngRoute can do as well as many extra functions.

Here are some common reason ui-router is chosen over ngRoute:

- ui-router allows for [nested views](#) and [multiple named views](#). This is very useful with larger app where you may have pages that inherit from other sections.
- ui-router allows for you to have strong-type linking between states based on state names. Change the url in one place will update every link to that state when you build your links with [ui-sref](#). Very useful for larger projects where URLs might change.
- There is also the concept of the [decorator](#) which could be used to allow your routes to be dynamically created based on the URL that is trying to be accessed. This could mean that you will not need to specify all of your routes before hand.
- [states](#) allow you to map and access different information about different states and you can easily pass information between states via [\\$stateParams](#).
- You can easily determine if you are in a state or parent of a state to adjust UI element (highlighting the navigation of the current state) within your templates via [\\$state](#) provided by ui-router which you can expose via setting it in \$rootScope on run.

In essence, ui-router is ngRouter with more features, under the sheets it is quite different. These additional features are very useful for larger applications.

### **Mention what are the advantages of using AngularJS ?**

AngularJS has several advantages in web development.

- AngularJS supports MVC pattern
- Can do two ways data binding using AngularJS
- It has per-defined form validations
- It supports both client server communication
- It supports animations

### **Mention the steps for the compilation process of HTML happens?**

Compilation of HTML process occurs in following ways

- Using the standard browser API, first the HTML is parsed into DOM
- By using the call to the \$compile () method, compilation of the DOM is performed. The method traverses the DOM and matches the directives.
- Link the template with scope by calling the linking function returned from the previous step

### **Explain what is linking function and type of linking function?**

Link combines the directives with a scope and produce a live view. For registering DOM listeners as well as updating the DOM, link function is responsible. After the template is cloned it is executed.

- Pre-linking function: Pre-linking function is executed before the child elements are linked. It is not considered as the safe way for DOM transformation.
- Post linking function: Post linking function is executed after the child elements are linked. It is safe to do DOM transformation by post-linking function

### **Explain what is injector?**

An injector is a service locator. It is used to retrieve object instances as defined by provider, instantiate types, invoke methods and load modules. There is a single injector per Angular application, it helps to look up an object instance by its name.

### **Explain what is the difference between link and compile in Angular.js?**

- Compile function: It is used for template DOM Manipulation and collect all of the directives.
- Link function: It is used for registering DOM listeners as well as instance DOM manipulation. It is executed once the template has been cloned.

### **Explain what is factory method in AngularJS?**

For creating the directive, factory method is used. It is invoked only once, when compiler matches the directive for the first time. By using \$injector.invoke the factory method is invoked.

### **Mention what are the characteristics of “Scope”?**

- To observer model mutations scopes provide APIs (\$watch)
- To propagate any model changes through the system into the view from outside of the Angular realm
- A scope inherits properties from its parent scope, while providing access to shared model properties, scopes can be nested to isolate application components
- Scope provides context against which expressions are evaluated

### **Explain the concept of scope hierarchy? How many scope can an application have?**

Each angular application consist of one root scope but may have several child scopes. As child controllers and some directives create new child scopes, application can have multiple scopes. When

new scopes are formed or created they are added as a children of their parent scope. Similar to DOM, they also creates a hierarchical structure.

### **Distinguish between AngularJs and JavaScript expressions.**

**Answer:** There are several differences between AngularJs and JavaScript expressions:

- We can write AngularJs expressions in HTML, but we cannot write JavaScript expressions in HTML.
- We cannot use conditional iterations, loops, and exceptions in AngularJs, but we can use all of these conditional properties in JavaScript expressions.
- Filters are supported in AngularJs whereas filters are not supported in JavaScript.

### **Write all the steps to configure an Angular App(ng-app).**

**Answer:** To set up an Angular App we must follow certain steps as mentioned below:

- angular.module will be created at first.
- A controller will be assigned to the module.
- The module will be linked with the HTML template(i.e. UI or View) with an angular app(ng-app).
- The HTML template will be linked with the controller(i.e JS) with an ng-controller directive.

### **What are the directive scopes in AngularJs?**

**Answer:** Three directive scopes are available in AngularJs.

**They are:**

- **Parent scope** – Whatever change you make in your directive that comes from the parent scope, will also reflect in the parent scope, and it is also a default scope.
- **Child scope** – It is a nested scope which inherits a property from the parent scope. Also if any properties and function on the scope are not connected with the parent scope directive, then a new child scope directive is created.
- **Isolated scope** – It is reusable and is used when we build a self-contained directive. It is only used for private and internal use which means it does not contain any properties of the parent scope.

### **How can we share the data between controllers in AngularJs?**

**Answer:** First, we have to create a service. Service is used to share the data between controllers in AngularJs in a very lucid, easy and fastest way. We use events, \$parent, next sibling, and controller by using \$rootScope.

