7/15/2019

<https://www.tutorialsteacher.com/angularjs/angularjs-filters>

1. What is $scope?

It transfers data from controller (!model) and the view

1. Which of the following is a valid AngularJS expression?  {{ 2 + 2 }}
2. AngularJS directives are used in View.
3. AngularJS directives can be written in HTML element as:

Tag, Attribute and Class name.

1. The ng-change directive must be used with ng-model directives.

True.

1. Which of the following directive allows us to use form?

Ng-form

The ng-app Directive

The **ng-app** directive is a starting point of AngularJS Application. It initializes the AngularJS framework automatically. AngularJS framework will first check for ng-app directive in a HTML document after the entire document is loaded and if ng-app is found, it bootstraps itself and compiles the HTML template.

tipCompiling HTML in AngularJS means attaching event listeners to the HTML to make it interactive.

Typically ng-app directives should be placed at the root of an HTML document e.g. <html> or <body> tag, so that it can control the entire DOM hierarchy. However, you can place it in any DOM element.

# AngularJS Expression

AngularJS expression is like JavaScript expression surrounded with braces - {{ expression }}. AngularJS evaluates the specified expression and binds the result data to HTML.

AngularJS expression can contain literals, operators and variables like JavaScript expression. For example, an expression {{2/2}} will produce the result 1 and will be bound to HTML.

Example: Expression

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<h1>AngularJS Expression Demo:</h1>

<div ng-app>

2 + 2 = {{2 + 2}} <br />

2 - 2 = {{2 - 2}} <br />

2 \* 2 = {{2 \* 2}} <br />

2 / 2 = {{2 / 2}}

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-11)

Result:

2 + 2 = 4

2 - 2 = 0

2 \* 2 = 4

2 / 2 = 1

AngularJS expression is like JavaScript code expression except for the following differences:

1. AngularJS expression cannot contain conditions, loops, exceptions or regular expressions e.g. if-else, ternary, for loop, while loop etc.
2. AngularJS expression cannot declare functions.
3. AngularJS expression cannot contain comma or void.
4. AngularJS expression cannot contain return keyword.

AngularJS expression contains literals of any data type.

Example: Expression

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<h1>AngularJS Expression Demo:</h1>

<div ng-app>

{{"Hello World"}}<br />

{{100}}<br />

{{true}}<br />

{{10.2}}

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-12)

Result:

Hello World

100

True

10.2

AngularJS expression can contain arithmetic operators which will produce the result based on the type of operands, similar to JavaScript:

Example: Expression

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<div ng-app>

{{"Hello" + " World"}}<br />

{{100 + 100 }}<br />

{{true + false}}<br />

{{10.2 + 10.2}}<br />

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-13)

Result:

Hello World

200

1

20.4

AngularJS expression can contain variables declared via ng-init directive. The ng-init directive is used to declare AngularJS application variables of any data type.

Example: Expression

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<div ng-app ng-init="greet='Hello World!'; amount= 10000;rateOfInterest = 10.5; duration=10; myArr = [100, 200]; person = { firstName:'Steve', lastName :'Jobs'}">

{{ (amount \* rateOfInterest \* duration)/100 }}<br />

{{myArr[1]}} <br />

{{person.firstName + " " + person.lastName}}

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-14)

Result:

10500

200

Steve Jobs

# AngularJS Expression

AngularJS expression is like JavaScript expression surrounded with braces - {{ expression }}. AngularJS evaluates the specified expression and binds the result data to HTML.

AngularJS expression can contain literals, operators and variables like JavaScript expression. For example, an expression {{2/2}} will produce the result 1 and will be bound to HTML.

Example: Expression

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<h1>AngularJS Expression Demo:</h1>

<div ng-app>

2 + 2 = {{2 + 2}} <br />

2 - 2 = {{2 - 2}} <br />

2 \* 2 = {{2 \* 2}} <br />

2 / 2 = {{2 / 2}}

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-11)

Result:

2 + 2 = 4

2 - 2 = 0

2 \* 2 = 4

2 / 2 = 1

AngularJS expression is like JavaScript code expression except for the following differences:

1. AngularJS expression cannot contain conditions, loops, exceptions or regular expressions e.g. if-else, ternary, for loop, while loop etc.
2. AngularJS expression cannot declare functions.
3. AngularJS expression cannot contain comma or void.
4. AngularJS expression cannot contain return keyword.

AngularJS expression contains literals of any data type.

Example: Expression

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<h1>AngularJS Expression Demo:</h1>

<div ng-app>

{{"Hello World"}}<br />

{{100}}<br />

{{true}}<br />

{{10.2}}

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-12)

Result:

Hello World

100

True

10.2

AngularJS expression can contain arithmetic operators which will produce the result based on the type of operands, similar to JavaScript:

Example: Expression

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<div ng-app>

{{"Hello" + " World"}}<br />

{{100 + 100 }}<br />

{{true + false}}<br />

{{10.2 + 10.2}}<br />

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-13)

Result:

Hello World

200

1

20.4

AngularJS expression can contain variables declared via ng-init directive. The ng-init directive is used to declare AngularJS application variables of any data type.

Example: Expression

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<div ng-app ng-init="greet='Hello World!'; amount= 10000;rateOfInterest = 10.5; duration=10; myArr = [100, 200]; person = { firstName:'Steve', lastName :'Jobs'}">

{{ (amount \* rateOfInterest \* duration)/100 }}<br />

{{myArr[1]}} <br />

{{person.firstName + " " + person.lastName}}

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-14)

Result:

10500

200

Steve Jobs

AngularJS Validation CSS Classes

AngularJS includes following CSS classes to allow styling of form and input controls based on the state of form field.

| CSS Class | Description |
| --- | --- |
| ng-valid | Angular will set this CSS class if the input field is valid without errors. |
| ng-invalid | Angular will set this CSS class if the input does not pass validations. |
| ng-pristine | Angular will set this CSS class if a user has not interacted with the control yet. |
| ng-dirty | Angular will set this CSS class if the value of form field has been changed. |
| ng-touched | Angular will set this CSS class if a user tabbed out from the input control. |
| ng-untouched | Angular will set this CSS class if a user has not tabbed out from the input control. |
| ng-submitted | Angular will set this CSS class if the form has been submitted. |

Note that you must provide implementation of these CSS classes and include in your CSS file. AngularJS automatically includes these classes based on the current state of input controls.

