**AngularJS Interview Questions & Answers.**

**Q-1. What Is AngularJS?**

**Answer.**

* It has been developed by one of the biggest technology giants Google. It is a JavaScript framework that helps you to create dynamic Web applications.
* It supports to use HTML as the template language and enables the developer to create extended HTML tags that help to represent the application’s components clearly. These tags make the code efficient by reducing the lines of code that a developer may need to write when using JavaScript.
* It is open-source and is licensed under the Apache License version 2.0.
* It helps to develop an easy to maintain architecture that can be tested at client side code.

**Q-2. What Are The Key Features Of AngularJS?**

**Answer.**

Here is the list of AngularJS features that makes it the hottest tech for web dev.

* **Data-binding –** Handles synchronization of data across model, controllers, and view.
* **Scope –** Object representing the model, acts as a glue layer between controller and view.
* **Controllers –** JS functions bound to the scope object.
* **Services –** Substitutable objects that are wired together using dependency injection. e.g. $location service.
* **Filters –** Formats the value of an expression for displaying to the user. e.g. uppercase, lowercase.
* **Directives –** These are extended HTML attributes start with the “ng-” prefix. e.g. ng-app directive used to initialize the angular app.
* **Templates –** HTML code including AngularJS specific elements and attributes.
* **Routing –** It’s an approach to switch views.
* **MVC pattern –** A design pattern made up of three parts.
  + **Model –** Represents data, could be static data from a JSON file or dynamic data from a database.
  + **View –** Renders data for the user.
  + **Controller –** Gives control over the model and view for collating information to the user.
* **Deep linking –** Enables the encoding of the application state in the URL and vice versa.
* **Dependency injection –** A design pattern to let the components injected into each other as dependencies.

**Q-3. Explain The Reasons That Compel A Web Developer To Choose AngularJS For His Web Development Project?**

**Answer.**

Following are some of the key reasons to choose AngularJS as you web development framework:

* It uses MVC design pattern which allows segregating an application into different components (called Model, View, and Controller) thus making it easy to maintain.
* It allows extending the HTML by attaching directives to your HTML markup. This provides the capability to define new powerful templates having new attributes, tags, and expressions.
* It allows you to create your own directives and also make reusable components. Directives help the developer to concentrate on creating logic, thus enabling them to work efficiently.
* It supports two-way data binding i.e. enables automatic synchronization of data between model and view components. Thus, any update in the model gets reflected in the view automatically. And there is no need to add any Javascript code or event listeners to reflect the data changes.
* It encapsulates the behavior of your application in controllers which gets instantiated with the help of dependency injection.
* It supports built-in services that perform the common tasks for web applications. For example, it provides $http service to communicate with REST service.
* It makes the development and testing of the application’s JavaScript code easy.
* Also, AngularJS has a mature community to help the developers. It has wide support over the internet.

**Q-4. List Down The Popular AngularJS IDE Plugins/Extensions For Web Development?**

**Answer.**

Here is a list of IDE Plugins and Extensions which can enhance the way you code with AngularJS:

* Sublime Text
* WebStorm
* Eclipse
* Netbeans
* Visual Studio 2012/2013 Express or higher
* TextMate
* Brackets
* ATOM

**Q-5. Explain The Steps Involved In The Boot Process For AngularJS?**

**Answer.**

Whenever a web page loads in the browser, following steps execute in the background.

* First, the HTML file containing the code gets loaded into the browser. After that, the JavaScript file mentioned in the HTML code gets loaded. It then creates a global object for angular. Now, the JavaScript which displays the controller functions gets executed.
* In this step, AngularJS browses the complete HTML code to locate the views. If a view is found, it is linked it to the corresponding controller function.
* In this step, AngularJS initiates the execution of required controller functions. Next, it populates the views with data from the model identified by the controller. With this the page is ready.

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-6. What Browsers Do AngularJS Support?**

**Answer.**

AngularJS works fine with the latest versions of Safari, Chrome, Firefox, Opera 15+, and IE9+ (Internet Explorer).  
It also supports various mobile browsers like Android, Chrome Mobile, iOS Safari, and Opera Mobile.

**Note:** Versions 1.3 and later of AngularJS dropped support for Internet Explorer 8.

**Q-7. What Are The Security Features Provided By AngularJS?**

**Answer.**

AngularJS provides built-in protection from the following security flaws.

* It prevents cross-site scripting attacks: Cross-site scripting is a technique where anyone can send a request from client side and can get the confidential information easily.
* It prevents HTML injection attacks.
* It prevents XSRF protection for server-side communication: It can be handled by “Auth token” mechanism. When the user logins for the first time a user id and password is sent to the server and it will, in turn, return an auth token. Now, this token does the authentication in the future transactions.

**Q-8. What Are The Web Application Security Risks That A Web Developer Should Avoid While Doing Development Using AngularJS?**

**Answer.**

Following are the most critical web application development flaws that a developer should take care of:

* Injection attack.
* Broken Authentication and Session Management.
* Cross-Site Scripting (XSS)
* Insecure direct object references.
* Security Misconfiguration.
* Sensitive Data Exposure.
* Missing Function Level Access Control.
* Cross-Site Request Forgery (CSRF).
* Using components that possess vulnerabilities.
* In-validated redirects and forwards.

**+**[**Top JavaScript Interview Questions**](http://www.techbeamers.com/javascript-interview-questions-answers/)

**Q-9. Explain What Are Directives? Mention Some Of The Most Commonly Used Directives In AngularJS Application?**

**Answer.**

AngularJS extends the behavior of HTML and DOM elements with new attributes called Directives. It directs the AngularJS’s HTML compiler ($compile) to attach a special behavior to that DOM element. This AngularJS component starts with prefix “ng”.

Following is the list of AngularJS built-in directives.

* **ng-bind –** The ng-bind directive tells AngularJS to replace the content of an HTML element with the value of a given variable, or expression.  
  If there is any change in the value of the given variable or expression, then the content of the specified HTML element will also be updated accordingly. It supports one-way binding only.
* **ng-model –** This directive is used to bind the value of HTML controls (input, select, text area) to application data. It is responsible for binding the view into the model. Directives such as input, text area, and select require it. It supports two-way data binding.
* **ng-class –**This directive dynamically binds one or more CSS classes to an HTML element. The value of the ng-class directive can be a string, an object, or an array.
* **ng-app –** Just like the “Main()” function of Java language, this directive marks the beginning of the application to AngularJS’s HTML compiler ($compile). If we do not use this directive first, an error gets generated.
* **ng-init –** This is used to initialize the application data so that we can use it in the block where it is declared. If an application requires local data like a single value or an array of values, this can be achieved using the ng-init directive.
* **ng-repeat –** This repeats a set of HTML statements for the defined number of times. The set of HTML statements will be repeated once per item in a collection. This collection must be an array or an object.