### **What is the digest cycle in AngularJs?**

**Answer:** It is a part of the process of data binding in AngularJs. It compares the old and new version of the scope model value in each digest cycle.

The digest cycle is triggered automatically. We can also enhance the usability by using `$apply()`, if we want to trigger the digest cycle manually.

### **Explain the differences between one-way binding and two-way binding.**

**Answer:** One-way binding is used to bind the data from the model to view without updating the HTML template or view automatically.

Thus in order to update the HTML template, we need to write a custom code which will update the view every time whenever a data is binded from model to view.

Whereas, two-way binding is used to bind the data from the model to view and vice versa (i.e. view to model) by automatically updating the HTML template without writing any custom code.

### **Difference between sessionStorage, cookies, and localStorage.**

**Answer:** Given below are the various differences.

**SessionStorage** – The data is stored for a particular session. The data will be lost whenever the browser tab will be closed or after some particular session. Maximum size stored can be up to 5MB.

**LocalStorage** – The data is stored with no expiration date. The data can only be cleared by JavaScript or by clearing the browser cache. Storage limit is maximum than the sessionStorage and cookie.

**Cookies** – It stores the data that has to be sent back to the server with some requests. The cookie's expiration varies on the type and duration set from either the server side or client side. Maximum size stored can be less than 4KB.

### **What is the role of \$routeProvider in AngularJs?**

**Answer:** It is the \$routeProvider that helps in navigating between different pages/links without separately loading the page/link whenever a user clicks on a link.

`ngRoute config()` method is used to configure the routeProvider.

### **What is the difference between \$scope and scope?**

**Answer:** In AngularJs \$scope is used to achieve dependency injection and scope is used for linking between View (i.e. HTML) and Controller (i.e. JS).

### **How are AngularJs prefixes \$ and \$\$ used?**

**Answer:** \$\$ variable in AngularJS is used as a private variable, as it is used to prevent accidental code collision with the user code.

Whereas \$ prefix can be used to denote angular core functionalities (like a variable, parameter, property or method).

### **Two-way data binding**

Two-way data binding in AngularJS means binding data from Model to View and vice versa (Data flows from the Scope/Controller to the View and from the View to the scope/controller). '**ng-model**' is

an angular directive used for achieving two-way data binding. Any modifications to that model from the Scope/Controller will be automatically propagated to the view regardless of whether the view is asking for the updated data and any modifications to that model from the View will be immediately reflected back to the Scope/Controller.

### One-way data binding

One-way data binding in AngularJS means binding data from Model to View (Data flows from the scope/controller to the view). '**ng-bind**' is an angular directive used for achieving one-way data binding. After the binding, any modifications to that model from the scope/controller will be automatically propagated to the view regardless of whether the view is asking for the updated data. No propagation happens for any change to the model from the view to the controller.

### One-time data binding

As its name suggests, the binding happens **only once**, ie, in the first digest cycle. One-time binding allows for a model or view to be updated ONCE from the value set by the controller upon the first digest cycle. As of AngularJS 1.3, you can use the "::" token to create one-time data bindings. These are bindings that deregister their own \$watch() functions once the value has stabilized (which basically means the value is defined).

One-time binding is used for values that won't change after the page is stable. "Stable" generally means that the expression has been assigned a value. Once the value is set, changes to the value in the controller won't change the displayed value until the page is reloaded. The syntax is {{::expression}}.

So, for those values or lists that won't change after the page is stable, always try to use one-time binding. This will reduce the number of unnecessary watchers in our application.

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.0/angular.min.js"></script>
```

```
<div ng-app="myApp" ng-controller="myCtrl">
  <div ng-repeat="customer in ::customers" >
    {{::customer.name}}
    ({{customer.address}})
    <button ng-click="change(customer)">Change</button>
  </div>
</div>
```

```
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.customers = [{
    name: 'Gloria Jane',
    address: 'Silo Park, 9300 East Orchard Road, Greenwood Village, CO, 80114'}, {
    name: 'Nicholas Michael',
    address: 'Village Park, East Lake Avenue, Aurora, CO, 80016'
  }];
});
```

```
$scope.change = function(customer) {  
    customer.address = 'Cherry Creek State Park, 4201 S Parker Rd, Aurora, CO 80014, USA';  
    customer.name = 'Tessy Thomas';  
};  
});  
</script>
```

### **What is SPA (Single page application) in AngularJs?**

**Answer:** It is a web application which loads a single HTML page and dynamically updates the page as the user connects with the app.

By using AJAX and HTML a fluid and responsive web app can be created by SPA without invariant page reloads. Through this, we can make responsive UI with no page flicker.

### **Explain ng-bind and ng-bind-html directives.**

**Answer:**

**ng-bind:** It is a directive which replaces the content of the HTML element with the value of the assigned variable or expression.

The content of the HTML element will change by changing the value of the variable or expression.

It is like ({{expression}}) and the syntax for this is,