The following example demonstrates ng-pristine, ng-touched, ng-valid, and ng-invalid classes to display validity of each form control.

Example: AngularJS Validation CSS Classes

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

<style>

input.ng-pristine {

background-color:yellow;

}

input.ng-touched.ng-invalid {

background-color:red;

}

input.ng-touched.ng-valid {

background-color:green;

}

</style>

</head>

<body ng-app>

<form name="studentForm" novalidate class="student-form">

<label for="firstName">First Name: </label> <br />

<input type="text" name="firstName" ng-model="firstName" ng-required="true" />

<span ng-show="studentForm.firstName.$touched && studentForm.firstName.$error.required">First name is required.</span><br /><br />

<label for="lastName">Last Name</label><br />

<input type="text" name="lastName" ng-minlength="3" ng-maxlength="10" ng-model="lastName" />

<span ng-show="studentForm.lastName.$touched && studentForm.lastName.$error.minlength">min 3 chars.</span>

<span ng-show="studentForm.lastName.$touched && studentForm.lastName.$error.maxlength">Max 10 chars.</span><br /><br />

<label for="dob">Email</label><br />

<input type="email" id="email" ng-model="email" name="email" />

<span ng-show="studentForm.email.$touched && studentForm.email.$error.email">Please enter valid email id.</span><br /><br />

<input type="submit" value="Save" />

</form>

</body>

</html>



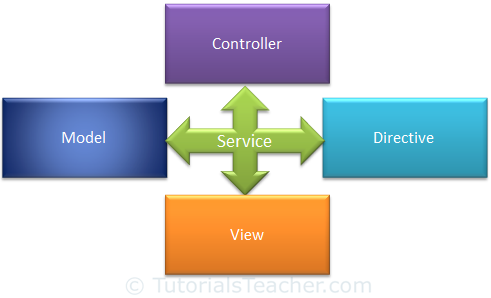
# AngularJS Service

AngularJS services are JavaScript functions for specific tasks, which can be reused throughout the application.

tipAngularJS built-in services starts with $, same as other built-in objects.

AngularJS includes services for different purposes. For example, $http service can be used to send an AJAX request to the remote server. AngularJS also allows you to create custom service for your application.

Most AngularJS services interact with the controller, model or custom directives. However, some services interact with view (UI) also for UI specific tasks.

[](https://www.tutorialsteacher.com/Content/images/ng/ng-service.png)Services

The following table lists all the built-in AngularJS services.

|  |  |  |  |
| --- | --- | --- | --- |
| $anchorScroll | $exceptionHandler | $interval | $rootScope |
| $animate | $filter | $locale | $sceDelegate |
| $cacheFactory | $httpParamSerializer | $location | $sce |
| $templateCache | $httpParamSerializerJQLike | $log | $templateRequest |
| $compile | $http | $parse | $timeout |
| $controller | $httpBackend | $q | $window |
| $document | $interpolate | $rootElement |  |

All the Angular services are **lazy instantiated** and **singleton**. It means AngularJS framework instantiates a service when an application component depends on it. Also, all the components share the same instance of a service.

Learn some of the important built-in services in the next section.

# $http Service

The $http service is used to send or receive data from the remote server using browser's XMLHttpRequest or JSONP.

$http is a service as an object. It includes following shortcut methods.

| Method | Description |
| --- | --- |
| $http.get() | Perform Http GET request. |
| $http.head() | Perform Http HEAD request. |
| $http.post() | Perform Http POST request. |
| $http.put() | Perform Http PUT request. |
| $http.delete() | Perform Http DELETE request. |
| $http.jsonp() | Perform Http JSONP request. |
| $http.patch() | Perform Http PATCH request. |

Let's look at some of the important methods of $http.

## **$http.get()**

$http.get() method sends http GET request to the remote server and retrieves the data.

Signature: HttpPromise $http.get(url)

$http.get() method returns HttpPromise object, which includes various methods to process the response of http GET request.

The following example demonstrates the use of $http service in a controller to send HTTP GET request.

Example: $http.get()

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app ="myApp">

<div>

<div ng-controller="myController">

Response Data: {{data}} <br />

Error: {{error}}

</div>

</div>

<script>

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

myApp.controller("myController", function ($scope, $http) {

var onSuccess = function (data, status, headers, config) {

$scope.data = data;

};

var onError = function (data, status, headers, config) {

$scope.error = status;

}

var promise = $http.get("/demo/getdata");

promise.success(onSuccess);

promise.error(onError);

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-51)

In the above example, 'myController' controller includes $http parameter, so that it can be used to send GET request. AngularJS automatically injects $scope parameter at runtime. The $http.get() method returns HttpPromise which includes methods like success() and error(). The success() method registers a callback method which is called when a request completes successfully. The error() method registers a callback method which is called when a request fails and returns an error.

The onSuccess() method above, attaches the response data to the $scope. The onError() method attaches status property to the $scope. These methods can be called in chain, as shown below.

Example: $http.get()

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app ="myApp">

<div>

<div ng-controller="myController">

Response Data: {{data}} <br />

Error: {{error}}

</div>

</div>

<script>

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

myApp.controller("myController", function ($scope, $http) {

var onSuccess = function (data, status, headers, config) {

$scope.data = data;

};

var onError = function (data, status, headers, config) {

$scope.error = status;

}

var promise = $http.get("/demo/getdata").success(onSuccess).error(onError);

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-52)

## **$http.post**

The $http.post() method sends Http POST request to the remote server to submit and retrieve the data.

