Created By www.ebooktutorials.in

www.ebooktutorials.in

AngularJS Tutorial

TIP

Press **CTRL+B** or **CLICK Bookmarks** in your PDF reader for easy navigation

Content Downloaded from www.w3schools.com

Created By www.ebooktutorials.in

AngularJS Tutorial

AngularJS extends HTML with new attributes.

AngularJS is perfect for Single Page Applications (SPAs).

AngularJS is easy to learn.

This Tutorial

This tutorial is specially designed to help you learn AngularJS as quickly and efficiently as possible.

First, you will learn the basics of AngularJS: directives, expressions, filters, modules, and controllers.

Then you will learn everything else you need to know about AngularJS:

Events, DOM, Forms, Input, Validation, Http, and more.

Content Downloaded from www.w3schools.com

AngularJS Example

```
<!DOCTYPE html>
<html lang="en-US">
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.
js"></script>
<body>

<div ng-app="">
  <p>Name : <input type="text" ng-model="name"></p>
  <h1>Hello {{name}}</h1>
</div>

</body>
</html>
```

What You Should Already Know

Before you study AngularJS, you should have a basic understanding of:

- HTML
- CSS
- JavaScript

AngularJS History

AngularJS version 1.0 was released in 2012.

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

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

Content Downloaded from www.w3schools.com

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:

```
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/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.

AngularJS Example

```
<!DOCTYPE html>
<html>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.
js"></script>
<body>
<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div></body></html>
```

Content Downloaded from www.w3schools.com

Example explained:

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 initialize AngularJS application variables.

AngularJS Example

```
<div ng-app="" ng-init="firstName='John'">

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

</div>
```

Alternatively with valid HTML:

AngularJS Example

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

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

</div>
```

You can use **data-ng-**, instead of **ng-**, if you want to make your page HTML valid.

You will learn a lot more about directives later in this tutorial.

AngularJS Expressions

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

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

AngularJS Example

```
<!DOCTYPE html>
<html>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.
js"></script>
<body>
<div ng-app="">
  <p>My first expression: {{ 5 + 5 }}</p>
</div>
</body>
</html>
```

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

AngularJS Example

```
<!DOCTYPE html>
<html>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.
js"></script>
<body>
<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p>{{name}}</p>
</div>
</body>
</html>
```

You will learn more about expressions later in this tutorial.

Content Downloaded from www.w3schools.com

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.

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>
```

AngularJS modules define applications:

AngularJS Module

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

AngularJS controllers control applications:

AngularJS Controller

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

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 }}

Example

```
<!DOCTYPE html>
<html>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.
js"></script>
<body>

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

</body>
</html>
```

Content Downloaded from www.w3schools.com

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

Example

```
<!DOCTYPE html>
<html>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/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 it's value:

Example

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

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

</div>
```

AngularJS Numbers

AngularJS numbers are like JavaScript numbers:

Example

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

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

</div>
```

Same example using **ng-bind**:

Example

```
<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:

Example

```
<div ng-app="" ng-init="firstName='John';lastName='Doe'">

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

</div>
```

Same example using `ng-bind`:

Example

```
<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:

Example

```
<div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">

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

</div>
```

Same example using **ng-bind**:

Example

```
<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:

Example

```
<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**:

Example

```
<div ng-app="" ng-init="points=[1,15,19,2,40]">

<p>The third result is <span ng-bind="points[2]"></span></p>

</div>
```

Content Downloaded from www.w3schools.com

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.

Content Downloaded from www.w3schools.com

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>
```

You will learn more about controllers later in this tutorial.

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:

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>
```

Content Downloaded from www.w3schools.com

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>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/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";
});
```

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>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/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 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.

Read about all AngularJS directives in our AngularJS directive reference.

Example

```
<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.

Content Downloaded from www.w3schools.com

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.

Content Downloaded from www.w3schools.com

Create New Directives

In addition to all the built-in AngularJS directives, you can create your own directives.

New directives are created by using the `.directive` function.

To invoke the new directive, make an HTML element with the same tag name as the new directive.

When naming a directive, you must use a camel case name, `w3TestDirective`, but when invoking it, you must use `-` separated name, `w3-test-directive`:

Example

```
<body ng-app="myApp">

<w3-test-directive></w3-test-directive>

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

</body>
```

You can invoke a directive by using:

- Element name
- Attribute
- Class
- Comment

The examples below will all produce the same result:

Element name

```
<w3-test-directive></w3-test-directive>
```

Content Downloaded from www.w3schools.com

Attribute

```
<div w3-test-directive></div>
```

Class

```
<div class="w3-test-directive"></div>
```

Comment

```
<!-- directive: w3-test-directive -->
```

Restrictions

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

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

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

The legal restrict values are:

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

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

Content Downloaded from www.w3schools.com

AngularJS ng-model Directive

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

The ng-model Directive

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

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
    Name: <input ng-model="name">
</div>

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

Two-Way Binding

The binding goes both ways. If the user changes the value inside the input field, the AngularJS property will also change it's value:

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
    Name: <input ng-model="name">
    <h1>You entered: {{name}}</h1>
</div>
```

Content Downloaded from www.w3schools.com

Validate User Input

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

Example

```
<form ng-app="" name="myForm">
    Email:
    <input type="email" name="myAddress" ng-model="text">
    <span ng-show="myForm.myAddress.$error.email">Not a valid e-mail
address</span>
</form>
```

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

If the property in the ng-model attribute does not exist, AngularJS will create one for you.

Application Status

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

Example

```
<form ng-app="" name="myForm" ng-init="myText = 'post@myweb.com'">
    Email:
    <input type="email" name="myAddress" ng-model="myText" required></p>
    <h1>Status</h1>
    {{myForm.myAddress.$valid}}
    {{myForm.myAddress.$dirty}}
    {{myForm.myAddress.$touched}}
</form>
```

Content Downloaded from www.w3schools.com

CSS Classes

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

Example

```
<style>
input.ng-invalid {
    background-color: lightblue;
}
</style>
<body>
<form ng-app="" name="myForm">
    Enter your name:
    <input name="myAddress" ng-model="text" required>
</form>
```

The **ng-model** directive adds/removes the following classes, according to the status of the form field:

- ng-empty
- ng-not-empty
- ng-touched
- ng-untouched
- ng-valid
- ng-invalid
- ng-dirty
- ng-pending
- ng-pristine

Content Downloaded from www.w3schools.com

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.

Content Downloaded from www.w3schools.com

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):

AngularJS Example

```
<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>
```

Content Downloaded from www.w3schools.com

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:

AngularJS Example

```
<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: {{firstName + " " + lastName}}

</div>

<script src="personController.js"></script>
```

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:

AngularJS Example

```
<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>
```

Content Downloaded from www.w3schools.com

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:

Example

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

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

When adding properties to the `$scope` object in the controller, the view (HTML) gets access to these properties.

In the view, you do not use the prefix `$scope`, you just refer to a propertyname, like `{{carname}}`.

Content Downloaded from www.w3schools.com

Understanding the Scope

If we consider an AngularJS application to consist of:

- View, which is the HTML.
- Model, which is the data available for the current view.
- Controller, which is the JavaScript function that makes/changes/removes/controls the data.

Then the scope is the Model.

The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.

Example

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

```
<div ng-app="myApp" ng-controller="myCtrl">
<input ng-model="name">
<h1>My name is {{name}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.name = "John Dow";
});
</script>
```

Know Your Scope

It is important to know which scope you are dealing with, at any time.

In the two examples above there is only one scope, so knowing your scope is not an issue, but for larger applications there can be sections in the HTML DOM which can only access certain scopes.

Content Downloaded from www.w3schools.com

Example

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

```
<div ng-app="myApp" ng-controller="myCtrl">
<ul>
    <li ng-repeat="x in names">{{x}}</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.names = ["Emil", "Tobias", "Linus"];
});</script>
```

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

Root Scope

All applications have a **$rootScope** which is the scope created on the HTML element that contains the **ng-app** directive.

The rootScope is available in the entire application.

If a variable has the same name in both the current scope and in the rootScope, the application use the one in the current scope.

Example

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

```
<body ng-app="myApp">
<p>The rootScope's favorite color:</p>
<h1>{{color}}</h1>
<div ng-controller="myCtrl">
    <p>The scope of the controller's favorite color:</p>
    <h1>{{color}}</h1>
</div>
<p>The rootScope's favorite color is still:</p>
<h1>{{color}}</h1>
<script>
var app = angular.module('myApp', []);
app.run(function($rootScope) {
    $rootScope.color = 'blue';
});
app.controller('myCtrl', function($scope) {
    $scope.color = "red";
});</script></body>
```

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.

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:

Example

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

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

</div>
```

The **lowercase** filter format strings to lower case:

Example

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

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

</div>
```

Adding Filters to Directives

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

Example

The `orderBy` filter sorts an array:

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

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

The currency Filter

The **currency** filter formats a number as currency:

Example

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

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

</div>
```

Content Downloaded from www.w3schools.com

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":

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

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

</div>
```

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:

- Jani
- Carl
- Margareth
- Hege
- Joe
- Gustav
- Birgit
- Mary
- Kai

Content Downloaded from www.w3schools.com

Example

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

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

<ul>
  <li ng-repeat="x in names | filter : test">
    {{ x }}
  </li>
</ul>

</div>
```

Sort an Array Based on User Input

Click the table headers to change the sort order::

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

Content Downloaded from www.w3schools.com

By adding the **ng-click** directive on the table headers, we can run a function that changes the sorting order of the array:

Example

```
<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>
```

Content Downloaded from www.w3schools.com

Custom Filters

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

Example

Make a custom filter called "myFormat":

```
<ul ng-app="myApp" ng-controller="namesCtrl">
    <li ng-repeat="x in names">
        {{x | myFormat}}
    </li>
</ul>

<script>
var app = angular.module('myApp', []);
app.filter('myFormat', function() {
    return function(x) {
        var i, c, txt = "";
        x = x.split("")
        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>
```

Content Downloaded from www.w3schools.com

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:

Example

Use the `$location` service in a controller:

```
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $location) {
    myUrl = $location.absUrl();
});
```

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

Why use Services?

For many services, like the `$location` service, it seems like you could use objects that are already in the DOM, like the `window.location` object, and you could, but it would have some limitations, at least for your AngularJS application.

Content Downloaded from www.w3schools.com

AngularJS constantly supervise 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.

Example

Use the **$http** service to request data from the server:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
    $http.get("welcome.htm").then(function (response) {
        $scope.myWelcome = response.data;
    });
});
```

This example demonstrates a very simple use of the **$http** service.

The $timeout Service

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

Example

Display a new message after two seconds:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $timeout) {
    $scope.myHeader = "Hello World!";
    $timeout(function () {
        $scope.myHeader = "How are you today?";
    }, 2000);
});
```

Content Downloaded from www.w3schools.com

The $interval Service

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

Example
Display the time every second:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $interval) {
    $scope.theTime = new Date().toLocaleTimeString();
    $interval(function () {
        $scope.theTime = new Date().toLocaleTimeString();
    }, 1000);
});
```

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 filter:

Example
Use the custom made service named **hexafy** to convert a number into a hexadecimal number:

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

Content Downloaded from www.w3schools.com

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:

The service **hexafy** used in the filter **myFormat**:

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

You can use the filter when displaying values from an object, or an array:

Create a service named **hexafy**:

```
<ul>
<li ng-repeat="x in counts">{{x | myFormat}}</li>
</ul>
```

Content Downloaded from www.w3schools.com

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.

Example
Make a simple request to the server, and display the result in a header:

```
<div ng-app="myApp" ng-controller="myCtrl">
<p>Today's welcome message is:</p>
<h1>{{myWelcome}}</h1></div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
    $http.get("welcome.htm")
    .then(function(response) {
        $scope.myWelcome = response.data;
    });
});</script>
```

Methods

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

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

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

Content Downloaded from www.w3schools.com

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

Example

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
    $http({
        method : "GET",
        url : "welcome.htm"
    }).then(function mySucces(response) {
        $scope.myWelcome = response.data;
    }, function myError(response) {
        $scope.myWelcome = response.statusText;
    });
});
```

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

Example

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

To handle errors, add one more function to the **.then** method:

Example

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
    $http.get("wrongfilename.htm")
    .then(function(response) {
        //First function handles success
        $scope.content = response.data;
    }, function(response) {
        //Second function handles error
        $scope.content = "Something went wrong";
    });
});
```

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**.

Example

The **ng-repeat** directive is perfect for looping through an array:

