



# AngularJS

## Interview Questions & Answers

- [www.dotnet-tricks.com](http://www.dotnet-tricks.com)
- [www.webgeekschool.com](http://www.webgeekschool.com)

by Shailendra Chauhan

# AngularJS Interview Questions and Answers

All rights reserved. No part of this book can be reproduced or stored in any retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, uploading on server and scanning without the prior written permission of the author.

The author of this book has tried his best to ensure the accuracy of the information described in this book. However, the author cannot guarantee the accuracy of the information contained in this book and the book has been written by referring AngularJS official documents. The author will not be liable for any damages, incidental or consequential caused directly or indirectly by this book.

Further, readers should be aware that the websites or reference links listed in this book may have changed or disappeared between when this book was written and when it is read.

All other trademarks referred to in this book are the property of their respective owners.

# Dedication

*My mother Mrs. Vriksha Devi and my wife Reshu Chauhan deserve to have their name on the cover as much as I do for all their support made this possible. I would like to say thanks to all my family members Virendra Singh(father), Jaishree and Jyoti(sisters), Saksham and Pranay(sons), friends, to you and to readers or followers of my blog [www.dotnet-tricks.com](http://www.dotnet-tricks.com) to encourage me to write this book.*

**-Shailendra Chauhan**

# Introduction

Writing a book has never been an easy task. It takes a great effort, patience and consistency with strong determination to complete it. Also, one should have a depth knowledge over the subject is going to write.

**So, what where my qualification to write this book?** My qualification and inspiration come from my enthusiasm for and the experience with the technology and from my analytic and initiative nature. Being a trainer, analyst, consultant and blogger, I have through knowledge and understandings of web technologies like HTML5, AngularJS, Backbone, Knockout.js, jQuery, jQueryUI jQuery Mobile and Ionic. My inspiration and knowledge has also come from many years of my working experience and research over it.

**So, the next question is who this book is for?** AngularJS Interview Questions and Answers book has been written with the intent of preparing yourself with solid foundations on AngularJS – in a short time without wasting your time in googling. As an author, I have tried to present each topic with the help of appropriate questions with answers which have suitable examples.

This book will help you to prepare yourself for an interview on AngularJS as well as to learn core Angular concepts - from how Angular works under the hood to advance concepts.

**I hope you will** enjoy this book and find it useful. At the same time I also encourage you to become a continue reader of the blog [www.dotnet-tricks.com](http://www.dotnet-tricks.com) and be the part of the discussion. But most importantly practice a lot and enjoy the technology. That's what it's all about.

To get the latest information on AngularJS, I encourage you to follow the official AngularJS website at [www.angularjs.org](http://www.angularjs.org). I also encourage you to subscribe to my blog at [www.dotnet-tricks.com](http://www.dotnet-tricks.com) that contains AngularJS, jQuery Mobile, .NET, C#, ASP.NET MVC, EF and many more tips, tricks and tutorials.

**As the author**, I am absolutely delighted about this. It's been a work of love and I want it to reach as many techy people as possible.

**All the best for your interview and happy programming!**

# About the Author

## Shailendra Chauhan



Works as a trainer & analyst and have more than 6 years of hand over Microsoft .NET technologies and other web technologies. He is a .NET Consultant, Author, Blogger, founder and chief editor of [www.dotnet-tricks.com](http://www.dotnet-tricks.com) and [www.webgeekschool.com](http://www.webgeekschool.com).

A number of articles of him has become articles-of-the-day, selected in daily-community-spotlight, and listed in [Recommended Resources for MVC](#) section in The Official Microsoft ASP.NET Site. His blog [www.dotnet-tricks.com](http://www.dotnet-tricks.com) is a well-known knowledge and support resource in the field of .NET technologies worldwide and is listed as a [non-Microsoft resource](#) in The Microsoft Official

He likes to share his working experience, research and knowledge through his well-known blogs. He is also the author of books [ASP.NET MVC Interview Questions and Answers](#) and [LINQ Interview Questions and Answers](#). He is a technical reviewer of book [ASP.NET MVC 4 Mobile App Development](#).