Signature: HttpPromise $http.post(url, dataToSubmit);

The following example demonstrates $http.post() method.

Example: $http.post()

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

Response Data: {{data}} <br />

Error: {{error}}

</div>

<script>

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

myApp.controller("myController", function ($scope, $http) {

var onSuccess = function (data, status, headers, config) {

$scope.data = data;

};

var onError = function (data, status, headers, config) {

$scope.error = status;

}

$http.post('/demo/submitData', { myData: 'Hello World!' })

.success(onSuccess)

.error(onError);

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-53)

## **$http()**

You can use construction function of $http service to perform http request, as shown below.

Example: $http()

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

Response Data: {{data}} <br />

Error: {{error}}

</div>

<script>

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

myApp.controller("myController", function ($scope, $http) {

var onSuccess = function (data, status, headers, config) {

$scope.data = data;

};

var onError = function (data, status, headers, config) {

$scope.error = status;

}

var getReq = {

method: 'GET',

url: '/demo/getdata'

};

$http(getReq).success(onSuccess).error(onError);

var postReq = {

method: 'POST',

url: '/demo/submitData',

data: { myData: 'test data' }

};

$http(postReq).success(onSuccess).error(onError);

});

</script>

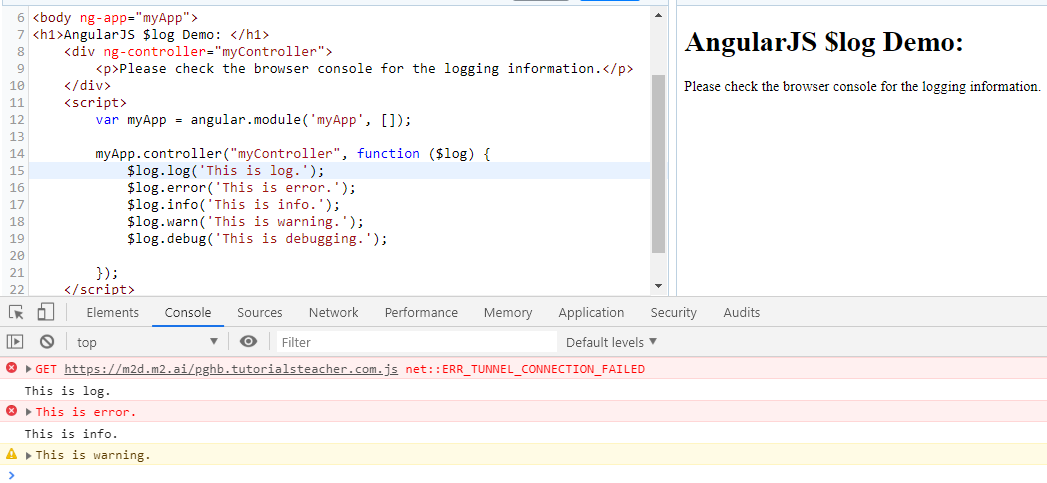
</body>

</html>

# $log Service

AngularJs includes logging service $log, which logs the messages to the browser's console.

The $log service includes different methods to log the error, information, warning or debug information. It can be useful in debugging and auditing.



# $interval Service

AngularJS includes $interval service which performs the same task as setInterval() method in JavaScript. The $interval is a wrapper for setInterval() method, so that it will be easy to override, remove or mocked for testing.

The $interval service executes the specified function on every specified milliseconds duration.

Signature: $interval(fn, delay, [count], [invokeApply], [Pass]);

The following example demonstrates $interval service that displays a counter on each 1000 milliseconds.

Example: $interval

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

{{counter}}

</div>

<script>

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

myApp.controller("myController", function ($scope, $interval) {

$scope.counter = 0;

var increaseCounter = function () {

$scope.counter = $scope.counter + 1;

}

$interval(increaseCounter, 1000);

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-62)

In the above example, $interval service calls increaseCounter() function on every 1000 milliseconds. The increaseCounter() function increases the $scope.counter property by 1. Thus, counter increases on every milliseconds.

## **Execution Count**

The $interval service also executes the specified function for the specified number of times as count parameter.

Example: $interval

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

{{counter}}

</div>

<script>

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

myApp.controller("myController", function ($scope, $interval) {

$scope.counter = 0;

var increaseCounter = function () {

$scope.counter = $scope.counter + 1;

}

$interval(increaseCounter, 1000, 10);

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-63)

In the above example, increaseCounter() method will be executed on each 1000 milliseconds but not more than 10 times.

## **Cancel Execution**

The $interval service returns an object of HttpPromise which can be used to stop the counter by using $interval.cancel(promise) method.

Example: $interval.cancel()

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div>

<div ng-controller="myController">

{{counter}} <br /><br />

<button ng-click="cancel()">Cancel</button>

</div>

</div>

<script>

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

myApp.controller("myController", function ($scope, $interval) {

$scope.counter = 0;

var increaseCounter = function () {

$scope.counter = $scope.counter + 1;

}

var promise = $interval(increaseCounter, 1000);

$scope.cancel = function () {

$interval.cancel(promise);

$scope.counter = "Cancelled!";

};

});