```
<ANY ELEMENT ng-bind="expression"> </ANY ELEMENT>
```

**ng-bind-html:** It is a directive which binds the content to the HTML element(view) in a secure way. \$sanitize service is used to sanitize the content to bind into an HTML element. To do this 'angular-sanitize.js' must be included in our application.

### **Syntax to write this,**

```
<ANY ELEMENT ng-bind-html=" expression "> </ANY ELEMENT>
```

### **Define Factory method in AngularJs.**

**Answer:** It is quite similar to service, factories implement a module pattern in which we use a factory method to generate an object which is used for building models.

In a factory, a method object is returned at the end by creating a new object and adding functions as properties.

### **Syntax:**

```
module.factory('factoryName', function);
```

### **What is a controller in AngularJs?**

**Answer:** A controller is a JavaScript function which is bound to the specified scope. Angular instantiates the new controller object and injects the new scope as a dependency.

A controller can be used to set up the initial state of the scope object and to add behavior to the object.

A controller cannot be used to share code or state across controllers, but instead of that Angular service can be used.

```
<Any ng-Controller=" expression">
</Any>
<div ng-app="mainApp" ng-controller="SimpleController">
</div>
```

### **What is ng-switch in AngularJs?**

**Answer:** It is used to conditionally exchange the structure of DOM on a template which is based on a scope-based expression.

This directive lets you show or hide the HTML element depending on the expression.

```
<element ng-switch="expression">
<element ng-switch-when="value"></element>
```

### **What is a representational state transfer(REST) in AngularJs?**

**Answer:** REST is an API style that operates over the HTTP request.

The requested URL identifies the data to be operated on, and the HTTP method identifies the operation that is to be performed. REST is a style of API rather than a formal specification, and there is a lot of debate and disagreement about what is and isn't RESTful, which is a term used to indicate an API that follows the REST style.

AngularJS is flexible about how RESTful web services are consumed.

### **What are the AngularJs Global API?**

**Answer:** It is a combination of global JavaScript function which is used to perform tasks like comparing objects, iterating objects and converting data.

**There are some common API functions like:**

- **angular. lowercase:** It converts a string to lowercase string.
- **angular. uppercase:** It converts a string to uppercase string.
- **angular. isString:** It will return true if the current reference is a string.
- **angular. isNumber:** It will return true if the current reference is a number.

### **What is a provider method in AngularJs?**

**Answer:** A provider is an object which creates a service object by allowing to take more control.

\$get() method is used in the provider which returns the service object. The service name and the factory function are the arguments that are passed into the provider method. AngularJS uses \$provide to register new providers.

**Syntax:**

```
serviceApp.provider("logService", function ()
```

**What is Event Handling?**

**Answer:** Event handling in AngularJs is very useful when you want to create advance AngularJs applications.

We need to handle DOM events like mouse clicks, moves, keyboard presses, change events and so on. AngularJs has some listener directives like ng-click, ng-dbl-click, ng-mousedown, ng-keydown, ng-keyup etc.

**What are the attributes that can be used during the creation of a new AngularJs directives?**

**Answer:** There are several attributes which can be used during a new directive creation.

**They include:**

1. **Template:** It describes an inline template as a string.
2. **Template URL:** This attribute specifies the AngularJs HTML compiler to replace the custom directive inside a template with the HTML content located inside a separate file.
3. **Replace:** It replaces the current element if the condition is true, if false append this directive to the current element.
4. **Transclude:** It allows you to move the original children of a directive to a location inside the new template.
5. **Scope:** It creates a new scope for this directive rather than inheriting the parent scope.
6. **Controller:** It creates a controller which publishes an API for communicating across the directives.
7. **Require:** It requires another directive to be present to function the current directive efficiently.
8. **Link:** It modifies resulting DOM element instances, adds event listeners, and set up data binding.
9. **Compile:** It modifies the DOM template for features across copies of a directive, as when used in other directives. Your compile function can also return link functions to modify the resulting element instances.

**Is Nested Controllers possible or not in AngularJs?**

**Answer:** Yes, it is possible as Nested Controllers are well-defined in a classified manner while using a view.

**Is AngularJS well-suited with all browsers?**



**Answer:** Yes, it is companionable with all browsers like Safari, Chrome, Mozilla, Opera, IE etc. as well as Mobile browsers.

### **Define services in AngularJS.**

**Answer:** AngularJS services are the singleton objects or functions which are used for carrying out definite tasks.

It embraces some corporate ideas and these purposes can be called as controllers, directive, filters and so on.

### **Explain the advantages of AngularJS.**

**Answer: Advantages of AngularJS include:**

- AngularJS supports MVC form.
- Organize two ways data binding using AngularJS.
- It supports mutual client-server communication.
- It supports simulations.

### **Difference between services and factory.**

**Answer:** Factories are functions that return the object, while services are constructor functions of the object which is used by a new keyword.

### **Syntax:**

**Factory** – `module.factory('factoryName', function);`

**Service** – `module.service('serviceName', function);`

### **If both factory and service are equivalent, then when should I use them?**

**Answer:** Factory provider is preferred using an object, whereas a service provider is preferred using with class.

### **Difference between ng-bind and ng-model directive.**

**Answer:** ng-bind directive has one way data bindings, data flows only from object to UI, not vice versa (i.e. `$scope>>view`) and ng-model directive has two way data bindings, data flows between UI to object and vice versa (i.e. `$scope>>view` and `view>>$scope`).

### **What is the difference between AJAX and AngularJs?**

**Answer:** AJAX stands for Asynchronous JavaScript which is used for sending and getting responses from the server without loading the page.

Whereas, AngularJS is a typescript language based JavaScript framework following the MVC pattern.

### **What is the difference between ngRoute and ui-router?**

**Answer:** ngRoute is a module developed by angularJS team which was a part of the core angularJS framework. Whereas ui-router was developed by a third-party community to overcome the problems of ngRoute.

ngRoute is a location or URL based routing, and ui-router is a state-based routing which allows nested views.

### **How to set, get and clear cookies in AngularJs?**

**Answer:** AngularJS has a module called ngCookies, so before injecting ngCookies angular-cookies.js should be included into the application.

**Set Cookies** – Put method is used to set cookies in a key-value format.