He loves to work with web applications, Mobile apps and Mobile websites using Microsoft technology including C#, ASP.NET, ASP.NET MVC, SQL Server, WCF, WEB API, LINQ, Entity Framework, jQuery, jQuery UI, AngularJS, jQuery Mobile, Knockout.js, Windows Azure, Backbone.js, PhoneGap and many more web technologies.

He strives to be the best he can be. He always keeps up with new technologies and learning new skills that allow him to provide better solutions to problems.

# About the Reviewer

## Kanishk Puri



Works as a Sr. Software Developer and have around 5 years of experience in .Net Technology using ASP. Net MVC, LINQ, Entity Framework, Web API, jQuery, SQL Server, AngularJS and mobile development.

Expert to develop mobile hybrid apps and N-tier web application using latest platform like PhoneGap, ASP.NET MVC, Entity Framework, Web API, PhoneGap, AngularJS, jQuery Mobile, Backbone and Ionic.

He always keep up with latest technologies and learning new skills to develop a better web app.

# How to Contact the Author

Although the author of this book has tried to make this book as accurate as it possible but if there is something strikes you as odd, or you find an error in the book please drop a line via e-mail.

The author e-mail addresses are listed as follows:

- [pro.shailendra@dotnet-tricks.com](mailto:pro.shailendra@dotnet-tricks.com)
- [pro.dotnettricks@gmail.com](mailto:pro.dotnettricks@gmail.com)

I am always happy to hear from my readers. Please provide with your valuable feedback and comments!

You can also follow [www.dotnet-tricks.com](http://www.dotnet-tricks.com) on [facebook](#), [twitter](#), [linkedin](#), [google plus](#) or subscribe to [RSS feed](#).

## Table of Contents

<b>AngularJS Interview Questions and Answers .....</b>	<b>1</b>
Dedication .....	2
Introduction .....	3
About the Author .....	4
About the Reviewer.....	5
How to Contact the Author .....	6
<b>AngularJS .....</b>	<b>11</b>
Q1. What is AngularJS? .....	11
Q2. Why to use AngularJS? .....	11
Q3. Why this project is called "AngularJS"? .....	12
Q4. What are the advantages of AngularJS?.....	12
Q5. How AngularJS is different from other JavaScript Framework?.....	12
Q6. What IDEs you can use for AngularJS development?.....	12
Q7. Does AngularJS has dependency on jQuery? .....	12
Q8. How to use jQuery with AngularJS? .....	13
Q9. Compare the features of AngularJS and jQuery? .....	13
Q10. What is jQLite or jQuery lite? .....	13
Q11. How to access jQLite or jQuery with AngularJS?.....	14
Q12. Is AngularJS a library, framework, plugin or a browser extension? .....	15
Q13. What browsers AngularJS support? .....	15
Q14. What is the size of angular.js file? .....	16
Q15. What are AngularJS features? .....	16
Q16. How AngularJS handle the security? .....	16
Q17. What are Modules in AngularJS?.....	16
Q18. What components can be defined within AngularJS modules?.....	17
Q19. What is core module in AngularJS? .....	17
Q20. How angular modules load the dependencies? .....	17
Q21. What is difference between config() and run() method in AngularJS? .....	18