</script>

</body>

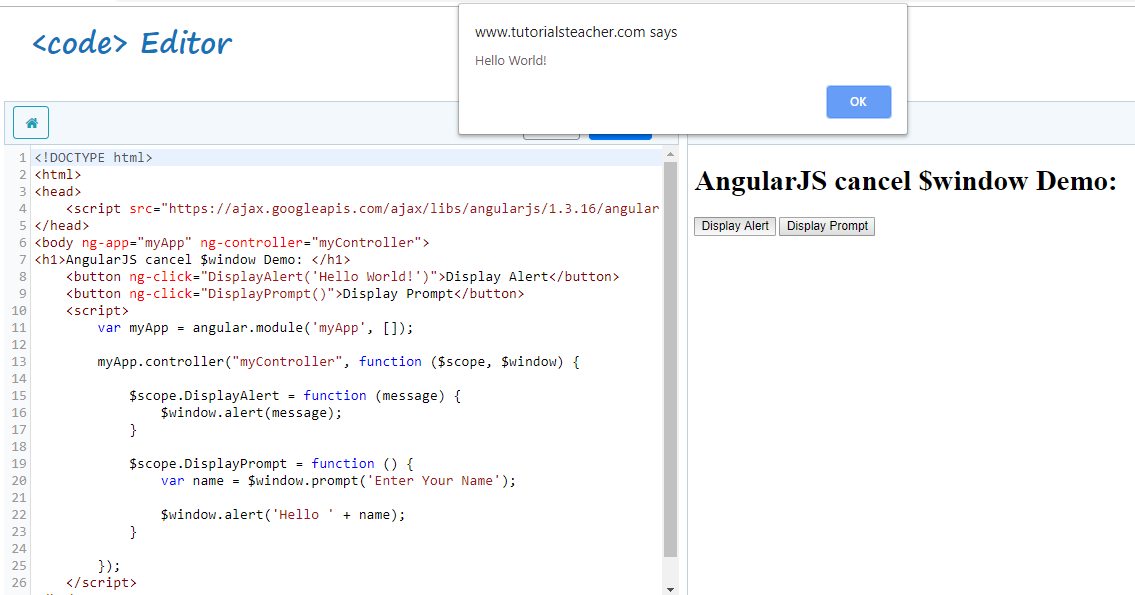
</html>

# $window Service

AngularJs includes $window service which refers to the browser window object.

In the JavaScript, window is a global object which includes many built-in methods like alert(), prompt() etc.

The $window service is a wrapper around window object, so that it will be easy to override, remove or mocked for testing. It is recommended to use $window service in AngularJS instead of global window object directly.



# AngularJS Filters

AngularJS Filters allow us to format the data to display on UI without changing original format.

Filters can be used with an expression or directives using pipe | sign.

{{expression | filterName:parameter }}

Angular includes various filters to format data of different data types. The following table lists important filters.

| Filter Name | Description |
| --- | --- |
| Number | Formats a numeric data as text with comma and fraction. |
| Currency | Formats numeric data into specified currency format and fraction. |
| Date | Formats date to string in specified format. |
| Uppercase | Converts string to upper case. |
| Lowercase | Converts string to lower case. |
| Filter | Filters an array based on specified criteria and returns new array. |
| orderBy | Sorts an array based on specified predicate expression. |
| Json | Converts JavaScript object into JSON string |
| limitTo | Returns new array containing specified number of elements from an existing array. |

**Number Filter**

A number filter formats numeric data as text with comma and specified fraction size.

{{ number\_expression | number:fractionSize}}

If a specified expression does not return a valid number then number filter displays an empty string.

The following example demonstrates how to use number filter with number expression or a model property.

Example: Number filter

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app >

Enter Amount: <input type="number" ng-model="amount" /> <br />

100000 | number = {{100000 | number}} <br />

amount | number = {{amount | number}} <br />

amount | number:2 = {{amount | number:2}} <br />

amount | number:4 = {{amount | number:4}} <br />

amount | number = <span ng-bind="amount | number"></span>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-81)

**Currency Filter**

The currency filter formats a number value as a currency. When no currency symbol is provided, default symbol for current locale is used.

{{ expression | currency : 'currency\_symbol' : 'fraction'}}

Example: Currency filter

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

Default currency: {{person.salary | currency}} <br />

Custom currency identifier: {{person.salary | currency:'Rs.'}} <br />

No Fraction: {{person.salary | currency:'Rs.':0}} <br />

Fraction 2: <span ng-bind="person.salary| currency:'GBP':2"></span>

</div>

<script>

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

myApp.controller("myController", function ($scope) {

$scope.person = { firstName: 'James', lastName: 'Bond', salary: 100000}

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-86)

Result:

Default currency: $100,000.00

Custom currency identifier: Rs.100,000.00

No Fraction: Rs.100,000

Fraction 2: GBP100,000.00

In the above example, we have applied currency filter to person.salary, which is a numeric property. It can be displayed with different currency symbols and fractions.

**Date filter**

Formats date to string based on the specified format.

{{ date\_expression | date : 'format'}}

Example: date filter

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app>

<div ng-init="person.DOB = 323234234898">

Default date: {{person.DOB| date}} <br />

Short date: {{person.DOB| date:'short'}} <br />

Long date: {{person.DOB | date:'longDate'}} <br />

Year: {{person.DOB | date:'yyyy'}} <br />

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-82)

Result:

Default date: Mar 30, 1980

short date: 3/30/80 8:47 AM

long date: March 30, 1980

Year: 1980

Visit Angular documentation for more information on [date](https://docs.angularjs.org/api/ng/filter/date) filter.

**Uppercase/lowercase filter**

The uppercase filter converts the string to upper case and lowercase filter converts the string to lower case.

Example: uppercase & lowercase filters

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app>

<div ng-init="person.firstName='James';person.lastName='Bond'">

Lower case: {{person.firstName + ' ' + person.lastName | lowercase}} <br />

Upper case: {{person.firstName + ' ' + person.lastName | uppercase}}

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-83)

Result:

Lower case: james bond

Upper case: JAMES BOND

**Filter**

Filter selects items from an array based on the specified criteria and returns a new array.

{{ expression | filter : filter\_criteria }}

Example: filter

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

Enter Name to search: <input type="text" ng-model="searchCriteria" /> <br />

Result: {{myArr | filter:searchCriteria}}

</div>

<script>

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

myApp.controller("myController", function ($scope) {

$scope.myArr = ['Steve', 'Bill', 'James', 'Rob', 'Ram', 'Moin']

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-84)

In the above example, searchCriteria contains a text entered in the input box, which will be used to filter items of an array myArr using filter:searchCriteria expression.

