SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

CUSTOMER INTERFACE DESIGN AND DEVELOPMENT -SITA1502

Vinodhini.M

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

CUSTOMER INTERFACE DESIGN AND DEVELOPMENT -SITA1502

Vinodhini.M

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT 4 ANGULAR JS AND JQUERY

Angular JS Expression – Modules – Directives – Data Binding –Controllers – Scopes – Filters – Services – Tables – Events– Form – Validation. jQuery Syntax – Selects – Events – jQuery Effects – jQuery – jQuery HTML – jQuery Traversing.

What is AngularJS

MVC Javascript Framework by Google for Rich Web Application Development

Why AngularJS

Other frameworks deal with HTML's shortcomings by either abstracting away HTML, CSS, and/or JavaScript or by providing an imperative way for manipulating the DOM.

- Structure, Quality and Organization
- Lightweight (< 36KB compressed and minified)
- Free
- Separation of concern
- Modularity
- Extensibility & Maintainability
- Reusable Components

"

jQuery

- Allows for DOM Manipulation
- Does not provide structure to your code
- Does not allow for two-way binding

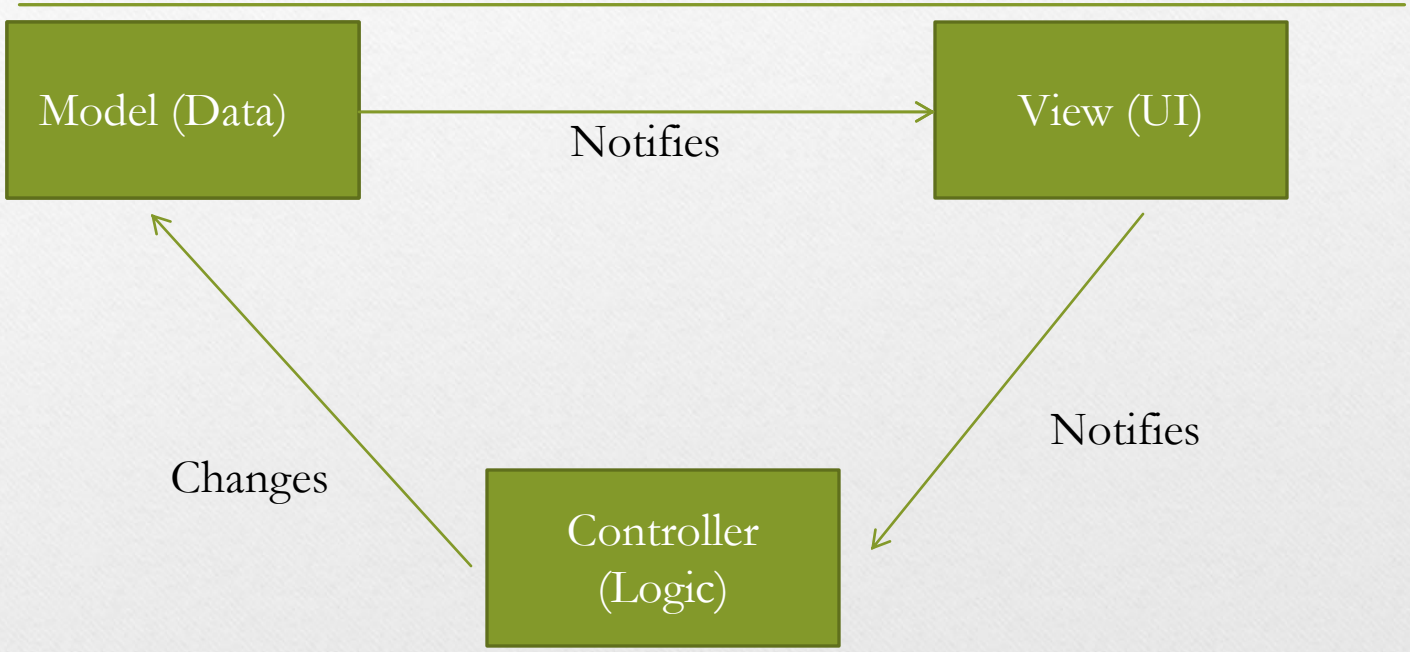
Features of AngularJS

- Two-way Data Binding – Model as single source of truth
- Directives – Extend HTML
- MVC
- Dependency Injection
- Testing
- Deep Linking (Map URL to route Definition)
- Server-Side Communication

