AngularJS Introduction

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/1.6.9/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>Input something in the input box:</p>
```

```
<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 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**.

AngularJS Directives

As you have already seen, AngularJS directives are HTML attributes with an **ng** prefix.

The **ng-init** directive initializes AngularJS application variables.

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<body>

<div ng-app=""  ng-init="firstName='Anil'">

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

</div>

</body>

</html>
```

Example 2:

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<body>

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

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

</div>

</body>

</html>
```

AngularJS Expressions

AngularJS expressions are written inside double braces: **{{ expression }}**.

AngularJS will "output" data exactly where the expression is written:

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<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.

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<body>

<div ng-app="">

<p>Input something in the input box:</p>

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

<p>{{name}}</p>

</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.

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<body>

<p>Try to change the names.</p>
```

```
<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= "Anil";

   $scope.lastName= "Sharma";

});

</script>

</body>

</html>
```

AngularJS Expressions

AngularJS binds data to HTML using **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 }}

```
<!DOCTYPE html>
```

```
<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<body>

<div ng-app>

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

</div>

</body>

</html>
```

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

```
<!DOCTYPE html>

<html>

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

<body>

<p>Change the value of the input field:</p>

<div ng-app="" ng-init="myCol='pink'">

<input style="background-color:{{myCol}}" ng-model="myCol">

</div>

<p>AngularJS resolves the expression and returns the result.</p>

<p>The background color of the input box will be whatever you write in the
input field.</p>

</body>

</html>
```

AngularJS Numbers

AngularJS numbers are like JavaScript numbers:

```
<!DOCTYPE html>

<html>

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

<body>

<div ng-app="" ng-init="quantity=1;cost=5">

<p>Total in dollar: {{ quantity * cost }}</p>

</div>

</body>

</html>
```

Using ng-bind :

```
<!DOCTYPE html>

<html>

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

<body>

<div ng-app="" ng-init="quantity=1;cost=5">

<p>Total in dollar: <span ng-bind="quantity * cost"></span></p>

</div>

</body>

</html>
```

AngularJS Strings

AngularJS strings are like JavaScript strings:

```
<!DOCTYPE html>

<html>

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

<body>

<div ng-app="" ng-init="firstName='Anil';lastName='Sharma'">

<p>The full name is: {{ firstName + " " + lastName }}</p>

</div>

</body>

</html>
```

Same example using `ng-bind`:

```
<!DOCTYPE html>

<html>

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

<body>

<div ng-app="" ng-init="firstName='Anil';lastName='Sharma'">

<p>The full name is: <span ng-bind="firstName + ' ' +
lastName"></span></p>

</div>

</body>
```

```
</html>
```

AngularJS Objects

AngularJS objects are like JavaScript objects:

```
<!DOCTYPE html>

<html>

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

<body>

<div ng-app="" ng-init="person={firstName:'Anil',lastName:'Sharma'}">

<p>The name is {{ person.lastName }}</p>

</div>

</body>

</html>
```

Same example using `ng-bind`:

```
<!DOCTYPE html>

<html>

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

<body>

<div ng-app="" ng-init="person={firstName:'Anil',lastName:'Sharma'}">

<p>The name is <span ng-bind="person.lastName"></span></p>

</div>

</body>
```

```
</html>
```

AngularJS Arrays

AngularJS arrays are like JavaScript arrays:

```
<!DOCTYPE html>

<html>

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

AngularJS Modules

An AngularJS module defines an application.

The module is a container for the different parts of an application.

The module is a container for the application controllers.

Controllers always belong to a module.

Creating a Module

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

```
<div ng-app="myApp">...</div>

<script>
```

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

</script>
```

The "myApp" parameter refers to an HTML element in which the application will run.

Adding a Controller

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

Example

```
<!DOCTYPE html>

<html>

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

<body>

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

{{ firstName + " " + lastName }}

</div>

<script>

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

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

    $scope.firstName = "Anil";

    $scope.lastName = "Sharma";

});

</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:

```
<!DOCTYPE html>

<html>

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

<body>

<div ng-app="myApp" test></div>

<script>

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

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

    return {

        template : "Acme Group"

    };

});

</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:

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/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 = "John";
    $scope.lastName= "Doe";
});
```

AngularJS Directives

Directives are markers on a DOM element that tell AngularJS to attach a specified behavior to that DOM element or even transform the DOM element and its children. In short, it extends the HTML.