**orderBy filter**

The orderBy filter sorts an array based on specified expression predicate.

{{ expression | orderBy : predicate\_expression : reverse}}

Example: orderBy filter

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

<select ng-model="SortOrder">

<option value="+name">Name (asc)</option>

<option value="-name">Name (dec)</option>

<option value="+phone">Phone (asc)</option>

<option value="-phone">Phone (dec)</option>

</select>

<ul ng-repeat="person in persons | orderBy:SortOrder">

<li>{{person.name}} - {{person.phone}}</li>

</ul>

</div>

<script>

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

myApp.controller("myController", function ($scope) {

$scope.persons = [{ name: 'John', phone: '512-455-1276' },

{ name: 'Mary', phone: '899-333-3345' },

{ name: 'Mike', phone: '511-444-4321' },

{ name: 'Bill', phone: '145-788-5678' },

{ name: 'Ram', phone: '433-444-8765' },

{ name: 'Steve', phone: '218-345-5678' }]

$scope.SortOrder = '+name';

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-85)

The above example displays a list of person names and phone numbers in a particular order specified using orderBy:SortOrder filter. SortOrder is a model property and will be set to the selected value in the dropdown. Therefore, based on the value of SortOrder, ng-repeat directive will display the data.

# Ng-click vs onclick

ng-click holds an angular expression. Angular expressions are evaluated in the context of an Angular [scope](http://docs.angularjs.org/guide/scope), which is bound to the element having the ng-click attribute or an ancestor of that element.

The Angular expression language doesn't include flow control statements and can't declare variables or define functions. These limitations mean templates can only access variables and run functions made available by a controller or directive.

**onclick vs addEventListener**

What’s the difference between these two lines of code?

|  |  |
| --- | --- |
|  | element.onclick = function() { /\* do stuff \*/ } |
|  | element.addEventListener('click', function(){ /\* do stuff \*/ }, false); | |

They apparently do the same thing: listen for the click event and execute a callback function. Nevertheless, they’re not equivalent. If you ever need to choose between the two, [this](https://gist.github.com/simonewebdesign/4017724) could help you to figure out which one is the best for you.

The main difference is that onclick **is just a property**, and like all object properties, if you write on more than once, **it will be overwritten**. With addEventListener() instead, we can simply bind an event handler to the element, and we can call it each time we need it without being worried of any overwritten properties.

In first place I was tempted to keep using onclick, because it’s shorter and looks simpler… and in fact it is. But I don’t recommend using it anymore. It’s just like using **inline JavaScript**. Using something like <button onclick="doSomething()"> – *that*’s inline JavaScript – is highly discouraged nowadays (inline CSS is discouraged too, but that’s another topic).

However, the addEventListener() function, despite it’s the standard, just **doesn’t work in old browsers** (Internet Explorer below version 9), and this is another big difference. If you need to support these ancient browsers, you should follow the onclick way. But you could also use [jQuery](https://jquery.com/) (or one of its [alternatives](http://www.jscripters.com/popular-jquery-alternatives/)): it basically simplifies your work and reduces the differences between browsers, therefore can save you a lot of time.

=============&=============

The difference you could see if you had another couple of functions:

var h = document.getElementById('a');

h.onclick = doThing\_1;

h.onclick = doThing\_2;

h.addEventListener('click', doThing\_3);

h.addEventListener('click', doThing\_4);

Functions 2, 3 and 4 work, but 1 does not.  1 gets overridden by 2 and is never called. This is because addEventListener does not overwrite existing event handlers, whereas onclick overrides any existing onclick = fn event handlers.

The other significant difference, of course, is that onclick will always work, whereas addEventListener does not work in Internet Explorer before version 9. You can use the analogous attachEvent (which has slightly different syntax) in IE <9.

So if I need multiple functions for one event, I am stuck with addEventListener, and I have to write more code for attachEvent just to accomodate IE.

# AngularJS Events

AngularJS includes certain directives which can be used to provide custom behavior on various DOM events, such as click, dblclick, mouseenter etc.

The following table lists AngularJS event directives.

| Event Directive |
| --- |
| ng-blur |
| ng-change |
| ng-click |
| ng-dblclick |
| ng-focus |
| ng-keydown |
| ng-keyup |
| ng-keypress |
| ng-mousedown |
| ng-mouseenter |
| ng-mouseleave |
| ng-mousemove |
| ng-mouseover |
| ng-mouseup |

Let's take a look at some of the important event directives.

## **ng-click**

The ng-click directive is used to provide event handler for click event.

Example: ng-click

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

Enter Password: <input type="password" ng-model="password" /> <br />

<button ng-click="DisplayMessage(password)">Show Password</button

</div>

<script>

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

myApp.controller("myController", function ($scope, $window) {

$scope.DisplayMessage = function (value) {

$window.alert(value)

}

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-91)

In the above example, ng-click directive is used to call a DisplayMessage() function with the 'password' parameter when a user clicks a button. A 'password' is a model property defined using ng-model directive in the input box. The DisplayMessage() function is attached to a $scope object in myController, so it will be accessible from button click as button comes under myController. The [$window](https://www.tutorialsteacher.com/angularjs/angularjs-window-service) service is used to display an alert.

## **Mouse Events**

The following example demonstrates important mouse event directives - ng-mouseenter and ng-mouseleave.