```
<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>
```

Content Downloaded from www.w3schools.com

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 data** from http://www.w3schools.com/angular/customers.php.

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

AngularJS Tables

The ng-repeat directive is perfect for displaying tables.

Displaying Data in a Table

Displaying tables with angular is very simple:

AngularJS Example

```
<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) {
    $http.get("http://www.w3schools.com/angular/customers.php")
    .then(function (response) {$scope.names = response.data.records;});
});</script>
```

Displaying with CSS Style

To make it nice, add some CSS to the page:

CSS Style

```
<style>
table, th , td {
  border: 1px solid grey;
  border-collapse: collapse;
  padding: 5px;
}
table tr:nth-child(odd) {
  background-color: #f1f1f1;
}
table tr:nth-child(even) {
  background-color: #ffffff;
}</style>
```

Display with orderBy Filter

To sort the table, add an **orderBy** filter:

AngularJS Example

```
<table>
  <tr ng-repeat="x in names | orderBy : 'Country'">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
```

Display with uppercase Filter

To display uppercase, add an **uppercase** filter:

AngularJS Example

```
<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country | uppercase }}</td>
  </tr>
</table>
```

Display the Table Index ($index)

To display the table index, add a <td> with **$index**:

AngularJS Example

```
<table>
  <tr ng-repeat="x in names">
    <td>{{ $index + 1 }}</td>
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
```

Content Downloaded from www.w3schools.com

Using $even and $odd

AngularJS Example

```
<table>
<tr ng-repeat="x in names">
<td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Name }}</td>
<td ng-if="$even">{{ x.Name }}</td>
<td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Country }}</td>
<td ng-if="$even">{{ x.Country }}</td>
</tr>
</table>
```

AngularJS Select Boxes

AngularJS lets you create dropdown lists based on items in an array, or an object.

Creating a Select Box Using ng-options

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

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
<select ng-model="selectedName" ng-options="x for x in names">
</select>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.names = ["Emil", "Tobias", "Linus"];
});
</script>
```

ng-options vs ng-repeat

You can also use the **ng-repeat** directive to make the same dropdown list:

Example

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

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

Content Downloaded from www.w3schools.com

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

What Do I Use?

Assume you have an array of objects:

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

The **ng-repeat** directive has it's limitations, the selected value must be a string:

Example

Using **ng-repeat**:

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

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

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

Example

Using **ng-options**:

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

<h1>You selected: {{selectedCar.model}}</h1>
<p>It's color is: {{selectedCar.color}}</p>
```

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

We will use the **ng-options** directive in this tutorial.

Content Downloaded from www.w3schools.com

The Data Source as an Object

In the previous examples the data source was an array, but we can also use an object.

Assume you have an object with key-value pairs:

```
$scope.cars = {
    car01 : "Ford",
    car02 : "Fiat",
    car03 : "Volvo"
};
```

The expression in the **ng-options** attribute is a bit different for objects:

Example

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

```
<select ng-model="selectedCar" ng-options="x for (x, y) in cars">
</select>

<h1>You selected: {{selectedCarl}}</h1>
```

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

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

Example

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

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

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

Example

```
<select ng-model="selectedCar" ng-options="y.brand for (x, y) in cars">
</select>
```

Content Downloaded from www.w3schools.com

The code from the previous chapter can also be used to read from databases.

Fetching Data From a PHP Server Running MySQL

AngularJS Example

```
<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) {
    $http.get("http://www.w3schools.com/angular/customers_mysql.php")
    .then(function (response) {$scope.names = response.data.records;});
});
</script>
```

Content Downloaded from www.w3schools.com

Fetching Data From an ASP.NET Server Running SQL

AngularJS Example

```
<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) {
    $http.get("http://www.w3schools.com/angular/customers_sql.aspx")
    .then(function (response) {$scope.names = response.data.records;});
});
</script>
```

Server Code Examples

The following section is a listing of the server code used to fetch SQL data.

1. Using PHP and MySQL. Returning JSON.
2. Using PHP and MS Access. Returning JSON.
3. Using ASP.NET, VB, and MS Access. Returning JSON.
4. Using ASP.NET, Razor, and SQL Lite. Returning JSON.

Content Downloaded from www.w3schools.com

Cross-Site HTTP Requests

Requests for data from a different server (than the requesting page), are called **cross-site** HTTP requests.

Cross-site requests are common on the web. Many pages load CSS, images, and scripts from different servers.

In modern browsers, cross-site HTTP requests **from scripts** are restricted to **same site** for security reasons.

The following line, in our PHP examples, has been added to allow cross-site access.

```
header("Access-Control-Allow-Origin: *");
```

1. Server Code PHP and MySQL

```
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

$conn = new mysqli("myServer", "myUser", "myPassword", "Northwind");

$result = $conn->query("SELECT CompanyName, City, Country FROM Customers");

$outp = "";
while($rs = $result->fetch_array(MYSQLI_ASSOC)) {
    if ($outp != "") {$outp .= ",";}
    $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';
    $outp .= '"City":"'   . $rs["City"]        . '",';
    $outp .= '"Country":"'. $rs["Country"]     . '"}';
}
$outp ='{"records":['.$outp.']}';
$conn->close();

echo($outp);
?>
```

2. Server Code PHP and MS Access

```
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=ISO-8859-1");

$conn = new COM("ADODB.Connection");
$conn->open("PROVIDER=Microsoft.Jet.OLEDB.4.0;Data Source=Northwind.mdb");

$rs = $conn->execute("SELECT CompanyName, City, Country FROM Customers");

$outp = "";
while (!$rs->EOF) {
    if ($outp != "") {$outp .= ",";}
    $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';
    $outp .= '"City":"'   . $rs["City"]        . '",';
    $outp .= '"Country":"'. $rs["Country"]     . '"}';
    $rs->MoveNext();
}
$outp ='{"records":['.$outp.']}';

$conn->close();

echo ($outp);
?>
```

3. Server Code ASP.NET, VB and MS Access

```
<%@ Import Namespace="System.IO"%>
<%@ Import Namespace="System.Data"%>
<%@ Import Namespace="System.Data.OleDb"%>
<%
Response.AppendHeader("Access-Control-Allow-Origin", "*")
Response.AppendHeader("Content-type", "application/json")
Dim conn As OleDbConnection
Dim objAdapter As OleDbDataAdapter
Dim objTable As DataTable
Dim objRow As DataRow
Dim objDataSet As New DataSet()
Dim outp
Dim c
conn = New OledbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data
```

```
source=Northwind.mdb")
objAdapter = New OledbDataAdapter("SELECT CompanyName, City, Country FROM
Customers", conn)
objAdapter.Fill(objDataSet, "myTable")
objTable=objDataSet.Tables("myTable")

outp = ""
c = chr(34)
for each x in objTable.Rows
if outp <> "" then outp = outp & ","
outp = outp & "{" & c & "Name"    & c & ":" & c & x("CompanyName") & c & ","
outp = outp &       c & "City"    & c & ":" & c & x("City")        & c & ","
outp = outp &       c & "Country" & c & ":" & c & x("Country")     & c & "}"
next

outp ="{" & c & "records" & c & ":[" & outp & "]}"
response.write(outp)
conn.close
%>
```

4. Server Code ASP.NET, Razor C# and SQL Lite

```
@{
Response.AppendHeader("Access-Control-Allow-Origin", "*")
Response.AppendHeader("Content-type", "application/json")
var db = Database.Open("Northwind");
var query = db.Query("SELECT CompanyName, City, Country FROM Customers");
var outp =""
var c = chr(34)
}
@foreach(var row in query)
{
if outp <> "" then outp = outp + ","
outp = outp + "{" + c + "Name"    + c + ":" + c + @row.CompanyName + c + ","
outp = outp +       c + "City"    + c + ":" + c + @row.City        + c + ","
outp = outp +       c + "Country" + c + ":" + c + @row.Country     + c + "}"
}
outp ="{" + c + "records" + c + ":[" + outp + "]}"
@outp
```

Content Downloaded from www.w3schools.com

AngularJS HTML DOM

AngularJS has directives for binding application data to the attributes of HTML DOM elements.

The ng-disabled Directive

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

AngularJS Example

```
<div ng-app="" ng-init="mySwitch=true">

<p>
<button ng-disabled="mySwitch">Click Me!</button>
</p>

<p>
<input type="checkbox" ng-model="mySwitch">Button
</p>

<p>
{{ mySwitch }}
</p>

</div>
```

Application explained:

The **ng-disabled** directive binds the application data **mySwitch** to the HTML button's **disabled**attribute.

The **ng-model** directive binds the value of the HTML checkbox element to the value of**mySwitch**.

If the value of **mySwitch** evaluates to **true**, the button will be disabled:

Content Downloaded from www.w3schools.com

```
<p>
<button disabled>Click Me!</button>
</p>
```

If the value of **mySwitch** evaluates to **false**, the button will not be disabled:

```
<p>
<button>Click Me!</button>
</p>
```

The ng-show Directive

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

AngularJS Example

```
<div ng-app="">

<p ng-show="true">I am visible.</p>

<p ng-show="false">I am not visible.</p>

</div>
```

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

You can use any expression that evaluates to true or false:

AngularJS Example

```
<div ng-app="" ng-init="hour=13">

<p ng-show="hour > 12">I am visible.</p>

</div>
```

In the next chapter, there are more examples, using the click of a button to hide HTML elements.

Content Downloaded from www.w3schools.com

The ng-hide Directive

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

AngularJS Example

```
<div ng-app="">

<p ng-hide="true">I am not visible.</p>

<p ng-hide="false">I am visible.</p>

</div>
```

AngularJS has its own HTML events directives.

AngularJS Events

You can add AngularJS event listeners to your HTML elements by using one or more of these directives:

- **ng-blur**
- **ng-change**
- **ng-click**
- **ng-copy**
- **ng-cut**
- **ng-dblclick**
- **ng-focus**
- **ng-keydown**
- **ng-keypress**
- **ng-keyup**
- **ng-mousedown**
- **ng-mouseenter**
- **ng-mouseleave**
- **ng-mousemove**
- **ng-mouseover**
- **ng-mouseup**
- **ng-paste**

The event directives allows us to run AngularJS functions at certain user events.

An AngularJS event will not overwrite an HTML event, both events will be executed.

Content Downloaded from www.w3schools.com

Mouse Events

Mouse events occur when the cursor moves over an element, in this order:

1. ng-mouseenter
2. ng-mouseover
3. ng-mousemove
4. ng-mouseleave

Or when a mouse button is clicked on an element, in this order:

1. ng-mousedown
2. ng-mouseup
3. ng-click

You can add mouse events on any HTML element.

Example

Increase the count variable when the mouse moves over the DIV element:

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

<div ng-mousemove="count = count + 1">Mouse over me!</button>

<h1>{{ count }}</h1>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.count = 0;
});
</script>
```

The ng-click Directive

The **ng-click** directive defines AngularJS code that will be executed when the element is being clicked.

Example

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

<button ng-click="count = count + 1">Click me!</button>

<p>{{ count }}</p>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.count = 0;
});
</script>
```

You can also refer to a function:

Example

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

<button ng-click="myFunction()">Click me!</button>

<p>{{ count }}</p>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.count = 0;
    $scope.myFunction = function() {
        $scope.count++;
    }
});
</script>
```

Content Downloaded from www.w3schools.com

Toggle, True/False

If you want to show a section of HTML code when a button is clicked, and hide when the button is clicked again, like a dropdown menu, make the button behave like a toggle switch:

Click Me

Example

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

<button ng-click="myFunc()">Click Me!</button>

<div ng-show="showMe">
    <h1>Menu:</h1>
    <div>Pizza</div>
    <div>Pasta</div>
    <div>Pesce</div>
</div>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.showMe = false;
    $scope.myFunc = function() {
        $scope.showMe = !$scope.showMe;
    }
});
</script>
```

The **showMe** variable starts out as the Boolean value **false**.

The **myFunc** function sets the **showMe** variable to the opposite of what it is, by using the **!** (not) operator.

$event Object

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

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

Content Downloaded from www.w3schools.com

Example

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

<h1 ng-mousemove="myFunc($event)">Mouse Over Me!</h1>

<p>Coordinates: {{x + ', ' + y}}</p>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.myFunc = function(myE) {
        $scope.x = myE.clientX;
        $scope.y = myE.clientY;
    }
});
</script>
```

Content Downloaded from www.w3schools.com

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:

Example

```
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
    $scope.firstname = "John";
});
</script>
```

It can also be referred to elsewhere in the application:

Example

```
<form>
First Name: <input type="text" ng-model="firstname">
</form>

<h1>You entered: {{firstname}}</h1>
```

Checkbox

A checkbox has the value **true** or **false**. Apply the **ng-model** directive to a checkbox, and use it's value in your application.