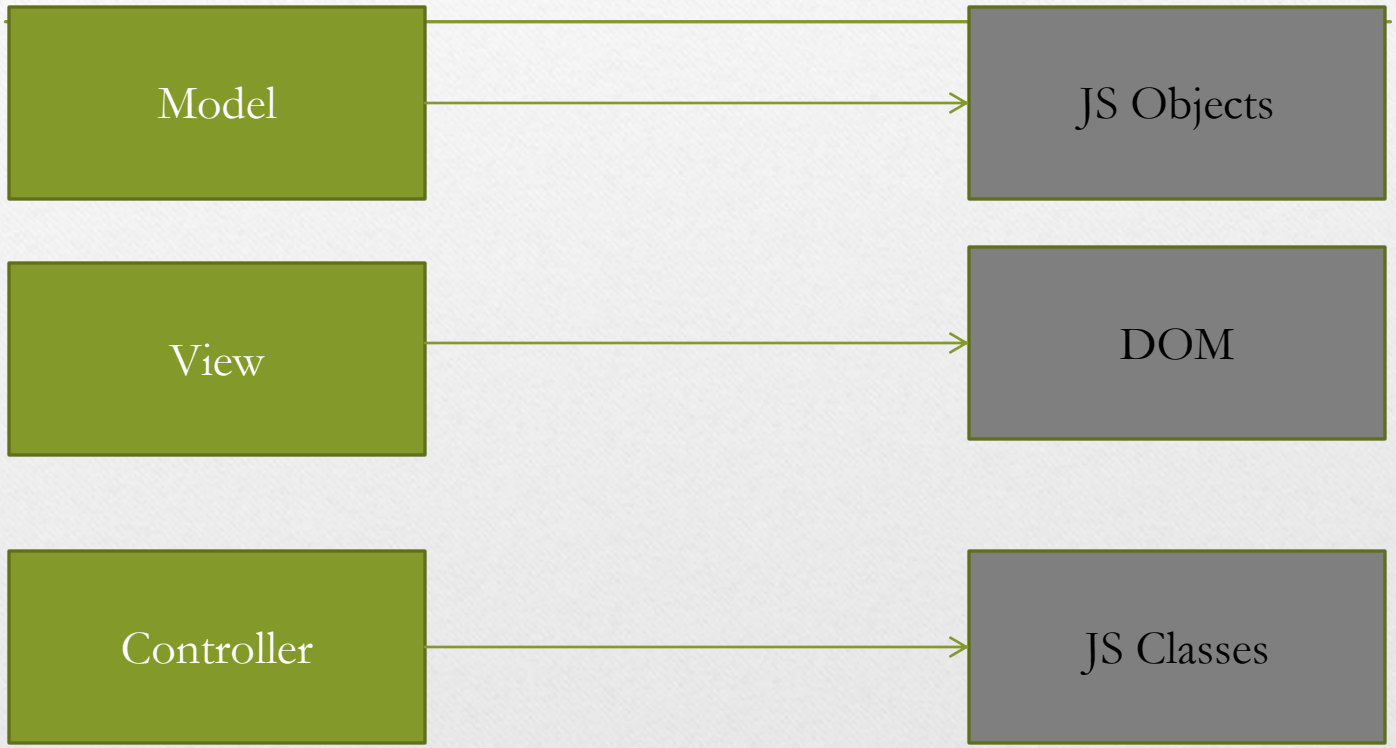
Data Binding

```
<html ng-app>
<head>
  <script src='angular.js'></script>
</head>
<body>
  <input ng-model='user.name'>
  <div ng-show='user.name'>Hi {{user.name}}</div>
</body>
</html>
```

MVC


Model (Data)
Notifies
View (UI)
Changes
Notifies
Controller
(Logic)

MVC



MVC

```
<html ng-app>
<head>
  <script src='angular.js'></script>
  <script src='controllers.js'></script>
</head>
<body ng-controller='UserController'>
  <div>Hi {{user.name}}</div>
</body>
</html>


function  XXXX($scope) {
  $scope.user = { name:'Larry' };
}
```

Hello Javascript

```
<p id="greeting1"></p>

<script>
var isIE = document.attachEvent;
var addListener = isIE
  ? function(e, t, fn) {
      e.attachEvent('on' + t, fn);}
  : function(e, t, fn) {
      e.addEventListener(t, fn, false);};
addListener(document, 'load', function(){
  var greeting = document.getElementById('greeting1');
  if (isIE) {
    greeting.innerText = 'Hello World!';
  } else {
    greeting.textContent = 'Hello World!';
 }
});
</script>
```

Hello JQuery

```
<p id="greeting2"></p>
```

```
<script>
$(function(){
  $('#greeting2').text('Hello World!');
});
</script>
```

Example

Hello AngularJS

```
<p ng:init="greeting = 'Hello
   World!'">{{greeting}}</p>
```

Sample Angular Powered View

```
<body ng-app="F1FeederApp" ng-controller="driversController">
 <table>
  <thead>
   <tr><th colspan="4">Drivers Championship Standings</th></tr>
  </thead>
  <tbody>
   <tr ng-repeat="driver in driversList">
    <td>{{$index + 1}}</td>
    <td>
     <img src="img/flags/{{driver.Driver.nationality}}.png" />
     {{driver.Driver.givenName}} {{driver.Driver.familyName}}
    </td>
    <td>{{driver.Constructors[0].name}}</td>
    <td>{{driver.points}}</td>
   </tr>
  </tbody>
 </table>
</body>
```

Expressions

Expressions allow you to execute some computation in order to return a desired value.

- {{ 1 + 1 }}
- {{ 946757880 | date }}
- {{ user.name }}

you shouldn't use expressions to implement any higher-level logic.

Directives

Directives are markers (such as attributes, tags, and class names) that tell AngularJS to attach a given behaviour to a DOM element (or transform it, replace it, etc.)

Some angular directives

- The ng-app - Bootstrapping your app and defining its scope.
- The ng-controller - defines which controller will be in charge of your view.
- The ng-repeat - Allows for looping through collections

What is module ?

AngularJS module defines an application.

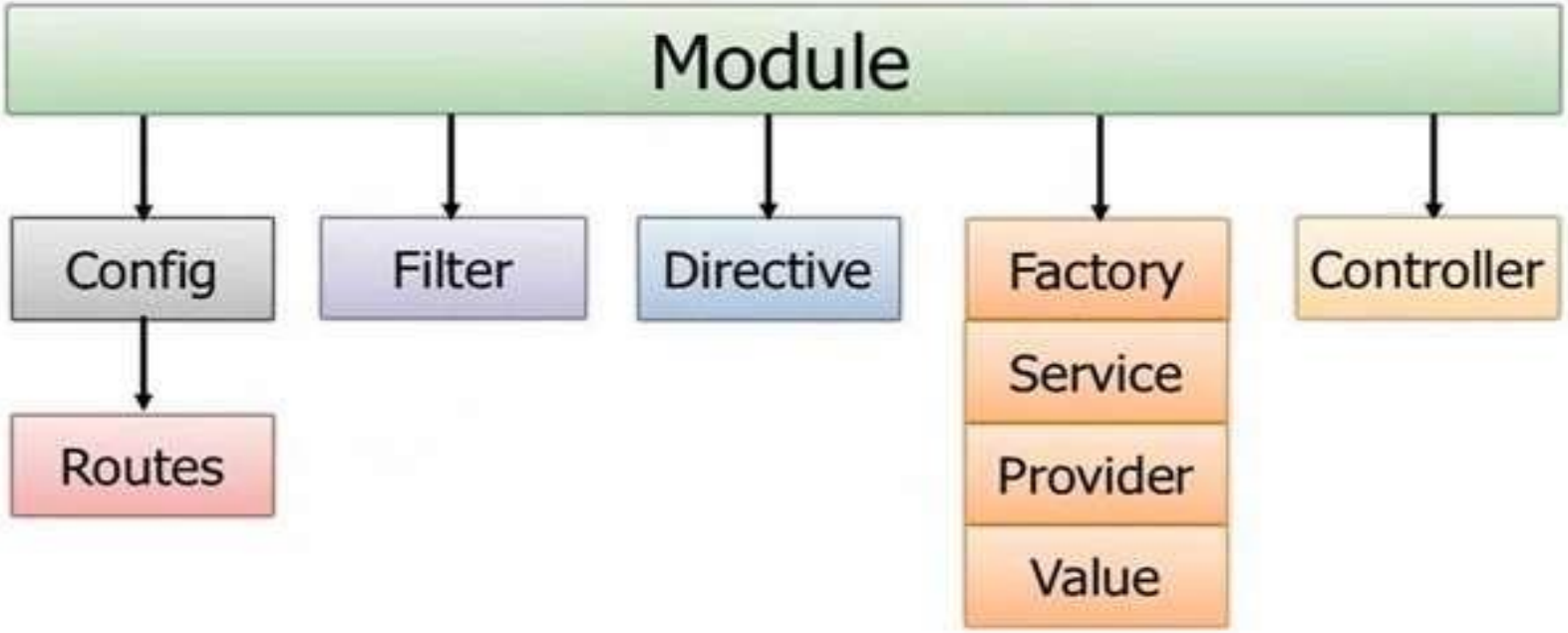
- The module is container for different parts of an application.I.e controllers,services,directives,filters.