We can even create our own directives too and use them in our AngularJS Application.

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-10. What Are Expressions In AngularJS?**

**Answer.**

AngularJS binds data to HTML using Expressions. It can be written inside double braces: **{{ expression}}** or inside a directive as **ng-bind=”expression”**. AngularJS solves the expression and returns the result exactly where that expression is written.

AngularJS expressions are much like JavaScript expressions. They can include literals, operators, and variables.

**For example –**

{{ 2 + 2 }} (numbers)

{{Name + " " + email}} (string)

{{ Country.Name }} (object)

{{ fact[4] }} (array)

**Q-11. What Are Filters? Explain Different Filters Provided By AngularJS?**

**Answer.**

An AngularJS Filter changes or transforms the data before passing it to the view. These Filters work in combination with AngularJS expressions or directives. AngularJS uses pipe character **(“|”)** to add filters to the expressions or directives. For example:

<p> {{ bid | currency }} </p>

The above example is an expression enclosed in the curly braces. The filter used in this expression is currency. Also important to note that filters are case-sensitive.

AngularJS provides following filters to transform data.

* **currency –** It is used to format a number to a currency format.
* **date –** It is required to format a date to a specified format.
* **filter –** It chooses a subset of items from an array.
* **json –** It formats an object to a JSON string.
* **limitTo –** Its purpose is to create an array or string containing a specified number of elements/characters. The elements are selected, either from the beginning or the end of the source array or string. This depends on the value and sign (positive or negative) of the limit.
* **lowercase –** This filter converts a string to lower case.
* **number –** It formats a number as a text.
* **orderBy –** It enables to sort an array. By default, sorting of strings happens alphabetically. And sorting of numbers is done numerically. It also supports a comparator function where we can define what will be counted as a match or not.
* **uppercase –** This filter converts a string to upper case.

**Q-12. What Are Angular Prefixes $ And $$?**

**Answer.**

To prevent accidental name collisions within the code, AngularJS prefixes the names of public objects with **$** and the names of private objects with **$$**.

Use of these literals (**$** or **$$)** for any other reason is not recommended.

**Q-13. What Are Different Ways To Invoke A Directive?**

**Answer.**

There are four different ways to invoke a directive in an angular application. They are as follows.

1) As an attribute:

<span my-directive></span>

2) As a class:

<span class="my-directive: expression;"></span>

3) As an element:

<my-directive></my-directive>

4) As a comment:

<!-- directive: my-directive expression -->

**Q-14. What Is Singleton Pattern? How Does Angular Use It?**

**Answer.**

A singleton pattern is an approach that we adopt to limit the instantiation of a Class to have only one object. In Angular, the dependency injection and the services implement the singleton pattern.

Technically, if we call the **“new Class()”** two times without following the singleton pattern, the outcome will be two objects of the same class. Whereas a singleton enabled class will create the object first time and return the same object onwards.

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-15. What Is $Scope In AngularJS?**

**Answer.**

It is an application object. And behaves as the owner of the apps variables and functions. Scope object has access to both View and controller. Thus it works as a medium of communication between both of them. This object contains both data and functions. We can use it to access model data of the controller.

Following are the key characteristics of the scope object.

* It provides observers to watch for all the model changes.
* Provides the ability to propagate model changes through the application as well as outside the system to other associated components.
* Scopes can be nested in such a way that they can isolate functionality and model properties.
* Provides an execution environment in which expressions are evaluated.

**Q-16. What Is “$RootScope” In AngularJS?**

**Answer.**

Every AngularJS application has a **“$rootScope**” that is the topmost scope created on the DOM element. An app can have only one **$rootScope** which will be shared among all its components. It contains the ng-app directive. Every other scope is its child scope. It can watch expressions and propagate events. Using root scope we can set the value in one controller and read it from the other controller.

**Q-17. Explain The Concept Of Scope Hierarchy? How Many Scopes Can An Application Have?**

**Answer.**

Every AngularJS application consists of one root scope but may have several child scopes. As child controllers and directives create new child scopes, they get attached to the application. These new scopes get added as children of their parent scope. Similar to DOM, they also create a hierarchical structure.

**Q-18. What Is SPA (Single Page Application) In AngularJS?**

**Answer.**

Single-Page Applications (SPAs) are web applications that fit on a single HTML page. It dynamically updates the web page as the user performs actions on the app.

SPAs use AJAX and HTML to create quick and responsive web apps. A single page load extracts all the web app code (JS, HTML, CSS).

Thus the user navigates to different parts of the application quickly as it happens without refreshing the whole page.

Key Characteristics of Single-Page Applications are as follows.

* Its UI is fast and responsive. Also, the Back/Forward buttons present in the UI work properly.
* IT contains more JavaScript code than actual HTML as compared to other applications.
* Dynamic data loading occurs from the server-side. The API uses restful web service with JSON format.
* It allows to pre-load and cache all the app pages. Thus fewer data download requests are made towards the server.
* Applications written in AngularJS are cross-browser compliant. It automatically handles the JavaScript code suitable for each browser.
* Even if the user has lost the internet connection, then also the SPA can work. As all the pages load in the starting itself.

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-19. What Is The Difference Between $Scope And Scope?**

**Answer.**

It is mandatory to use **<$scope>** while defining a controller. However, the “scope” will be used to create a link function for the custom directive. Both of them refer to the **“scope”** object in AngularJS. The difference between them is that <$scope> uses dependency injection whereas “scope” does not.

Factory methods like controller, filter, service etc receive its arguments via dependency injection (DI). In DI, the order of passing the arguments does not matter. For example, a controller may be defined as (let’s define $scope as the first parameter in this case):

myApp.controller(‘MyController’, [‘$scope’, function($scope, $http) {

//rest of the code goes here }

OR ( if $scope is the second parameter)

myApp.controller(‘MyController’, [‘$scope’, function($http, $scope) {

//rest of the code goes here }

Thus, AngularJS does not care for the position of “$scope” in both the cases. It uses the argument name to retrieve an object out of the dependency injection container.

But, in the case of directive linker function, the position of scope matters, as it does not use DI. The reason being that the supplied arguments are received by its caller. In this case, the very first parameter has to be the scope as per AngularJS syntax.

app.directive(“myDirective”, function() {

return {

scope: {};

link: function(scope, element, attrs) {

// code goes here.

}

};

});

In the case of non-dependency injected arguments, we can also give a name to injected objects. The above code can be re-written as:

app.directive(“myDirective”, function() {

return {

scope: {};

link: function(foo, bar, biz) {

// code goes here.

}

};

});

To summarize, in DI case, we pass the <scope object> as <$scope> whereas, in non-DI cases, the <scope object> is returned either as a scope or with any name.