Example

Show the header if the checkbox is checked:

```
 <form>
    Check to show a header:
    <input type="checkbox" ng-model="myVar">
</form>

<h1 ng-show="myVar">My Header</h1>
```

Radiobuttons

Bind radio buttons to your application with the **ng-model** directive.

Radio buttons with the same **ng-model** can have different values, but only the selected one will be used.

Example

Display some text, based on the value of the selected radio button:

```
<form>
Pick a topic:
<input type="radio" ng-model="myVar" value="dogs">Dogs
<input type="radio" ng-model="myVar" value="tuts">Tutorials
<input type="radio" ng-model="myVar" value="cars">Cars
</form>
```

Content Downloaded from www.w3schools.com

The value of myVar will be either **dogs**, **tuts**, or **cars**.

Selectbox

Bind select boxes to your application with the **ng-model** directive.

The property defined in the **ng-model** attribute will have the value of the selected option in the selectbox.

Example
Display some text, based on the value of the selected option:

```
<form>
Select a topic:
<select ng-model="myVar">
    <option value="">
    <option value="dogs">Dogs
    <option value="tuts">Tutorials
    <option value="cars">Cars
</select>
</form>
```

The value of myVar will be either **dogs**, **tuts**, or **cars**.

An AngularJS Form Example

First Name:

Last Name:

RESET

form = {"firstName":"John","lastName":"Doe"}

master = {"firstName":"John","lastName":"Doe"}

Application Code

```
<div ng-app="myApp" ng-controller="formCtrl">
  <form novalidate>
    First Name:<br>
    <input type="text" ng-model="user.firstName"><br>
    Last Name:<br>
    <input type="text" ng-model="user.lastName">
    <br><br>
    <button ng-click="reset()">RESET</button>
  </form>
  <p>form = {{user}}</p>
  <p>master = {{master}}</p>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
    $scope.master = {firstName: "John", lastName: "Doe"};
    $scope.reset = function() {
        $scope.user = angular.copy($scope.master);
    };
    $scope.reset();
});
</script>
```

The novalidate **attribute is new in HTML5. It disables any default browser validation.**

Example Explained

The **ng-app** directive defines the AngularJS application.

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

The **ng-model** directive binds two input elements to the **user** object in the model.

The **formCtrl** controller sets initial values to the **master** object, and defines the **reset()** method.

The **reset()** method sets the **user** object equal to the **master** object.

The **ng-click** directive invokes the **reset()** method, only if the button is clicked.

The novalidate attribute is not needed for this application, but normally you will use it in AngularJS forms, to override standard HTML5 validation.

Content Downloaded from www.w3schools.com

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:

Example

The input field is required:

```
<form name="myForm">
<input name="myInput" ng-model="myInput" required>
</form>

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

E-mail

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

Example
The input field has to be an e-mail:

```
<form name="myForm">
<input name="myInput" ng-model="myInput" type="email">
</form>

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

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**.

Content Downloaded from www.w3schools.com

You can use these states to show meaningful messages to the user. Example, if a field is required, and the user leaves it blank, you should give the user a warning:

Example
Show an error message if the field has been touched AND is empty:
`<input name="myName" ng-model="myName" required>` `<span ng-show="myForm.myName.$touched && myForm.myName.$invalid">The name is required.</span>`

CSS Classes

AngularJS adds CSS classes to forms and input fields depending on their states.

The following classes are added to, or removed from, input fields:

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

The following classes are added to, or removed from, forms:

- **ng-pristine** No fields has not been modified yet
- **ng-dirty** One or more fields has been modified
- **ng-valid** The form content is valid
- **ng-invalid** The form content is not valid
- **ng-valid-*key*** One *key* for each validation. Example: **ng-valid-required**, useful when there are more than one thing that must be validated
- **ng-invalid-*key*** Example: **ng-invalid-required**

Content Downloaded from www.w3schools.com

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.

Example
Apply styles, using standard CSS:

```
<style>
input.ng-invalid {
    background-color: pink;
}
input.ng-valid {
    background-color: lightgreen;
}
</style>
```

Forms can also be styled:

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

```
<style>
form.ng-pristine {
    background-color: lightblue;
}
form.ng-dirty {
    background-color: pink;
}
</style>
```

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.

Example

Create your own directive, containing a custom validation function, and refer to it by using **my-directive**.

The field will only be valid if the value contains the character "e":

```
<form name="myForm">
<input name="myInput" ng-model="myInput" required my-directive>
</form>
<script>
var app = angular.module('myApp', []);
app.directive('myDirective', function() {
  return {
    require: 'ngModel',
    link: function(scope, element, attr, mCtrl) {
      function myValidation(value) {
        if (value.indexOf("e") > -1) {
          mCtrl.$setValidity('charE', true);
        } else {
          mCtrl.$setValidity('charE', false);
        }
        return value;
      }
      mCtrl.$parsers.push(myValidation);
    }
  };
});
</script>
```

Example Explained:

In HTML, the new directive will be referred to by using the attribute **my-directive**.

In the JavaScript we start by adding a new directive named **myDirective**.

Remember, when naming a directive, you must use a camel case name, **myDirective**, but when invoking it, you must use `-` separated name, **my-directive**.

Then, return an object where you specify that we require **ngModel**, which is the ngModelController.

Content Downloaded from www.w3schools.com

Make a linking function which takes some arguments, where the fourth argument, **mCtrl**, is the **ngModelController**,

Then specify a function, in this case named **myValidation**, which takes one argument, this argument is the value of the input element.

Test if the value contains the letter "e", and set the validity of the model controller to either **true** or **false**.

At last, **mCtrl.$parsers.push(myValidation);** will add the **myValidation** function to an array of other functions, which will be executed every time the input value changes.

Validation Example

```
<!DOCTYPE html>
<html>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/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.**

Example Explained

The AngularJS directive **ng-model** binds the input elements to the model.

The model object has two properties: **user** and **email**.

Because of **ng-show**, the spans with color:red are displayed only when user or email is **$dirty**and **$invalid**.

API stands for **A**pplication **P**rogramming **I**nterface.

AngularJS Global API

The AngularJS Global API is a set of global JavaScript functions for performing common tasks like:

- Comparing objects
- Iterating objects
- Converting data

The Global API functions are accessed using the angular object.

Below is a list of some common API functions:

API	**Description**
angular.lowercase()	Converts a string to lowercase
angular.uppercase()	Converts a string to uppercase
angular.isString()	Returns true if the reference is a string
angular.isNumber()	Returns true if the reference is a number

Content Downloaded from www.w3schools.com

angular.lowercase()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
<p>{{ x1 }}</p>
<p>{{ x2 }}</p>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
$scope.x1 = "JOHN";
$scope.x2 = angular.lowercase($scope.x1);
});
</script>
```

angular.uppercase()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
<p>{{ x1 }}</p>
<p>{{ x2 }}</p>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
$scope.x1 = "John";
$scope.x2 = angular.uppercase($scope.x1);
});
</script>
```

angular.isString()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
<p>{{ x1 }}</p>
<p>{{ x2 }}</p>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
$scope.x1 = "JOHN";
$scope.x2 = angular.isString($scope.x1);
});</script>
```

Content Downloaded from www.w3schools.com

angular.isNumber()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
<p>{{ x1 }}</p>
<p>{{ x2 }}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
$scope.x1 = "JOHN";
$scope.x2 = angular.isNumber($scope.x1);
});
</script>
```

AngularJS and W3.CSS

You can easily use w3.css style sheet together with AngularJS. This chapter demonstrates how.

W3.CSS

To include W3.CSS in your AngularJS application, add the following line to the head of your document:

```
<link rel="stylesheet" href="http://www.w3schools.com/lib/w3.css">
```

If you want to study W3.CSS, visit W3.CSS Tutorial.

Below is a complete HTML example, with all AngularJS directives and W3.CSS classes explained.

HTML Code

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="http://www.w3schools.com/lib/w3.css">
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.
js"></script>
<body ng-app="myApp" ng-controller="userCtrl">

<div class="w3-container">

<h3>Users</h3>

<table class="w3-table w3-bordered w3-striped">
  <tr>
    <th>Edit</th>
    <th>First Name</th>
    <th>Last Name</th>
  </tr>
```

```
  <tr ng-repeat="user in users">
    <td>
      <button class="w3-btn w3-ripple" ng-click="editUser(user.id)">&#9998;
Edit</button>
    </td>
    <td>{{ user.fName }}</td>
    <td>{{ user.lName }}</td>
  </tr>
</table>
<br>
<button class="w3-btn w3-green w3-ripple" ng-click="editUser('new')">&#9998;
Create New User</button>

<form ng-hide="hideform">
  <h3 ng-show="edit">Create New User:</h3>
  <h3 ng-hide="edit">Edit User:</h3>
    <label>First Name:</label>
    <input class="w3-input w3-border" type="text" ng-model="fName" ng-
disabled="!edit" placeholder="First Name">
  <br>
    <label>Last Name:</label>
    <input class="w3-input w3-border" type="text" ng-model="lName" ng-
disabled="!edit" placeholder="Last Name">
  <br>
    <label>Password:</label>
    <input class="w3-input w3-border" type="password" ng-
model="passw1"placeholder="Password">
  <br>
    <label>Repeat:</label>
    <input class="w3-input w3-border" type="password" ng-
model="passw2"placeholder="Repeat Password">
 <br>
<button class="w3-btn w3-green w3-ripple" ng-disabled="error ||
incomplete">&#10004; Save Changes</button>
</form>

</div>

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

</body>
</html>
```

Content Downloaded from www.w3schools.com

Directives (Used Above) Explained

AngularJS Directive	**Description**
<body ng-app	Defines an application for the <body> element
<body ng-controller	Defines a controller for the <body> element
<tr ng-repeat	Repeats the <tr> element for each user in users
<button ng-click	Invoke the function editUser() when the <button> element is clicked
<h3 ng-show	Show the <h3>s element if edit = true
<h3 ng-hide	Hide the form if hideform = true, and hide the <h3> element if edit = true
<input ng-model	Bind the <input> element to the application
<button ng-disabled	Disables the <button> element if error or incomplete = true

Content Downloaded from www.w3schools.com

W3.CSS Classes Explained

Element	Class	Defines
	w3-container	A content container
<table>	w3-table	A table
<table>	w3-bordered	A bordered table
<table>	w3-striped	A striped table
<button>	w3-btn	A button
<button>	w3-green	A green button
<button>	w3-ripple	A ripple effect when you click the button
<input>	w3-input	An input field
<input>	w3-border	A border on the input field

Content Downloaded from www.w3schools.com

JavaScript Code

myUsers.js

```
angular.module('myApp', []).controller('userCtrl', function($scope) {
$scope.fName = '';
$scope.lName = '';
$scope.passw1 = '';
$scope.passw2 = '';
$scope.users = [
{id:1, fName:'Hege', lName:"Pege" },
{id:2, fName:'Kim',  lName:"Pim" },
{id:3, fName:'Sal',  lName:"Smith" },
{id:4, fName:'Jack', lName:"Jones" },
{id:5, fName:'John', lName:"Doe" },
{id:6, fName:'Peter',lName:"Pan" }
];
$scope.edit = true;
$scope.error = false;
$scope.incomplete = false;
$scope.hideform = true;
$scope.editUser = function(id) {
  $scope.hideform = false;
  if (id == 'new') {
    $scope.edit = true;
    $scope.incomplete = true;
    $scope.fName = '';
    $scope.lName = '';
    } else {
    $scope.edit = false;
    $scope.fName = $scope.users[id-1].fName;
    $scope.lName = $scope.users[id-1].lName;
  }
};

$scope.$watch('passw1',function() {$scope.test();});
$scope.$watch('passw2',function() {$scope.test();});
$scope.$watch('fName', function() {$scope.test();});
$scope.$watch('lName', function() {$scope.test();});

$scope.test = function() {
  if ($scope.passw1 !== $scope.passw2) {
```

```
    $scope.error = true;
    } else {
    $scope.error = false;
  }
  $scope.incomplete = false;
  if ($scope.edit && (!$scope.fName.length ||
  !$scope.lName.length ||
  !$scope.passw1.length || !$scope.passw2.length)) {
     $scope.incomplete = true;
  }
};

});
```

JavaScript Code Explained

Scope Properties	Used for
$scope.fName	Model variable (user first name)
$scope.lName	Model variable (user last name)
$scope.passw1	Model variable (user password 1)
$scope.passw2	Model variable (user password 2)
$scope.users	Model variable (array of users)
$scope.edit	Set to true when user clicks on 'Create user'.
$scope.hideform	Set to true when user clicks on 'Edit' or 'Create user'.
$scope.error	Set to true if passw1 not equal passw2

Content Downloaded from www.w3schools.com

$scope.incomplete	Set to true if any field is empty (length = 0)
$scope.editUser	Sets model variables
$scope.$watch	Watches model variables
$scope.test	Tests model variables for errors and incompleteness

Content Downloaded from www.w3schools.com

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

AngularJS Includes

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

Example

```
<body ng-app="">

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

</body>
```

Include AngularJS Code

The HTML files you include with the ng-include directive, can also contain AngularJS code:

myTable.htm:

```
<table>
<tr ng-repeat="x in names">
<td>{{ x.Name }}</td>
<td>{{ x.Country }}</td>
</tr>
</table>
```

Include the file "myTable.htm" in your web page, and all AngularJS code will be executed, even the code inside the included file:

Content Downloaded from www.w3schools.com

Example

```
<body>
<div ng-app="myApp" ng-controller="customersCtrl">
  <div ng-include="'myTable.htm'"></div>
</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>
```

Include Cross Domains

By default, the ng-include directive does not allow you to include files from other domains.