```
$cookies.put("username", $scope.username);
```

**Get Cookies** – Get method is used to get cookies.

```
$cookies.get('username');
```

**Clear Cookies** – Remove method is used to remove cookies.

```
$cookies.remove('username');
```

### **What is a Directive?**

In the simplest terms, a directive in AngularJS is a reusable component. Directives in AngularJS encapsulate all the behavioral properties and functionalities of an element in a semantic way, thereby keeping all of the functionality grouped together. This helps to keep track of changes of one HTML section in one place rather than tracking the changes on a global level in a script.

A more formal definition of a directive is: In AngularJS, a directive is a JavaScript factory function defined inside an AngularJS module that holds a set of instructions for the HTML compiler for defining a specified behavior of a DOM element.

### **Defining a Directive**

This section lists simple steps to define a custom directive in an AngularJS module. First, we need to define an Angular app.

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

Now, define a directive.

```
myApp.directive('myDirective', function() {  
    return {  
        restrict: 'E',  
        template: '<h1>I made a directive!</h1>'  
    };  
});
```

```
});
```

This defines a directive. restrict: 'E' means “restrict the usage of this directive to only Elements.” Thus we embed this directive in the HTML page as

```
<body ng-app="myApp">
  <my-directive></my-directive>
</body>
```

This code piece is equivalent to

```
<body ng-app="myApp">
  <h1>I made a directive!</h1>
</body>
```

Note that AngularJS maps the naming conventions from HTML’s my-directive to JavaScript’s myDirective.

This is a basic example of definition and usage of a directive. The following is a more advanced example.

Consider a use case where we need an HTML structure to display movie titles. The HTML snippet looks like this:

```
<div class="movie">
  <div class="title">
    Movie Title
  </div>
  <div class="movie">
    Movie name
  </div>
</div>
```

The code given above seems rich in terms of CSS classes and HTML tags, but the code is not repeatable. In a case where multiple movies will need to be displayed, this code structure would become messy. Moreover, if at a later stage there comes a requirement to add a movie poster and some other functionality to the movie section, the code will need to be modified at multiple places, which is not sustainable.

Instead, we can use directive in this situation.

First, we define the Angular app and controller. Note that the list of movies is defined in the controller.

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

myApp.controller('myController', function($scope) {
  $scope.movies = ['Ice Age', 'Frozen'];
});
```

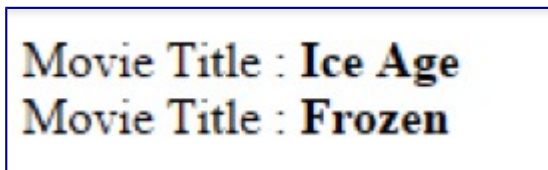
Next, define the directive.

```
myApp.directive('myMovie', function() {
  return {
    restrict: 'E',
    transclude: 'true',
    template: '<span ng-transclude></span>',
    link: function(scope, element, attr){
      element.append("<strong>" + attr.title + "</strong>");
    }
  };
});
```

Now, embed the directive in the HTML code.

```
<div ng-app="myApp" ng-controller="myController">
  <div ng-repeat="movie in movies">
    <my-movie title="{{movie}}">
      Movie Title :
    </my-movie>
  </div>
</div>
```

The output is



Ta-da!

