Prerequisite:

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

* HTML
* CSS
* JavaScript: OOPS concept is required

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript>

**Lets Begin:**

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**.

Example-1:

Hello AngularJS:

Download Framework from NuGet packages

Including angular scripts from the Google CDN

The quickest way to get started is to point your html <script> tag to a Google CDN URL. This way, you don't have to download anything or maintain a local copy.

There are two types of angular script URLs you can point to, one for development and one for production:

* **angular.js** — This is the human-readable, non-minified version, suitable for web development.
* **angular.min.js** — This is the minified version, which we strongly suggest you use in production.

To point your code to an angular script on the Google CDN server, use the following template. This example points to the minified version 1.3.14:

<!doctype html>

<html ng-app>

<head>

<title>My Angular App</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

</head>

<body>

</body>

</html>

Note that only versions 1.0.1 and above are available on the CDN, if you need an earlier version you can use the<http://code.angularjs.org/> URL which was the previous recommended location for hosted code source. If you're still using the angular server you should switch to the CDN version for even faster loading times.

Downloading and hosting angular files locally

This option is for those who want to work with angular offline, or those who want to host the angular files on their own servers.

If you navigate to <http://code.angularjs.org/>, you'll see a directory listing with all of the angular versions since we started releasing versioned build artifacts (quite late in the project lifetime). Each directory contains all artifacts that we released for a particular version. Download the version you want and have fun.

Each directory under <http://code.angularjs.org/> includes the following set of files:

* **angular.js** — This file is non-obfuscated, non-minified, and human-readable by opening it it any editor or browser. In order to get better error messages during development, you should always use this non-minified angular script.
* **angular.min.js** — This is a minified and obfuscated version of angular.js created with the Closure compiler. Use this version for production in order to minimize the size of the application that is downloaded by your user's browser.
* **angular.zip** — This is a zip archive that contains all of the files released for this angular version. Use this file to get everything in a single download.
* **angular-mocks.js** — This file contains an implementation of mocks that makes testing angular apps even easier. Your unit/integration test harness should load this file after angular.js is loaded.
* **angular-scenario.js** — This file is a very nifty JavaScript file that allows you to write and execute end-to-end tests for angular applications.
* **angular-loader.min.js** — Module loader for Angular modules. If you are loading multiple script files containing Angular modules, you can load them asynchronously and in any order as long as you load this file first. Often the contents of this file are copy&pasted into the index.html to avoid even the initial request to angular-loader.min.js. See [angular-seed](https://github.com/angular/angular-seed/blob/master/app/index-async.html) for an example of usage.
* **Additional Angular modules:** optional modules with additional functionality. These files should be loaded after the coreangular.js file:
  + **angular-animate.js** - Enable animation support
  + **angular-cookies.js** - A convenient wrapper for reading and writing browser cookies
  + **angular-resource.js** - Interaction support with RESTful services via the $resource service
  + **angular-route.js** - Routing and deeplinking services and directives for angular apps
  + **angular-sanitize.js** - Functionality to sanitize HTML
  + **angular-touch.js** - Touch events and other helpers for touch-enabled devices
* 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 **innerHTML** of the <p> element to the application variable **name**.
* The **ng-init** directive initialize AngularJS application variables.

Example-2:

## **AngularJS Applications**

AngularJS **modules** define AngularJS applications.

AngularJS **controllers** control AngularJS applications view.

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

AngularJS **modules**:

You can think of a module as a container for the different parts of your app – controllers, services, filters, directives, etc.

Why?

Most applications have a main method that instantiates and wires together the different parts of the application.

Angular apps don't have a main method. Instead modules declaratively specify how an application should be bootstrapped. There are several advantages to this approach:

* The declarative process is easier to understand.
* You can package code as reusable modules.
* The modules can be loaded in any order (or even in parallel) because modules delay execution.
* Unit tests only have to load relevant modules, which keeps them fast.
* End-to-end tests can use modules to override configuration.

<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>

* The reference to myApp module in <div ng-app="myApp">. This is what bootstraps the app using your module.
* The empty array in angular.module('myApp', []). This array is the list of modules myApp depends on.