To include files from another domain, you can add a whitelist of legal files and/or domains in the config function of your application:

Example:

```
<body ng-app="myApp">
<div ng-include="'http://www.refsnesdata.no/angular_include.asp'"></div>
<script>
var app = angular.module('myApp', [])
app.config(function($sceDelegateProvider) {
    $sceDelegateProvider.resourceUrlWhitelist([
        'http://www.refsnesdata.no/**'
    ]);
});
</script>
</body>
```

Be sure that the server on the destination allows cross domain file access.

AngularJS Animations

AngularJS provides animated transitions, with help from CSS.

What is an Animation?

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

Example:
Check the checkbox to hide the DIV:

```
<body ng-app="ngAnimate">
Hide the DIV: <input type="checkbox" ng-model="myCheck">

<div ng-hide="myCheck"></div>
</body>
```

Applications should not be filled with animations, but some animations can make the application easier to understand.

What do I Need?

To make your applications ready for animations, you must include the AngularJS Animate library:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-animate.js"></script>
```

Then you must refer to the **ngAnimate** module in your application:

```
<body ng-app="ngAnimate">
```

Or if your application has a name, add **ngAnimate** as a dependency in your application module:

Example

```
<body ng-app="myApp">
<h1>Hide the DIV: <input type="checkbox" ng-model="myCheck"></h1>
<div ng-hide="myCheck"></div>
<script>
var app = angular.module('myApp', ['ngAnimate']);
</script>
```

What Does ngAnimate Do?

The ngAnimate module adds and removes classes.

The ngAnimate module does not animate your HTML elements, but when ngAnimate notice certain events, like hide or show of an HTML element, the element gets some pre-defined classes which can be used to make animations.

The directives in AngularJS who add/remove classes are:

- **ng-show**
- **ng-hide**
- **ng-class**
- **ng-view**
- **ng-include**
- **ng-repeat**
- **ng-if**
- **ng-switch**

The **ng-show** and **ng-hide** directives adds or removes a **ng-hide** class value.

The other directives adds a **ng-enter** class value when they enter the DOM, and a **ng-leave** attribute when they are removed from the DOM.

The **ng-repeat** directive also adds a **ng-move** class value when the HTML element changes position.

Content Downloaded from www.w3schools.com

In addition, *during* the animation, the HTML element will have a set of class values, which will be removed when the animation has finished. Example: the **ng-hide** directive will add these class values:

- **ng-animate**
- **ng-hide-animate**
- **ng-hide-add** (if the element will be hidden)
- **ng-hide-remove** (if the element will be showed)
- **ng-hide-add-active** (if the element will be hidden)
- **ng-hide-remove-active** (if the element will be showed)

Animations Using CSS

We can use CSS transitions or CSS animations to animate HTML elements. This tutorial will show you both.

To learn more about CSS Animation, study CSS Transition Tutorial and CSS Animation Tutorial.

CSS Transitions

CSS transitions allows you to change CSS property values smoothly, from one value to another, over a given duration:

Example:
When the DIV element gets the **.ng-hide** class, the transition will take 0.5 seconds, and the height will smoothly change from 100px to 0:

```
<style>
div {
    transition: all linear 0.5s;
    background-color: lightblue;
    height: 100px;
}
.ng-hide {
    height: 0;
}
</style>
```

CSS Animations

CSS Animations allows you to change CSS property values smoothly, from one value to another, over a given duration:

Example
When the DIV element gets the `.ng-hide` class, the `myChange` animation will run, which will smoothly change the height from 100px to 0:

```
<style>
@keyframes myChange {
    from {
        height: 100px;
    } to {
        height: 0;
    }
}
div {
    height: 100px;
    background-color: lightblue;
}
div.ng-hide {
    animation: 0.5s myChange;
}
</style>
```

AngularJS Application

It is time to create a real AngularJS Single Page Application (SPA).

An AngularJS Application Example

You have learned more than enough to create your first AngularJS application:

My Note

Save Clear

Number of characters left: **100**

Application Explained

AngularJS Example

```
<html ng-app="myNoteApp">
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.
js"></script>
<body>

<div ng-controller="myNoteCtrl">

<h2>My Note</h2>

<p><textarea ng-model="message" cols="40" rows="10"></textarea></p>
```

```
<p>
<button ng-click="save()">Save</button>
<button ng-click="clear()">Clear</button>
</p>

<p>Number of characters left: <span ng-bind="left()"></span></p>

</div>

<script src="myNoteApp.js"></script>
<script src="myNoteCtrl.js"></script>

</body>
</html>
```

The application file "myNoteApp.js":

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

The controller file "myNoteCtrl.js":

```
app.controller("myNoteCtrl", function($scope) {
    $scope.message = "";
    $scope.left  = function() {return 100 - $scope.message.length;};
    $scope.clear = function() {$scope.message = "";};
    $scope.save  = function() {alert("Note Saved");};
});
```

The <html> element is the container of the AngularJS application: ng-app="**myNoteApp**":

```
<html ng-app="myNoteApp">
```

A a in the HTML page is the scope of a controller: ng-controller="**myNoteCtrl**":

```
<div ng-controller="myNoteCtrl">
```

An **ng-model** directive binds a <textarea> to the controller variable **message**:

```
<textarea ng-model="message" cols="40" rows="10"></textarea>
```

Content Downloaded from www.w3schools.com

The two **ng-click** events invoke the controller functions **clear()** and **save()**:

```
<button ng-click="save()">Save</button>
<button ng-click="clear()">Clear</button>
```

An **ng-bind** directive binds the controller function **left()** to a displaying the characters left:

```
Number of characters left: <span ng-bind="left()"></span>
```

Your application libraries are added to the page (after the library):

```
<script src="myNoteApp.js"></script>
<script src="myNoteCtrl.js"></script>
```

AngularJS Application Skeleton

Above you have the skeleton (scaffolding) of a real life AngularJS, single page application (SPA).

The <html> element is the "container" for the AngularJS application (**ng-app=**).

A elements defines the scope of an AngularJS controller (**ng-controller=**).

You can have many controllers in one application.

An application file (**my...App.js**) defines the application module code.

One or more controller files (**my...Ctrl.js**) defines the controller code.

Content Downloaded from www.w3schools.com

Summary - How Does it Work?

The ng-app directive is placed at the root element of the application.

For single page applications (SPA), the root of the application is often the <html> element.

One or more ng-controller directives define the application controllers. Each controller has its own scope: the HTML element where they were defined.

AngularJS starts automatically on the HTML DOMContentLoaded event. If an ng-app directive is found, AngularJS will load any module named in the directive, and compile the DOM with ng-app as the root of the application.

The root of the application can be the whole page, or a smaller portion of the page. The smaller the portion, the faster the application will compile and execute.

Content Downloaded from www.w3schools.com

AngularJS References

AngularJS Directives

ng-app

Defines the root element of an application.

Example
Let the body element become the root element for the AngularJS application:

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

Definition and Usage

The **ng-app** directive tells AngularJS that this is the root element of the AngularJS application.

All AngularJS applications must have a root element.

You can only have one **ng-app** directive in your HTML document. If more than one **ng-app**directive appears, the first appearance will be used.

Syntax

```
<element ng-app="modulename">
...
  content inside the ng-app root element can contain AngularJS code
...
</element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
modulename	Optional. Specifies the name of a module to load with the application

Example
Load a module to run in the application

```
<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>
```

ng-bind

Example
Bind the innerHTML of the <p> element to the variable *myText*:

```
<div ng-app="" ng-init="myText='Hello World!'">

<p ng-bind="myText"></p>

</div>
```

Definition and Usage

The **ng-bind** directive tells AngularJS to replace the content of an HTML element with the value of a given variable, or expression.

If the value of the given variable, or expression, changes, the content of the specified HTML element will be changed as well.

Content Downloaded from www.w3schools.com

Syntax

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

Or as a CSS class:

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

Supported by all HTML elements.

Parameter Values

Value	Description
expression	Specifies a variable, or an expression to evaluate.

ng-bind-html

Example

Bind the innerHTML of the <p> element to the variable *myText*:

```
<scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.
js"></script>
<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.0-beta.2/angular-
sanitize.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
    <p ng-bind-html="myText"></p>
</div>
<script>
var app = angular.module("myApp", ['ngSanitize']);
app.controller("myCtrl", function($scope) {
    $scope.myText = "My name is: <h1>John Doe</h1>";
});
</script>
```

Definition and Usage

The **ng-bind-html** directive is a secure way of binding content to an HTML element.

When you are letting AngularJS write HTML in your application, you should check the HTML for dangerous code. By including the "angular-santize.js" module in your application you can do so by running the HTML code through the ngSanitize function.

Syntax

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

Supported by all HTML elements.

Parameter Values

Value	Description
expression	Specifies a variable, or an expression to evaluate.

ng-bind-template

Example
Bind two expressions to the <p> element:

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

Definition and Usage

The **ng-bind-template** directive tells AngularJS to replace the content of an HTML element with the value of the given expressions.

Use the **ng-bind-template** directive when you want to bind more thant one expression to your HTML element.

Syntax

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

Supported by all HTML elements.

Parameter Values

Value	Description
expression	One or more expressions to evaluate, each surrounded by {{ }}.

ng-blur

Example
Execute an expresson when the input field loses focus (onblur):

`<input ng-blur="count = count + 1" ng-init="count=0" />`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-blur** directive tells AngularJS what to do when an HTML element loses focus.

The **ng-blur** directive from AngularJS will not override the element's original onblur event, both the **ng-blur** expression and the original onblur event will be executed.

Syntax

```
<element ng-blur="expression"></element>
```

Supported by <a>, <input>, <select>, <textarea>, and the window object.

Parameter Values

Value	Description
expression	An expression to execute when an element loses focus.

ng-change

Example

Execute a function when the value of the input field changes:

```
<body ng-app="myApp">
<div ng-controller="myCtrl">
    <input type="text" ng-change="myFunc()" ng-model="myValue" />
    <p>The input field has changed {{count}} times.</p>
</div>
<script>
angular.module('myApp', [])
.controller('myCtrl', ['$scope', function($scope) {
    $scope.count = 0;
    $scope.myFunc = function() {
        $scope.count++;
    };
}]);
</script>
</body>
```

Definition and Usage

The **ng-change** directive tells AngularJS what to do when the value of an HTML element changes.

The **ng-change** directive requires a **ng-model** directive to be present.

The **ng-change** directive from AngularJS will not override the element's original onchange event, both the **ng-change** expression and the original onchange event will be executed.

The **ng-change** event is triggered at every change in the value. It will not wait until all changes are made, or when the input field loses focus.

The **ng-change** event is only triggered if there is a actual change in the input value, and not if the change was made from a JavaScript.