Q22.	When dependent modules of a module are loaded? .....	18
Q23.	What is Global API? .....	18
Q24.	What is Angular Prefixes \$ and \$\$? .....	19
Q25.	What are Filters in AngularJS? .....	19
Q26.	What are Expressions in AngularJS? .....	19
Q27.	How AngularJS expressions are different from the JavaScript expressions? .....	20
Q28.	What are Directives in AngularJS? .....	20
Q29.	What is the role of ng-app, ng-init and ng-model directives? .....	20
Q30.	How to create custom directives in AngularJS? .....	20
Q31.	What are different ways to invoke a directive? .....	21
Q32.	What is restrict option in directive? .....	21
Q33.	Can you define multiple restrict options on a directive? .....	21
Q34.	What is auto bootstrap process in AngularJS? .....	21
Q35.	What is manual bootstrap process in AngularJS? .....	23
Q36.	How to bootstrap your angular app for multiple modules? .....	23
Q37.	What is scope in AngularJS? .....	25
Q38.	What is \$scope and \$rootScope? .....	25
Q39.	What is scope hierarchy? .....	27
Q40.	What is the difference between \$scope and scope? .....	29
Q41.	How AngularJS is compiled? .....	29
Q42.	How AngularJS compilation is different from other JavaScript frameworks? .....	30
Q43.	How Directives are compiled? .....	30
Q44.	What are Compile, Pre, and Post linking in AngularJS? .....	31
Q45.	What directives are used to show and hide HTML elements in AngularJS? .....	32
Q46.	Explain directives ng-if, ng-switch and ng-repeat? .....	33
Q47.	What are ng-repeat special variables? .....	34
Q48.	What are Templates in AngularJS? .....	35
Q49.	What is ng-include and when to use it? .....	35
Q50.	What angular components can be defined within AngularJS templates? .....	35
Q51.	What is data binding in AngularJS? .....	36
Q52.	Explain Two-way and One-way data binding in AngularJS? .....	36
Q53.	What is issue with two-way data binding? .....	37

Q54.	How AngularJS handle data binding? .....	37
Q55.	What is the difference between \$watch, \$digest and \$apply?.....	37
Q56.	Which one is fast between \$digest and \$apply? .....	39
Q57.	Which one handles exception automatically between \$digest and \$apply? .....	40
Q58.	Explain \$watch(), \$watchgroup() and \$watchCollection() functions of scope?.....	40
Q59.	Explain AngularJS scope life-cycle? .....	41
Q60.	Explain digest life-cycle in AngularJS? .....	42
Q61.	When to use \$destroy() function of scope? .....	43
Q62.	What is difference between \$evalAsync and \$timeout? .....	43
Q63.	What is the difference between \$watch and \$observe? .....	43
Q64.	What is the difference between \$parse and \$eval?.....	43
Q65.	What is Isolate Scope and why it is required?.....	44
Q66.	Does AngularJS support MVC? .....	45
Q67.	What is Model in AngularJS? .....	45
Q68.	What is ViewModel in AngularJS? .....	45
Q69.	What is Controller in AngularJS? .....	45
Q70.	How to share information between controllers in AngularJS? .....	46
Q71.	What is \$emit, \$broadcast and \$on in AngularJS? .....	46
Q72.	What is View in AngularJS? .....	49
Q73.	How to apply validation in AngularJS? .....	49
Q74.	How to do custom form validation in AngularJS? .....	50
Q75.	What are different Angular form properties? .....	51
Q76.	What are different states of a form in AngularJS? .....	52
Q77.	What is Service in AngularJS? .....	52
Q78.	What are different ways to create service in AngularJS? .....	52
Q79.	What is the difference between Factory, Service and Provider? .....	52
Q80.	What is difference between value and constant? .....	55
Q81.	What is the difference between \$http and \$resource? .....	56
Q82.	What methods \$http service support? .....	56
Q83.	How to enable caching in \$http service? .....	56
Q84.	What methods \$resource service object support? .....	56
Q85.	What is \$q service and when to use it? .....	57

