AngularJS Expressions, Modules, Directive and Controller Basics

AngularJS binds data to HTML using **Expressions**

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 }}

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<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>
<div>
  <p>My first expression: {{ 5 + 5 }}</p>
</div>
</body>
</html>
```

You can write expressions wherever you like, AngularJS will simply resolve the expression and return the result.Example: Let AngularJS change the value of CSS properties.
Change the color of the input box below, by changing its value:

```
<div ng-app="" ng-init="myCol='lightblue'">

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

</div>
```

AngularJS Numbers

AngularJS numbers are like JavaScript numbers:

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

Same example using ng-bind:

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

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

AngularJS Strings

AngularJS strings are like JavaScript strings:

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

Same example using ng-bind:

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

AngularJS Objects

AngularJS objects are like JavaScript objects:

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

Same example using ng-bind:

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

AngularJS Arrays

AngularJS arrays are like JavaScript arrays:

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

Same example using ng-bind:

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

AngularJS Expressions vs. JavaScript Expressions

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

AngularJS Modules

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

Creating a Module

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

```
<div ng-app="myApp">...</div>
<script>
var app = angular.module("myApp", []);
</script>
```

The "myApp" parameter refers to an HTML element in which the application will run.
Now you can add controllers, directives, filters, and more, to your AngularJS application.

Adding a Controller

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

Example

```
<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.
For a full reference, visit our AngularJS directive reference.
In addition you can use the module to add your own directives to your applications:

Example

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

Modules and Controllers in Files

It is common in AngularJS applications to put the module and the controllers in JavaScript files.

In this example, "myApp.js" contains an application module definition, while "myCtrl.js" contains the controller:

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 src="myApp.js"></script>
<script src="myCtrl.js"></script>
</body>
</html>
```

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

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

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

myCtrl.js

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

Functions can Pollute the Global Namespace

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

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

When to Load the Library

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

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

Example

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

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

AngularJS Directives

AngularJS 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.

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

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

Data Binding

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

Example

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

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

Repeating HTML Elements

The ng-repeat directive repeats an HTML element:

Example

```
<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:

Example

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

AngularJS is perfect for database CRUD (Create Read Update Delete) applications.
Just imagine if these objects were records from a database.

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.

Read more about the ng-model directive in the next chapter.

Create New Directives

In addition to all the built-in AngularJS directives, you can create your own directives.
New directives are created by using the .directive function.
To invoke the new directive, make an HTML element with the same tag name as the new directive.
When naming a directive, you must use a camel case name, w3TestDirective, but when invoking it, you must use - separated name, w3-test-directive:

Example

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

You can invoke a directive by using:

- Element name
- Attribute
- Class
- Comment

The examples below will all produce the same result:

Element name

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

Attribute

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="myApp">
```

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

Class

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="myApp">
<div class="w3-test-directive"></div>
<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
   return {
      restrict : "C",
      template : "<h1>Made by a directive!</h1>"
   };
});
</script>
<p><strong>Note:</strong> You must add the value "C" to the restrict property to be able to invoke the directive from a class name.</p>
</body>
</html>
```

Comment

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="myApp">
<!-- directive: w3-test-directive -->

<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
   return {
      restrict : "M",
      replace : true,
      template : "<h1>Made by a directive!</h1>"
   };
```

```
});
</script>
<p><strong>Note:</strong> We've added the <strong>replace</strong> property in this example, otherwise the comment would be invisible.</p>
<p><strong>Note:</strong> You must add the value "M" to the <strong>restrict</strong> property to be able to invoke the directive from a comment.</p>
</body>
</html>
```

Restrictions

You can restrict your directives to only be invoked by some of the methods.

Example

By adding a restrict property with the value "A", the directive can only be invoked by attributes:

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

The legal restrict values are:

- E for Element name
- A for Attribute
- C for Class
- M for Comment

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

AngularJS Controllers

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

AngularJS controllers are regular **JavaScript Objects**.

AngularJS Controllers

AngularJS applications are controlled by controllers.

The **ng-controller** directive defines the application controller.

A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

AngularJS Example

```
<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>
```

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):

```
<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">
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.
Just copy the code between the <script> tags into an external file
named personController.js:

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

Another Example

For the next example we will create a new controller file:

```
angular.module('myApp', []).controller('namesCtrl', function($scope) {
  $scope.names = [
    {name:'Jani',country:'Norway'},
    {name:'Hege',country:'Sweden'},
    {name:'Kai',country:'Denmark'}
  ];
});
```

Save the file as namesController.js:
And then use the controller file in an application:

Now use above file named namesController.js

```
<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>
```