Content Downloaded from www.w3schools.com

Syntax

```
<element ng-change="expression"></element>
```

Supported by <input>, <select>, and <textarea>.

Parameter Values

Value	Description
expression	An expression to execute when an element's value changes.

ng-checked

Example
Check one to check them all:

```
<body ng-app="">
<p>My cars:</p>
<input type="checkbox" ng-model="all"> Check all<br><br>

<input type="checkbox" ng-checked="all">Volvo<br>
<input type="checkbox" ng-checked="all">Ford<br>
<input type="checkbox" ng-checked="all">Mercedes
</body>
```

Definition and Usage

The **ng-checked** directive sets the checked attribute of a checkbox or a radiobutton.

The checkbox, or radiobutton, will be checked if the expression inside the **ng-checked** attribute returns true.

Syntax

```
<input type="checkbox|radio" ng-checked="expression"></input>
```

Supported by <input> elements of type checkbox or radio.

Parameter Values

Value	Description
expression	An expression that will set the element's checked attribute if it returns true.

ng-class

Example
Change class of a element:

```
<select ng-model="home">
    <option value="sky">Sky</option>
    <option value="tomato">Tomato</option>
</select>

<div ng-class="home">
    <h1>Welcome Home!</h1>
    <p>I like it!</p>
</div>
```

Definition and Usage

The **ng-class** directive dynamically binds one or more CSS classes to an HTML element.

The value of the **ng-class** directive can be a string, an object, or an array.

If it is a string, it should contain one or more, space-separated class names.

Content Downloaded from www.w3schools.com

As an object, it should contain key-value pairs, where the key is a boolean value, and the value is the class name of the class you want to add. The class will only be added if the key is set to true.

As an array, it can be a combination of both. Each array element can be either a string, or an object, described as above.

Syntax

```
<element ng-class="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression that returns one or more class names.

ng-class-even

Example
Set class="striped" for every other (even) table row:

```
<table ng-controller="myCtrl">
<tr ng-repeat="x in records" ng-class-even="'striped'">
    <td>{{x.Name}}</td>
    <td>{{x.Country}}</td>
</tr>
</table>
```

Definition and Usage

The **ng-class-even** directive dynamically binds one or more CSS classes to an HTML element, but will take effect only on every other (even) appearence of the HTML element.

The **ng-class-even** directive will only work if it is used together with the **ng-repeat** directive.

The **ng-class-even** directive is perfect for styling items in a list or rows in a table, but it can be used on any HTML element.

Syntax

```
<element ng-class-even="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression that returns one or more class names.

ng-class-odd

Example
Set class="striped" for every other (odd) table row:

```
<table ng-controller="myCtrl">
<tr ng-repeat="x in records" ng-class-odd="'striped'">
    <td>{{x.Name}}</td>
    <td>{{x.Country}}</td>
</tr>
</table>
```

Definition and Usage

The **ng-class-odd** directive dynamically binds one or more CSS classes to an HTML element, but will take effect only on every other (odd) appearence of the HTML element.

The **ng-class-odd** directive will only work if it is used together with the **ng-repeat** directive.

The **ng-class-odd** directive is perfect for styling items in a list or rows in a table, but it can be used on any HTML element.

Syntax

```
<element ng-class-odd="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression that returns one or more class names.

ng-click

Example
Increase the count variable by one, each time the button is clicked:
`<button ng-click="count = count + 1" ng-init="count=0">OK</button>`

Definition and Usage

The **ng-click** directive tells AngularJS what to do when an HTML element is clicked.

Syntax

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

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to execute when an element is clicked.

Example
Execute a function, in AngularJS, when a button is clicked:

```
<body ng-app="myApp">
<div ng-controller="myCtrl">
    <button ng-click="myFunc()">OK</button>
    <p>The button has been clicked {{count}} times.</p>
</div>
<script>
angular.module('myApp', [])
.controller('myCtrl', ['$scope', function($scope) {
    $scope.count = 0;
    $scope.myFunc = function() {
        $scope.count++;
    };
}]);</script></body>
```

Content Downloaded from www.w3schools.com

ng-cloak

Example
Prevent the application from flicker at page load:

`<div ng-app="">`
 	
`<p ng-cloak>{{ 5 + 5 }}</p>`
 	
`</div>`

Definition and Usage

The `ng-cloak` directive prevents the document from showing unfinished AngularJS code while AngularJS is being loaded.

AngularJS applications can make HTML documents flicker when the application is being loaded, showing the AngularJS code for a second, before all code are executed. Use the `ng-cloak` directive to prevent this.

Syntax

```
<element ng-cloak></element>
```

Supported by all HTML elements.

Parameter Values

The **ng-cloak** directive has no parameters.

ng-controller

Example
Add a controller to handle your application variables:

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

Definition and Usage

The **ng-controller** directive adds a controller to your application.

In the controller you can write code, and make functions and variables, which will be parts of an object, available inside the current HTML element. In AngularJS this object is called a scope.

Syntax

```
<element ng-controller="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	The name of the controller.

ng-copy

Example
Execute an expresson when the text of the input field is being copied:
`<input ng-copy="count = count + 1" ng-init="count=0" value="Copy this text"/>`

Definition and Usage

The **ng-copy** directive tells AngularJS what to do when an HTML element is being copied.

The **ng-copy** directive from AngularJS will not override the element's original oncopy event, both the **ng-copy** expression and the original oncopy event will be executed.

Syntax

```
<element ng-copy="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to execute when the text of an element is being copied.

Content Downloaded from www.w3schools.com

ng-csp

Example
Change the way AngularJS behaves regarding "eval" and inline styles:
`<body ng-app="" ng-csp>` `...`

Definition and Usage

The **ng-csp** directive is used to change the security policy of AngularJS.

With the **ng-csp** directive set, AngularJS will not run any eval functions, and it will not inject any inline styles.

Setting the value of the **ng-csp** directive to **no-unsafe-eval**, will stop AngularJS from running any eval functions, but allow injecting inline styles.

Setting the value of the **ng-csp** directive to **no-inline-style**, will stop AngularJS from injecting any inline styles, but allow eval functions.

Using the **ng-csp** directive is necessary when developing apps for Google Chrome Extensions or Windows Apps.

Note: The **ng-csp** directive does not affect JavaScript, but it changes the way AngularJS works, meaning: you can still write eval functions, and they will be executed as you expect, but AngularJS will not run its own eval functions. It uses a compatability mode which can slow down the evaluation time up to 30%.

Syntax

```
<element ng-csp="no-unsafe-eval | no-inline-style"></element>
```

Parameter Values

Value	Description
no-unsafe-eval no-inline-style	The value can be empty, meaning neither eval or inline styles are allowed. The value can be one of the two values described. The value can be both values, separated by a semicolon, but that will have the same meaning as an empty value.

Content Downloaded from www.w3schools.com

ng-cut

Example
Execute an expresson when the text of the input field is being cut:
`<input ng-cut="count = count + 1" ng-init="count=0" value="Cut this text" />`

Definition and Usage

The **ng-cut** directive tells AngularJS what to do when you cut the text of an HTML element.

The **ng-cut** directive from AngularJS will not override the element's original oncut event, both the **ng-cut** expression and the original oncut event will be executed.

Syntax

```
<element ng-cut="expression"></element>
```

Supported by <a>, <input>, <select>, <textarea>, and the window object.

Parameter Values

Value	Description
expression	An expression to execute when the text of an element is being cut.

ng-dblclick

Example
Increase the count variable by one, each time the header is double-clicked:
`<h1 ng-dblclick="count = count + 1" ng-init="count=0">Welcome</h1>`

Definition and Usage

The **ng-dblclick** directive tells AngularJS what to do when an HTML element is double-clicked.

The **ng-dblclick** directive from AngularJS will not override the element's original ondblclick event, both are executed.

Syntax

```
<element ng-dblclick="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to execute when an element is cdouble-licked.

ng-disabled

Example
Disable / enable the input field:

```
Disable form fields: <input type="checkbox" ng-model="all">
<br>
<input type="text" ng-disabled="all">
<input type="radio" ng-disabled="all">
<select ng-disabled="all">
<option>Female</option>
<option>Male</option>
</select>
```

Definition and Usage

The **ng-disabled** directive sets the disabled attribute of a form field (input, select, or textarea).

The form field will be disabled if the expression inside the **ng-disabled** attribute returns true.

Syntax

```
<input ng-disabled="expression"></input>
```

Supported by <input>, <select>, and <textarea> elements.

Parameter Values

Value	Description
expression	An expression that will set the element's disabled attribute if it returns true.

Content Downloaded from www.w3schools.com

ng-focus

Example
Execute an expression when the input field gets focus:

`<input ng-focus="count = count + 1" ng-init="count=0" />`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-focus** directive tells AngularJS what to do when an HTML element gets focus.

The **ng-focus** directive from AngularJS will not override the element's original onfocus event, both will be executed.

Syntax

```
<element ng-focus="expression"></element>
```

Supported by <a>, <input>, <select>, <textarea>, and the window object.

Parameter Values

Value	Description
expression	An expression to execute when an element gets focus.

ng-form

Specifies an HTML form to inherit controls from.

ng-hide

Example
Hide a section when a checkbox is checked:

```
Hide HTML: <input type="checkbox" ng-model="myVar">
<div ng-hide="myVar">
<h1>Welcome</h1>
<p>Welcome to my home.</p>
</div>
```

Definition and Usage

The **ng-hide** directive hides the HTML element if the expression evaluates to true.

ng-hide is also a predefined CSS class in AngularJS, and sets the element's **display** to **none**.

Syntax

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

When used as a CSS class:

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

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression that will hide the element if the expression returns true.

Content Downloaded from www.w3schools.com

ng-href

Example
Make a href using AngularJS:

```
<div ng-init="myVar = 'http://www.w3schools.com'">
    <h1>Tutorials</h1>
    <p>Go to <a ng-href="{{myVar}}">{{myVar}}</a> to learn!</p>
</div>
```

Definition and Usage

The **ng-href** directive overrides the original href attribute of an <a> element.

The **ng-href** directive should be used instead of **href** if you have AngularJS code inside the href value.

The **ng-href** directive makes sure the link is not broken even if the user clicks the link before AngularJS has evaluated the code.

Syntax

```
<a ng-href="string"></a>
```

Supported by the <a> element.

Parameter Values

Value	Description
string	A string value, or an expression resulting in a string.

ng-if

Example
Uncheck a checkbox to remove a section:

```
Keep HTML: <input type="checkbox" ng-model="myVar" ng-init="myVar = true">
<div ng-if="myVar">
<h1>Welcome</h1>
<p>Welcome to my home.</p>
<hr>
</div>
```

Definition and Usage

The **ng-if** directive removes the HTML element if the expression evaluates to false.

If the if statement evaluates to true, a copy of the Element is added in the DOM.

The **ng-if** directive is different from the ng-hide which hides the display of the element, were the ng-if directive completely removes the element from the DOM.

Syntax

```
<element ng-if="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression that will completely remove the element if it returns true. If it returns true, a copy of the element will be inserted instead.

Content Downloaded from www.w3schools.com

ng-include

Example
Include HTML from an external file:
`<div ng-include="'myFile.htm'"></div>`

Definition and Usage

The **ng-include** directive includes HTML from an external file.

The included content will be included as childnodes of the specified element.

The value of the **ng-include** attribute can also be an expression, returning a filename.

By default, the included file must be located on the same domain as the document.

Syntax

```
<element ng-
include="filename" onload="expression" autoscroll="expression" ></element>
```

The ng-include directive can also be used as an element:

```
<ng-include src="filename" onload="expression" autoscroll="expression" ></ng-
include>
```

Supported by all HTML elements.

Parameter Values

Value	Description
filename	A filename, written with apostrophes, or an expression which returns a filename.
onload	Optional. An expression to evaluate when the included file is loaded.
autoscroll	Optional. Whether or not the included section should be able to scroll into a specific view.

Content Downloaded from www.w3schools.com

ng-init

Example
Create a variable when initiating the application:

`<div ng-app="" ng-init="myText='Hello World!'">`
 	
`<h1>{{myText}}</h1>`

Definition and Usage

The **ng-init** directive evaluates the given expression(s).

The **ng-init** directive can add some unnecessary logic into the scope, and you are recommended to do your evaluations in a controller instead, see the ng-controller directive.

Syntax

```
<element ng-init="expression" ></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to evaluate.

ng-jq

Specifies that the application must use a library, like jQuery.

ng-keydown

Example
Execute an expression at every keystroke:

`<input ng-keydown="count = count + 1" ng-init="count=0" />`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-keydown** directive tells AngularJS what to do when the keyboard is used on the specific HTML element.