**+**[**Top HTML5 Interview Questions**](http://www.techbeamers.com/latest-html-interview-questions/)

**Q-20. How Is AngularJS Compiled?**

**Answer.**

Angular’s HTML compiler allows you to teach the browser, new HTML syntax. It allows the developer to attach new behaviors or attributes to any HTML element called as directives. AngularJS compilation process takes place in the web browser itself. It does not involve any server-side or pre-compilation step.

AngularJS uses <$compiler> service to compile the angular HTML page. Its compilation begins after the HTML page (static DOM) is fully loaded.

It occurs in two phases.

* **Compile –** It looks into the entire DOM and collects all of the directives. The result is a linking function.
* **Link –** It combines the directives with a scope and produces a live view. Any changes in the <scope model> get reflected in the view, and any operations done by the user in the view gets reflected in the <scope model>.

The concept of compile and link has come from C language. Here the code is compiled first and then linked.

**Q-21. How Is AngularJS Compilation Different From Other JavaScript Frameworks?**

**Answer.**

Javascript frameworks like backbone and jQuery process the template as a string and returns the result as a string. You have to dump this resulting string into the DOM where you wanted it with **innerHTML()**.

AngularJS process the template in another way. It directly works on HTML DOM rather than strings and manipulates it as required. It uses two-way data binding between model and view to sync the data.

**Q-22. What Is Ng-View In AngularJS?**

**Answer.**

The ng-view tag creates a placeholder where an HTML or ng-template view can be placed based on the configuration.

Let’s take an example.

<div ng-app = "testApp">

<div ng-view>

<!-- Target Html Template here -->

</div>

</div>

In other words, ng-view is the directive that works as a container for angularJS to switch between views.

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-23. What Is Ng-Template In AngularJS?**

**Answer.**

The ng-template directive is used to create an HTML view using script tag. It contains **“id”** attribute which is used by **$routeProvider** to map a view with a controller.

While defining ng-template, it is mandatory to specify the type of the **<script>** element as the **text/ng-template**. Also, assign a cache name to the template using the element’s id. Later on, this name gets used as directive’s templateUrl.

Following is the syntax of using a ng-template directive in angularJS application.

<div ng-app = "mainApp">

<scrip t type = "text/ng-template" id = "addEmployee.htm">

<h2> Add Employee </h2>

{{message}}

</scrip t>

</div>

$routeProvider part.

var mainApp = angular.module("mainApp", ['ngRoute']);

mainApp.config(['$routeProvider', function($routeProvider) {

$routeProvider.

when('/addEmployee', {

templateUrl: 'addEmployee.htm', controller: 'AddEmployeeController'

}).

otherwise({

redirectTo: '/addEmployee'

});

}]);

**Q-24. What Is $RouteProvider In AngularJS?**

**Answer.**

$routeProvider is the key service which set the configuration of URLs, map them with the corresponding HTML page or ng-template, and attach a controller with the same.

Let’s see the following example:

var mainApp = angular.module("mainApp", ['ngRoute']);

mainApp.config(['$routeProvider', function($routeProvider) {

$routeProvider.

when('/addEmployee', {

templateUrl: 'addEmployee.htm', controller: 'AddEmployeeController'

}).

otherwise({

redirectTo: '/addEmployee'

});

}]);

Following are the important points to be considered in above example.

* routeProvider is defined as a function under config of the mainApp module using a key as ‘$routeProvider’.
* $routeProvider.when defines a URL “/addEmployee” which is then mapped to “addEmployee.htm”. This <addEmployee.htm> should be present in the same path as main HTML page.
* “otherwise” is used to set the default view.
* “controller” is used to set the corresponding controller for the view.

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-25. What Is Data Binding? How Many Types Of Data Binding Directives Are Provided By AngularJS?**

**Answer.**

Data binding is the connection bridge between view and business logic (view model) of the application. Data binding in AngularJs is the automatic synchronization between the model and view. When the model changes, the view is automatically updated and vice versa. AngularJs support one-way binding as well as two-way binding.

AngularJS provides the following data binding directives:

**1) <ng-bind>-** It updates the text content of the specified HTML element with the value of the given expression. This text content gets updated when there is any change in the expression. It is very similar to double curly markup ( {{expression }}) but less verbose.

It has the following Syntax.

<ANY ELEMENT ng-bind="expression"> </ANY ELEMENT>

**2) <ng-bind-html>-** It evaluates the expression and inserts the HTML content into the element in a secure way. To use this functionality, it has to use $sanitize service. For this, it is mandatory that $sanitize is available.

It has the following Syntax.

<ANY ELEMENT ng-bind-html=" expression "> </ANY ELEMENT>

**3) <ng-bind-template>-**It replaces the element text content with the interpolation of the template. It can contain multiple double curly markups.

It has the following Syntax.

<ANY ELEMENT ng-bind-template=" {{expression1}} {{expression2}} … {{expressionn}} "> </ANY ELEMENT>

**4) <ng-non-bindable>-** This directive informs AngularJS, not to compile or bind the contents of the current DOM element. It is useful in the case when the user wants to display the expression only and do not want to execute it.

It has the following Syntax.

<ANY ELEMENT ng-non-bindable > </ANY ELEMENT>

**5) <ng-model>-** This directive can be bound with input, select, text area or any custom form control. It provides two-way data binding. It also provides validation behavior. It also retains the state of the control (like valid/invalid, touched/untouched and so on).

It has the following Syntax.

<input ng-bind="expression"/>

**Q-26. What Directives Are Used To Show And Hide HTML Elements In AngularJS?**

**Answer.**

The directives used to show and hide HTML elements in the AngularJS are <ng-show> and <ng-hide>. They do this based on the result of an expression.

Its syntax is as follows.

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

When the expression for <ng-show> evaluates to true, then HTML element(s) are shown on the page, otherwise the HTML element is hidden. Similarly, <ng-hide> directive hides the HTML element if the expression evaluates to true.

Let’s take the following example.

<div ng-controller="MyCtrl">

<div ng-show="data.isShow">ng-show Visible</div>

<div ng-hide="data.isHide">ng-hide Invisible</div>

</div>

<script>

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

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

$scope.data = {};

$scope.data.isShow = true;

$scope.data.isHide = true;

});

</script>

**Q-27. Explain The Directives Ng-If, Ng-Switch, And Ng-Repeat?**

**Answer.**

**A) <Ng-If>.**

This directive can add/remove HTML elements from the DOM based on an expression. If the expression is true, it adds a copy of HTML elements to the DOM. If the expression evaluates to false, this directive removes the HTML element from the DOM.

<div ng-controller="MyCtrl">

<div ng-if="data.isVisible">ng-if Visible</div>

</div>

<script>

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

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

$scope.data = {};

$scope.data.isVisible = true;

});

</script>

**B) <Ng-Switch>.**

This directive can add/remove HTML elements from the DOM conditionally based on scope expression.