Most of the directives in AngularJS are starting with `ng-` where ng stands for Angular. AngularJS includes various built-in directives. In addition to this, you can create custom directives for your application.

The following table lists the important built-in AngularJS directives.

Directive	Description
ng-app	Auto bootstrap AngularJS application.
ng-init	Initializes AngularJS variables
ng-model	Binds HTML control's value to a property on the $scope object.
ng-controller	Attaches the controller of MVC to the view.
ng-bind	Replaces the value of HTML control with the value of specified AngularJS expression.
ng-repeat	Repeats HTML template once per each item in the specified collection.
ng-show	Display HTML element based on the value of the specified expression.

Directive	Description
ng-readonly	Makes HTML element read-only based on the value of the specified expression.
ng-disabled	Sets the disable attribute on the HTML element if specified expression evaluates to true.
ng-if	Removes or recreates HTML element based on an expression.
ng-click	Specifies custom behavior when an element is clicked.

ng-app:

The ng-app directive initializes AngularJS and makes the specified element a root element of the application. Visit ng-app section for more information.

ng-init:

The ng-init directive can be used to initialize variables in AngularJS application.

The following example demonstrates ng-init directive that initializes variable of string, number, array, and object.

Example: ng-init

```
<!DOCTYPE html>

<html >

<head>

    <script src="~/Scripts/angular.js"></script>

</head>

<body >

    <div ng-app ng-init="greet='Hello World!'; amount= 100; myArr = [100, 200];
person = { firstName:'Steve', lastName :'Jobs'}">
```

```
      {{amount}}      <br />
      {{myArr[1]}}    <br />
      {{person.firstName}}
   </div>
</body>
</html>
```

Result:

```
100
200
Steve
```

In the above example, we initialized variables of string, number, array and object. These variables can be used anywhere in the DOM element hierarchy where it is declared e.g variables in the above example cannot be used out of element.

ng-model:

The ng-model directive is used for two-way data binding in AngularJS. It binds <input>, <select> or <textarea> elements to a specified property on the $scope object. So, the value of the element will be the value of a property and vica-versa.

Example: ng-model

```
<!DOCTYPE html>
<html >
<head>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app>
    <input type="text" ng-model="name" />
    <div>
       Hello {{name}}
    </div>
</body>
</html>
```

The property set via ng-model can be accessed in a controller using $scope object. We will look at it in the next section.

Note : Variables initialized in ng-init are different from the properties defined using ng-model directive. The variables initialized in ng-init are not attached to $scope object whereas ng-model properties are attached to $scope object.

ng-bind:

The ng-bind directive binds the model property declared via $scope or ng-model directive or the result of an expression to the HTML element. It also updates an element if the value of an expression changes.

Example: ng-bind

```
<!DOCTYPE html>

<html >

<head>

   <script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="">

   <div>

      5 + 5 = <span ng-bind="5 + 5"></span> <br />

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

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

   </div>

</body>

</html>
```

In the above example, ng-bind directive binds a result of an expression "5 + 5" to the . The same way, it binds a value of a model property "name" to the . The value of "name" property will be the value entered in a textbox.

ng-repeat:

The ng-repeat directive repeats HTML once per each item in the specified array collection.

Example:

```
<!DOCTYPE html>

<html>

<head>

    <script src="~/Scripts/angular.js"></script>

    <style>

        div {

            border: 1px solid green;

            width: 100%;

            height: 50px;

            display: block;

            margin-bottom: 10px;

            text-align:center;

            background-color:yellow;

        }

    </style>

</head>

<body ng-app="" ng-init="students=['Bill','Steve','Ram']">

    <ol>

        <li ng-repeat="name in students">

            {{name}}

        </li>
```

```
    </ol>
    <div ng-repeat="name in students">
        {{name}}
    </div>
</body>
</html>
```

In the above example, ng-repeat is used with students array. It creates <li> element for each item in the students array. Using the same way it repeats the element.

ng-if:

The ng-if directive creates or removes an HTML element based on the Boolean value returned from the specified expression. If an expression returns true then it recreates an element otherwise removes an element from the HTML document.

ng-readonly:

The ng-readonly directive makes an HTML element read-only, based on the Boolean value returned from the specified expression. If an expression returns true then the element will become read-only, otherwise not.

ng-disabled:

The ng-disabled directive disables an HTML element, based on the Boolean value returned from the specified expression. If an expression returns true the element will be disabled, otherwise not.