- Controllers always belong to a module.

- Module is used as a main() method.angular is an object and module is a method. Module is which present in angular object.

Modules are Containers

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



Why module

- Module is starting point of angularjs application.
- ng-app that have empty module make its controller function global.

- Controller function will become global if we don’t register it with the module then there may be chances of overriding data. To avoid this we have register controller function with the module.
- So that can solve global value problem.

How to create a module

Syntax:

```
Var app = angular.module(“myModule”, []);
```

How to add controller to Module

```
var myApp=angular.module("myModule",[])
myApp.controller("myController",function($scope){//Register
the controller with the module.

});
```

kinds of directives in Angular

- There are **three** kinds of directives in Angular:

  - **Components—** directives with a template.

  - **Structural directives—** change the DOM layout by adding and removing DOM elements.

  - **Attribute directives—** change the appearance or behavior of an element, component, or another directive.

Directives

- Structural Directives change the structure of the view. Two examples are **NgFor and NgIf.**
- Structural **directives are easy to recognize. An asterisk (*) precedes the directive attribute name as in this.**
- Attribute directives are used as **attributes of elements**. The built-in **NgStyle** directive in the Template
- Three of the common, built-in structural directives—**NgIf, NgFor, and NgSwitch.**

NgIf

**<div *ngIf="hero" class="name">{{hero.name}} **

- No brackets. No parentheses. Just ***ngIf** set to a string.
- **The asterisk (*) is a convenience** notation and the string is a micro syntax rather than the usual template expression.

  Render to

  <ng-template **[ngIf]="hero"**>
  {{hero.name}}
  </ng-template>

- Angular desugars this notation into a marked-up **<ng-template>** that surrounds the host element and its descendents. Each structural directive does something different with that template.

https://www.ifourtechnolab.com/

NgFor

<li *ngFor="**let hero of heroes**">{{hero.name}}

Ref -> https://blog.angular-university.io/angular-2-ngfor/

- Angular transforms the ***ngFor in similar fashion from asterisk (*)** syntax to <ng-template> element.

<div *ngFor="**let hero of heroes; let i=index; let odd=odd; trackBy: trackById**" [class.odd]="odd">
({{i}}) {{hero.name}}

<ng-template ngFor let-hero **[ngForOf]**="heroes" let-i="index" let-odd="odd" **[ngForTrackBy]="trackById"**>
<div [class.odd]="odd">({{i}}) {{hero.name}}
</ng-template>

Angular transforms the *ngFor in similar fashion from asterisk (*) syntax to <ng-template> element.

NgSwitch

- The Angular **NgSwitch** is actually a **set of cooperating directives**:

a. NgSwitch,

b. NgSwitchCase

a. NgSwitchDefault

NgSwitch Example

```
<ul [ngSwitch]="person.country">
 <li *ngSwitchCase="'UK'"   class="text-success">
        {{ person.name }} ({{ person.country }})
 </li>
 <li *ngSwitchCase="'USA'" class="text-primary">
        {{ person.name }} ({{ person.country }})
 </li>
 <li *ngSwitchDefault class="text-warning">
        {{ person.name }} ({{ person.country }})
 </li>
</ul>
```

NgClass

- **Adds and removes CSS classes on an HTML element.**
- The CSS classes are updated as follows, depending on the type of the expression evaluation:

a. **string** - the CSS classes listed in the string (space delimited) are added,
b. **Array** - the CSS classes declared as Array elements are added,
c. **Object** - keys are CSS classes that get added when the expression given in the value evaluates to a truthy value, otherwise they are removed.

```
<some-element [ngClass]="'first second'">...</some-element>
<some-element [ngClass]="['first', 'second']">...</some-element>
<some-element [ngClass]="{'first': true, 'second': true, 'third': false}">...</some-element>
<some-element [ngClass]="string: Exp|array: Exp|obj: Exp">...</some-element>
<some-element [ngClass]="{'class1 class2 class3' : true}">...</some-element>
```

NgStyle

- **A**n attribute directive that updates styles for the containing HTML element.
- Sets one or more style properties, specified as colon-separated key-value pairs.
- The key is a style name, with an optional .
- <unit> suffix (such as 'top.px', 'font-style.em').
- The value is an expression to be evaluated.
- The resulting non-null value, expressed in the given unit, is assigned to the given style property.
- If the result of evaluation is null, the corresponding style is removed.

```
<some-element [ngStyle]="{'font-style': styleExp}">...</some-element>
<some-element [ngStyle]="{'max-width.px': widthExp}">...</some-element>
<some-element [ngStyle]="objExp">...</some-element>
```

Data Binding in Angular

Types of Data Binding in Angular

- 1. **String Interpolation**:
  - The type of one-way data binding where text is between a set of curly braces often uses the name of a component property. Angular replaces that name with the string value of the corresponding component property. The syntax of string interpolation is to use double curly braces.