Child elements with the <ng-switch-when> directive will be displayed if it gets a match, else the element and its children get removed. It also allows defining a default section, by using the <ng-switch-default> directive. It displays a section if none of the other sections match.

Let’s see the following example that displays the syntax for <ng-switch>.

<div ng-controller="MyCtrl">

<div ng-switch on="data.case">

<div ng-switch-when="1">Shown when case is 1</div>

<div ng-switch-when="2">Shown when case is 2</div>

<div ng-switch-default>Shown when case is anything else than 1 and 2</div>

</div>

</div>

<script>

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

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

$scope.data = {};

$scope.data.case = true;

});

</script>

**C) <Ng-Repeat>.**

This directive is used to iterate over a collection of items and generate HTML from it.

<div ng-controller="MyCtrl">

<ul>

<li ng-repeat="name in names">

{{ name }}

</li>

</ul>

</div>

<script>

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

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

$scope.names = [ 'Mahesh', 'Raj', 'Diksha' ];

});

</script>

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-28. Explain The Set Of “Special” Variables Supported With <Ng-Repeat> Directive?**

**Answer.**

The <ng-repeat> directive has a set of special variables that are useful while iterating the collection.

These variables are as follows.

* $index
* $first
* $middle
* $last

The “$index” contains the index of the element being iterated. The variables $first, $middle and $last returns a boolean value depending on whether the current item is the first, middle or last element in the collection being iterated.

<html>

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

<head>

<script>

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

app.controller("ctrl", function ($scope)

{

$scope.employees = [

{

name: 'A',

gender: 'alphabet'

},

{

name: 'B',

gender: 'number'

},

{

name: 'C',

gender: 'alphanumeric'

},

{

name: 'D',

gender: 'special character'

}];

});

</script>

</head>

<body ng-app="app">

<div ng-controller="ctrl">

<ul>

<li ng-repeat="employee in employees">

<div> {{employee.name}} is a {{employee.gender}}. <span ng-if="$first">

<strong>(first element found)</strong>

</span> <span ng-if="$middle">

<strong>(middle element found)</strong>

</span> <span ng-if="$last">

<strong>(last element found)</strong>

</span>

</div>

</li>

</ul>

</div>

</body>

</html>

The output is as follows.

A is a alphabet. (first element found)

B is a number. (middle element found)

C is a alphanumeric. (middle element found)

D is a special character. (last element found)

**Q-29. What Is A Factory Method In AngularJS?**

**Answer.**

A factory is a simple function which allows you to add some logic before creating the object. In the end, it returns the created object.

**Syntax.**

app.factory('serviceName',function(){ return serviceObj;})

**Creating Service Using The Factory Method.**

<script>

//creating module

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

//define a factory using factory() function

app.factory('MyFactory', function () {

var serviceObj = {};

serviceObj.function1 = function () {

//TO DO:

};

serviceObj.function2 = function () {

//TO DO:

};

return serviceObj;

});

</script>

**When To Use Factory?**

It is just a collection of functions, like a class. Hence, it can be instantiated in different controllers when you are using it with a constructor function.

**Q-30. Explain What Is String Interpolation In AngularJS?**

**Answer.**

During the compilation process, AngularJS compiler matches text and attributes using interpolate service to see if it contains embedded expressions.

During normal, digest life cycle, these expressions are updated and registered as watches.

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-31. Explain AngularJS Application Life-Cycle?**

**Answer.**

Understanding the life cycle of an AngularJS application makes it easier to learn about the way to design and implement the code. Apps life cycle consists of following three phases- bootstrap, compilation, and runtime.  
These three phases of the life cycle occur each time a web page of an AngularJS application gets loaded in the browser. Let’s learn about each of the three phases in detail:

* **The Bootstrap Phase –** In this phase, the browser downloads the AngularJS javascript library. After this, AngularJS initializes its necessary components and the modules to which the ng-app directive points. Now that the module has loaded, required dependencies are injected into it and become available to the code within that module.
* **The Compilation Phase –** The second phase of the AngularJS life cycle is the HTML compilation stage. Initially, when a web page loads in the browser, a static form of the DOM gets loaded. During the compilation phase, this static DOM gets replaced with a dynamic DOM which represents the app view. There are two main steps – first, is traversing the static DOM and collecting all the directives. These directives are now linked to the appropriate JavaScript functionality which lies either in the AngularJS built-in library or custom directive code. The combination of directives and the scope, produce the dynamic or live view.
* **The Runtime Data Binding Phase –** This is the final phase of the AngularJS application. It remains until the user reloads or navigates to a different web page. At this point, any changes in the scope get reflected in the view, and any changes in the view are directly updated in the scope, making the scope the single source of data for the view.

This shows that AngularJS behaves differently from traditional methods of binding data. The traditional methods combine a template with data, received from the engine and then manipulate the DOM each time there is any change in the data.

However, AngularJS compiles the DOM only once and then links the compiled template as necessary, making it much more efficient than the traditional methods.

**Q-32. Explain AngularJS Scope Life-Cycle?**

**Answer.**

After the angular app gets loaded into the browser, scope data passes through different stages called as its life cycle. Learning about this cycle helps us to understand the interaction between scope and other AngularJS components.

The scope data traverses through the following phases.

* **Creation –** This phase initializes the scope. During the bootstrap process, the $injector creates the root scope for the application. And during template linking, some directives create new child scopes. A digest loop also gets created in this phase that interacts with the browser event loop. This loop is responsible for updating DOM elements with the changes made to the model as well as executing any registered watcher functions.
* **Watcher registration –** This phase registers watchers for scope created in the above point. These watches propagate the model changes to the DOM elements, automatically. We can also register our own watcher’s on a scope by using the $watch() function.
* **Model mutation –** This phase occurs when there is any change in the scope data. When we do any modification in the angular app code, the scope function <$apply()> updates the model and then calls the <$digest()> function to update the DOM elements and the registered watches. However, when we change the scope inside the angular code like within controllers or services, angular internally calls <$apply()> function for us. But, when we do the changes to the scope outside the angular code, we have to call the <$apply()> function explicitly on the scope, to force the model and DOM to be updated correctly.
* **Mutation observation –** This phase occurs, when the digest loop execute the $digest() function at the end of $apply() call. When the $digest() function executes, it evaluates all watches for model changes. If there is a change in the value, $digest() calls the $watch listener and updates the DOM elements.
* **Scope destruction –** This phase occurs when the child scopes that are no longer needed, are removed from the browser’s memory by using the $destroy() function. It is the responsibility of the child scope creator to destroy them via scope.$destroy() API. This stops propagation of $digest calls into the child scopes and enables the browsers’ garbage collector to reclaim the unused memory.