The **ng-keydown** directive from AngularJS will not override the element's original onkeydown event, both will be executed.

The order of a key stroke is:

1. Keydown

2. Keypress

3. Keyup

Syntax

```
<element ng-keydown="expression"></element>
```

Supported by <input>, <select>, <textarea>, and other editable elements.

Parameter Values

Value	Description
expression	An expression to execute when a key is pressed.

ng-keypress

Example
Execute an expression at every keystroke:

`<input ng-keypress="count = count + 1" ng-init="count=0" />`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-keypress** directive tells AngularJS what to do when the keyboard is used on the specific HTML element.

The **ng-keypress** directive from AngularJS will not override the element's original onkeypress event, both will be executed.

The order of a key stroke is:

1. Keydown

2. Keypress

3. Keyup

Syntax

```
<element ng-keypress="expression"></element>
```

Supported by <input>, <select>, <textarea>, and other editable elements.

Parameter Values

Value	Description
expression	An expression to execute when a key is pressed.

ng-keyup

Example
Execute an expression at every keystroke:

`<input ng-keyup="count = count + 1" ng-init="count=0" />`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-keyup** directive tells AngularJS what to do when the keyboard is used on the specific HTML element.

The **ng-keyup** directive from AngularJS will not override the element's original onkeyup event, both will be executed.

The order of a key stroke is:

1. Keydown

2. Keypress

3. Keyup

Syntax

```
<element ng-keyup="expression"></element>
```

Supported by <input>, <select>, <textarea>, and other editable elements.

Parameter Values

Value	Description
expression	An expression to execute when a keypress is finished.

Content Downloaded from www.w3schools.com

ng-list

Example
Convert user input into an array:

`<div ng-app="">`
 	
`<input ng-model="customers" ng-list/>`
 	
`<pre>{{customers}}</pre>`

Definition and Usage

The **ng-list** directive converts a string into an array of strings, using a comma as the default separator.

The **ng-list** directive also converts the other way around, if you have a array of strings you wish to display in an input field as a string, then put the **ng-list** directive on the input field.

The value of the **ng-list** attribute defines the separator.

Syntax

```
<element ng-list="separator"></element>
```

Supported by <input> and <textarea> elements.

Parameter Values

Value	Description
separator	Optional, defines the separator, default value is ", "

ng-model

Example
Bind the value of an input field to a variable in the scope:

```
<div ng-app="myApp" ng-controller="myCtrl">
    <input ng-model="name">
</div>

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

Definition and Usage

The **ng-model** directive binds an HTML form element to a variable in the scope.

If the variable does not exist in the scope, it will be created.

Syntax

```
<element ng-model="name"></element>
```

Supported by <input>, <select>, and <textarea> elements.

Parameter Values

Value	Description
name	The name of the property you want to bind to the form field.

Content Downloaded from www.w3schools.com

ng-model-options

Example
Wait with the data-binding until the field loses focus:

```
<div ng-app="myApp" ng-controller="myCtrl">
    <input ng-model="name" ng-model-options="{updateOn: 'blur'}">
</div>

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

Definition and Usage

The **ng-model-options** directive is used to control the binding of an HTML form element and a variable in the scope.

You can specify that the binding should wait for a specific event to occur, or wait a specific number of milliseconds, and more, see the legal values listed in the parameter values below.

Syntax

```
<element ng-model-options="option"></element>
```

Supported by <input>, <select>, and <textarea> elements.

Parameter Values

Value	Description
option	An object specifying what options the data-binding must follow. Legal objects are: {updateOn: '*event*'} specifies that the binding should happen when the specific event occur.

	{debounce : 1000} specifies how many milliseconds to wait with the binding. {allowInvalid : true|false} specify if the binding can happen if the value di not validate. {getterSetter : true|false} specifies if functions bound to the model should be treated as getters/setters. {timezone : '0100'} Specifies what timezone should be used when working with the Date object.

ng-mousedown

Example
Execute an expression when a mouse click occurs:

`<div ng-mousedown="count = count + 1" ng-init="count=0">Click me!</div>`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-mousedown** directive tells AngularJS what to do when a mouse button is clicked on the specific HTML element.

The **ng-mousedown** directive from AngularJS will not override the element's original onmousedown event, both will be executed.

The order of a mouse click is:

1. Mousedown
2. Mouseup
3. Click

Content Downloaded from www.w3schools.com

Syntax

```
<element ng-mousedown="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to execute when a mouse button is clicked.

ng-mouseenter

Example
Execute an expression when the mouse cursor enters a element:

`<div ng-mouseenter="count = count + 1" ng-init="count=0">Mouse over me!</div>`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-mouseenter** directive tells AngularJS what to do when a mouse cursor enters the specific HTML element.

The **ng-mouseenter** directive from AngularJS will not override the element's original onmouseenter event, both will be executed.

Syntax

```
<element ng-mouseenter="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to execute when the mouse cursor enters an element.

ng-mouseleave

Example
Execute an expression when the mouse cursor leaves a element:

`<div ng-mouseleave="count = count + 1" ng-init="count=0">Mouse over me! (and mouse away from me...)</div>`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-mouseleave** directive tells AngularJS what to do when a mouse cursor leaves the specific HTML element.

The **ng-mouseleave** directive from AngularJS will not override the element's original onmouseleave event, both will be executed.

Syntax

```
<element ng-mouseleave="expression"></element>
```

Supported by all HTML elements.

Content Downloaded from www.w3schools.com

Parameter Values

Value	Description
expression	An expression to execute when the mouse cursor leaves an element.

ng-mousemove

Example
Execute an expression when the mouse cursor moves over a element:

`<div ng-mousemove="count = count + 1" ng-init="count=0">Mouse over me!</div>`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-mousemove** directive tells AngularJS what to do when a mouse cursor moves over the specific HTML element.

The **ng-mousemove** directive from AngularJS will not override the element's original onmousemove event, both will be executed.

Syntax

```
<element ng-mousemove="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to execute when the mouse cursor moves over an element.

ng-mouseover

Example
Execute an expression when the mouse cursor moves over a element:

`<div ng-mouseover="count = count + 1" ng-init="count=0">Mouse over me!</div>`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-mouseover** directive tells AngularJS what to do when a mouse cursor moves over the specific HTML element.

The **ng-mouseover** directive from AngularJS will not override the element's original onmouseover event, both will be executed.

Syntax

```
<element ng-mouseover="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to execute when the mouse cursor moves over an element.

ng-mouseup

Example
Execute an expression when a mouse click is finished:

`<div ng-mouseup="count = count + 1" ng-init="count=0">Click me!</div>`
 	
`<h1>{{count}}</h1>`

Definition and Usage

The **ng-mouseup** directive tells AngularJS what to do when a mouse click is finished.

The **ng-mouseup** directive from AngularJS will not override the element's original onmouseup event, both will be executed.

The order of a mouse click is:

1. Mousedown

2. Mouseup

3. Click

Syntax

```
<element ng-mouseup="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression to execute when a mouse click is finished.

Content Downloaded from www.w3schools.com

ng-non-bindable

Example
This paragraph should not be compiled by AngularJS:

`<div ng-app="">`
`<p ng-non-bindable>This code is not compiled by AngularJS: {{ 5+5 }}</p>`
`...`
`</div>`

Definition and Usage

The **ng-non-bindable** directive specifies that the content of this HTML element, and it's child nodes, should not be compiled by AngularJS.

Syntax

```
<element ng-non-bindable></element>
```

Supported by all HTML elements.

Parameter Values

The ng-non-bindable directive does not have any parameters.

ng-open

Example
Show / Hide a <details> list by clicking a checkbox:

```
<input type="checkbox" ng-model="showDetails">

<details ng-open="showDetails">
    <summary>Copyright 1999-2016.</summary>
    <p> - by Refsnes Data. All Rights Reserved.</p>
</details>
```

Definition and Usage

The **ng-open** directive sets the open attribute of a details list.

The details list will be visible if the expression inside the ng-open attribute returns true.

Syntax

```
<details ng-open="expression">...</details>
```

Supported by the <details> element.

Parameter Values

Value	Description
expression	An expression that will set the element's open attribute if it returns true.

Content Downloaded from www.w3schools.com

ng-options

Example
Fill options in a dropdown list by using the items of an array:

```
<div ng-app="myApp" ng-controller="myCtrl">
<select ng-model="selectedName" ng-options="item for item in names"></select>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.names = ["Emil", "Tobias", "Linus"];
});
</script>
```

Definition and Usage

The **ng-options** directive fills a <select> element with <options>.

The **ng-options** directive uses an array to fill the dropdown list. In many cases it would be easier to use the **ng-repeat** directive, but you have more flexibility when using the **ng-options** directive.

Syntax

```
<select ng-options="array expression"></select>
```

Supported by the <select> element.

Parameter Values

Value	Description
array expression	An expression that selects the specified parts of an array to fill the select element. Legal expressions:

	label for *value* in *array* *select* as *label* for *value* in *array* *label* group by *group* for *value* in *array* *label* disabled when *disable* for *value* in *array* *label* group by *group* for *value* in *array* track by *expression* *label* disabled when *disable* for *value* in *array* track by *expression* *label* for *value* in *array* | orderBy *expression* track by *expression*

ng-paste

Example
Execute an expression when text is pasted into the the input:
`<input ng-paste="count = count + 1" ng-init="count=0" value="Cut this text"/>`

Definition and Usage

The **ng-paste** directive tells AngularJS what to do when text is pasted into an HTML element.

The **ng-paste** directive from AngularJS will not override the element's original onpaste event, both will be executed.

Syntax

```
<element ng-paste="expression"></element>
```

Supported by <input>, <select>, and <textarea> and other editable elements

Content Downloaded from www.w3schools.com

Parameter Values

Value	Description
expression	An expression to execute when text is being pasted into an element.

ng-pluralize

Specifies a message to display according to en-us localization rules.

ng-readonly

Example
Make the input field readonly:
`Readonly: <input type="checkbox" ng-model="all">` ` ` `<input type="text" ng-readonly="all">`

Definition and Usage

The **ng-readonly** directive sets the readonly attribute of a form field (input or textarea).

The form field will be readonly if the expression inside the **ng-readonly** attribute returns true.

Syntax

```
<input ng-readonly="expression"></input>
```

Supported by <input> <textarea> elements.

Parameter Values

Value	Description
expression	An expression that will set the element's readonly attribute if it returns true.

ng-repeat

Example

Write one header for each item in the records array:

```
<body ng-app="myApp" ng-controller="myCtrl">
<h1 ng-repeat="x in records">{{x}}</h1>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.records = [
        "Alfreds Futterkiste",
        "Berglunds snabbköp",
        "Centro comercial Moctezuma",
        "Ernst Handel",
    ]
});
</script>
</body>
```

Definition and Usage

The **ng-repeat** directive repeats a set of HTML, a given number of times.

The set of HTML will be repeated once per item in a collection.

The collection must be an array or an object.

Note: Each instance of the repetition is given its own scope, which consist of the current item.

If you have an collection of objects, the **ng-repeat** directive is perfect for making a HTML table, displaying one table row for each object, and one table data for each object property. See example below.

Content Downloaded from www.w3schools.com

Syntax

```
<element ng-repeat="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description

expression	An expression that specifies how to loop the collection.
	
	Legal Expression examples:
	
	`x in records`
	
	`(key, value) in myObj`
	
	`x in records track by $id(x)`

More Examples

Example
Write one table row for each item in the records array:

```
<table ng-controller="myCtrl" border="1">
    <tr ng-repeat="x in records">
        <td>{{x.Name}}</td>
        <td>{{x.Country}}</td>
    </tr>
</table>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.records = [
       {
            "Name" : "Alfreds Futterkiste",
            "Country" : "Germany"
        },{
            "Name" : "Berglunds snabbköp",
```

```
            "Country" : "Sweden"
        },{
            "Name" : "Centro comercial Moctezuma",
            "Country" : "Mexico"
        },{
            "Name" : "Ernst Handel",
            "Country" : "Austria"
        }
    ]
});</script>
```

Example

Write one table row for each property in an object:

```
<table ng-controller="myCtrl" border="1">
    <tr ng-repeat="(x, y) in myObj">
        <td>{{x}}</td>
        <td>{{y}}</td>
    </tr>
</table>

<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.myObj = {
        "Name" : "Alfreds Futterkiste",
        "Country" : "Germany",
        "City" : "Berlin"
    }
});
</script>
```

Content Downloaded from www.w3schools.com

ng-selected

Example
Make the input field readonly:

```
Click here to select BMW as your favorite car:
<input type="checkbox" ng-model="mySel">

<p>My Favourite car:</p>

<select>
<option>Volvo</option>
<option ng-selected="mySel">BMW</option>
<option>Ford</option>
</select>
```

Definition and Usage

The **ng-selected** directive sets the selected attribute of an <option> element in a <select> list.