### Examine the Directive

A directive is, thus, implemented as a function, which returns an object, called Data Definition Object (DDO), that configures the directive's behavior and template. restrict, template, etc., are the fields of this object.

restrict: defines the type of HTML element which can act as a trigger for the directive.

- E: Element – the directive is used as an HTML tag
- A: Attribute – the directive is used as an HTML attribute
- C: Class – the directive is used as a CSS (in an element's class="..." definition)

transclude: specifies whether to transfer and include the original inner content of the directive's HTML markup ('Movie Title :') in the destination markup (which is defined in the template).

template: specifies the HTML content that should be added to the HTML result of the directive.

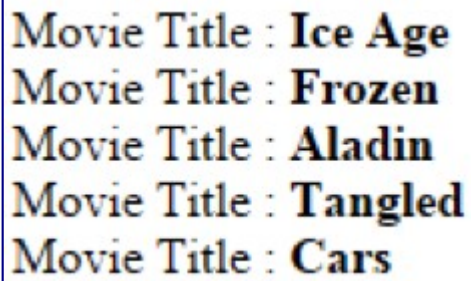
## There's More...

Let's further experiment with the directive.

1. More movies can be added to the movies in the controller.

```
$scope.movies = ['Ice Age', 'Frozen', 'Aladdin', 'Tangled', 'Cars'];
```

The output



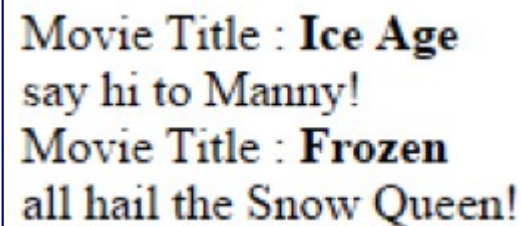
```
Movie Title : Ice Age  
Movie Title : Frozen  
Movie Title : Aladdin  
Movie Title : Tangled  
Movie Title : Cars
```

2. Different functions for different movies can be added. Let's add some content to the movies section depending on the movie title.

Modify the directive's link function

```
link: function(scope, element, attr){  
    element.append("<strong>" + attr.title + "</strong>");  
    if(attr.title === 'Ice Age'){  
        element.append("<br> say hi to Manny!");  
    }  
    else {  
        element.append("<br> all hail the Snow Queen!");  
    }  
}
```

The output



```
Movie Title : Ice Age  
say hi to Manny!  
Movie Title : Frozen  
all hail the Snow Queen!
```

Angular provides several other fields in directive. When these fields are used in conjunction, they truly make directive perform magic on the HTML page.

**The link function normally accepts 3 parameters including the scope, the element that the directive is associated with, and the attributes of the target element.**

Some of commonly used properties are:

#### 1. restrict

It specifies how a directive is implemented in an Angular app. There are 4 restrict options:

restrict : 'A' – attribute (default one) | `<div my-directive></div>`

restrict : 'C' – Class | `<div class="my-directive: expression;"></div>`

restrict : 'E' – element | `<my-directive></my-directive>`

restrict : 'M' – Comment | `<!-- Directive : my-directive expression; -->`

You can also specify multiple restrict like as follows:

restrict : 'EC'

#### 2. scope

Scope accesses data or methods inside template and link function. By default the directives do not produce their own scope without explicitly set.

Different types of scopes are able to define which are as follows:

- scope : true – Get new scope
- scope : false – Use its parent scope
- scope : {} – Get new isolated scope that doesn't inherit from parent and exists on its own

#### 3. template

It specifies the HTML content that will be generated when directive is compiled.

#### 4. templateUrl

It specifies the path of template that should be used by directives.

```
details.directive('intellipaat', function () {  
  return {  
    restrict: 'E',  
    scope: false,  
    templateUrl: 'intellipaat.tpl.html'  
  }  
});
```

**The directive.scope.user property is set to "=user". That means, that the directive.scope.user property is bound to the property in the scope property (not in the isolate scope) with the name passed to the user attribute of the <userinfo> element. It sounds confusing, so look at this HTML example:**

```
<userinfo user="jakob"></userinfo>  
<userinfo user="john"></userinfo>
```

These two <userinfo> elements contain a user attribute. The value of these attributes contain the names of properties in the \$scope object which are to be referenced by the isolate scope object's userinfo property.

Here is a full example:

```
<userinfo user="jakob"></userinfo>
<userinfo user="john"></userinfo>

<script>
myapp.directive('userinfo', function() {
    var directive = {};

    directive.restrict = 'E';

    directive.template = "User : <b>{{user.firstName}}</b> <b>{{user.lastName}}</b>";

    directive.scope = {
        user : "=user"
    }

    return directive;
});

myapp.controller("MyController", function($scope, $http) {
    $scope.jakob = {};
    $scope.jakob.firstName = "Jakob";
    $scope.jakob.lastName = "Jenkov";

    $scope.john = {};
    $scope.john.firstName = "John";
    $scope.john.lastName = "Doe";
});

</script>
```

## The compile() and link() Functions

If you need to do something more advanced inside your directive, something that you cannot do with an HTML template, you can use the compile() and link() functions instead.