**Q-33. What Is An Auto Bootstrap Process In AngularJS?**

**Answer.**

AngularJS initializes automatically upon the **“DOMContentLoaded**” event or when the browser downloads the angular.js script and at the same time **document.readyState** is set to **‘complete’**. At this point, AngularJS looks for the ng-app directive which is the root of Angular app compilation process.

If the ng-app directive is located, then AngularJS will do the following.

* Load the module associated with the directive.
* Create the application injector.
* Compile the DOM starting from the ng-app root element.

We term this process as **Auto-bootstrapping**.

Following is the sample code that helps to understand it more clearly:

<html>

<body ng-app="myApp">

<div ng-controller="Ctrl">Hello {{msg}}!</div>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.5/angular.min.js"></script>

<script>

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

app.controller('Ctrl', function($scope) {

$scope.msg = 'Welcome';

});

</script>

</body>

</html>

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-34. What Is The Manual Bootstrap Process In AngularJS?**

**Answer.**

Sometimes we may need to manually initialize Angular app in order to have more control over the initialization process. We can do that by using **angular.bootstrap()** function within **angular.element(document).ready()**function. AngularJS fires this function when the DOM is ready for manipulation.

The **angular.bootstrap()** function takes two parameters, the document, and module name injector.

Following is the sample code that helps to understand the concept more clearly.

<html>

<body>

<div ng-controller="Ctrl">Hello {{msg}}!</div>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.5/angular.min.js"></script>

<script>

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

app.controller('Ctrl', function($scope) {

$scope.msg = 'Welcome';

});

//manual bootstrap process

angular.element(document).ready(function () { angular.bootstrap(document, ['myApp']); });

</script>

</body>

</html>

**Q-35. How To Bootstrap Your Angular App For Multiple Modules?**

**Answer.**

Bootstrap for multiple modules can be achieved by using following two methods.

**1. Automatic bootstrap –** AngularJS is automatically initialized for one module. When we have multiple modules, we combine them into a single module and thus the angular app will be automatically initialized for the newly created module. Other modules act as dependent modules for this newly created module.

Let’s take an example, suppose we have two modules: module1 and model2. To initialize the app automatically, based on these two modules following code is used:

<html>

<head>

<title>Multiple modules bootstrap</title>

<script src="lib/angular.js"></script>

<script>

//module1

var app1 = angular.module("module1", []);

app1.controller("Controller1", function ($scope) {

$scope.name = "Welcome";

});

//module2

var app2 = angular.module("module2", []);

app2.controller("Controller2", function ($scope) {

$scope.name = "World";

});

//module3 dependent on module1 & module2

angular.module("app", ["module1", "module2"]);

</script>

</head>

<body>

<!--angularjs autobootstap process-->

<div ng-app="app">

<h1>Multiple modules bootstrap</h1>

<div ng-controller="Controller2">

</div>

<div ng-controller="Controller1">

</div>

</div>

</body>

</html>

**2. Manual bootstrap –** We can manually bootstrap the app by using angular.bootstrap() function, for multiple modules.

The above example can be rewritten for a manual bootstrap process as given below.

<html>

<head>

<title>Multiple modules bootstrap</title>

<script src="lib/angular.js"></script>

<script>

//module1

var app1 = angular.module("module1", []);

app1.controller("Controller1", function ($scope) {

$scope.name = "Welcome";

});

//module2

var app2 = angular.module("module2", []);

app2.controller("Controller2", function ($scope) {

$scope.name = "World";

});

//manual bootstrap process

angular.element(document).ready(function () {

var div1 = document.getElementById('div1');

var div2 = document.getElementById('div2');

//bootstrap div1 for module1 and module2

angular.bootstrap(div1, ['module1', 'module2']);

//bootstrap div2 only for module1

angular.bootstrap(div2, ['module1']);

});

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-36. What Are Compile, Pre, And Post Linking In AngularJS?**

**Answer.**

**A) Compile –** It collects an HTML string or DOM into a template and produces a template function. It can then be used to link the scope and the template together.

AngularJS uses the compile function to change the original DOM before creating its instance and before the creation of scope.

Before discussing the Pre-Link and the Post-Link functions let’s see the Link function in detail.

**B) Link –** It has the duty of linking the model to the available templates. AngularJS does the data binding to the compiled templates using Link.

Following is the Link syntax.

link: function LinkFn(scope, element, attr, ctrl){}

where each of the four parameters is as follows.

* **Scope –** It is the scope of the directive.
* **element –** It is the DOM element where the directive has to be applied.
* **attr-** It is the Collection of attributes of the DOM element.
* **ctrl –**It is the array of controllers required by the directive.

AngularJS allows setting the link property to an object also. The advantage of having an object is that we can split the link function into two separate methods called, pre-link and post-link.

**C) Post-Link –** Execution of Post-Link function starts after the linking of child elements. It is safer to do DOM transformation during its execution. The post-link function is suitable to execute the logic.

**D) Pre-Link –** It gets executed before the child elements are linked. It is not safe to do DOM transformation. As the compiler linking function will fail to locate the correct elements.

It is good to use the pre-link function to implement the logic that runs when AngularJS has already compiled the child elements. Also, before any of the child element’s post-link functions have been called.

Let’s see an example that talks about Compile, Pre-Link, and Post-Link functions.

<html>

<head>

<title>Compile vs Link</title>

<script src="lib/angular.js"></script>

<script type="text/javascript">

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

function createDirective(name){

return function(){

return {

restrict: 'E',

compile: function(tElem, tAttrs){

console.log(name + ': compile');

return {

pre: function(scope, iElem, iAttrs){

console.log(name + ': pre link');

},

post: function(scope, iElem, iAttrs){

console.log(name + ': post link');

}

}

}

}

}

}

app.directive('levelOne', createDirective('levelOne'));

app.directive('levelTwo', createDirective('levelTwo'));

app.directive('levelThree', createDirective('levelThree'));

</script>

</head>

<body ng-app="app">

<level-one>

<level-two>

<level-three>

Hello {{name}}

</level-three>

</level-two>

</level-one>

</body>

</html>

Output:

Hello

**Q-37. What Is A Controller In AngularJS?**

**Answer.**

A Controller is a set of JavaScript functions which is bound to a specified scope, the ng-controller directive. Angular creates a new instance of the Controller object to inject the new scope as a dependency. The role of the Controller is to expose data to our view via $scope and add functions to it, which contains business logic to enhance view behavior.

**Controller Rules.**

* A Controller helps in setting up the initial state of the scope object and define its behavior.
* The Controller should not be used to manipulate the DOM as it contains only business logic. Rather, for manipulating the DOM, we should use data binding and directives.
* Do not use Controllers to format input. Rather, using angular form controls is recommended for that.
* Controllers should not be used to share code or states. Instead, use angular services for it.