Q86.	What is the difference between Kris Kowal's Q and \$q? .....	57
Q87.	What is Restangular? .....	57
Q88.	What are the advantages of Restangular over \$resource and \$http? .....	57
Q89.	What is difference between \$window and window in AngularJS? .....	58
Q90.	What is difference between \$document and window.document in AngularJS? .....	58
Q91.	What is difference between \$timeout and window.setTimeout in AngularJS? .....	59
Q92.	What is difference between \$interval and window. setInterval in AngularJS? .....	59
Q93.	What is Routing in AngularJS? .....	59
Q94.	What is AngularUI router and how it is different from ngRoute? .....	60
Q95.	What is \$injector and \$inject? .....	60
Q96.	What is Dependency Injection in AngularJS? .....	61
Q97.	How to do Language Internationalization in AngularJS? .....	61
Q98.	What is i18n and L10n? .....	62
Q99.	What is \$locale service? .....	62
Q100.	What is a locale ID? .....	62
Q101.	How to manage cookie in AngularJS? .....	63
Q102.	What is difference between \$cookies and \$cookieStore service? .....	63
Q103.	How to handle mobile browsers/devices events in AngularJS? .....	63
Q104.	How to detect swipe event in mobile browsers/devices in AngularJS? .....	64
Q105.	How to do animation with the help of AngularJS? .....	64
Q106.	What directives support animations? .....	64
Q107.	How to debug AngularJS app in browser? .....	64
Q108.	How to securely parse and manipulate your HTML data in AngularJS? .....	65
Q109.	What is Angular 2.0? .....	65
Q110.	What is AtScript? .....	65
Others Free Interview Books .....		66

# AngularJS

## Q1. What is AngularJS?

**Ans.** AngularJS is an open-source JavaScript framework developed by Google. It helps you to create single-page applications or one-page web applications that only require HTML, CSS, and JavaScript on the client side. It is based on MV-\* pattern and allow you to build well structured, easily testable, and maintainable front-end applications.



AngularJS has changed the way to web development. It is not based on jQuery to perform its operations. In spite of using ASP.NET Web form, ASP.NET MVC, PHP, JSP, Ruby on Rails for web development, you can do your complete web development by using most powerful and adaptive JavaScript Framework AngularJS. There is no doubt, JavaScript frameworks like AngularJS, Ember etc. are the future of web development.

## Q2. Why to use AngularJS?

**Ans.** There are following reasons to choose AngularJS as a web development framework:

1. It is based on MVC pattern which helps you to organize your web apps or web application properly.
2. It extends HTML by attaching directives to your HTML markup with new attributes or tags and expressions in order to define very powerful templates.
3. It also allows you to create your own directives, making reusable components that fill your needs and abstract your DOM manipulation logic.
4. It supports two-way data binding i.e. connects your HTML (views) to your JavaScript objects (models) seamlessly. In this way any change in model will update the view and vice versa without any DOM manipulation or event handling.
5. It encapsulates the behaviour of your application in controllers which are instantiated with the help of dependency injection.
6. It supports services that can be injected into your controllers to use some utility code to fulfil your need.  
**For example**, it provides \$http service to communicate with REST service.
7. It supports dependency injection which helps you to test your angular app code very easily.
8. Also, AngularJS is mature community to help you. It has widely support over the internet.

### Q3. Why this project is called "AngularJS"?

**Ans.** Html has angle brackets i.e. <,> and *ng* sound like Angular. That's why it is called AngularJS.

### Q4. What are the advantages of AngularJS?

**Ans.** There are following advantages of AngularJS:

- **Data Binding** - AngularJS provides a powerful data binding mechanism to bind data to HTML elements by using scope.
- **Customize & Extensible** - AngularJS is customized and extensible as per you requirement. You can create your own custom components like directives, services etc.
- **Code Reusability** - AngularJS allows you to write code which can be reused. **For example** custom directive which you can reuse.
- **Support** – AngularJS is mature community to help you. It has widely support over the internet. Also, AngularJS is supported by Google which gives it an advantage.
- **Compatibility** - AngularJS is based on JavaScript which makes it easier to integrate with any other JavaScript library and runnable on browsers like IE, Opera, FF, Safari, Chrome etc.
- **Testing** - AngularJS is designed to be testable so that you can test your AngularJS app components as easy as possible. It has dependency injection at its core, which makes it easy to test.

### Q5. How AngularJS is different from other JavaScript Framework?

**Ans.** Today, AngularJS is the most popular and dominant JavaScript framework for professional web development. It is well suited for small, large and any sized web app and web application.

AngularJS is different from other JavaScript framework in following ways:

1. AngularJS mark-up lives in the DOM.
2. AngularJS uses plain old JavaScript objects (POJO).
3. AngularJS is leverages with Dependency Injection.