The compile() and link() functions define how the directive is to modify the HTML that matched the directive.

The compile() function is called once for each occurrence of the directive in the HTML page. The compile() function can then do any one-time configuration needed of the element containing the directive.

The compile() function finishes by returning the link() function. The link() function is called every time the element is to be bound to data in the \$scope object.

As mentioned, you add the compile() function to the directive definition object, and the compile() function has to return the link() function when executed. Here is how that looks:

```

<script>
myapp = angular.module("myapp", []);
myapp.directive('userinfo', function() {
    var directive = {};

    directive.restrict = 'E'; /* restrict this directive to elements */

    directive.compile = function(element, attributes) {
        // do one-time configuration of element.

        var linkFunction = function($scope, element, attributes) {
            // bind element to data in $scope
        }

        return linkFunction;
    }

    return directive;
});
</script>

```

The compile() function takes two parameters: The element and attributes parameters.

The element parameter is a jqLite wrapped DOM element. AngularJS contains a lite version of jQuery to help you do DOM manipulation, so the element's DOM manipulation methods are the same as you know from jQuery.

The attributes parameter is a JavaScript object containing properties for all the attributes of the DOM element. Thus, to access an attribute named type you would write attributes.type.

The link() function takes three parameters: The \$scope parameter, the element parameter and the attributes parameter. The element and attributes parameter is the same as passed to the compile() function. The \$scope parameter is the normal scope object, or an isolate scope in case you have specified one in the directive definition object.

The compile() and link() function names are actually confusing. They are inspired by compiler terms. I can see the resemblance, but a compiler parses an input once, and creates an output. A directive configures an HTML element and then updates that HTML subsequently whenever the \$scope object changes.

A better name for the compile() function would have been something like create(), init() or configure(). Something that signals that this function is only called once.

A better name for the link() function would have been something like bind() or render(), which signals that this function is called whenever the directive needs to bind data to it, or to re-render it.

Here is a full example that shows a directive that uses both a compile() and link() function:



```

<div ng-controller="MyController" >
  <userinfo >This will be replaced</userinfo>
</div>

<script>
  myapp = angular.module("myapp", []);
  myapp.directive('userinfo', function() {
    var directive = {};

    directive.restrict = 'E'; /* restrict this directive to elements */

    directive.compile = function(element, attributes) {
      element.css("border", "1px solid #cccccc");

      var linkFunction = function($scope, element, attributes) {
        element.html("This is the new content: " + $scope.firstName);
        element.css("background-color", "#ffff00");
      }

      return linkFunction;
    }

    return directive;
  })
  myapp.controller("MyController", function($scope, $http) {
    $scope.cssClass = "notificationDiv";

    $scope.firstName = "Jakob";

    $scope.doClick = function() {
      console.log("doClick() called");
    }
  });
</script>

```

The compile() function sets a border on the HTML element. This is only executed once because the compile() function is only executed once.

The link() function replaces the content of the HTML element, and sets the background color to yellow.

There is no particular reason why the border was set in the compile() function, and the background color in the link() function. Both could have been set in the compile() function, or both in the link() function. If set in the compile() function they would only have been set once (which is often what you want). If set in the link() function they would be set every time the HTML element is bound to data in the \$scope object. This might be useful if you needed to set the border and background color differently depending on data in the \$scope object.

## Setting Only a link() Function

Sometimes you do not need the compile() step for your directive. You only need the link() function. In that case you can set the link() function directly on the directive definition object. Here is the example from before, with only a link function:

```
<div ng-controller="MyController" >
  <userinfo >This will be replaced</userinfo>
</div>

<script>
  myapp = angular.module("myapp", []);
  myapp.directive('userinfo', function() {
    var directive = {};

    directive.restrict = 'E'; /* restrict this directive to elements */

    directive.link = function($scope, element, attributes) {
      element.html("This is the new content: " + $scope.firstName);
      element.css("background-color", "#ffff00");
    }

    return directive;
  })
  myapp.controller("MyController", function($scope, $http) {
    $scope.cssClass = "notificationDiv";

    $scope.firstName = "Jakob";

    $scope.doClick = function() {
      console.log("doClick() called");
    }
  });
</script>
```

## How to create a service in AngularJS ?

The service in AngularJS is created by registering it with the module it is going to operate in. There are three ways to create the angular service. They are:

- Factory
- Service
- Provider

## How AngularJS is different from JQuery?

AngularJS is a framework with key features like models, two-way binding, directives, routing, dependency injections, unit tests etc., whereas the JQuery is a library used for DOM manipulation with no two-way binding features.