{{ code }}

2: **Property Binding**:

- Property Binding allows us to bind the view of the template expression. Property binding in simple term is defined as updating the value of a certain variable in component (model) and displaying it in view (presentation layer).
- This is a one-way mechanism, thus it allows you to change the value whenever you want but only at the component level.
- binding.component.html

```
<h1>PropertyBinding</h1><img [src]="imagePath" class="image-adjustment"/><br>
```

Sample Program

- binding.component.ts
- import { Component, OnInit } from '@angular/core';@Component({ selector: 'app-binding', templateUrl: './binding.component.html', styleUrls: ['./binding.component.css']})export class BindingComponent implements OnInit { constructor() { } imagePath: string = 'assets/images/Databinding.png'; ngOnInit() {} }}
- Though, both doing the same thing, so what is the difference between Property Binding and Interpolation?

3: **Event Binding**:

- Event binding is defined as the updating/sending of the value/information of a certain variable from the presentation layer (view) to the component (model). For example, clicking a button.

binding.component.html

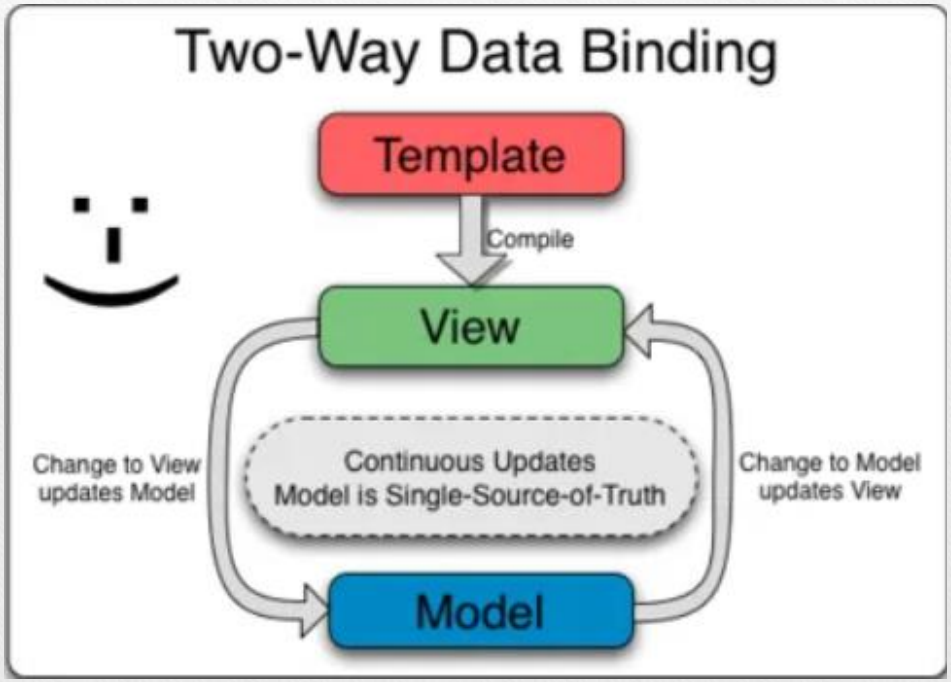
<h1 Event Binding></h1><h1>{{title}}</h1><button (click)="changeMyTitle()">Title is changed on click of this button.</button>

Sample Program

- binding.component.ts
- export class BindingComponent implements OnInit { constructor() { } title = 'Learning string interpolation'; ngOnInit() { }changeMyTitle() { this.title = 'Learning Event Binding';}}
- **Output:**

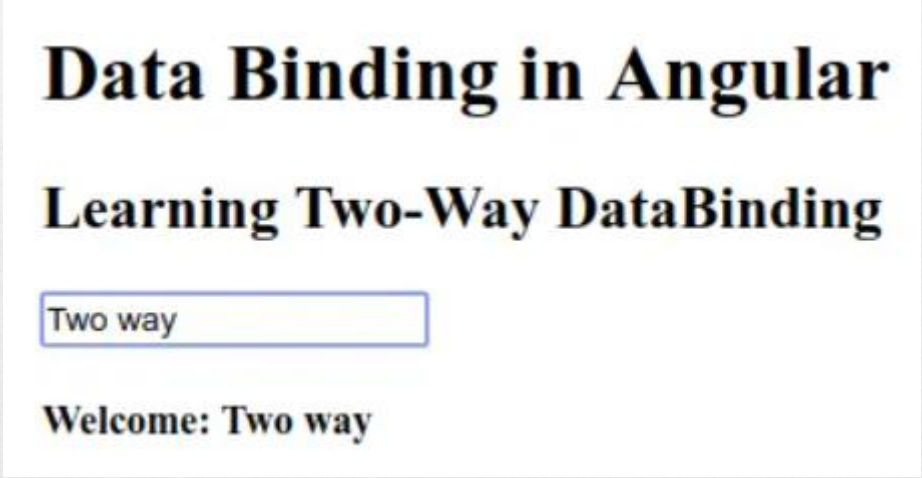
4: **Two-Way Data Binding**:

- Two-way data binding is a combination of both Property and Event binding and it is a continuous synchronization of a data from view to the component and component to the view.



Sample Program

- binding.component.html
- <h1>Data Binding in Angular</h1><h2>Learning Two-Way DataBinding</h2><input type = "text" [(ngModel)]="userName"/>
<h4>Welcome: {{userName}}</h4>
- **Output**



Controllers

- AngularJS application mainly relies on controllers to control the flow of data in the application.
- A controller is defined using *ng-controller* directive.
- A controller is a JavaScript object that contains attributes/properties, and functions.
- Each controller accepts $scope as a parameter, which refers to the application/module that the controller needs to handle.

Example

-
  ...