### Q6. What IDEs you can use for AngularJS development?

**Ans.** AngularJS development can be done with the help of following IDEs:

1. Visual Studio 2012, 2013, 2015 or higher
2. Eclipse
3. WebStorm
4. Sublime Text
5. TextMate

### Q7. Does AngularJS has dependency on jQuery?

**Ans.** AngularJS has no dependency on jQuery library. But it can be used with jQuery library.

**Q8. How to use jQuery with AngularJS?**

**Ans.** By default AngularJS use **jQLite** which is the subset of jQuery. If you want to use jQuery then simply load the jQuery library before loading the AngularJS. By doing so, Angular will skip jQLite and will started to use jQuery library.

**Q9. Compare the features of AngularJS and jQuery?**

**Ans.** The comparison of AngularJS and jQuery features are given below:

Features	jQuery	AngularJS
Abstract The DOM	Y	Y
Animation Support	Y	Y
AJAX/JSONP	Y	Y
Cross Module Communication	Y	Y
Deferred Promises	Y	Y
Form Validation	N	Y
Integration Test Runner	N	Y
Unit Test Runner	Y	Y
Localization	N	Y
MVC Pattern	N	Y
Template	N	Y
Two-way Binding	N	Y
One-way Binding	N	Y
Dependency Injection	N	Y
Routing	N	Y
Restful API	N	Y

**Q10. What is jQLite or jQuery lite?**

**Ans.** jQLite is a **subset** of jQuery that is built directly into AngularJS. jQLite provides you all the useful features of jQuery. In fact it provides you limited features or functions of jQuery.

Here is a table of supported jQuery methods by jQLite.

jQuery Method	Limitation, if any
addClass()	
after()	
append()	
attr()	

bind()	Does not support namespace, selectors and eventData
children	Does not support selectors
clone()	
contents()	
css()	
data()	
detach()	
empty()	
eq()	
find()	Limited to lookups by tag name
hasClass()	
html()	
text()	Does not support selectors
on()	Does not support namespace, selectors and eventData
off()	Does not support namespace, selectors
one()	Does not support namespace, selectors
parent()	Does not support selectors
prepend()	
prop	
ready()	
remove	
removeAttr()	
removeClass()	
removeData()	
replaceWith()	
toggleClass()	
triggerHandler()	Passes a dummy event object to handlers
unbind()	Does not support namespace
val()	
wrap()	

### Q11. How to access jQLite or jQuery with AngularJS?

OR

#### What is angular.element() in AngularJS?

**Ans.** jQuery lite or the full jQuery library if available, can be accessed via the AngularJS code by using the *element()* function in AngularJS. Basically, *angular.element()* is an alias for the jQuery function i.e.

```
angular.element() === jQuery() === $()
```

**For Example**

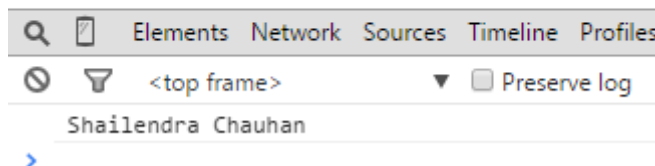
```

<html>
<head>
  <script src="lib/angular.js"></script>
  <script type="text/javascript">
    var app = angular.module('app', []);
    app.controller("mainCtrl", function ($scope, $element) {
      $scope.clickme = function () {
        var elem = angular.element(document.querySelector('#txtName'));
        console.log(elem.val())// console the value of textbox
      };
    });
  </script>
</head>
<body ng-app="app">
  <div ng-controller="mainCtrl">
    <input type="text" id="txtName" value="Shailendra Chauhan" />
    <button type="button" ng-click="clickme()">Click me</button>
  </div>
</body>
</html>

```

When you will click on the *Click me* button, the value of textbox will be logged on console as given below:

Shailendra Chauhan

**Q12. Is AngularJS a library, framework, plugin or a browser extension?**

**Ans.** AngularJS is a first class JavaScript framework which allows you to build well structured, easily testable, and maintainable front-end applications. It is not a library since library provides you limited functionality or has dependencies to other libraries.