We should break our application to multiple modules like:

* A module for each feature
* A module for each reusable component (especially directives and filters)
* And an application level module which depends on the above modules and contains any initialization code.

Module Loading & Dependencies

A module is a collection of configuration and run blocks which get applied to the application during the bootstrap process. In its simplest form the module consist of a collection of two kinds of blocks:

1. **Configuration blocks** - get executed during the provider registrations and configuration phase. Only providers and constants can be injected into configuration blocks. This is to prevent accidental instantiation of services before they have been fully configured.
2. **Run blocks** - get executed after the injector is created and are used to kickstart the application. Only instances and constants can be injected into run blocks. This is to prevent further system configuration during application run time.

For more details please follow the link:

<https://docs.angularjs.org/guide/module>

**AngularJS Controller:**

In Angular, a Controller is a JavaScript **constructor function** that is used to augment the [Angular Scope](https://docs.angularjs.org/guide/scope).

When a Controller is attached to the DOM via the [ng-controller](https://docs.angularjs.org/api/ng/directive/ngController) directive, Angular will instantiate a new Controller object, using the specified Controller's **constructor function**. A new **child scope** will be available as an injectable parameter to the Controller's constructor function as $scope.

Use controllers to:

* Set up the initial state of the $scope object.
* Add behavior to the $scope object.

Do not use controllers to:

* Manipulate DOM — Controllers should contain only business logic. Putting any presentation logic into Controllers significantly affects its testability. Angular has [databinding](https://docs.angularjs.org/guide/databinding) for most cases and [directives](https://docs.angularjs.org/guide/directive) to encapsulate manual DOM manipulation.
* Format input — Use [angular form controls](https://docs.angularjs.org/guide/forms) instead.
* Filter output — Use [angular filters](https://docs.angularjs.org/guide/filter) instead.
* Share code or state across controllers — Use [angular services](https://docs.angularjs.org/guide/services) instead.
* Manage the life-cycle of other components (for example, to create service instances).

## **Setting up the initial state of a $scope object**

Typically, when you create an application you need to set up the initial state for the Angular $scope. You set up the initial state of a scope by attaching properties to the $scope object. The properties contain the **view model** (the model that will be presented by the view). All the $scope properties will be available to the [template](https://docs.angularjs.org/guide/templates) at the point in the DOM where the Controller is registered.

## **Adding Behavior to a Scope Object**

In order to react to events or execute computation in the view we must provide behavior to the scope. We add behavior to the scope by attaching methods to the $scope object. These methods are then available to be called from the template/view.

Any objects (or primitives) assigned to the scope become model properties. Any methods assigned to the scope are available in the template/view, and can be invoked via angular expressions and ng event handler directives (e.g. [ngClick](https://docs.angularjs.org/api/ng/directive/ngClick)).

## **Using Controllers Correctly**

In general, a Controller shouldn't try to do too much. It should contain only the business logic needed for a single view.

The most common way to keep Controllers slim is by encapsulating work that doesn't belong to controllers into services and then using these services in Controllers via dependency injection. This is discussed in the [Dependency Injection](https://docs.angularjs.org/guide/di) [Services](https://docs.angularjs.org/guide/services) sections of this guide.

## **Scope Inheritance Example**

It is common to attach Controllers at different levels of the DOM hierarchy. Since the [ng-controller](https://docs.angularjs.org/api/ng/directive/ngController) directive creates a new child scope, we get a hierarchy of scopes that inherit from each other. The $scope that each Controller receives will have access to properties and methods defined by Controllers higher up the hierarchy.

For more details please follow the link:

<https://docs.angularjs.org/guide/controller>

Example-3:

# AngularJS Directives:

# <https://docs.angularjs.org/api/ng/directive>

# ng-app, ng-init, ng-model.

# ng-repeat:

# ngDisabled: This directive sets the disabled attribute on the element if the [expression](https://docs.angularjs.org/guide/expression) inside ngDisabled evaluates to truthy.

<div ng-init="isDisabled = false">

<button disabled="{{isDisabled}}">Disabled</button>

</div>

**ngChecked:** The HTML specification does not require browsers to preserve the values of boolean attributes such as checked. (Their presence means true and their absence means false.) If we put an Angular interpolation expression into such an attribute then the binding information would be lost when the browser removes the attribute. The ngChecked directive solves this problem for the checked attribute. This complementary directive is not removed by the browser and so provides a permanent reliable place to store the binding information.