Here, we declare a controller named *studentController*, using the ng-controller directive

Scope in a Directive

- This scope object is used for accessing the variables and functions defined in the AngularJS controllers
- The controller and link functions of the directive. By default, directives do not create their own scope; instead they use the scope of their parent, generally a controller (within the scope of which the directive is defined).

Scope in a Directive

- We can change the default scope of the directive using the scope field of the DDO (Data Definition Object).
- Note that scope is a property of the directive, just like restrict, template, etc. The value of the scope field defines how the scope of the directive will be created and used. These values are ‘true’, ‘false,’ and ‘{}’.

scope: false

- This is the default value of scope in the directive definition. In this case, the directive has the same scope as its parent controller.

scope: false

- **Creating an Angular app and controller**

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

movieApp.controller("movieController",function($scope){
    $scope.movie = "Ice Age";
});
```

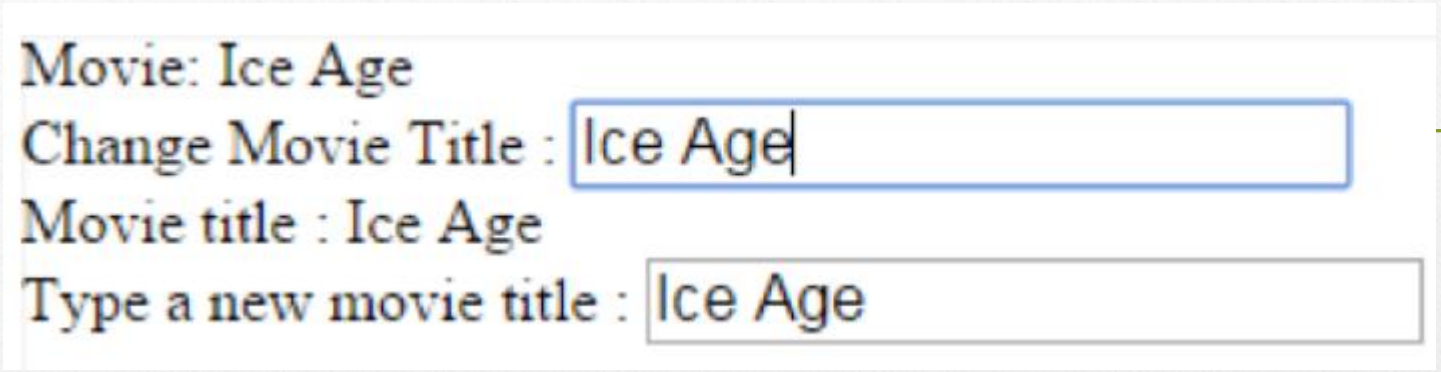
scope: false

- **Embedding the directive in HTML**

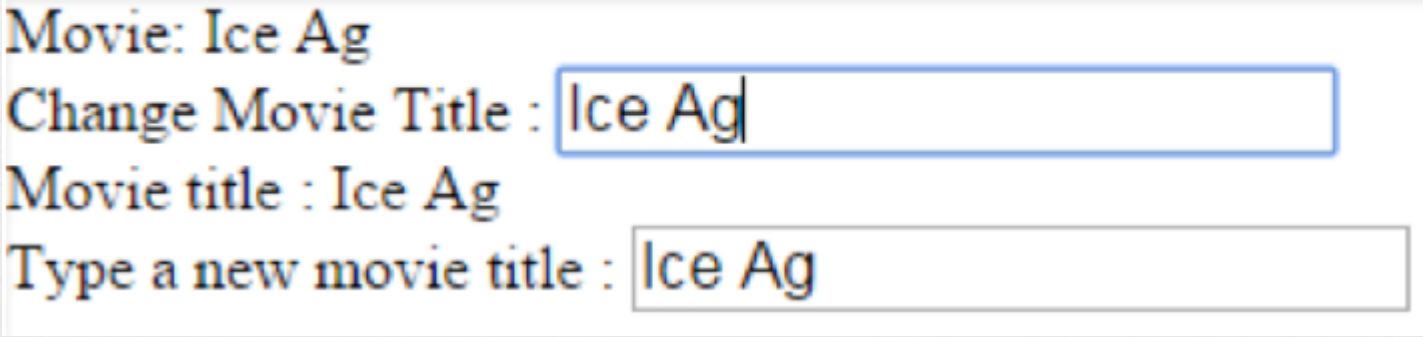
```
<div ng-app="movieApp">
    <div ng-controller="movieController">
        <h2>Movie: {{movie}}</h2>
  Change Movie Title : <input type='text' ng-model='movie'/>
        <movie-directive></movie-directive>
    </div>
</div>
```

scope: false

Output



On changing the `Movie Title` in the text box above:

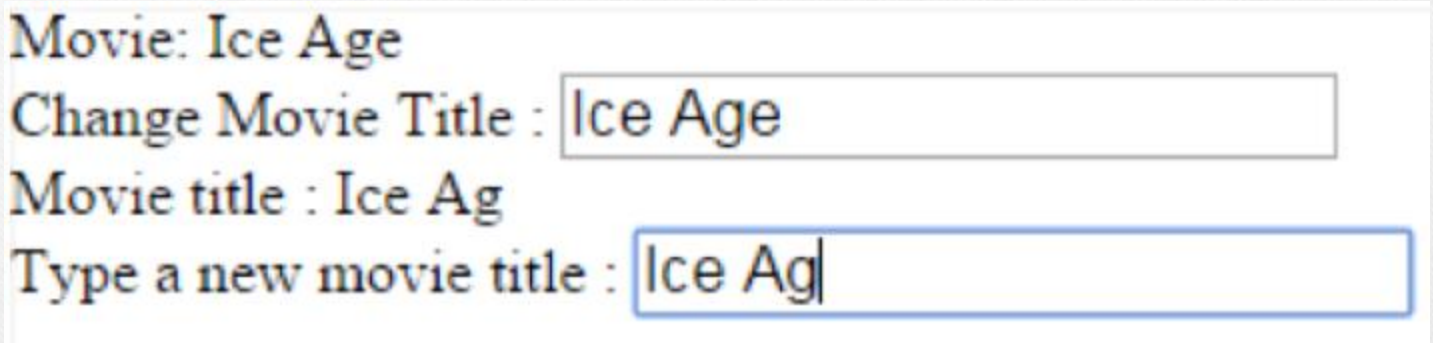


scope: true

- In this case, the movie title in both the header section and the directive is initialized to the same value defined in the controller's scope, *Ice Age*.