It is not a plugin or browser extension since it is based on JavaScript and compatible with both desktop and mobile browsers.

**Q13. What browsers AngularJS support?**

**Ans.** The latest version of **AngularJS 1.3** support Safari, Chrome, Firefox, Opera 15+, IE9+ and mobile browsers (Android, Chrome Mobile, iOS Safari, Opera Mobile).

AngularJS 1.3 has dropped support for IE8 but AngularJS 1.2 will continue to support IE8.



**Q14. What is the size of angular.js file?**

**Ans.** The size of the compressed and minified file is < 36KB.

**Q15. What are AngularJS features?**

**Ans.** The features of AngularJS are listed below:

1. Modules
2. Directives
3. Templates
4. Scope
5. Expressions
6. Data Binding
7. MVC (Model, View & Controller)
8. Validations
9. Filters
10. Services
11. Routing
12. Dependency Injection
13. Testing

**Q16. How AngularJS handle the security?**

**Ans.** AngularJS provide following built-in protection from basic security holes:

1. Prevent HTML injection attacks.
2. Prevent Cross-Site-Scripting (CSS) attacks.
3. Prevent XSRF protection for server side communication.

Also, AngularJS is designed to be compatible with other security measures like Content Security Policy (CSP), HTTPS (SSL/TLS) and server-side authentication and authorization that greatly reduce the possible attacks.

**Q17. What are Modules in AngularJS?**

**Ans.** AngularJS modules are containers just like namespace in C#. They divide an angular app into small, reusable and functional components which can be integrated with other angular app. Each module is identified by a unique name and can be dependent on other modules. In AngularJS, every web page (view) can have a single module assigned to it via *ng-app* directive.

**Creating an AngularJS module**

```
<script type="text/javascript">
  // defining module
  angular.module('myApp', []);

  //OR defining module which has dependency on other modules
  angular.module('myApp', ['dependentModule1', 'dependentModule2']);
</script>
```

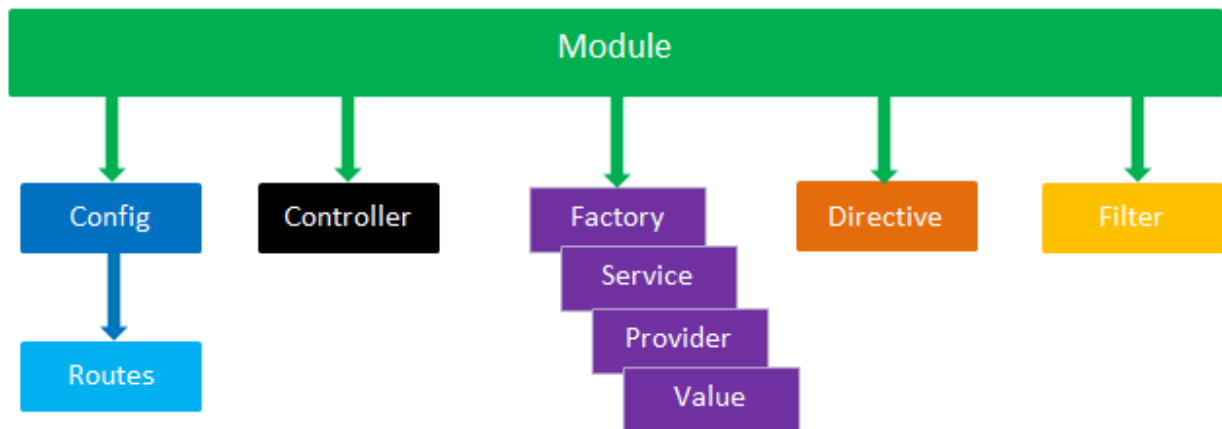
### Using an AngularJS module into your app

You can bootstrap your app by using your AngularJS module as given below:

```
<html ng-app="myApp">
<head>
  ...
</head>
<body>
  ...
</body>
```

### Q18. What components can be defined within AngularJS modules?

Ans. You can define following components with in your angular module:



1. Directive
2. Filter
3. Controller
4. Factory
5. Service
6. Provider
7. Value
8. Config settings and Routes

### Q19. What is core module in AngularJS?

Ans. **ng** is the core module in angular. This module is loaded by default when an angular app is started. This module provides the essential components for your angular app like directives, services/factories, filters, global APIs and testing components.