The option will be selected if the expression inside the **ng-selected** attribute returns true.

Syntax

```
<option ng-selected="expression"></option>
```

Supported by the <option> element.

Parameter Values

Value	Description
expression	An expression that will set the element's selected attribute if it returns true.

ng-show

Example
Show a section when a checkbox is checked:

```
Show HTML: <input type="checkbox" ng-model="myVar">
<div ng-show="myVar">
<h1>Welcome</h1>
<p>Welcome to my home.</p>
</div>
```

Definition and Usage

The **ng-show** directive shows the specified HTML element if the expression evaluates to true, otherwise the HTML element is hidden.

Syntax

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

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression that will show the element only if the expression returns true.

Content Downloaded from www.w3schools.com

ng-src

Example
Add an image, where the src is evaluated by AngularJS:
`<div ng-init="myVar = 'pic_angular.jpg'">   <h1>Angular</h1>   <img ng-src="{{myVar}}"> </div>`

Definition and Usage

The **ng-src** directive overrides the original src attribute of an <img> element.

The **ng-src** directive should be used instead of **src** if you have AngularJS code inside the src value.

The **ng-src** directive makes sure the image is not displayed wrong before AngularJS has evaluated the code.

Syntax

```
<img ng-src="string"></img>
```

Supported by the <img> element.

Parameter Values

Value	Description
string	A string value, or an expression resulting in a string.

ng-srcset

Example
Add an image, where the srcset is evaluated by AngularJS:

```
<div ng-init="myVar = 'pic_angular.jpg'">
    <h1>Angular</h1>
    <img ng-srcset="{{myVar}}">
</div>
```

Definition and Usage

The **ng-srcset** directive overrides the original srcset attribute of an element.

The **ng-srcset** directive should be used instead of **srcset** if you have AngularJS code inside the srcset value.

The **ng-srcset** directive makes sure the image is not displayed wrong before AngularJS has evaluated the code.

Syntax

```
<img ng-srcset="string"></img>
```

Supported by the <img> and <source> elements.

Parameter Values

Value	Description
string	A string value, or an expression resulting in a string.

ng-style

Example
Add som style with AngularJS, using an object with CSS keys and values:

```
<body ng-app="myApp" ng-controller="myCtrl">
<h1 ng-style="myObj">Welcome</h1>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.myObj = {
        "color" : "white",
        "background-color" : "coral",
        "font-size" : "60px",
        "padding" : "50px"
    }
});
</script></body>
```

Definition and Usage

The **ng-style** directive specifies the style attribute for the HTML element.

The value of the **ng-style** attribute must be an object, or an expression returning an object.

The object consists of CSS properties and values, in key value pairs.

Syntax

```
<element ng-style="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression which returns an object where the keys are CSS properties, and the values are CSS values.

Content Downloaded from www.w3schools.com

ng-submit

Example

Run a function when the form is submitted:

```
<body ng-app="myApp" ng-controller="myCtrl">
<form ng-submit="myFunc()">
    <input type="text">
    <input type="submit">
</form>
<p>{{myTxt}}</p>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.myTxt = "You have not yet clicked submit";
    $scope.myFunc = function () {
        $scope.myTxt = "You clicked submit!";
    }
});
</script></body>
```

Definition and Usage

The **ng-submit** directive specifies a function to run when the form is submitted.

If the form does not have an **action ng-submit** will prevent the form from being submitted.

Syntax

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

Supported by the <form> element.

Parameter Values

Value	Description
expression	A function to be called when the form is being submitted, or an expression to be evaluated, which should return a function call.

Content Downloaded from www.w3schools.com

ng-switch

Example
Show a section of HTML, only if it matches a certain value:

```
<div ng-switch="myVar">
  <div ng-switch-when="dogs">
    <h1>Dogs</h1>
    <p>Welcome to a world of dogs.</p>
  </div>
  <div ng-switch-when="tuts">
    <h1>Tutorials</h1>
    <p>Learn from examples.</p>
  </div>
  <div ng-switch-when="cars">
    <h1>Cars</h1>
    <p>Read about cars.</p>
  </div>
  <div ng-switch-default>
    <h1>Switch</h1>
    <p>Select topic from the dropdown, to switch the content of this DIV.</p>
  </div>
</div>
```

Definition and Usage

The **ng-switch** directive lets you hide/show HTML elements depending on an expression.

Child elements with the **ng-switch-when** directive will be displayed if it gets a match, otherwise the element, and it's children will be removed.

You can also define a default section, by using the **ng-switch-default** directive, to show a section if non of the other sections get a match.

Content Downloaded from www.w3schools.com

Syntax

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

Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression that will remove elements with no match, and display elements with a match.

ng-transclude

Specifies a point to insert transcluded elements.

ng-value

Example
Set the value of the input field:

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

<input ng-value="myVar">

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.myVar = "Hello World!";
});
</script>
```

Definition and Usage

The **ng-value** directive sets the value attribute of a input element, or a select element.

Syntax

```
<input ng-value="expression"></input>
```

Supported by <input> and <select> elements.

Parameter Values

Value	Description
expression	An expression that will set the element's value attribute.

AngularJS Directives on HTML Elements

AngularJS modifies the default behavior of some HTML elements.

AngularJS a Directive

Example
The link inside the AngularJS application will not reload the page:

```
<a href="">Click me!</a>
<div ng-app="">
    <a href="">Click me too!</a>
</div>
```

Definition and Usage

AngularJS modifies the default behavior of the **<a>** element.

AngularJS prevents the page from reloading when a user clicks on an **<a>** element with an empty **href** attribute.

AngularJS form Directive

Example
This form's "valid state" will not be consider "true", as long as the required input field is empty:

```
<form name="myForm">
<input name="myInput" ng-model="myInput" required>
</form>

<p>The forms's valid state is:</p>
<h1>{{myForm.$valid}}</h1>
```

Content Downloaded from www.w3schools.com

Definition and Usage

AngularJS modifies the default behavior of the `<form>` element.

Forms inside an AngularJS application are given certain properties. These properties describes the current state of the form.

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

The value of each state represents a Boolean value, and is either **`true`** or **`false`**.

Forms in AngularJS prevents the default action, which is submitting the form to the server, if the action attribute is not specified.

Syntax

```
<form name="formname"></form>
```

Forms are being referred to by using the value of the name attribute.

CSS Classes

Forms inside an AngularJS application are given certain *classes*. These classes can be used to style forms according to their state.

The following classes are added:

- **`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

Content Downloaded from www.w3schools.com

- **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**.

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

```
<style>
form.ng-pristine {
    background-color: lightblue;
}
form.ng-dirty {
    background-color: pink;
}
</style>
```

AngularJS input Directive

Example

An input field with data-binding:

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

<p>The value of the input field is:</p>
<h1>{{myInput}}</h1>
```

Definition and Usage

AngularJS modifies the default behavior of **<input>** elements, but only if the **ng-model** attribute is present.

They provide data-binding, which means they are part of the AngularJS model, and can be referred to, and updated, both in AngularJS functions and in the DOM.

They provide validation. Example: an **<input>** element with a **required** attribute, has the**$valid** state set to **false** as long as it is empty.

They also provide state control. AngularJS holds the current state of all input elements.

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

The value of each state represents a Boolean value, and is either `true` or `false`.

Syntax

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

Input elements are being referred to by using the value of the **ng-model** attribute.

CSS Classes

<input> elements inside an AngularJS application are given certain *classes*. These classes can be used to style input elements according to their state.

The following classes are added:

- **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

Content Downloaded from www.w3schools.com

- **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 classes are removed if the value they represent is **false**.

Example
Apply styles for valid and invalid input elements, using standard CSS:

```
<style>
input.ng-invalid {
    background-color: pink;
}
input.ng-valid {
    background-color: lightgreen;
}
</style>
```

AngularJS Script Directive

AngularJS modifies the <script> element's default behaviors.

AngularJS Select Directive

AngularJS modifies the <select> element's default behaviors.

AngularJS textarea Directive

Example
An textarea with data-binding:

```
<textarea ng-model="myTextarea"></textarea>

<p>The value of the textarea field is:</p>
<h1>{{myTextarea}}</h1>
```

Definition and Usage

AngularJS modifies the default behavior of **<textarea>** elements, but only if the **ng-model**attribute is present.

They provide data-binding, which means they are part of the AngularJS model, and can be referred to, and updated, both in AngularJS functions and in the DOM.

They provide validation. Example: an **<textarea>** element with a **required** attribute, has the**$valid** state set to **false** as long as it is empty.

They also provide state control. AngularJS holds the current state of all textarea elements.

Textarea 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

The value of each state represents a Boolean value, and is either `true` of `false`.

Syntax

```
<textarea ng-model="name"></textarea>
```

Textarea elements are being referred to by using the value of the **ng-model** attribute.

CSS Classes

<textarea> elements inside an AngularJS application are given certain *classes*. These classes can be used to style textarea elements according to their state.

The following classes are added:

- **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 classes are removed if the value they represent is **false**.

Example
Apply styles for valid and invalid textarea elements, using standard CSS:

```
<style>
textarea.ng-invalid {
    background-color: pink;
}
textarea.ng-valid {
    background-color: lightgreen;
}
</style>
```

Content Downloaded from www.w3schools.com

AngularJS Filters

AngularJS currency Filter

Example
Display the number as a currency format:

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

<p>Price = {{ price | currency }}</p>

</div>
```

Definition and Usage

The **currency** filter formats a number to a currency format.

By default, the locale currency format is used.

Syntax

```
{{ number | currency : symbol : fractionsize }}
```

Parameter Values

Value	Description
symbol	Optional. The currency symbol to be displayed. The symbol can be any character or text.
fractionsize	Optional. The number of decimals.

Content Downloaded from www.w3schools.com

More Examples

Example
Display the price in the Norwegian currency format:

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

<p>Price = {{ price | currency : "NOK" }}</p>

</div>
```

Example
Display the price with three deciamls:

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

<p>Price = {{ price | currency : "NOK" : 3 }}</p>

</div>
```

AngularJS date Filter

Example
Display the number as a currency format:

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

<p>Date = {{ today | date }}</p>

</div>
```

Definition and Usage

The **date** filter formats a date to a specified format.

The date can be a date object, milliseconds, or a datetime string like "2016-01-05T09:05:05.035Z"

Content Downloaded from www.w3schools.com

By default, the format is "MMM d, y" (Jan 5, 2016).

Syntax

```
{{ date | date : format : timezone }}
```

Parameter Values

Value	Description
format	Optional. The date format to display the date in, which can be one or more of the following: "yyyy" year (2016) "yy" year (16) "y" year (2016) "MMMM" month (January) "MMM" month (Jan) "MM" month (01) "M" month (1) "dd" day (06) "d" day (6) "EEEE" day (Tuesday) "EEE" day (Tue) "HH" hour, 00-23 (09) "H" hour 0-23 (9) "hh" hour in AM/PM, 00-12 (09) "h" hour in AM/PM, 0-12 (9) "mm" minute (05) "m" minute (5) "ss" second (05) "s" second (5) "sss" millisecond (035) "a" (AM/PM) "Z" timezone (from -1200 to +1200) "ww" week (00-53) "w" week (0-53) "G" era (AD) "GG" era (AD) "GGG" era (AD) "GGGG" era (Anno Domini) The format value can also be one of the following predefined formats:

	"short" same as "M/d/yy h:mm a" (1/5/16 9:05 AM) "medium" same as "MMM d, y h:mm:ss a" (Jan 5, 2016 9:05:05 AM) "shortDate" same as "M/d/yy" (1/5/16) "mediumDate" same as "MMM d, y" (Jan 5, 2016) "longDate" same as "MMMM d, y" (January 5, 2016) "fullDate" same as "EEEE, MMMM d, y" (Tuesday, January 5, 2016) "shortTime" same as "h:mm a" (9:05 AM) "mediumTime" same as "h:mm:ss a" (9:05:05 AM)
timezone	Optional. The timezone used to format the date.

Example

Display a date in a custom format:

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

<p>Date = {{ today | date :  "dd.MM.y" }}</p>

</div>
```

Example

Display a date using a predefined format:

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

<p>Date = {{ today | date : "fullDate" }}</p>

</div>
```

Content Downloaded from www.w3schools.com

Example

Display a date combination of text and a predefined format:

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

<p>Date = {{ today | date : "fullDate" }}</p>

</div>
```

Example

The date as a datetime string:

```
<div ng-app="">

<p>Date = {{ "2016-01-05T09:05:05.035Z" | date }}</p>

</div>
```

AngularJS filter Filter

Example

Display the items that contains the letter "A":

```
<div ng-app="myApp" ng-controller="arrCtrl">
<ul>
<li ng-repeat="x in cars | filter : 'A'">{{x}}</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('arrCtrl', function($scope) {
    $scope.cars = ["Aston Martin", "Audi", "Bentley", "BMW", "Bugatti"];
});
</script>
```

Definition and Usage

The `filter` filter allows us to filter an array, and return an array containing only the matching items.