**Steps For Creating A Controller.**

* It needs ng-controller directive.
* Next step is to add Controller code to a module.
* Name your Controller based on functionality. Its name should follow camel case format (i.e. SampleController).
* Set up the initial state of the scope object.

Declaring a Controller using ng-Controller directive.

<div ng-app="mainApp" ng-controller="SampleController">

</div>

Following code displays the definition of SampleController.

<script>

function SampleController($scope) {

$scope.sample = {

firstSample: "INITIAL",

lastSample: "Initial",

fullName: function() {

var sampleObject;

sampleObject = $scope.sample;

return sampleObject.firstSample + " " + sampleObject.lastSample;

}

};

}

</script>

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-38. What Does A Service Mean In AngularJS? Explain Its Built-In Services?**

**Answer.**

Services are functions that are bound to perform specific tasks in an application.

* It gives us a method, that helps in maintaining the angular app data for its lifetime.
* It gives us methods, that facilitate to transfer data across the controllers in a consistent way.
* It is a singleton object and its instance is created only once per application.
* It is used to organize and share, data and function across the application.

Two main execution characteristics of angular services are that they are Singleton and lazy instantiated.

**1. Lazily instantiated –**

It means that AngularJS instantiates a service only when a component of an application needs it. This is done by using dependency injection method, that makes the Angular codes, robust and less error-prone.

**2. Singletons –**

Each application component dependent on the service, work with the single instance of the service created by the AngularJS.

Let us take an example of a very simple service that calculates the square of a given number:

var CalculationService = angular.module('CalculationService', [])

.service('Calculation', function () {

this.square = function (a) { return a\*a};

});

**AngularJS Internal Services –**

AngularJS provides many built-in services. Each of them is responsible for a specific task. Built-in services are always prefixed with the $ symbol.

Some of the commonly used services in any AngularJS application are as follows:

* **$http –** used to make an Ajax call to get the server data.
* **$window –** Provides a reference to a DOM object.
* **$Location –** Provides reference to the browser location.
* **$timeout –** Provides a reference to window.set timeout function.
* **$Log –** used for logging.
* **$sanitize –** Used to avoid script injections and display raw HTML in the page.
* **$Rootscope –** Used for scope hierarchy manipulation.
* **$Route –** Used to display browser-based path in browser URL.
* **$Filter –** Used for providing filter access.
* **$resource –** Used to work with Restful API.
* **$document –** Used to access the window. Document object.
* **$exceptionHandler –** Used for handling exceptions.
* **$q –** Provides a promise object.
* **$cookies –** Use this service for reading, writing and deleting the browser cookies.
* **$parse –** This service intends to convert an AngularJS expression into a function.
* **$cacheFactory –** This service evaluates the specified expression when the user changes the input.

**Q-39. What Are Different Ways To Create Service In AngularJS?**

**Answer.**

There are 5 different ways to create services in AngularJS.

* Value
* Factory
* Service
* Provider
* Constant

Let’s discuss, each of the above AngularJS service types one by one with code example:

**1. AngularJS Value.**

It is the simplest service type supported by AngularJS that we can create and use. It is similar to a key-value pair or like a variable having a value. It can store only a single value. Let’s take an example and create a service that displays username:

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

app.value("username","Madhav");

**Code to use “Value”:**

We can use this service anywhere by using dependency injection. Following example injects the service in a controller:

app.controller("MainController",function($scope, username){

$scope.username=username;

});

In the above example, we have created a Value service “username” and used it in MainController.

**2. AngularJS Factory.**

Value service may be very easy to write but, it lacks many important features. So, the next service type we will look at is “Factory” service. After its creation, we can even inject other services into it. Unlike Value service, we cannot add any dependency in it.

Let’s take an example to create a Factory service.

app.factory("username",function(){

var name="John";

return {

name:name

}

});

The above code shows that Factory service takes “function” as an argument. We can inject any number of dependencies or methods in this “function” as required by this service. This function must return some object. In our example, it returns an object with the property name. Now, let us look, as to how we can use this service:

**Code to use “Factory”:**

The function returns an object from service which has a property name so we can access it and use it anywhere. Let’s see how we can use it in the controller:

app.controller("MainController",function($scope, username){

$scope.username=username.name;

});

We are assigning the username from factory service to our scope username.

**3. AngularJS Service.**

It works same as the “Factory” service. But, instead of a function, it receives a Javascript class or a constructor function as an argument. Let’s take an example. Suppose we have a function:

function MyExample(num){

this.variable="value";

}

Now, we want to convert the function into a service. Let’s take a look at how we can do this with “Factory” method:

app.factory("MyExampleService",["num" ,function(num){

return new MyExample(num);

}]);

Thus in this way, we will create its new instance and return it. Also, we have injected <num> as a dependency in Factory service. Now, let’s see how we can do this using Service type:

app.service("MyExampleService",["num", MyExample]);

Thus, we have called the service method on the module and provided its name, dependency, and the name of the function in an array.

**4. AngularJS Provider.**

It is the parent of all the service types supported by AngularJS, except the “Constant” that we will discuss in the next section. It is the core of all the service types. Thus we can say that other services work on top of it. It allows us to create a configurable service that must implement the <$get> method.

We use this service to expose the API that is responsible for doing the application-wide configuration. The configuration should complete before starting the application.

Let’s take an example.

app.provider('authentication', function() {

var username = "John";

return {

set: function(newUserName) {

username = newUserName;

},

$get: function() {

function getUserName() {

return username;

}

return {

getUserName: getUserName

};

}

};

});

This example initializes a provider with its name as “authentication”. It also implements a <$get> function, which returns a method “getUsername” which in turn returns the private variable called username. This also has a setter, using it we can set the username on application startup as follows:

app.config(["authenticationProvider", function(authenticationProvider) {

authenticationProvider.set("Mihir");

}]);

**5. AngularJS Constant.**

As the name suggests, this service helps us to declare constants in our application. We can then use them wherever needed, just by adding it as a dependency. There are many places, where we use constants like some base URLs, application name, etc.

We just define them once and use them anywhere as per our need. Thus, this technique allows us to write the definition at one place. If there is any change in the value later, we have to do the modifications at one place only.

Here is an example of how we can create constants:

app.constant('applicationName', 'Service Tutorials');

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-40. What Is The Difference Between The $Watch, $Digest, And $Apply?**

**Answer.**

In AngularJS $scope object is having different functions like **$watch()**, **$digest()** and **$apply()** and we will call these functions as central functions. The AngularJS central functions **$watch()**, **$digest()**, and **$apply()** are used to bind data to variables in view and observe changes happening in variables.

**A) $Watch() –**

The use of this function is to observe changes in a variable on the $scope. It triggers a function call when the value of that variable changes. It accepts three parameters: expression, listener, and equality object. Here, listener and equality objects are optional parameters.

$watch(watchExpression, listener, [objectEquality]).