<label>Check me to check both: <input type="checkbox" ng-model="master"></label><br/>

<input id="checkSlave" type="checkbox" ng-checked="master" aria-label="Slave input">

**ngReadonly:** The HTML specification does not require browsers to preserve the values of boolean attributes such as readonly. (Their presence means true and their absence means false.) If we put an Angular interpolation expression into such an attribute then the binding information would be lost when the browser removes the attribute. The ngReadonly directive solves this problem for the readonly attribute. This complementary directive is not removed by the browser and so provides a permanent reliable place to store the binding information.

<label>Check me to make text readonly: <input type="checkbox" ng-model="checked"></label><br/>

<input type="text" ng-readonly="checked" value="I'm Angular" aria-label="Readonly field" />

**ngSelected:** The HTML specification does not require browsers to preserve the values of boolean attributes such as selected. (Their presence means true and their absence means false.) If we put an Angular interpolation expression into such an attribute then the binding information would be lost when the browser removes the attribute. The ngSelected directive solves this problem for the selected attribute. This complementary directive is not removed by the browser and so provides a permanent reliable place to store the binding information.

<label>Check me to select: <input type="checkbox" ng-model="selected"></label><br/>

<select aria-label="ngSelected demo">

<option>Hello!</option>

<option id="greet" ng-selected="selected">Greetings!</option>

</select>

**ngValue:** Binds the given expression to the value of <option> or [input[radio]](https://docs.angularjs.org/api/ng/input/input%5bradio%5d), so that when the element is selected, the [ngModel](https://docs.angularjs.org/api/ng/directive/ngModel) of that element is set to the bound value.

<script>

angular.module('valueExample', [])

.controller('ExampleController', ['$scope', function($scope) {

$scope.names = ['pizza', 'unicorns', 'robots'];

$scope.my = { favorite: 'unicorns' };

}]);

</script>

<form ng-controller="ExampleController">

<h2>Which is your favorite?</h2>

<label ng-repeat="name in names" for="{{name}}">

{{name}}

<input type="radio"

ng-model="my.favorite"

ng-value="name"

id="{{name}}"

name="favorite">

</label>

<div>You chose {{my.favorite}}</div>

</form>

**ngBind:**

The ngBind attribute tells Angular to replace the text content of the specified HTML element with the value of a given expression, and to update the text content when the value of that expression changes.

<script>

angular.module('bindExample', [])

.controller('ExampleController', ['$scope', function($scope) {

$scope.name = 'Whirled';

}]);

</script>

<div ng-controller="ExampleController">

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

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

</div>

**ngChange:** Evaluate the given expression when the user changes the input. The expression is evaluated immediately, unlike the JavaScript onchange event which only triggers at the end of a change (usually, when the user leaves the form element or presses the return key).

The ngChange expression is only evaluated when a change in the input value causes a new value to be committed to the model.

It will not be evaluated: if the model is changed programmatically and not by a change to the input value

<script>

angular.module('changeExample', [])

.controller('ExampleController', ['$scope', function($scope) {

$scope.counter = 0;

$scope.change = function() {

$scope.counter++;

};

}]);

</script>

<div ng-controller="ExampleController">

<input type="checkbox" ng-model="confirmed" ng-change="change()" id="ng-change-example1" />