The following example demonstrates ng-if, ng-readonly, and ng-disabled directives.

Example: ng-if, ng-readonly, ng-disabled

```
<!DOCTYPE html>
<html>
```

```
<head>
    <script src="~/Scripts/angular.js"></script>
    <style>
        div {
            width: 100%;
            height: 50px;
            display: block;
            margin: 15px 0 0 10px;
        }
    </style>
</head>
<body ng-app ng-init="checked=true" >
    Click Me: <input type="checkbox" ng-model="checked" /> <br />
    <div>
        New: <input ng-if="checked" type="text" />
    </div>
    <div>
        Read-only: <input ng-readonly="checked" type="text" value="This is
read-only." />
    </div>
    <div>
   Disabled: <input ng-disabled="checked" type="text" value="This is
disabled." />
    </div>
</body>
```

```
</html>
```

Directive Syntax:

AngularJS directives can be applied to DOM elements in many ways. It is not mandatory to use `ng-` syntax only.

Directive can start with `x-` or `data-`, for example ng-model directive can be written as data-ng-model or x-ng-model.

Also, the `-` in the directive can be replaced with `:` or `_` or both. For example, ng-model can be written as ng_model or ng:model. It can also be a mix with `data-` or `x-`.

The following example demonstrates all the rules of a directive syntax.

Example: Directives syntax variation

```
<!DOCTYPE html>

<html >

<head>

    <script src="~/Scripts/angular.js"></script>

</head>

<body ng-app>

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

    data-ng-bind: <span data-ng-bind="name"></span><br />

    data-ng:bind: <span data-ng:bind="name"></span><br />

    data:ng:bind: <span data:ng:bind="name"></span><br />

    x:ng:bind:    <span x:ng:bind="name"></span><br />

    ng:bind:      <span ng:bind="name"></span><br />

    x-ng-bind:    <span x-ng-bind="name"></span><br />

    x_ng_bind:    <span x_ng_bind="name"></span><br />
```

```
    ng_bind:      <span ng_bind="name"></span>

</body>

</html>
```

Example :

```
<!DOCTYPE html>

<html>

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

<body>

<div data-ng-app="" data-ng-init="quantity=1;price=5">

<h2>Cost Calculator</h2>

Quantity: <input type="number" ng-model="quantity">

Price: <input type="number" ng-model="price">

<p><b>Total in dollar:</b> {{quantity * price}}</p>

</div>

</body>

</html>
```

Repeating HTML Elements

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<body>
```

```
<div ng-app="" ng-init="names=['Jani','Hege','Kai']">
  <p>Looping with ng-repeat:</p>
  <ul>
    <li ng-repeat="x in names">
      {{ x }}
    </li>
  </ul>
</div>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>
<body>
<div ng-app="" ng-init="names=[
{name:'Jani',country:'Norway'},
{name:'Hege',country:'Sweden'},
{name:'Kai',country:'Denmark'}]">
<p>Looping with objects:</p>
<ul>
  <li ng-repeat="x in names">
  {{ x.name + ', ' + x.country }}</li>
</ul>
```

```
</div>

</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.

