# AngularJS Expressions, Modules, Directive, Controller, Scope, Filter Services ,AJAX ,JSON and Form Validations

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>
```

# AngularJS Scope

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

# How to Use the Scope?

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

```
<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>{{carname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.carname = "Volvo";
});
</script>
</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 property name, 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.

```
<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 = "Sangola College";
});
</script>
</body>
</html>
```

## 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.

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="myApp">
<p>color  in  rootScope's :</p>
<h1>{{color}}</h1>
<div ng-controller="myCtrl">
<p>color controller :</p>
<h1>{{color}}</h1>
</div>
<p>still color in rootScope:</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>
</html>
```

## 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.

## Adding Filters to Expressions

Filters can be added to expressions by using the pipe character `|`, followed by a filter.The `uppercase` filter format strings to upper case:

```
<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 {{ lastName | uppercase }}</p>
</div>
<script>
angular.module('myApp', []).controller('personCtrl', function($scope) {
   $scope.firstName = "John",
   $scope.lastName = "Doe"
});
</script>
</body>
</html>
```

The `lowercase` filter format strings to lower case:

```
<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 {{ lastName | lowercase }}</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:

```
<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>
```

The `currency` filter formats a number as currency:

```
<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">
<h1>Price: {{ price | currency }}</h1>
</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>
```

## 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.

```
<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>
```

## 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:

```
<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

```
<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>
```

## Custom Filters

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

```
<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>
</body>
</html>
```

## AngularJS Services

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

## 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:

```
<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">
<p>The url of this page is:</p>
<h3>{{myUrl}}</h3>
</div>
<p>This example uses the built-in $location service to get the absolute url of the
page.</p>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $location) {
  $scope.myUrl = $location.absUrl();
});
</script>
</body>
</html>
```

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

## 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.

```
<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">
<p>Today's welcome message is:</p>
<h1>{{myWelcome}}</h1>
</div>
<p>The $http service requests a page on the server, and the response is set as the value of the "myWelcome" variable.</p>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm").then(function (response) {
     $scope.myWelcome = response.data;
  });
});
</script>
</body>
</html>
```

his example demonstrates a very simple use of the `$http` service.

## The $timeout Service

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

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

```
<h1>{{myHeader}}</h1>
</div>
<p>The $timeout service runs a function after a specified number of
milliseconds.</p>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $timeout) {
  $scope.myHeader = "Hello World!";
  $timeout(function () {
     $scope.myHeader = "How are you today?";
  }, 2000);
});
</script>
</body>
</html>
```

## The $interval Service

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

```
<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">
<p>The time is:</p>
<h1>{{theTime}}</h1>
</div>
<p>The $interval service runs a function every specified millisecond.</p>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $interval) {
  $scope.theTime = new Date().toLocaleTimeString();
  $interval(function () {
     $scope.theTime = new Date().toLocaleTimeString();
  }, 1000);
});
</script>
</body>
</html>
```

## Create Your Own Service

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

```
<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">
<p>The hexadecimal value of 255 is:</p>
<h1>{{hex}}</h1>
</div>
<p>A custom service with a method that converts a given number into a hexadecimal number.</p>
<script>
var app = angular.module('myApp', []);

app.service('hexafy', function() {
   this.myFunc = function (x) {
      return x.toString(16);
   }
});
app.controller('myCtrl', function($scope, hexafy) {
  $scope.hex = hexafy.myFunc(255);
});
</script>
</body>
</html>
```

## 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:

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp">
Convert the number 255, using a custom made service inside a custom made filter:
<h1>{{255 | myFormat}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
```

```
app.service('hexafy', function() {
   this.myFunc = function (x) {
      return x.toString(16);
   }
});
app.filter('myFormat',['hexafy', function(hexafy) {
   return function(x) {
      return hexafy.myFunc(x);
   };
}]);
</script>
</body>
</html>
```

## AngularJS AJAX - $http

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

## AngularJS $http

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

```
<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">
<p>Today's welcome message is:</p>
<h1>{{myWelcome}}</h1>
</div>
<p>The $http service requests a page on the server, and the response is set as the
value of the "myWelcome" variable.</p>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm")
  .then(function(response) {
     $scope.myWelcome = response.data;
  });
});
</script>
</body>
</html>
```

## 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:

```
<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">
<p>Today's welcome message is:</p>
<h1>{{myWelcome}}</h1>
</div>
<p>The $http service requests a page on the server, and the response is set as the
value of the "myWelcome" variable.</p>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
 $http({
   method : "GET",
   url : "welcome.htm"
 }).then(function mySuccess(response) {
     $scope.myWelcome = response.data;
   }, function myError(response) {
     $scope.myWelcome = response.statusText;
 });
});
</script>
</body>
</html>
```

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.

```
<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">
<p>Data : {{content}}</p>
<p>Status : {{statuscode}}</p>
<p>StatusText : {{statustext}}</p>
</div>
<p>The response object has many properties. This example demonstrate the value of
the data, status, and statusText properties.</p>
<script>
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;
  });
});
</script>
</body>
</html>
```

To handle errors, add one more functions to the `.then` method:

```
<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>{{content}}</h1>
</div>
<p>The $http service executes different functions on success and failure.</p>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("wrongfilename.htm")
  .then(function(response) {
      $scope.content = response.data;
  }, function(response) {
      $scope.content = "Something went wrong";
  });
});
</script>
</body>
</html>
```