Example: Mouse Events

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

<style>

.redDiv {

width: 100px;

height: 100px;

background-color: red;

padding:2px 2px 2px 2px;

}

.yellowDiv {

width: 100px;

height: 100px;

background-color: yellow;

padding:2px 2px 2px 2px;

}

</style>

</head>

<body ng-app>

<div ng-class="{redDiv: enter, yellowDiv: leave}" ng-mouseenter="enter=true;leave=false;" ng-mouseleave="leave=true;enter=false">

Mouse <span ng-show="enter">Enter</span> <span ng-show="leave">Leave</span>

</div>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-92)

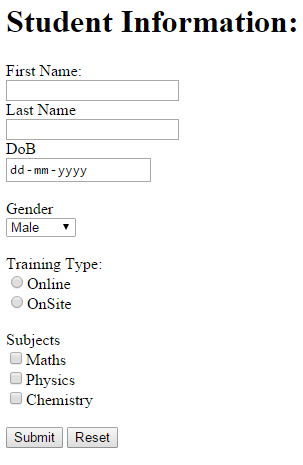
In the above example, the ng-class directive includes map of CSS classes, so redDiv will be applied if enter=true and yellowDiv will be applied if leave=true. The ng-mouseenter directive sets 'enter' to true, which will apply redDiv class to the <div> element. In the same way, ng-mouseleave will set leave to true, which will apply yellowDiv class.

# AngularJS Forms

The HTML form is a collection of input controls where user can enter the data. Here, you will learn how to display AngularJS form and submit the data.

## **An AngularJS Form Example**

We will create following Student Information form with submit and reset functionality.

[](https://www.tutorialsteacher.com/Content/images/ng/angular-form.png)Sample AngularJS Form

The following is the code of the above form.

Example: AngularJS Form

<!DOCTYPE html>

<html ng-app="studentApp">

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-controller="studentController">

<h1>Student Information:</h1>

<form **ng-submit="submitStudnetForm()"** >

<label for="firstName" >First Name: </label><br />

<input type="text" id="firstName" ng-model="student.firstName" /> <br />

<label for="lastName">Last Name</label><br />

<input type="text" id="lastName" ng-model="student.lastName" /> <br />

<label for="dob" >DoB</label><br />

<input type="date" id="dob" ng-model="student.DoB" /> <br /><br />

<label for="gender" >Gender</label> <br />

<select id="gender" ng-model="student.gender">

<option value="male">Male</option>

<option value="female">Female</option>

</select><br /> <br />

<span>Training Type:</span><br />

<label><input value="online" type="radio" name="training" ng-model="student.trainingType" />Online</label><br />

<label><input value="onsite" type="radio" name="training" ng-model="student.trainingType" />OnSite</label> <br /><br />

<span>Subjects</span><br />

<label><input type="checkbox" ng-model="student.maths" />Maths</label> <br />

<label><input type="checkbox" ng-model="student.physics" />Physics</label> <br />

<label><input type="checkbox" ng-model="student.chemistry" />Chemistry</label><br /><br />

<input type="submit" value="Submit" />

<input type="reset" ng-click="resetForm()" value="Reset" />

</form>

<script>

//1. create app module

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

//2. create controller

studentApp.controller("studentController", function ($scope, $http) {

//3. attach originalStudent model object

$scope.originalStudent = {

firstName: 'James',

lastName: 'Bond',

DoB: new Date('01/31/1980'),

gender: 'male',

trainingType: 'online',

maths: false,

physics: true,

chemistry: true

};

//4. copy originalStudent to student. student will be bind to a form

$scope.student = angular.copy($scope.originalStudent);

//5. create submitStudentForm() function. This will be called when user submits the form

$scope.submitStudnetForm = function () {

var onSuccess = function (data, status, headers, config) {

alert('Student saved successfully.');

};

var onError = function (data, status, headers, config) {

alert('Error occured.');

}

$http.post('/student/submitData', { student:$scope.student })

.success(onSuccess)

.error(onError);

};

//6. create resetForm() function. This will be called on Reset button click.

$scope.resetForm = function () {

$scope.student = angular.copy($scope.OriginalStudent);

};

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-111)

The following is a step by step explanation of the above example:

1. Create an HTML page and wrap all the necessary input controlls into <form> tag.
2. Create the AngularJS application module in the <script> tag.
3. Create studentController in application module.
4. Create originalStudent object and attach to the $scope with required properties. This will stay unchanged during entire life cycle.
5. Create new student object and attach to the $scope and copy all the properties and values from originalStudent. This student object will be bound to the form using ng-model directive. Therefore, if user changes form values then the student object will also get changed.
6. Create submitStudnetForm function which will get called when user submits the form using Submit button. Here, send http POST request to the remote server to submit the data using [$http service](https://www.tutorialsteacher.com/angularjs/angularjs-service-http).
7. Create resetForm() function, which will reset the form values to the originalStudent values by copying it to student object.
8. Apply ng-app, ng-controller directives.
9. Apply ng-model directives to each HTML input element to bind appropriate properties of student object.
10. Apply ng-submit directive to form which will call submitStudentForm() on the form submit event.
11. Apply ng-click directive to reset button which will call resetForm() on the button click event.

An AngularJS form can be submitted using either ng-submit or ng-click directive but not both.

**Ng-submit:** Binds angular expression to onsubmit event when form does not include action attribute.

**Ng-click:** Binds angular expression to onclick event.

Note : The angular form can be submitted using ng-submit directive on the form tag or using ng-click directive on <input type="submit" /> element. Use either ng-submit or ng-click directive but not both to submit the form. The form will be submitted twice if both ng-submit and ng-click directives are used.

What is Novalidate form tag?

**novalidate** attribute is used to disable browser's native form validation.

You can use it when you need do your own AngularJS custom validation.

You can use the same ones used by the HTML 5 specification in Angular,so you can add the **novalidate** attribute to the form element, which tells the browser **not** to use its **native validation**. Because different browsers have different implementation validations. Since Angular get validation itself, the browser don't need to do validation implementation.

# What is the difference between novalidate and formnovalidate attributes?

The novalidate and formnovalidate attributes are used to bypass validation. The novalidate attribute is applied to a form and prevents it from validation. The formnovalidate is applied to input type submit button, which overrides the novalidate. It submits the form without validating.