### Q20. How angular modules load the dependencies?

Ans. An angular module use **configuration and run** blocks to inject dependencies (like providers, services and constants) which get applied to the angular app during the bootstrap process.

**Q21. What is difference between config() and run() method in AngularJS?**

**Ans. Configuration block** – This block is executed during the provider registration and configuration phase. Only providers and constants can be injected into configuration blocks. This block is used to inject module wise configuration settings to prevent accidental instantiation of services before they have been fully configured. This block is created using *config()* method.

```
angular.module('myModule', []).
  config(function (injectables) { // provider-injector
    // This is an example of config block.
    // You can have as many of these as you want.
    // You can only inject Providers (not instances)
    // into config blocks.
  }).
  run(function (injectables) { // instance-injector
    // This is an example of a run block.
    // You can have as many of these as you want.
    // You can only inject instances (not Providers)
    // into run blocks
  });
```

**Run block** – This block is executed after the configuration block. It is used to inject instances and constants. This block is created using *run()* method. This method is like as main method in C or C++.

The run block is a great place to put event handlers that need to be executed at the root level for the application.

**For example**, authentication handlers.

**Q22. When dependent modules of a module are loaded?**

**Ans.** A module might have dependencies on other modules. The dependent modules are loaded by angular before the requiring module is loaded.

In other words the configuration blocks of the dependent modules execute before the configuration blocks of the requiring module. The same is true for the run blocks. Each module can only be loaded once, even if multiple other modules require it.

**Q23. What is Global API?**

**Ans.** Global API provides you global functions to perform common JavaScript tasks such as comparing objects, deep copying, iterating through objects, and converting JSON data etc. All global functions can be accessed by using the *angular* object. The list of global functions is given below:

Name	Description
angular.lowercase	Converts the specified string to lowercase.
angular.uppercase	Converts the specified string to uppercase.
angular.forEach	Invokes the iterator function once for each item in obj collection, which can be either an object or an array.
angular.isUndefined	Determines if a reference is undefined.
angular.isDefined	Determines if a reference is defined.
angular.isObject	Determines if a reference is an Object.

angular.isString	Determines if a reference is a String.
angular.isNumber	Determines if a reference is a Number.
angular.isDate	Determines if a value is a date.
angular.isArray	Determines if a reference is an Array.
angular.isFunction	Determines if a reference is a Function.
angular.isElement	Determines if a reference is a DOM element (or wrapped jQuery element).
angular.copy	Creates a deep copy of source, which should be an object or an array.
angular.equals	Determines if two objects or two values are equivalent. Supports value types, regular expressions, arrays and objects.
angular.bind	Returns a function which calls function fn bound to self
angular.toJson	Serializes input into a JSON-formatted string. Properties with leading \$\$ characters will be stripped since angular uses this notation internally.
angular.fromJson	Deserializes a JSON string.
angular.bootstrap	Use this function to manually start up angular application.
angular.reloadWithDebugInfo	Use this function to reload the current application with debug information turned on.
angular.injector	Creates an injector object that can be used for retrieving services as well as for dependency injection
angular.element	Wraps a raw DOM element or HTML string as a jQuery element.
angular.module	Used for creating, registering and retrieving Angular modules.

## Q24. What is Angular Prefixes \$ and \$\$?

**Ans.** To prevent accidental name collisions with your code, Angular prefixes names of public objects with \$ and names of private objects with \$\$\$. So, do not use the \$ or \$\$\$ prefix in your code.

## Q25. What are Filters in AngularJS?

**Ans.** Filters are used to format data before displaying it to the user. They can be used in view templates, controllers, services and directives. There are some built-in filters provided by AngularJS like as Currency, Date, Number, OrderBy, Lowercase, Uppercase etc. You can also create your own filters.