```
<div ng-app="movieApp">
    <div ng-controller="movieController">
        <h2>Movie: {{movie}}</h2>
          Change Movie Title : <input type='text' ng-model='movie'/>
        <movie-directive></movie-directive>
    </div>
</div>
```

Output

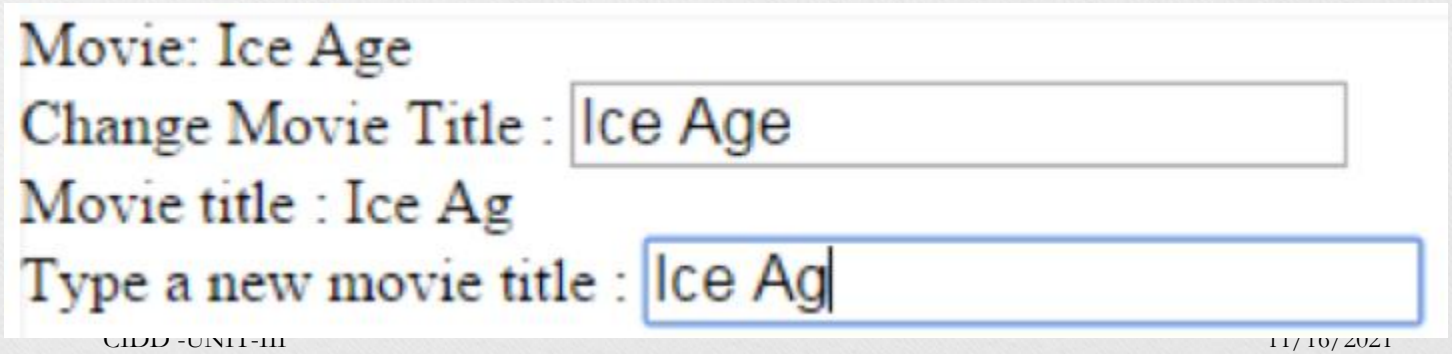

Movie: Ice Age
Change Movie Title : Ice Age
Movie title : Ice Ag
Type a new movie title : Ice Ag

scope: true

Let's change the scope field of the directive in the last example to true

- scope: true

```
<div ng-app="movieApp">
    <div ng-controller="movieController">
        <h2>Movie: {{movie}}</h2>
          Change Movie Title : <input type='text' ng-model='movie'/>
        <movie-directive></movie-directive>
    </div>
</div>
```



scope: {}

This is how the directive gets its isolated scope. We pass an object for the scope field in this case. This way, the scope of the directive is not inherited from the parent and is instead completely detached from it. Thus, the directive has an isolated scope.

```
movieApp.directive("movieDirective", function(){
    return {
        restrict: "E",
        scope: {},
        template: "<div>Movie title : {{movie}}</div>"+
        "Type a new movie title : <input type='text' ng-model='movie' />"
    };
});
```

Isolated scope - Prefixes

The directive scope uses prefixes to achieve that. Using prefixes helps establish a two-way or one-way binding between parent and directive scopes, and also make calls to parent scope methods. To access any data in the parent scope requires passing the data at two places – the directive scope and the directive tag.

1. Passing data to the directive scope

```
movieApp.directive("movieDirective", function(){
    return {
        restrict: "E",
        scope: {
                movie: '='
        },
        template: "<div>Movie title : {{movie}}</div>"+
        "Type a new movie title : <input type='text' ng-model='movie' />"
    };
});
```

2. Passing data in the directive tag

```
<movie-directive movie="movie"></movie-directive>
```

There are 3 types of prefixes in AngularJS:

1. '@' - Text binding / one-way binding
2. '=' - Direct model binding / two-way binding
3. '&' - Behavior binding / Method binding

Filters

- Filters are used to modify the data. They can be clubbed in expression or directives using pipe (|) character. The following list shows the commonly used filters.

Sr.No.	Name & Description
1	**uppercase** converts a text to upper case text.
2	**lowercase** converts a text to lower case text.
3	**currency** formats text in a currency format.
4	**filter** filter the array to a subset of it based on provided criteria.
5	**orderby** orders the array based on provided criteria.

2. Uppercase Filter

Add uppercase filter to an expression using pipe character. Here we've added uppercase filter to print student name in all capital letters.

```
Enter first name:<input type = "text" ng-model = "student.firstName">
Enter last name: <input type = "text" ng-model = "student.lastName">
Name in Upper Case: {{student.fullName() | uppercase}}
```

3. Lowercase Filter

- Add lowercase filter to an expression using pipe character. Here we've added lowercase filter to print student name in all lowercase letters.

```
Enter first name:<input type = "text" ng-model = "student.firstName">
Enter last name: <input type = "text" ng-model = "student.lastName">
Name in Lower Case: {{student.fullName() | lowercase}}
```

4. Currency Filter

- Add currency filter to an expression returning number using pipe character. Here we've added currency filter to print fees using currency format.

```
Enter fees: <input type = "text" ng-model = "student.fees">
fees: {{student.fees | currency}}
```

4. Filter

To display only required subjects, we use subjectName as filter.

```
Enter subject: <input type = "text" ng-model = "subjectName">
Subject:
<ul>
   <li ng-repeat = "subject in student.subjects | filter: subjectName">
      {{ subject.name + ', marks:' + subject.marks }}
   </li>
</ul>
```

5. OrderBy Filter

To order subjects by marks, we use orderBy marks.