Following is the example of using the **$watch()** function in AngularJS applications.

<html>

<head>

<title>AngularJS Watch</title>

<script src="lib/angular.js"></script>

<script>

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

var myController = myapp.controller("myController", function

($scope) {

$scope.name = 'dotnet-tricks.com';

$scope.counter = 0;

//watching change in name value

$scope.$watch('name', function (newValue, oldValue) {

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

});

});

</script>

</head>

<body ng-app="myapp" ng-controller="myController">

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

<br /><br />

Counter: {{counter}}

</body>

</html>

**B) $Digest() –**

This function iterates through all the watch list items in the $scope object, and its child objects (if it has any). When **$digest()** iterates over the watches, it checks if the value of the expression has changed or not. If the value has changed, AngularJS calls the listener with the new value and the old value.

The **$digest()** function is called whenever AngularJS thinks it is necessary. For example, after a button click, or after an AJAX call. You may have some cases where AngularJS does not call the **$digest()** function for you. In that case, you have to call it yourself.

**+**[**Top NodeJS Interview Questions**](http://www.techbeamers.com/top-30-node-js-interview-questions-answers/)

Following is the example of using the **$digest()** function in AngularJS applications:

<html>

<head>

<title>AngularJS Digest Example</title>

<script src="lib/jquery-1.11.1.js"></script>

<script src="lib/angular.js"></script>

</head>

<body ng-app="app">

<div ng-controller="Ctrl">

<button class="digest">Digest my scope!</button>

<br />

<h2>obj value : {{obj.value}}</h2>

</div>

<script>

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

app.controller('Ctrl', function ($scope) {

$scope.obj = { value: 1 };

$('.digest').click(function () {

console.log("digest clicked!");

console.log($scope.obj.value++);

//update value

$scope.$digest();

});

});

</script>

</body>

</html>

**C) $Apply() –**

AngularJS automatically updates the model changes which are inside AngularJS context. When you apply changes to any model, that lies outside of the Angular context (like browser DOM events, setTimeout, XHR or third-party libraries), then you need to inform the Angular about the changes by calling **$apply()** manually. When the **$apply()** function call finishes, AngularJS calls **$digest()** internally, to update all data bindings.

Following are the key differences between **$apply()** and **$digest()**.

* Its use is to update the model properties forcibly.
* The **$digest()** method evaluates the watchers for the current scope. However, the **$apply()** method is used to evaluate watchers for root scope, that means it’s for all scopes.

Following is the example of using the **$apply()** function in AngularJS applications.

<html>

<head>

<title>AngularJS Apply Example</title>

<script src="lib/angular.js"></script>

<script>

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

var myController = myapp.controller("myController", function

($scope) {

$scope.datetime = new Date();

$scope.updateTime = function () {

$scope.datetime = new Date();

}

//outside angular context document.getElementById("updateTimeButton").addEventListener('click', function () {

//update the value

$scope.$apply(function () {

console.log("update time clicked");

$scope.datetime = new Date();

console.log($scope.datetime);

});

});

});

</script>

</head>

<body ng-app="myapp" ng-controller="myController">

<button ng-click="updateTime()">Update time - ng-click</button>

<button id="updateTimeButton">Update time</button>

<br />

{{datetime | date:'yyyy-MM-dd HH:mm:ss'}}

</body>

</html>

**Q-41. Which One Handles Exception Automatically Between $Digest And $Apply?**

**Answer.**

When an error occurs in one of the watchers, **$digest()** cannot handle them via **$exceptionHandler** service. In that case, you have to handle the exception yourself. However, **$apply()** uses try catch block internally to handle errors. But, if an error occurs in one of the watchers, then it transfers the errors to $exceptionHandler service.

Code for **$apply()** function.

$apply(expr) {

try {

return $eval(expr);

} catch (e) {

$exceptionHandler(e);

} finally {

$root.$digest();

}

}

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-42. Explain $Watch(), $Watchgroup() And $WatchCollection() Functions Of Scope?**

**Answer.**

**A) $Watch.**

Its use is to observe the changes in the variable on the $scope. It accepts three arguments – expression, listener, and equality object. The listener and Equality object are optional parameters.

$watch(watchExpression, listener, [objectEquality])

Here, **<watchExpression>** is the expression to be observed in the scope. This expression gets called on every **$digest()** and returns a value for the listener to monitor.

The listener defines a function which gets called when **watchExpression** changes its value. In case, its value does not change then the listener will not be called. The objectEquality is a boolean type to compare the objects for equality using angular.equals.

scope.name = 'shailendra';

scope.counter = 0;

scope.$watch('name', function (newVal, oldVal) {

scope.counter = scope.counter + 1; });

**B) $Watchgroup.**

This function was introduced in **Angular 1.3**. It works in the same way as **$watch()** function except that the first parameter is an array of expressions.

$watchGroup(watchExpression, listener)

The listener is also an array containing the new and old values of the variables that are being watched. The listener gets called whenever any expression contained in the **watchExpressions** array changes.

$scope.teamScore = 0;

$scope.time = 0;

$scope.$watchGroup(['teamScore', 'time'], function(newVal, oldVal) {

if(newVal[0] > 20){

$scope.matchStatus = 'win';

}

else if (newVal[1] > 60){

$scope.matchStatus = 'times up';

});

**C) $WatchCollection.**

This use of this function is to watch the properties of an object. It gets fired when there is any change in the properties. It takes an object as the first parameter and watches the properties of the object.

$watchCollection(obj, listener)

The listener is called whenever there is any change in the obj.

$scope.names = ['shailendra', 'deepak', 'mohit', 'kapil'];

$scope.dataCount = 4;

$scope.$watchCollection('names', function (newVal, oldVal) {

$scope.dataCount = newVal.length;

});

**Q-43. What Are The Form Validations Supported By AngularJS?**

**Answer.**

AngularJS allows form validation on the client-side in a simplistic way. First of all, it monitors the state of the form and its input fields. Secondly, it observes any change in the values and notifies the same to the user.  
Let’s discuss the different input field validations along with examples.

**A) Required Field Validation.**

By using “Required Field” validation we can prevent, form submission with a null value. It’s mandatory for the user to fill the form fields.

The syntax for required field validation is as follows.

<input type="text" required />

Example Code.

<form name="myForm">

<input name="myInput" ng-model="myInput" required>

</form>

<p>The input's valid state is:</p>

<h1>{{myForm.myInput.$valid}}</h1>

**B) Minimum & Maximum Field Length Validations.**

To prevent the user from providing less or excess number of characters in the input field, we use Minimum & Maximum length validation. The AngularJS directive used for Minimum & Maximum length validations are <ng-minlength> and <ng-maxlength>. Both of these attributes take integer values. The <ng-minlength> attribute is used to set the number of characters a user is limited to, whereas the <ng-maxlength> attribute sets the maximum number of characters that a user is allowed to enter.