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<body ng-app="myApp" test>

<script>

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

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

   return {

      template : "<h1>Acme</h1>"

   };

});

</script>

</body>

</html>
```

ng-model Directive

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

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

Two-Way Binding

```
<!DOCTYPE html>

<html>

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

<body>

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

Name: <input ng-model="name">

<h3>{{name}}</h3>

</div>

<script>

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

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

   $scope.name = "Acme";

});

</script>

</body>

</html>
```

Validate User Input

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

```
<!DOCTYPE html>

<html>

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

<body>

<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>Enter your e-mail address in the input field. AngularJS will display an
errormessage if the address is not an e-mail.</p>

</body>

</html>
```

Application Status

The **`ng-model`** directive can provide status for application data (invalid, dirty, touched, error):

```
<!DOCTYPE html>

<html>
```

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

<body>

<form ng-app="" name="myForm" ng-init="myText = 'post@myweb.com'">



Email:

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

<p>Edit the e-mail address, and try to change the status.</p>

<h1>Status</h1>

<p>Valid: {{myForm.myAddress.$valid}} (if true, the value meets all
criteria).</p>

<p>Dirty: {{myForm.myAddress.$dirty}} (if true, the value has been
changed).</p>

<p>Touched: {{myForm.myAddress.$touched}} (if true, the field has been in
focus).</p>

</form>

</body>

</html>
```

CSS Classes

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

```
<!DOCTYPE html>

<html>

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

```
<style>
input.ng-invalid {
   background-color: red;
}
</style>
<body>
<form ng-app="" name="myForm">
   First Name:
   <input name="firstname" ng-model="firstname" required><br><br>
   Last Name:
   <input name="lastname" ng-model="lastname" required><br><br>
   {{firstname+" "+lastname}}
</form>
<p>Edit the text field and it will get/lose classes according to the status.</p>
<p><b>Note:</b> A text field with the "required" attribute is not valid when it is empty.</p>
</body>
</html>
```

AngularJS Data Binding

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

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

Example :

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

HTML View

The HTML container where the AngularJS application is displayed, is called the view.

The view has access to the model, and there are several ways of displaying model data in the view.

You can use the `ng-bind` directive, which will bind the innerHTML of the element to the specified model property:

```
<p ng-bind="firstname"></p>
```

You can also use double braces `{{ }}` to display content from the model:

```
<p>First name: {{firstname}}</p>
```

Or you can use the `ng-model` directive on HTML controls to bind the model to the view.

The `ng-model` Directive

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

```
<input ng-model="firstname">
```

Example :

```
<!DOCTYPE html>

<html>

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

```
<body>


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

   <input ng-model="firstname">

</div>

<script>

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

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

   $scope.firstname = "John";

   $scope.lastname = "Doe";

});

</script>

<p>Use the ng-model directive on HTML controls (input, select, textarea) to
bind data between the view and the data model.</p>

</body>

</html>
```

AngularJS Controller

AngularJS controllers **control the data** of AngularJS applications.

AngularJS controllers are regular **JavaScript Objects**.

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.

```
<!DOCTYPE html>
```

```
<html>

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

<body>

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

   <h1 ng-click="changeName()">{{firstname}}</h1>

</div>

<script>

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

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

   $scope.firstname = "John";

   $scope.changeName = function() {

      $scope.firstname = "Nelly";

   }

});

</script>

<p>Click on the header to run the "changeName" function.</p>

<p>This example demonstrates how to use the controller to change model
data.</p>

</body>

</html>
```

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**.

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<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 = "John";

   $scope.lastName = "Doe";

});

</script>

</body>

</html>
```

Application explained:

The AngularJS application is defined by **ng-app="myApp"**. The application runs inside the .

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):

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<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 = "John";
   $scope.lastName = "Doe";
   $scope.fullName = function() {
      return $scope.firstName + " " + $scope.lastName;
   };
});
</script>
</body>
</html>
```

Controllers In External Files

In larger applications, it is common to store controllers in external files.

External_file.js

```
angular.module('myApp', []).controller('personCtrl', function($scope) {
   $scope.firstName = "John",
   $scope.lastName = "Doe",
   $scope.fullName = function() {
      return $scope.firstName + " " + $scope.lastName;
   }
});
```

Create another file :

```
<!DOCTYPE html>
<html>
```

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<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="External_file.js"></script>

</body>

</html>
```

Example 2 :

namesController.js

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

   $scope.names = [

      {name:'Jani',country:'Norway'},

      {name:'Hege',country:'Sweden'},

      {name:'Kai',country:'Denmark'}

   ];

});
```

index.html

```
<!DOCTYPE html>

<html>
```

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>

<body>

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

<ul>

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

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

  </li>

</ul>

</div>

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

</body>

</html>
```

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.

Properties made in the controller, can be referred to in the view:

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scri
pt>
```

```
<body>

<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>

<p>The property "carname" was made in the controller, and can be referred to in the view by using the {{ }} brackets.</p>

</body>

</html>
```

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.

```
<!DOCTYPE html>

<html>

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

<body>

<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>

<p>When you change the name in the input field, the changes will affect the
model, and it will also affect the name property in the controller.</p>

</body>

</html>
```

AngularJS Filters

Filters can be added in AngularJS to format data.