### What are scopeless controller in AngularJS ? why to use them?

**Scopeless controllers** have no \$scope injected and the functions and properties are binded directly onto the controller. Scopeless controllers are used in the situation where the controller becomes complex by using the \$scope to provide data.

### What is the use of ng-cloak directive in AngularJS ?

It is used to prevent the Angular HTML template from getting displayed briefly by the browser in its uncompiled form while the application is still loading. It is used to avoid the undesirable flicker effect caused by HTML template display.

### How \$location is different from window.location in AngularJS ?

**\$Location** is used when you don't want the full page to reload when the URL is changed and the **window.location** is used when you want to change the URL, reload the page or navigate to a different page.

### What is the difference between 'this' vs \$scope in AngularJS controllers?

The **\$scope** is an object with the available methods and properties. It is used to get all controls on the controller.JS files. It is available for both the controller and the view. "This" is the controller used when the controller's constructor function is called.

### How to use Multiple ng-app within a page in AngularJS ?

There are two ways to use a **Multiple ng-app** within a single page in AngularJS. One is the bootstrap method and the other is ngModules directive.

### **Define Multiple Angular Apps In One Page**

```
<div ng-app="firstApp">
  <div ng-controller="FirstController">
    <p>1: {{ desc }}</p>
  </div>
</div>

<div ng-app="secondApp">
  <div ng-controller="SecondController">
    <p>2: {{ desc }}</p>
  </div>
</div>
```

```

<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.7/angular.js"></script>
<script type="text/javascript">
var firstApp = angular.module('firstApp', []);
firstApp.controller('FirstController', function($scope) {
    $scope.desc = "First app.";
});
var secondApp = angular.module('secondApp', []);
secondApp.controller('SecondController', function($scope) {
    $scope.desc = "Second app.";
});
</script>

```

And below is the output. Out of 2 `<ng-app>` present on the page, the first one gets initialized and works as expected. Output also shows the same.

1: First app.

2: {{ desc }}

**Then how can you define 2 different angular apps on the same page? Well, there are a couple of ways to implement this.**

### Manually Bootstrapping Each App

You can manually bootstrap both the applications using `<angular.bootstrap()>`

```

var firstApp = angular.module('firstApp', []);
firstApp.controller('FirstController', function($scope) {
    $scope.desc = "First app. ";
});
var secondApp = angular.module('secondApp', []);
secondApp.controller('SecondController', function($scope) {
    $scope.desc = "Second app. ";
});
var dvFirst = document.getElementById('dvFirst');
var dvSecond = document.getElementById('dvSecond');
angular.element(document).ready(function() {
    angular.bootstrap(dvFirst, ['firstApp']);
    angular.bootstrap(dvSecond, ['secondApp']);

```

```
});
```

Since we are bootstrapping them manually, make sure to remove `<ng-app>` attribute from the HTML page.

### Manually Bootstrapping Second App

You can also manually bootstrap the second app, where the first app gets initialized via ng-app.

```
var firstApp = angular.module('firstApp', []);
firstApp.controller('FirstController', function($scope) {
    $scope.desc = "First app. ";
});
var secondApp = angular.module('secondApp', []);
secondApp.controller('SecondController', function($scope) {
    $scope.desc = "Second app. ";
});
var dvSecond = document.getElementById('dvSecond');
angular.element(document).ready(function() {
    angular.bootstrap(dvSecond, ['secondApp']);
});
```

### Using Dependency Injection to Inject Both in Root App

You can define a root level app using `<ng-app>` and then inject both the apps as module in root app.

```
var rootApp = angular.module('rootApp', ['firstApp','secondApp']);
var firstApp = angular.module('firstApp', []);
firstApp.controller('FirstController', function($scope) {
    $scope.desc = "First app. ";
});
var secondApp = angular.module('secondApp', []);
secondApp.controller('SecondController', function($scope) {
    $scope.desc = "Second app. ";
});
```

And make sure any parent node (body/Parent div) is assigned with `<ng-app=rootApp>`.

### Explain .config() and .run() methods in AngularJS ?

**.config()** – The .config() function is used to add configuration blocks on the module.

**.run()** - The .run() function is used to add run blocks on the module.

### How to use \$scope.\$watch and \$scope.\$apply in AngularJS ?

In AngularJS **\$scope.\$watch** the function is used to create a watch of some variables and \$scope.

\$apply function is used to execute some code and call the \$digest function.

### Explain the functionality angular.copy() method?

It is used to allocating the value of an object into another object however the object value must not be altered.

If you are adding any new property or altering any value of the property then the object noting to the same object will update by applying angular. Copy() method.