```
Subject:
<ul>
   <li ng-repeat = "subject in student.subjects | orderBy:'marks'">
      {{ subject.name + ', marks:' + subject.marks }}
   </li>
</ul>
```

Service

In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.

- AngularJS has about 30 built-in services.
- Syntax

```
$location
```

Example

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

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

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

The $timeout Service

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

TABLE

- Table data is generally repeatable.
- if we want to show data in table formats it's better to use ng-repeat directive because the table will contain data in repeated format.
- In angularjs the ng-repeat directive will loop through the array data objects in the DOM elements and help us to show data in tables easily.

Example Table

```
<table>
<tr>
 <th>Name</th>
 <th>Marks</th>
</tr> <tr ng-repeat = "subject in student.subjects">
<td>{{ subject.name }}</td>
<td>{{ subject.marks }}</td>
</tr>
 </table>
```

Table can be styled using CSS Styling.

```
<style>
table, th , td { border: 1px solid grey; border-collapse: collapse; padding: 5px; }
table tr:nth-child(odd) { background-color: #f2f2f2; }
 table tr:nth-child(even) { background-color: #ffffff; }
</style>
```

Display with orderBy Filter

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

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

Display with uppercase Filter

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

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

Display the Table Index ($index)

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

```
<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>
```

EVENTS

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

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

- ng-blur
- ng-change
- ng-click
- ng-copy
- ng-cut
- ng-dblclick

- ng-keyup
- ng-mousedown
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- ng-mouseover
- ng-mouseup
- ng-paste
- ng-focus
- ng-keydown
- ng-keypress

Mouse Events

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

- ng-mouseover
- ng-mouseenter
- ng-mousemove
- ng-mouseleave

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

- ng-mousedown
- ng-mouseup
- ng-click
- You can add mouse events on any HTML element.

Example

Increase the count variable when the mouse moves over the H1 element:

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

<h1 ng-mousemove="count = count + 1">Mouse over me!</h1>

<h2>{{ count }}</h2>

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

Forms

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

Input Controls

Input controls are the HTML input elements:

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

Data-Binding

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

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

The application does now have a property named firstname.

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

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

Example

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

It can also be referred to elsewhere in the application:

Example

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

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

Checkbox

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

Example

Show the header if the checkbox is checked:

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

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

Radiobuttons

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

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

Example

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

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

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

Selectbox

- Bind select boxes to your application with the ng-model directive.
- The property defined in the ng-model attribute will have the value of the selected option in the selectbox.

Example

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

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

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

Form Validation

- AngularJS offers client-side form validation.
- AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.
- AngularJS also holds information about whether they have been touched, or modified, or not.
- You can use standard HTML5 attributes to validate input, or you can make your own validation functions.
- Client-side validation cannot alone secure user input. Server side validation is also necessary.

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

The input field is required:

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

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

E-mail

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

Example

The input field has to be an e-mail:

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

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

Form State and Input State

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

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

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

Forms have the following states:

- $pristine No fields have been modified yet
- $dirty One or more have been modified
- $invalid The form content is not valid
- $valid The form content is valid
- $submitted The form is submitted

They are all properties of the form, and are either true or false.

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

Example

Show an error message if the field has been touched AND is empty:

```
<input name="myName" ng-model="myName" required>
<span ng-show="myForm.myName.$touched &&
myForm.myName.$invalid">The name is required.</span>
```

CSS Classes

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

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

- ng-untouched The field has not been touched yet
- ng-touched The field has been touched
- ng-pristine The field has not been modified yet
- ng-dirty The field has been modified

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

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

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

The classes are removed if the value they represent is false.

Add styles for these classes to give your application a better and more intuitive user interface.

Example

Apply styles, using standard CSS:

```
<style>

input.ng-invalid {
    background-color: pink;
  }
  input.ng-valid {
    background-color: lightgreen;
  }

</style>
```

Forms can also be styled:

Example

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

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

Custom Validation

- To create your own validation function is a bit more tricky; You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

Example

- Create your own directive, containing a custom validation function, and refer to it by using my-directive.
- The field will only be valid if the value contains the character "e":

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

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

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

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

```
    </form>
    <script>
    var app = angular.module('myApp', []);
```

```
app.directive('myDirective', function() {
    return {
      require: 'ngModel',
      link: function(scope, element, attr, mCtrl) {
        function myValidation(value) {
```

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

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

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

```
if (value.indexOf("e") > -1) {
            mCtrl.$setValidity('charE', true);
          } else {
            mCtrl.$setValidity('charE', false);
          }
          return value;
        }
        mCtrl.$parsers.push(myValidation);
      }
    };
  });
  </script>
```

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

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

Select

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

Creating a Select Box Using ng-options

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

Example

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

    <select ng-model="selectedName" ng-options="x for x in names">
    </select>

    </div>
```

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

ng-options vs ng-repeat

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

Example

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

Because the ng-repeat directive repeats a block of HTML code for each item in an array, it can be used to create options in a dropdown list, but the ng-options directive was made especially for filling a dropdown list with options.

You can use both the ng-repeat directive and the ng-options directive:

Assume you have an array of objects:

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

Example

Using ng-repeat:

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

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

When using the value as an object, use ng-value insead of value:

Example

Using ng-repeat as an object:

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

  <h1>You selected
  a {{selectedCar.color}} {{selectedCar.model}}</h1>
```

Example

Using ng-options:

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

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

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

The Data Source as an Object

- In the previous examples the data source was an array, but we can also use an object.
- Assume you have an object with key-value pairs:

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

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

Example

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

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

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

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

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

Example

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

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

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

Example

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

jQuery Effects

Hide, Show, Toggle, Slide, Fade, and Animate.

jQuery hide() and show()

- jQuery hide()
  Demonstrates a simple jQuery hide() method.
- jQuery hide()
  Another hide() demonstration. How to hide parts of text.

With jQuery, you can hide and show HTML elements with the hide() and show() methods:

Example

```
$("#hide").click(function(){
  $("p").hide();
});

$("#show").click(function(){
  $("p").show();
});
```

Syntax:

$(*selector*).hide(*speed,callback*);

$(*selector*).show(*speed,callback*);

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the hide() or show() method completes (you will learn more about callback functions in a later chapter).

The following example demonstrates the speed parameter with hide():

Example

