AngularJS extends HTML with **ng-directives**.

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

The **ng-model** directive binds the value of the input field to the application variable **name**.

The **ng-bind** directive binds the content of the <p> element to the application variable **name**.

<div ng-app="">

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

<h1 ng-bind="name"></h1>

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

<div ng-app="" ng-init="firstname='sujeet'">

<h1 ng-bind="firstname">

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

--<div data-ng-app="" data-ng-init="name='amit'">

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

Expression:

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

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

## 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.

Note:

AngularJS Expressions

AngularJS binds data to HTML using **Expressions**.

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

AngularJS expressions can also be written inside a directive: ng-bind="*expression*".

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

Using object:

<div ng-init="person={firstname:'sujeet', lastname:'alok'}"></div>

**{{**person.lastname**}}**

**Module:**

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.

Var app=angular.module(‘myapp’,[]);

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

Controller with module

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

## Adding a Directive

AngularJS has a set of built-in directives which you can use to add functionality to your application.

For a full reference, visit our [AngularJS directive reference](https://www.w3schools.com/angular/angular_ref_directives.asp).

In addition you can use the module to add your own directives to your applications:

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

You can invoke a directive by using:

* Element name
* Attribute
* Class
* Comment

The examples below

< w3TestDirective ></ w3TestDirective >

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

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

<script>

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

app.directive("w3TestDirective", function() {

return {

restrict : "A",

template : "<h1>Made by a directive!</h1>"

};

});

</script>

<p><strong>Note:</strong> By setting the <strong>restrict</strong> property to "A", only the HTML element with the "w3-test-directive" attribute has invoked the directive.</p>

</body>

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.

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.

## When to Load the Library

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

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

### Example

<!DOCTYPE html>  
<html>  
<body>  
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>  
  
<div ng-app="myApp" ng-controller="myCtrl">  
{{ firstName + " " + lastName }}  
</div>  
  
<script>  
var app = angular.module("myApp", []);  
app.controller("myCtrl", function($scope) {  
    $scope.firstName = "John";  
    $scope.lastName = "Doe";  
});  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/angular/tryit.asp?filename=try_ng_module_body)

# AngularJS ng-model Directive

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

Two way data binding

The ng-model directive provides a two-way binding between the model and the view.

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

## Validate User Input

<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>

## CSS Classes

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

<style>

input.ng-invalid {  
    background-color: lightblue;  
}

</style>  
<body>  
  
<form ng-app="" name="myForm">  
    Enter your name:  
    <input name="myName" ng-model="myText" required>  
</form>

## Two-way Binding

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

When data in the model changes, the view reflects the change, and when data in the view changes, the modelis updated as well. This happ

<div ng-app="myApp" ng-controller="myCtrl">  
    Name: <input ng-model="firstname">  
    <h1>{{firstname}}</h1>  
</div>  
  
<script>

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

</script>

# AngularJS Scope

The scope is the binding part between the HTML (view) and the JavaScript (controller).

The scope is an object with the available properties and methods.

The scope is available for both the view and the controller.

## Root Scope

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

The rootScope is available in the entire application.

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

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.

<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>

## Custom Filters

You can make your own filters by registering 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 = "";  
        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>

# AngularJS Services

## 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;  
    });  
});

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);  
    }  
});

# AngularJS AJAX - $http

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

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

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

<script>

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

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

$http({

method: "GET",

url: "Testpage.html"

}).then(function mySuccess(response) {

$scope.myWelcome = response.data;

}, function myError(response) {

$scope.myWelcome = response.statusText;

});

});

</script>

The response from the server is an object with these properties:

* .config the object used to generate the request.
* .data a string, or an object, carrying the response from the server.
* .headers a function to use to get header information.
* .status a number defining the HTTP status.
* .statusText a string defining the HTTP status.

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

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 <https://www.w3schools.com/angular/customers.php>.

# AngularJS Tables

The ng-repeat directive is perfect for displaying tables.

## Display the Table Index ($index)

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

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

## Using $even and $odd

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

# AngularJS Select Boxes

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

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

<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>

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

# AngularJS SQL

## Fetching Data From an ASP.NET Server Running SQL

<div ng-app="myApp" ng-controller="customersCtrl">  
  
<table>  
  <tr ng-repeat="x in names">  
    <td>{{ x.Name }}</td>  
    <td>{{ x.Country }}</td>  
  </tr>  
</table>  
  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('customersCtrl', function($scope, $http) {  
    $http.get("customers\_sql.aspx")  
    .then(function (response) {$scope.names = response.data.records;});  
});  
</script>

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: \*");

# AngularJS Events

AngularJS has its own HTML events 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

<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>

## $event Object

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

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

<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.count = 0;

$scope.myFunc = function (p) {

$scope.x = p.clientX;

$scope.y = p.clientY;

}

});

</script>

# AngularJS Forms

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

Input controls are the HTML input elements:

* input elements
* select elements
* button elements
* textarea elements

<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>

<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>

<p>The ng-switch directive hides and shows HTML sections depending on the value of the radio buttons.</p>

# AngularJS Forms

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>

# AngularJS API

## 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:

|  |  |
| --- | --- |
| 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 |

--angular.uppercase exampel

<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>

## AngularJS Includes

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

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

## 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:

<body ng-app="myApp">  
  
<div ng-include="'https://tryit.w3schools.com/angular\_include.php'"></div>  
  
<script>

var app = angular.module('myApp', [])  
app.config(function($sceDelegateProvider) {  
    $sceDelegateProvider.resourceUrlWhitelist([  
        'https://tryit.w3schools.com/\*\*'  
    ]);  
});

</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 do I Need?

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

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

<!DOCTYPE html>

<html>

<style>

div {

transition: all linear 0.5s;

background-color: lightblue;

height: 100px;

width: 100%;

position: relative;

top: 0;

left: 0;

}

.ng-hide {

height: 0;

width: 0;

background-color: transparent;

top: -200px;

left: 200px;

}

</style>

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

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

<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>

</body>

</html>

# AngularJS Routing

The ngRoute module helps your application to become a Single Page Application.

If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the ngRoute module.

The ngRoute module routes your application to different pages without reloading the entire application.

Navigate to "red.htm", "green.htm", and "blue.htm":

<body ng-app="myApp">  
  
<p><a href="#/!">Main</a></p>  
  
<a href="#!red">Red</a>  
<a href="#!green">Green</a>  
<a href="#!blue">Blue</a>  
  
<div ng-view></div>  
  
<script>  
var app = angular.module("myApp", ["ngRoute"]);  
app.config(function($routeProvider) {  
    $routeProvider  
    .when("/", {  
        templateUrl : "main.htm"  
    })  
    .when("/red", {  
        templateUrl : "red.htm"  
    })  
    .when("/green", {  
        templateUrl : "green.htm"  
    })  
    .when("/blue", {  
        templateUrl : "blue.htm"  
    });  
});  
</script>  
</body>

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

Then you must add the ngRoute as a dependency in the application module:

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

## Where Does it Go?

Your application needs a container to put the content provided by the routing.

This container is the ng-view directive.

There are three different ways to include the ng-view directive in your application:

<div class="ng-view"></div>

<ng-view></ng-view>

<div ng-view></div>