It has the following syntax.

<input type="text" ng-minlength=5 />

<input type="text" ng-maxlength=10 />

Example code:

<label>User Message:</label>

<textarea type="text" name="userMessage" ng-model="message"

ng-minlength="100" ng-maxlength="1000" required>

</textarea>

<div ng-messages="exampleForm.userMessage.$error">

<div ng-message="required">This field is required</div>

<div ng-message="minlength">Message must be over 100 characters</div>

<div ng-message="maxlength">Message must not exceed 1000 characters</div>

</div>

**C) Matches Pattern Validation.**

AngularJS provides a <ng-pattern> directive to ensure that input fields match the regular expressions that are passed into the attribute.

It has the following syntax.

<input type="text" ng-pattern="[a-zA-Z]" />

To activate the error message in <ng-pattern> we pass the value of pattern into ng-message.

Example code.

<label>Phone Number:</label>

<input type="email" name="userPhoneNumber" ng-model="phoneNumber"

ng-pattern="/^[\+]?[(]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$/"

required/>

<div ng-messages="exampleForm.userPhoneNumber.$error">

<div ng-message="required">This field is required</div>

<div ng-message="pattern">Must be a valid 10 digit phone number</div>

</div>

**D) Email Validation.**

In order to validate an email id, AngularJS provides ng-model directive. Using the following syntax, we can validate an email id from an input field.

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

Example code.

<label>Email Address:</label>

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

<div ng-messages="exampleForm.userEmail.$error">

<div ng-message="required">This field is required</div>

<div ng-message="email">Your email address is invalid</div>

</div>

**E) Number Validation.**

To validate an input against Number we can use ng-model directive from AngularJS.

Its syntax is as follows.

<input type="number" name="personage" ng-model="user.age" />

**F) URL Validation.**

To validate an input field for URL, we can use the following syntax.

<input type="url" name="weblink" ng-model="user.facebook\_url" />

Example Code.

<div ng-app="urlInputExample">

<form name="myForm" ng-controller="UrlController">

<label for="exampleInput">Enter Email</label>

<input type="url" name="input" ng-model="example.url" required />

<p style="font-family:Arial;color:red;background:steelblue;padding:3px;width:350px;"

ng-if='!myForm.input.$valid'>Enter Valid URL</p>

</form>

</div>

**Q-44. How Do You Exchange Data Among Different Modules Of Your Angular JS Application?**

**Answer.**

There are a no. of ways in Angular to share data among modules. A few of them are as follows.

* The most common method is to create an Angular service to hold the data and dispatch it across the modules.
* Angular has a matured event system which provides $broadcast(), $emit() and $on() methods to raise events and pass data among the controllers.
* We can also use $parent, $nextSibling, and $ controllerAs to directly access the controllers.
* Variables defined at the root scope level ($rootScope) are available to the controller scope via prototypical inheritance. But they behave like globals and hard to maintain.

[**Back to top**](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/#top)

**Q-45. How Would You Use An Angular Service To Pass Data Between Controllers? Explain With Examples?**

**Answer.**

Using a service is the best practice in Angular to share data between controllers. Here is a step by step example to demonstrate data transfer.

We can prepare the data service provider in the following manner.

app.service('dataService', function() {

var dataSet = [];

var addData = function(newData) {

dataSet.push(newData);

};

var getData = function(){

return dataSet;

};

return {

addData: addData,

getData: getData

};

});

Now, we’ll inject the service dependency into the controllers.

Say, we have two controllers – pushController and popController.

The first one will add data by using the data service provider’s addData method. And the latter will fetch this data using the service provider’s getData method.

app.controller('pushController', function($scope, dataService) {

$scope.callToAddToProductList = function(currObj){

dataService.addData(currObj);

};

});

app.controller('popController', function($scope, dataService) {

$scope.dataSet = dataService.getData();

});

**Q-46. How Will You Send And Receive Data Using The Angular Event System? Use Methods Like $Broadcast And $On To Send Data Across?**

**Answer.**

We can call the $broadcast method using the $rootScope object and send any data we want.

$scope.sendData = function() {

$rootScope.$broadcast('send-data-event', data);

}

To receive data, we can use the $scope object inside a controller.

$scope.$on('send-data-event', function(event, data) {

// process the data.

});

**+**[**Top AngularJS Coding Tips**](http://www.techbeamers.com/angularjs-tutorial-beginners-essential-tips-tricks/)

**Q-47. How Do You Switch To Different Views From A Controller Function?**

**Answer.**

With the help of **<ui-sref>** directive or using the **$state.go()** function, we can switch between different views from the controllers.

Here is a working example to bring you more clarity on using the above features.

**AngularJS Code.**

var myapp = angular.module('myapp', ['ui.router']);

//Setup a basic AngularJS app.

myapp.config(['$stateProvider', function ($stateProvider) {

$stateProvider.state('python', {

url: '/python',

template: '<div style="background: blue">{{pythonCtrl.title}}</div>',

controller: 'TestFirstCtrl',

controllerAs: 'pythonCtrl'

});

$stateProvider.state('java', {

url: '/java',

template: '<div style="background: green">{{javaCtrl.title}}</div>',

controller: 'TestSecondCtrl',

controllerAs: 'javaCtrl'

});

}]);

//These two controllers will control the same view with different routes.

myapp.controller('TestFirstCtrl', ['$scope', function($scope) {

this.title = 'Learn Python';

}]);

myapp.controller('TestSecondCtrl', ['$scope', function($scope) {

this.title = 'Learn Java';

}]);

**HTML Page.**

It is the HTML code to test the Angular code given above.

<html>

<head>

<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<script src="//angular-ui.github.io/ui-router/release/angular-ui-router.js"></script>

</head>

<body ng-app="myapp">

<input type="button" ui-sref="python">Go to Learn Python</button>

<input type="button" ui-sref="java">Go to Learn Java</button>

<div ui-view></div>

</body>

</html>

**Q-48. What Would You Do To Limit A Scope Variable To Have One-Time Binding?**

**Answer.**

By prefixing the **“::”** operator to the scope variable. It’ll make sure the candidate is aware of the available variable bindings in AngularJS.

**Q-49. What Is The Difference Between One-Way Binding And Two-Way Binding?**

**Answer.**

The main difference between one-way binding and two-way binding is as follows.

* In one-way binding, the scope variable in the HTML gets initialized with the first value its model specifies.
* In two-way binding, the scope variable will change its value whenever the model gets a different value.

**Q-50. Which Angular Directive Would You Use To Hide An Element From The DOM Without Modifying Its Style?**

**Answer.**

It is the conditional ngIf Directive which we can apply to an element. Whenever the condition becomes false, the ngIf Directive removes it from the DOM.