The novalidate attribute is also a Boolean attribute, but using it won’t validate the form of submission. The formnovalidate attribute in HTML is useful when you have a form with more than one submit button.

You can try to run the following code to learn how to use novalidate attribute in HTML. In the following example, if you will add text in the <input type=”number” > field, then it won’t show an error.

<!DOCTYPE html>

<html>

   <head>

      <title>HTML novalidate attribute</title>

   </head>

   <body>

      <form action = "" method = "get" novalidate>

         Team Name<br><input type = "name" name = "tname"><br>

         Team Rank<br><input type = "number" name = "trank"><br>

         <input type = "submit" value = "Submit">

      </form>

   </body>

</html>

You can try to run the following code to learn how to use the formnovalidate attribute in HTML. If you will select the submit button with no validation, then the form won’t get validate.

<!DOCTYPE html>

<html>

   <head>

      <title>HTML formnovalidate attribute</title>

   </head>

   <body>

      <form action = "" method = "get">

         Rank <input type="number" name="rank"><br>

         <input type="submit" value="Submit"><br>

         <input type="submit" formnovalidate="formnovalidate"

            value="Submit with no validation”>

      </form>

   </body>

</html>

# Validation in AngularJS

We created an HTML form in the previous section. Here, we will implement client side validation in AngularJS form.

AngularJS includes the following validation directives.

| Directive | Description |
| --- | --- |
| ng-required | Sets required attribute on an input field. |
| ng-minlength | Sets minlength attribute on an input field. |
| ng-maxlength | Sets maxlength attribute on an input field. Setting the attribute to a negative or non-numeric value, allows view values of any length. |
| ng-pattern | Sets pattern validation error key if the ngModel value does not match the specified RegEx expression. |

Let's implement validation in the student form which contains First Name, Last Name and Email fields.

Example: Form Validation

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app >

<form name="studentForm" novalidate>

<label for="firstName">First Name: </label> <br />

<input type="text" name="firstName" ng-model="student.firstName" ng-required="true" />

<span ng-show="studentForm.firstName.$touched && studentForm.firstName.$error.required">First name is required.</span><br /><br />

<label for="lastName">Last Name</label><br />

<input type="text" name="lastName" ng-minlength="3" ng-maxlength="10" ng-model="student.lastName" />

<span ng-show="studentForm.lastName.$touched && studentForm.lastName.$error.minlength">min 3 chars.</span>

<span ng-show="studentForm.lastName.$touched && studentForm.lastName.$error.maxlength">Max 10 chars.</span><br /><br />

<label for="dob">Email</label><br />

<input type="email" id="email" ng-model="student.email" name="email" />

<span ng-show="studentForm.email.$touched && studentForm.email.$error.email">Please enter valid email id.</span><br /><br />

<input type="submit" value="Submit" />

</form>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-131)

Let's understand the above example step by step:

1. Apply novalidate attribute in <form> tag. The novalidate attribute will disable the browser's default validation.
2. Set the name attribute in <form> and other elements, which will be used to obtain a reference of the elements.
3. Now, set ng-required="true" on the input element of First Name. Also, set name attribute to the name of model property, "firstName" in this case.
4. Create <span> element to specify an error message with every input filed where the validation directive is applied.
5. Set ng-show directives to <span> element to an expression "studentForm.firstName.$touched && studentForm.firstName.$error.required". This expression will return true if a user tabbed out without entering FirstName.
6. The same way set ng-minlength & ng-maxlength directives to last name. Also, set ng-show directive to "studentForm.lastName.$touched && studentForm.lastName.$error.minlength" expression to <span> element adjacent to LastName input field.
7. Create another <span> for maxlength validation message.
8. Email will be validated automatically with input type=email. Also, create <span> for email validation message.

We have applied an expression "studentForm.firstName.$touched && studentForm.firstName.$error.required" to the <span>, in the above example. $touched & $error are built-in properties which return the state of the specified input controls and form. Let's learn about the state properties.

## **State Properties**

Angular includes properties which return the state of form and input controls. The state of the form and control changes based on the user's interaction and validation errors. These built-in properties can be accessed using form name or input control name. To check the form status use formName.propertyName, and to check the state of input control, use formName.inputFieldName.propertyName.

The following table lists the state properties.

| Property | Description |
| --- | --- |
| $error | $error object contains all the validation attributes applied to the specified element. |
| $pristine | Returns true if the user has not interacted with control yet else returns false. |
| $valid | Returns true if the model is valid |
| $invalid | Returns true if the model is invalid |
| $dirty | Returns true if user changed the value of model at least once |
| $touched | Returns true if the user has tabbed out from the control. |
| $untouched | Returns true if the user has not tabbed out from the control. |

The following example demonstrates the state properties.

Example: State Properties

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app>

<form name="studentForm" novalidate>

<p>

First Name Status: <br />

Pristine: {{studentForm.firstName.$pristine}} <br />

Touched: {{studentForm.firstName.$touched}}<br />

Untouched: {{studentForm.firstName.$untouched}}<br />

Valid: {{studentForm.firstName.$valid}} <br />

Invalid: {{studentForm.firstName.$invalid}} <br />

Dirty: {{studentForm.firstName.$dirty}} <br />

Error: {{studentForm.firstName.$error}} <br />

</p>

<label for="firstName">First Name: </label> <br />

<input type="text" name="firstName" ng-model="student.firstName" ng-required="true" />

<span ng-show="studentForm.firstName.$touched && studentForm.firstName.$error.required">First name is required.</span><br /><br />

<label for="lastName">Last Name</label><br />

<input type="text" name="lastName" ng-minlength="3" ng-maxlength="10" ng-model="student.lastName" /> <br />

<span ng-show="studentForm.lastName.$error.minlength">min 3 chars.</span>

<span ng-show="studentForm.lastName.$error.maxlength">Max 10 chars.</span> <br />

<input type="submit" value="Save" />

</form>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-132)

Learn about built-in validation css classes in AngularJS in the next section.