# JSON

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

customers.php


```
{
  "records": [
    {
      "Name": "Alfreds Futterkiste",
      "City": "Berlin",
      "Country": "Germany"
    },
    {
      "Name": "Ana Trujillo Emparedados y helados",
      "City": "México D.F.",
      "Country": "Mexico"
    },
    {
      "Name": "Antonio Moreno Taquería",
      "City": "México D.F.",
      "Country": "Mexico"
    },
    {
      "Name": "Around the Horn",
      "City": "London",
      "Country": "UK"
    },
    {
      "Name": "B's Beverages",
      "City": "London",
      "Country": "UK"
    },
    {
      "Name": "Berglunds snabbköp",
      "City": "Luleå",
      "Country": "Sweden"
    },
    {
      "Name": "Blauer See Delikatessen",
      "City": "Mannheim",
      "Country": "Germany"
    },
    {
      "Name": "Blondel père et fils",
      "City": "Strasbourg",
      "Country": "France"
    },
    {
      "Name": "Bólido Comidas preparadas",
      "City": "Madrid",
```

```
    "Country": "Spain"
  },
  {
    "Name": "Bon app'",
    "City": "Marseille",
    "Country": "France"
  },
  {
    "Name": "Bottom-Dollar Marketse",
    "City": "Tsawassen",
    "Country": "Canada"
  },
  {
    "Name": "Cactus Comidas para llevar",
    "City": "Buenos Aires",
    "Country": "Argentina"
  },
  {
    "Name": "Centro comercial Moctezuma",
    "City": "México D.F.",
    "Country": "Mexico"
  },
  {
    "Name": "Chop-suey Chinese",
    "City": "Bern",
    "Country": "Switzerland"
  },
  {
    "Name": "Comércio Mineiro",
    "City": "São Paulo",
    "Country": "Brazil"
  }
 ]
}
```

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="customersCtrl">
<ul>
 <li ng-repeat="x in myData">
  {{ x.Name + ', ' + x.Country }}
 </li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
```

```
  $http.get("customers.php").then(function (response) {
     $scope.myData = response.data.records;
  });
});
</script>
</body>
</html>
```

Application explained:

The application defines the customersCtrl controller, with a $scope and $http object.

$http is an XMLHttpRequest object for requesting external data.

$http.get() reads JSON

On success, the controller creates a property, myData, in the scope, with JSON data from the server.

# AngularJS Tables

## Displaying Data in a Table

Displaying tables with angular is very simple:

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="customersCtrl">
<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
    $http.get("customers.php")
    .then(function (response) {$scope.names = response.data.records;});
});
</script>
</body>
</html>
```

## AngularJS Forms

Forms in AngularJS provides data-binding and validation of input controls.

### input Controls

Input controls are the HTML input elements:

- input elements
- select elements
- button elements
- textarea elements

## Data-Binding

Input controls provides data-binding by using the ng-model directive.

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

The application does now have a property named firstname.

The ng-model directive binds the input controller to the rest of your application.

The property firstname, can be referred to in a controller:

```
<html lang="en">
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="formCtrl">
 <form>
   First Name: <input type="text" ng-model="firstname">
 </form>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
   $scope.firstname = "John";
});
</script>
</body>
</html>
```

## AngularJS Form Validation

AngularJS can validate input data.

## Form Validation

AngularJS offers client-side form validation.

AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.

AngularJS also holds information about whether they have been touched, or modified, or not.

You can use standard HTML5 attributes to validate input, or you can make your own validation functions.

Client-side validation cannot alone secure user input. Server side validation is also necessary.

## Required

Use the HTML5 attribute `required` to specify that the input field must be filled out:

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="">
<p>Try writing in the input field:</p>
<form name="myForm">
<input name="myInput" ng-model="myInput" required>
```

```
</form>
<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
</body>
</html>
```

## E-mail

Use the HTML5 type `email` to specify that the value must be an e-mail:

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="">
<p>Try writing an E-mail address in the input field:</p>
<form name="myForm">
<input type="email" name="myInput" ng-model="myInput">
</form>
<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
<p>Note that the state of the input field is "true" before you start writing in it, even if it does not contain an e-mail address.</p>
</body>
</html>
```

## Form State and Input State

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

- `$untouched` The field has not been touched yet
- `$touched` The field has been touched
- `$pristine` The field has not been modified yet
- `$dirty` The field has been modified
- `$invalid` The field content is not valid
- `$valid` The field content is valid

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

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="">
<p>Try leaving the first input field blank:</p>
<form name="myForm">
<p>Name:
<input name="myName" ng-model="myName" required>
```

```
<span ng-show="myForm.myName.$touched && myForm.myName.$invalid">The
name is required.</span>
</p>
<p>Address:
<input name="myAddress" ng-model="myAddress" required>
</p>
</form>
<p>We use the ng-show directive to only show the error message if the field has been
touched AND is empty.</p>
</body>
</html>
```

## 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`

The classes are removed if the value they represent is `false`.
Add styles for these classes to give your application a better and more intuitive user interface.

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<style>
input.ng-invalid {
  background-color:pink;
}
input.ng-valid {
  background-color:lightgreen;
}
</style>
<body ng-app="">
<p>Try writing in the input field:</p>
<form name="myForm">
<input name="myName" ng-model="myName" required>
</form>
```

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

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

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<style>
form.ng-pristine {
  background-color:lightblue;
}
form.ng-dirty {
  background-color:pink;
}
</style>
<body ng-app="">
<form name="myForm">
<p>Try writing in the input field:</p>
<input name="myName" ng-model="myName" required>
<p>The form gets a "ng-dirty" class when the form has been modified, and will therefore turn pink.</p>
</form>
</body>
</html>
```

## 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":

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="myApp">
<p>Try writing in the input field:</p>
<form name="myForm">
<input name="myInput" ng-model="myInput" required my-directive>
</form>
<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
<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>
<p>The input field must contain the character "e" to be consider valid.</p>
</body>
</html>
```

## Validation Example

```
<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.**