AngularJS Filters

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.

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

The `uppercase` filter format strings to upper case:

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

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

</div>
```

Example :

```
<!DOCTYPE html>

<html>

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

<body>

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

<p>The name is {{ (firstName|lowercase) +" "+(lastName | uppercase
)}}</p>

</div>
```

```
<script>

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

    $scope.firstName = "JOHN",

    $scope.lastName = "Doe"

});

</script>

</body>

</html>
```

Adding Filters to Directives

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

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/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:'Jani',country:'Norway'},
      {name:'Carl',country:'Sweden'},
      {name:'Margareth',country:'England'},
      {name:'Hege',country:'Norway'},
      {name:'Joe',country:'Denmark'},
      {name:'Gustav',country:'Sweden'},
      {name:'Birgit',country:'Denmark'},
      {name:'Mary',country:'England'},
      {name:'Kai',country:'Norway'}
      ];
});
</script>
</body>
</html>
```

Result :

Looping with objects:

- Joe, Denmark
- Birgit, Denmark
- Margareth, England
- Mary, England
- Jani, Norway

- Hege, Norway
- Kai, Norway
- Carl, Sweden
- Gustav, Sweden

The currency Filter

The `currency` filter formats a number as currency:

```
<!DOCTYPE html>

<html>

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

<body>

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

<h3>Price: {{ price | currency }}</h3>

</div>

<script>

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

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

   $scope.price = 58;

});

</script>

<p>The currency filter formats a number to a currency format.</p>

</body>

</html>
```

Result :

Price: $58.00

The currency filter formats a number to a currency format.

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.

Example

Return the names that contains the letter "i":

```
<!DOCTYPE html>

<html>

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

<body>

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

<ul>

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

    {{ x }}

  </li>

</ul>

</div>

<script>

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

```
  $scope.names = [
      'Jani','Carl','Margareth','Hege','Joe','Gustav','Birgit','Mary','Kai'
   ];
});
</script>
<p>This example displays only the names containing the letter "i".</p>
</body>
</html>
```

Result :

- Jani
- Birgit
- Kai

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

Filter an Array Based on User Input

By setting the `ng-model` directive on an input field, we can use the value of the input field as an expression in a filter.

Type a letter in the input field, and the list will shrink/grow depending on the match.

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

       'Jani','Carl','Margareth','Hege','Joe','Gustav','Birgit','Mary','Kai'

    ];

});

</script>

<p>The list will only consists of names matching the filter.</p>

</body>

</html>
```

Sort an Array Based on User Input

```
<!DOCTYPE html>

<html>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/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:'Jani',country:'Norway'},
        {name:'Carl',country:'Sweden'},
        {name:'Margareth',country:'England'},
        {name:'Hege',country:'Norway'},
        {name:'Joe',country:'Denmark'},
        {name:'Gustav',country:'Sweden'},
        {name:'Birgit',country:'Denmark'},
```

```
        {name:'Mary',country:'England'},
        {name:'Kai',country:'Norway'}
        ];
    $scope.orderByMe = function(x) {
        $scope.myOrderBy = x;
    }
});
</script>
</body>
</html>
```

Result :

Click the table headers to change the sorting order:

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

Custom Filters

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

```
<!DOCTYPE html>
```

```
<html>

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

<body>

<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>

<p>Make your own filters.</p>

<p>This filter, called "myFormat", will uppercase every other character.</p>

</body>

</html>
```

Result :

- JaNi
- CaRl
- MaRgArEtH
- HeGe
- JoE
- GuStAv
- BiRgIt
- MaRy
- KaI

Make your own filters.

This filter, called "myFormat", will uppercase every other character.