### Explain \$templateCache in AngularJs ?

**\$templateCache** is a Cache object that is created by the \$cache factory. The first time you use a template, it is loaded in the \$templateCache for easy and quick retrieval.

### What is the difference between a stateful and stateless component in AngularJs ?

A **stateful component** is a detail implementation of the component that can change over time and the stateful components can have stateless components inside them, while the **stateless components** are a plain JavaScript function.

### What is an interceptor in Angular? Why it is used?

An interceptor is a middleware code in AngularJs where all the \$http requests go through. It is attached with \$httpProvider service and able to intercept request and response objects. Interceptor Middleware is useful for error handling, authentication and other filters you want to apply on request and response.

### What is \$emit, \$broadcast and \$on in AngularJS ?

\$broadcast, \$emit, and \$on are services in Angular Js. Below we have listed why they are used In Angular JS.

**\$broadcast()**: \$broadcast() service of Angular is used to propagate the event to all of his child controllers and it's registered parent \$rootScope.scope listeners.

```
$rootScope.$broadcast('SummaryEvent', {  
    priority: priority  
});
```

**\$on()**: AngularJS \$on() service is used to listen any type of event raised by \$broadcast and \$emit.

```
$scope.$on("SummaryEvent", function (event, args) {
```

```
Vm.priority=args.priority  
});
```

**\$emit:** \$emit is similar to \$broadcast service but it is used to propagate the event to upwards through the scope hierarchy and notify to the registered \$rootScope.Scope listeners.

### List some inbuilt services in AngularJs ?

There are 30 inbuilt **services in AngularJs**. Below are few most used services in AngularJs.

- \$location
- \$scope
- \$http
- \$timeout
- \$interval
- \$window

### What is transclusion in AngularJS ?

The transclusion in Angular JS will allow you to move the original children of a directive to a specific location inside a new template. The ng directive marks the insertion point for the transcluded DOM of the very near parent directive that uses transclusion.

**ng-transclude** or **ng-transclude-slot** attribute directives are used for transclusion in Angular JS.

### Explain the use of Ng-If, Ng-Switch, And Ng-Repeat directives in AngularJS ?

**ng-if** – This directive removes a portion of the DOM tree, which is based on the expression.

In case the expression is assigned to ng-if, it evaluates to a false value, and then the element is deleted from the DOM tree, or else a clone of the element is reinserted into the DOM.

**ng-switch** – This directive is used based on a scope expression to conditionally swap DOM structure on the template.

The ng-switch default directive will be preserved at the specific location in a template.

**ng-repeat** – This directive is used to instantiate the template once per item from a collection.

Each template which is instantiated gets its own scope where the given loop variable is set to the current collection of item.

### How to enable caching in Angular 1.x?

Caching can be enabled by setting the config.cache value or the default cache value to TRUE or to a cache object that is created with \$cacheFactory.

In case you want to cache all the responses, then you can set the default cache value to TRUE.

And, if you want to cache a specific response, then you can set the config.cache value to TRUE.

### How can you get URL parameters from AngularJS Controller?

The RouteProvider and the RouteParams can be used to get the URL parameters in the controller. As the route wires up the URL to the controller and RouteParams can be passed to the controller to get the URL parameters.

### How to access parent scope from child controller in AngularJS ?

In angular there is a scope variable called \$parent (i.e. \$scope.\$parent). \$parent is used to access parent scope from child controller in Angular JS.

Example:

```
<div ng-controller="ParentCtrl">
  <h1>{{ name }}</h1>
  <p>{{ address }}</p>
  <div ng-controller="ChildCtrl">
    <h1>{{ title }}</h1>
    <input type="text" ng-model="$parent.address" />
  </div>
```

### How to enable HTML5 mode in Angular 1.x?

**html5Mode** method of \$locationProvider module is used to enable HTML5 mode in Angular 1.x. For creating pretty URLs and removing # from URLs html5Mode need to true.

Enabling html5Mode to true in Angular 1.x.

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

.config(function($routeProvider, $locationProvider) {

  $routeProvider
    .when('/', {
      templateUrl : 'partials/home.html',
      controller : mainController
    })

  // use the HTML5 History API
  $locationProvider.html5Mode(true);
});
```

### Explain Directive scopes in AngularJs ?

There are three types of directive scopes available in Angular.

- **Parent Scope:** is default scope



- **Child Scope:** If the properties and functions you set on the scope are not relevant to other directives and the parent, you should probably create a new child scope.
- **Isolated Scope:** Isolated Scope is used if the directive you are going to build is self-contained and reusable. Does not inherit from parent scope, used for private/internal use.

### Explain AngularJS digest cycle?

AngularJS digest cycle is the process behind Angular JS data binding.

In each digest cycle, Angular compares the old and the new version of the scope model values. The digest cycle is triggered automatically. We can also use `$apply()` if we want to trigger the digest cycle manually.