<input type="checkbox" ng-model="confirmed" id="ng-change-example2" />

<label for="ng-change-example2">Confirmed</label><br />

<tt>debug = {{confirmed}}</tt><br/>

<tt>counter = {{counter}}</tt><br/>

</div>

|  |  |
| --- | --- |
| [**ngClass**](https://docs.angularjs.org/api/ng/directive/ngClass) | The ngClass directive allows you to dynamically set CSS classes on an HTML element by databinding an expression that represents all classes to be added. |
| [**ngClick**](https://docs.angularjs.org/api/ng/directive/ngClick) | The ngClick directive allows you to specify custom behavior when an element is clicked. |
| [**ngDblclick**](https://docs.angularjs.org/api/ng/directive/ngDblclick) | The ngDblclick directive allows you to specify custom behavior on a dblclick event. |
| [**ngMousedown**](https://docs.angularjs.org/api/ng/directive/ngMousedown) | The ngMousedown directive allows you to specify custom behavior on mousedown event. |
| [**ngMouseup**](https://docs.angularjs.org/api/ng/directive/ngMouseup) | Specify custom behavior on mouseup event. |
| [**ngMouseover**](https://docs.angularjs.org/api/ng/directive/ngMouseover) | Specify custom behavior on mouseover event. |
| [**ngMouseenter**](https://docs.angularjs.org/api/ng/directive/ngMouseenter) | Specify custom behavior on mouseenter event. |
| [**ngMouseleave**](https://docs.angularjs.org/api/ng/directive/ngMouseleave) | Specify custom behavior on mouseleave event. |
| [**ngMousemove**](https://docs.angularjs.org/api/ng/directive/ngMousemove) | Specify custom behavior on mousemove event. |
| [**ngKeydown**](https://docs.angularjs.org/api/ng/directive/ngKeydown) | Specify custom behavior on keydown event. |
| [**ngKeyup**](https://docs.angularjs.org/api/ng/directive/ngKeyup) | Specify custom behavior on keyup event. |
| [**ngKeypress**](https://docs.angularjs.org/api/ng/directive/ngKeypress) | Specify custom behavior on keypress event. |
| [**ngSubmit**](https://docs.angularjs.org/api/ng/directive/ngSubmit) | Enables binding angular expressions to onsubmit events. |
| [**ngFocus**](https://docs.angularjs.org/api/ng/directive/ngFocus) | Specify custom behavior on focus event. |
| [**ngBlur**](https://docs.angularjs.org/api/ng/directive/ngBlur) | Specify custom behavior on blur event. |
| [**ngCopy**](https://docs.angularjs.org/api/ng/directive/ngCopy) | Specify custom behavior on copy event. |
| [**ngCut**](https://docs.angularjs.org/api/ng/directive/ngCut) | Specify custom behavior on cut event. |
| [**ngPaste**](https://docs.angularjs.org/api/ng/directive/ngPaste) | Specify custom behavior on paste event. |
| [**ngShow**](https://docs.angularjs.org/api/ng/directive/ngShow) | The ngShow directive shows or hides the given HTML element based on the expression provided to the ngShowattribute. The element is shown or hidden by removing or adding the .ng-hide CSS class onto the element. The.ng-hide CSS class is predefined in AngularJS and sets the display style to none (using an !important flag). For CSP mode please add angular-csp.css to your html file (see [ngCsp](https://docs.angularjs.org/api/ng/directive/ngCsp)). |
| [**ngHide**](https://docs.angularjs.org/api/ng/directive/ngHide) | The ngHide directive shows or hides the given HTML element based on the expression provided to the ngHideattribute. The element is shown or hidden by removing or adding the ng-hide CSS class onto the element. The.ng-hide CSS class is predefined in AngularJS and sets the display style to none (using an !important flag). For CSP mode please add angular-csp.css to your html file (see [ngCsp](https://docs.angularjs.org/api/ng/directive/ngCsp)). |

### **Custom Directive:**

<https://docs.angularjs.org/guide/directive>

### **Directive types**

$compile can match directives based on element names, attributes, class names, as well as comments.

All of the Angular-provided directives match attribute name, tag name, comments, or class name. The following demonstrates the various ways a directive (myDir in this case) can be referenced from within a template:

<my-dir></my-dir>

<span my-dir="exp"></span>

<!-- directive: my-dir exp -->

<span class="my-dir: exp;"></span>

## **AngularJS Filters**

<https://docs.angularjs.org/api/ng/filter>

AngularJS filters can be used to transform data:

|  |  |
| --- | --- |
| **Filter** | **Description** |
| currency | Format a number to a currency format. |
| filter | Select a subset of items from an array. |
| lowercase | Format a string to lower case. |
| orderBy | Orders an array by an expression. |
| uppercase | Format a string to upper case. |

## **Form Validation:**

<https://docs.angularjs.org/guide/forms>