AngularJS Validation CSS Classes

AngularJS includes following CSS classes to allow styling of form and input controls based on the state of form field.

| CSS Class | Description |
| --- | --- |
| ng-valid | Angular will set this CSS class if the input field is valid without errors. |
| ng-invalid | Angular will set this CSS class if the input does not pass validations. |
| ng-pristine | Angular will set this CSS class if a user has not interacted with the control yet. |
| ng-dirty | Angular will set this CSS class if the value of form field has been changed. |
| ng-touched | Angular will set this CSS class if a user tabbed out from the input control. |
| ng-untouched | Angular will set this CSS class if a user has not tabbed out from the input control. |
| ng-submitted | Angular will set this CSS class if the form has been submitted. |

Note that you must provide implementation of these CSS classes and include in your CSS file. AngularJS automatically includes these classes based on the current state of input controls.

The following example demonstrates ng-pristine, ng-touched, ng-valid, and ng-invalid classes to display validity of each form control.

Example: AngularJS Validation CSS Classes

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

<style>

input.ng-pristine {

background-color:yellow;

}

input.ng-touched.ng-invalid {

background-color:red;

}

input.ng-touched.ng-valid {

background-color:green;

}

</style>

</head>

<body ng-app>

<form name="studentForm" novalidate class="student-form">

<label for="firstName">First Name: </label> <br />

<input type="text" name="firstName" ng-model="firstName" ng-required="true" />

<span ng-show="studentForm.firstName.$touched && studentForm.firstName.$error.required">First name is required.</span><br /><br />

<label for="lastName">Last Name</label><br />

<input type="text" name="lastName" ng-minlength="3" ng-maxlength="10" ng-model="lastName" />

<span ng-show="studentForm.lastName.$touched && studentForm.lastName.$error.minlength">min 3 chars.</span>

<span ng-show="studentForm.lastName.$touched && studentForm.lastName.$error.maxlength">Max 10 chars.</span><br /><br />

<label for="dob">Email</label><br />

<input type="email" id="email" ng-model="email" name="email" />

<span ng-show="studentForm.email.$touched && studentForm.email.$error.email">Please enter valid email id.</span><br /><br />

<input type="submit" value="Save" />

</form>

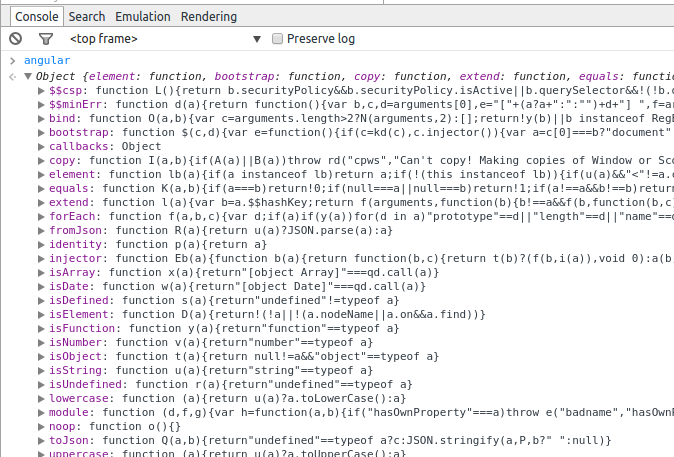
</body>

</html>

[How can I tell whether a web app was built using Angular (or other technologies)?](https://stackoverflow.com/questions/29950213/how-can-i-tell-whether-a-web-app-was-built-using-angular-or-other-technologies)

* Application declared using ng-app directive
* very simple controller and directive
* check for ng-model, ng-repeater attributes in the code. All these attibutes are written in small letters.
* Also you can check by typing in the console(ctrl + shift + i) and navigate to console tab. There type in window.angular.version --> it displays the version of the site your are currently inspecting.

=====&====

The best way to check is to write "angular" on browser console. If you get any object [With child objects as "bind","bootstrap","callbacks","module" etc.] then its an angular web app.

Exception Handling in AngularJS

Every application needs proper exception handling mechanism. You can use try, catch, and finally block of JavaScript to handle exceptions in AngularJS modules.

tip$exceptionHandler does not handle syntax errors.

AngularJS also includes built-in $exceptionHandler service, which handles uncaught exceptions in the application.

The default implementation of $exceptionHandler service logs the exception into the browser console. You can override this service as per your requirement.

The following example demonstrates uncaught exception handling using $exceptionHandler service.

Example: $exceptionHandler

<!DOCTYPE html>

<html ng-app="studentApp">

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body class="container" ng-controller="studentController">

Status: {{status}} <br />

Data: {{data}} <br />

<input type="button" value="Get Data" ng-click="getStudent()" />

<script>

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

app.config(function ($provide) {

$provide.decorator('$exceptionHandler', function ($delegate) {

return function (exception, cause) {

$delegate(exception, cause);

alert('Error occurred! Please contact admin.');

};

});

});

app.controller("studentController", function ($scope) {

var onSuccess = function (response) {

$scope.status = response.status;

$scope.data = response.data;

};

var onError = function (response) {

$scope.status = response.status;

$scope.data = response.data;

}

$scope.getStudent = function () {

$http.get("/getdata").then(onSuccess, onError);

};

});

</script>

</body>

</html>

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=ng-161)

In the above example, we override the $provide service's default behavior using $provide.decorate() method in the app.config() method. The decorate method allow us to override or modify the behavior of the service. So, in the decorate method, we display custom error messages along with logging exception messages to the browser console.

Note that we have used $http service in the studentController. However, we have not included $http service as a parameter in the controller function to raise an exception for demo purpose. Now, the exception will be handled by $exceptionHandler and displays an alert message.