This filter can only be used for arrays.

Syntax

```
{{ arrayexpression | filter : expression : comparator }}
```

Parameter Values

Value	Description
expression	The expression used when selecting items from the array. The expression can be of type: String: The array items that match the string will be returned. Object: The object is a pattern to search for in the array. Example: `{"name" : "H", "city": "London"}` will return the array items with a name containing the letter "A", where the city contains the word "Berlin". See example below. Function: A function which will be called for each array item, and items where the function returns true will be in the result array.
comparator	Optional. Defines how strict the comparison should be. The value can be: true : Returns a match only if the value of the array item is exactly what we compare it with. false : Returns a match if the value of the array item *contains* what we compare it with. This comparison is not case sensitiv. This is the default value. function : A function where we can define what will be considered a match or not.

Content Downloaded from www.w3schools.com

More Examples

Example

Use an object as a filter:

```
<div ng-app="myApp" ng-controller="arrCtrl">
<ul>
<li ng-repeat="x in customers | filter : {'name' : 'O', 'city' : 'London'}">
    {{x.name + ", " + x.city}}
</li>
</ul></div>
<script>
var app = angular.module('myApp', []);
app.controller('arrCtrl', function($scope) {
    $scope.customers = [
        {"name" : "Alfreds Futterkiste", "city" : "Berlin"},
        {"name" : "Around the Horn", "city" : "London"},
        {"name" : "B's Beverages", "city" : "London"},
        {"name" : "Bolido Comidas preparadas", "city" : "Madrid"},
        {"name" : "Bon app", "city" : "Marseille"},
        {"name" : "Bottom-Dollar Marketse" ,"city" : "Tsawassen"},
        {"name" : "Cactus Comidas para llevar", "city" : "Buenos Aires"}
    ];
});</script>
```

Example

Do a "strict" comparison, which does not return a match unless the value is *exactly* the same as the expression:

```
<div ng-app="myApp" ng-controller="arrCtrl">
<ul>
<li ng-repeat="x in customers | filter : 'London' : true">
    {{x.name + ", " + x.city}}
</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('arrCtrl', function($scope) {
    $scope.customers = [
        {"name" : "London Food", "city" : "London"},
        {"name" : "London Catering", "city" : "London City"},
        {"name" : "London Travel", "city" : "Heathrow, London"}
    ];
});</script>
```

Content Downloaded from www.w3schools.com

AngularJS json Filter

Example
Display a JavaScript object as a JSON string:

```
<div ng-app="myApp" ng-controller="jsCtrl">
<h1>Customer:</h1>
<pre>{{customer | json}}</pre>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('jsCtrl', function($scope) {
    $scope.customer = {
        "name" : "Alfreds Futterkiste",
        "city" : "Berlin",
        "country" : "Germany"
    };
});
</script>
```

Definition and Usage

The **json** filter converts a JavaScript object into a JSON string.

This filter can be useful when debugging your applications.

The JavaScript object can be any kind of JavaScript object.

Syntax

```
{{ object | json : spacing }}
```

Parameter Values

Value	Description
spacing	Optional. A number specifying how many spaces to user per indentation. The default value is 2

Content Downloaded from www.w3schools.com

More Examples

Example

Make sure that the JSON string is written with 12 spaces per indentation:

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

<h1>Customer:</h1>

<pre>{{customer | json : 12}}</pre>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('jsCtrl', function($scope) {
    $scope.customer = {
        "name" : "Alfreds Futterkiste",
        "city" : "Berlin",
        "country" : "Germany"
    };
});
</script>
```

Example

The JavaScript object as an array:

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

<h1>Carnames:</h1>

<pre>{{cars | json}}</pre>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('jsCtrl', function($scope) {
    $scope.cars = ["Audi", "BMW", "Ford"];
});
</script>
```

Content Downloaded from www.w3schools.com

AngularJS limitTo Filter

Example

Display only the first three items of an array:

```
<div ng-app="myApp" ng-controller="sizeCtrl">
<ul>
<li ng-repeat="x in cars | limitTo : 3">{{x}}</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('sizeCtrl', function($scope) {
    $scope.cars = ["Audi", "BMW", "Dodge", "Fiat", "Ford", "Volvo"];
});
</script>
```

Definition and Usage

The **limitTo** filter returns an array or a string containing only a specified number of elements.

When the **limitTo** filter is used for arrays, it returns an array containing only the specified number of items.

When the **limitTo** filter is used for strings, it returns a string containing, only the specified number of characters.

When the **limitTo** filter is used for numbers, it returns a string containing only the specified number of digits.

Use negative numbers to return elements starting from the end of the element, instead of the beginning.

Syntax

```
{{ object | limitTo : limit : begin }}
```

Content Downloaded from www.w3schools.com

Parameter Values

Value	Description
limit	A number, specifying how many elements to return
begin	Optional. A number specifying where to begin the limitation. Default is 0

Example

Display the *last* three items of the array:

```
<div ng-app="myApp" ng-controller="sizeCtrl">
<ul>
<li ng-repeat="x in cars | limitTo : -3">{{x}}</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('sizeCtrl', function($scope) {
    $scope.cars = ["Audi", "BMW", "Dodge", "Fiat", "Ford", "Volvo"];
});
</script>
```

Example

Display three items, starting at position 1:

```
<div ng-app="myApp" ng-controller="sizeCtrl">
<ul>
<li ng-repeat="x in cars | limitTo : 3 : 1">{{x}}</li>
</ul>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('sizeCtrl', function($scope) {
    $scope.cars = ["Audi", "BMW", "Dodge", "Fiat", "Ford", "Volvo"];
});
</script>
```

Content Downloaded from www.w3schools.com

Example

Display the first three characters of the string:

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

<h1>{{txt | limitTo : 3}}</h1>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('sizeCtrl', function($scope) {
    $scope.txt = "Hello, welcome to AngularJS";
});
</script>
```

Example

Display the first three digits og the number:

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

<h1>{{phone | limitTo : 3}}</h1>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('sizeCtrl', function($scope) {
$scope.phone = "123456789";
});
</script>
```

Content Downloaded from www.w3schools.com

AngularJS lowercase Filter

Example
Display the text in lowercase letters:

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

<h1>{{txt | lowercase}}</h1>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('caseCtrl', function($scope) {
    $scope.txt = "Hello World!";
});
</script>
```

Definition and Usage

The **lowercase** filter converts a string to lowercase letters.

Syntax

```
{{ string | lowercase }}
```

Content Downloaded from www.w3schools.com

AngularJS number Filter

Example
Format the prize as a number::

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

<h1>{{prize | number}}</h1>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('nCtrl', function($scope) {
    $scope.prize = 1000000;
});
</script>
```

Definition and Usage

The **number** filter formats a number to a string.

Syntax

{{ *string* | number : *fractionsize*}}

Parameter Values

Value	Description
fractionsize	A number, specifying the number of decimals.

More Examples

Example
Display the weight with 3 decimals:

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

<h1>{{weight | number : 3}} kg</h1>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('nCtrl', function($scope) {
    $scope.weight = 9999;
});
</script>
```

AngularJS orderBy Filter

Example
Display the items alphabetically:

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

<ul>
<li ng-repeat="x in cars | orderBy">{{x}}</li>
</ul>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('orderCtrl', function($scope) {
    $scope.cars = ["Dodge", "Fiat", "Audi", "Volvo", "BMW", "Ford"];
});
</script>
```

Definition and Usage

The **orderBy** filter allows us to sort an array.

By default, strings are sorted alphabetically, and numbers are sorted numerically.

Syntax

```
{{ array | orderBy : expression : reverse }}
```

Parameter Values

Value	Description
expression	The expression used to determine the order. The expression can be of type: String: If the array is an array of objects, you can sort the array by the value of one of the object properties. See the examples below. Function: You can create a function to organize the sorting. Array: Use an array if you need more than one object property to determine the sorting order. The array items can be both strings and functions.
reverse	Optional. Set to true if you want to reverse the order of the array.

Example

Sort the array by "city":

```
<div ng-app="myApp" ng-controller="orderCtrl">
<ul>
<li ng-repeat="x in customers | orderBy : 'city'">{{x.name + ", " +
x.city}}</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('orderCtrl', function($scope) {
    $scope.customers = [
        {"name" : "Bottom-Dollar Marketse" ,"city" : "Tsawassen"},
        {"name" : "Alfreds Futterkiste", "city" : "Berlin"},
        {"name" : "Bon app", "city" : "Marseille"},
        {"name" : "Cactus Comidas para llevar", "city" : "Buenos Aires"},
        {"name" : "Bolido Comidas preparadas", "city" : "Madrid"},
        {"name" : "Around the Horn", "city" : "London"},
        {"name" : "B's Beverages", "city" : "London"}
    ];
});
</script>
```

Content Downloaded from www.w3schools.com

Example

Sort the array by "city", in descending order:

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

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

</div>

<script>
var app = angular.module('myApp', []);
app.controller('orderCtrl', function($scope) {
    $scope.customers = [
        {"name" : "Bottom-Dollar Marketse" ,"city" : "Tsawassen"},
        {"name" : "Alfreds Futterkiste", "city" : "Berlin"},
        {"name" : "Bon app", "city" : "Marseille"},
        {"name" : "Cactus Comidas para llevar", "city" : "Buenos Aires"},
        {"name" : "Bolido Comidas preparadas", "city" : "Madrid"},
        {"name" : "Around the Horn", "city" : "London"},
        {"name" : "B's Beverages", "city" : "London"}
    ];
});
</script>
```

Content Downloaded from www.w3schools.com

AngularJS uppercase Filter

Example
Display the text in uppercase letters:

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

<h1>{{txt | uppercase}}</h1>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('caseCtrl', function($scope) {
    $scope.txt = "Hello World!";
});
</script>
```

Definition and Usage

The **uppercase** filter converts a string to uppercase letters.

Syntax

```
{{ string | uppercase}}
```

Content Downloaded from www.w3schools.com

AngularJS Validation Properties

- $dirty
- $invalid
- $error

AngularJS Global API Converting

API	**Description**
angular.lowercase()	Converts a string to lowercase
angular.uppercase()	Converts a string to uppercase
angular.copy()	Creates a deep copy of an object or an array
angular.forEach()	Executes a function for each element in an object or array

Content Downloaded from www.w3schools.com

Comparing

API	Description
angular.isArray()	Returns true if the reference is an array
angular.isDate()	Returns true if the reference is a date
angular.isDefined()	Returns true if the reference is defined
angular.isElement()	Returns true if the reference is a DOM element
angular.isFunction()	Returns true if the reference is a function
angular.isNumber()	Returns true if the reference is a number
angular.isObject()	Returns true if the reference is an object
angular.isString()	Returns true if the reference is a string
angular.isUndefined()	Returns true if the reference is undefined
angular.equals()	Returns true if two references are equal

Content Downloaded from www.w3schools.com

JSON

API	Description
angular.fromJSON()	Deserializes a JSON string
angular.toJSON()	Serializes a JSON string

Basic

API	Description
angular.bootstrap()	Starts AngularJS manually
angular.element()	Wraps an HTML element as an jQuery element
angular.module()	Creates, registers, or retrieves an AngularJS module

Content Downloaded from www.w3schools.com