```
$("button").click(function(){
    $("p").hide(1000);
  });
```

juery Effects - Fading

- jQuery fadeIn()
  Demonstrates the jQuery fadeIn() method.
- jQuery fadeOut()
  Demonstrates the jQuery fadeOut() method.
- jQuery fadeToggle()
  Demonstrates the jQuery fadeToggle() method.
- jQuery fadeTo()
  Demonstrates the jQuery fadeTo() method.

jQuery Fading Methods

With jQuery you can fade an element in and out of visibility.

jQuery has the following fade methods:

- fadeIn()
- fadeOut()
- fadeToggle()
- fadeTo()

jQuery Effects - Sliding

- The jQuery slide methods slide elements up and down.

Examples

- jQuery slideDown()
  Demonstrates the jQuery slideDown() method.
- jQuery slideUp()
  Demonstrates the jQuery slideUp() method.
- jQuery slideToggle()
  Demonstrates the jQuery slideToggle() method.

jQuery Effects - Animation

- With jQuery, you can create custom animations.

The jQuery animate() method is used to create custom animations.

Syntax:

$(*selector*).animate({*params*},*speed*,*callback*);

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.
The following example demonstrates a simple use of the animate() method; it moves a element to the right, until it has reached a left property of 250px:

jQuery Stop Animations

The jQuery stop() method is used to stop animations or effects before it is finished.

- jQuery stop() sliding
  Demonstrates the jQuery stop() method.
- jQuery stop() animation (with parameters)
  Demonstrates the jQuery stop() method.

jQuery Callback Functions

- A callback function is executed after the current effect is 100% finished.
- JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.
- To prevent this, you can create a callback function.
- A callback function is executed after the current effect is finished.
- syntax: **$(*selector*).hide(*speed,callback*);**

jQuery - Chaining

- jQuery, you can chain together actions/methods.
- Chaining allows us to run multiple jQuery methods (on the same element) within a single statement.

jQuery HTML

jQuery - Get Content and Attributes

- jQuery contains powerful methods for changing and manipulating HTML elements and attributes.

DOM = Document Object Model

The DOM defines a standard for accessing HTML and XML documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

Get Content - text(), html(), and val()

Three simple, but useful, jQuery methods for DOM manipulation are:

- text() - Sets or returns the text content of selected elements
- html() - Sets or returns the content of selected elements (including HTML markup)
- val() - Sets or returns the value of form fields

The following example demonstrates how to get content with the jQuery text() and html() methods:

Get Attributes - attr()

The jQuery attr() method is used to get attribute values

Set Content - text(), html(), and val()

text() - Sets or returns the text content of selected elements

html() - Sets or returns the content of selected elements (including HTML markup)

val() - Sets or returns the value of form fields

jQuery - Add Elements

Add New HTML Content

We will look at four jQuery methods that are used to add new content:

- append() - Inserts content at the end of the selected elements
- prepend() - Inserts content at the beginning of the selected elements
- after() - Inserts content after the selected elements
- before() - Inserts content before the selected elements

jQuery - Remove Elements

Remove Elements/Content

To remove elements and content, there are mainly two jQuery methods:

- remove() - Removes the selected element (and its child elements)
- empty() - Removes the child elements from the selected element

jQuery remove() Method

The jQuery remove() method removes the selected element(s) and its child elements.

jQuery empty() Method

The jQuery empty() method removes the child elements of the selected element(s).

jQuery - Get and Set CSS Classes

jQuery Manipulating CSS

jQuery has several methods for CSS manipulation. We will look at the following methods:

- addClass() - Adds one or more classes to the selected elements
- removeClass() - Removes one or more classes from the selected elements
- toggleClass() - Toggles between adding/removing classes from the selected elements
- css() - Sets or returns the style attribute

jQuery Traversing

- jQuery traversing, which means "move through", are used to "find" (or select) HTML elements based on their relation to other elements. Start with one selection and move through that selection until you reach the elements you desire.
- The image below illustrates an HTML page as a tree (DOM tree). With jQuery traversing, you can easily move up (ancestors), down (descendants) and sideways (siblings) in the tree, starting from the selected (current) element. This movement is called traversing - or moving through - the DOM tree.
-

- llustration explained:
- The element is the **parent** of <ul>, and an **ancestor** of everything inside of it
- The <ul> element is the **parent** of both <li> elements, and a **child** of
- The left <li> element is the **parent** of , **child** of <ul> and a **descendant** of
- The element is a **child** of the left <li> and a **descendant** of <ul> and
- The two <li> elements are **siblings** (they share the same parent)
- The right <li> element is the **parent** of <b>, **child** of <ul> and a **descendant** of
- The <b> element is a **child** of the right <li> and a **descendant** of <ul> and

- An ancestor is a parent, grandparent, great-grandparent, and so on.
  A descendant is a child, grandchild, great-grandchild, and so on.
  Siblings share the same parent.

Traversing the DOM

- jQuery provides a variety of methods that allow us to traverse the DOM.
- The largest category of traversal methods are tree-traversal.
- The next chapters will show us how to travel up, down and sideways in the DOM tree.

jQuery Traversing - Ancestors

Traversing Up the DOM Tree

Three useful jQuery methods for traversing up the DOM tree are:

- parent()
- parents()
- parentsUntil()

jQuery Traversing - Descendants

- With jQuery you can traverse down the DOM tree to find descendants of an element.
- A descendant is a child, grandchild, great-grandchild, and so on.

Traversing Down the DOM Tree

Two useful jQuery methods for traversing down the DOM tree are:

Children()

Find()

jQuery Traversing - Siblings

- With jQuery you can traverse sideways in the DOM tree to find siblings of an element.
- Siblings share the same parent.

Traversing Sideways in The DOM Tree

- siblings()
- next()
- nextAll()
- nextUntil()
- prev()
- prevAll()
- prevUntil()

jQuery Traversing - Filtering

- The first(), last(), eq(), filter() and not() Methods
- The most basic filtering methods are first(), last() and eq(), which allow you to select a specific element based on its position in a group of elements.
- Other filtering methods, like filter() and not() allow you to select elements that match, or do not match, a certain criteria.