### Filter Syntax

```
{{ expression | filter }}
```

### Filter Example

```
<script type="text/javascript">
  { { 14 | currency } } //returns $14.00
</script>
```

## Q26. What are Expressions in AngularJS?

**Ans.** AngularJS expressions are much like JavaScript expressions, placed inside HTML templates by using double braces such as: `{{expression}}`. AngularJS evaluates expressions and then dynamically adds the result to a web page. Like JavaScript expressions, they can contain literals, operators, and variables.

There are some valid AngularJS expressions:

- {{ 1 + 2 }}
- {{ x + y }}
- {{ x == y }}
- {{ x = 2 }}
- {{ user.Id }}

### Q27. How AngularJS expressions are different from the JavaScript expressions?

**Ans.** AngularJS expressions are much like JavaScript expressions but they are different from JavaScript expressions in the following ways:

1. Angular expressions can be added inside the HTML templates.
2. Angular expressions doesn't support control flow statements (conditionals, loops, or exceptions).
3. Angular expressions support filters to format data before displaying it.

### Q28. What are Directives in AngularJS?

**Ans.** AngularJS directives are a combination of AngularJS template markups (HTML attributes or elements, or CSS classes) and supporting JavaScript code. The JavaScript directive code defines the template data and behaviors of the HTML elements.

AngularJS directives are used to extend the HTML vocabulary i.e. they decorate html elements with new behaviors and help to manipulate html elements attributes in interesting way.

There are some built-in directives provided by AngularJS like as *ng-app*, *ng-controller*, *ng-repeat*, *ng-model* etc.

### Q29. What is the role of ng-app, ng-init and ng-model directives?

**Ans.** The main role of these directives is explained as:

- **ng-app** - Initialize the angular app.
- **ng-init** - Initialize the angular app data.
- **ng-model** - Bind the html elements like input, select, text area to angular app model.

### Q30. How to create custom directives in AngularJS?

**Ans.** You can create your own custom directive by using following syntax:

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

//creating custom directive syntax
app.directive("myDir", function () {
    return {
        restrict: "E", //define directive type like E = element, A = attribute, C =
class, M = comment
        scope: { //create a new child scope or an isolate scope

            title: '@' //@ reads the attribute value,
            // = provides two-way binding,
            // & works with functions
        }
    };
});
```

```

    },
    template: "<div>{{ myName }}</div>", // define HTML markup
    templateUrl: 'mytemplate.html', //path to the template, used by the directive
    replace: true | false, // replace original markup with template yes/no
    transclude: true | false, // copy original HTML content yes/no
    controller: function (scope) { //define controller, associated with the directive
template
        //TODO:
    },
    link: function (scope, element, attrs, controller) { //define function, used for
DOM manipulation
        //TODO:
    }
}
});

```

### Q31. What are different ways to invoke a directive?

**Ans.** There are four methods to invoke a directive in your angular app which are equivalent.

Method	Syntax
As an <b>attribute</b>	<code>&lt;span my-directive&gt;&lt;/span&gt;</code>
As a <b>class</b>	<code>&lt;span class="my-directive: expression;"&gt;&lt;/span&gt;</code>
As an <b>element</b>	<code>&lt;my-directive&gt;&lt;/my-directive&gt;</code>
As a <b>comment</b>	<code>&lt;!-- directive: my-directive expression --&gt;</code>

### Q32. What is restrict option in directive?

**Ans.** The restrict option in angular directive, is used to specify how a directive will be invoked in your angular app i.e. as an attribute, class, element or comment.

There are four valid options for restrict:

```

'A' (Attribute)- <span my-directive></span>
'C' (Class)- <span class="my-directive:expression;"></span>
'E' (Element)- <my-directive></my-directive>
'M' (Comment)- <!-- directive: my-directive expression -->

```

### Q33. Can you define multiple restrict options on a directive?

**Ans.** You can also specify multiple restrict options to support more than one methods of directive invocation as an element or an attribute. Make sure all are specified in the restrict keyword as:

```
restrict: 'EA'
```

### Q34. What is auto bootstrap process in AngularJS?

OR

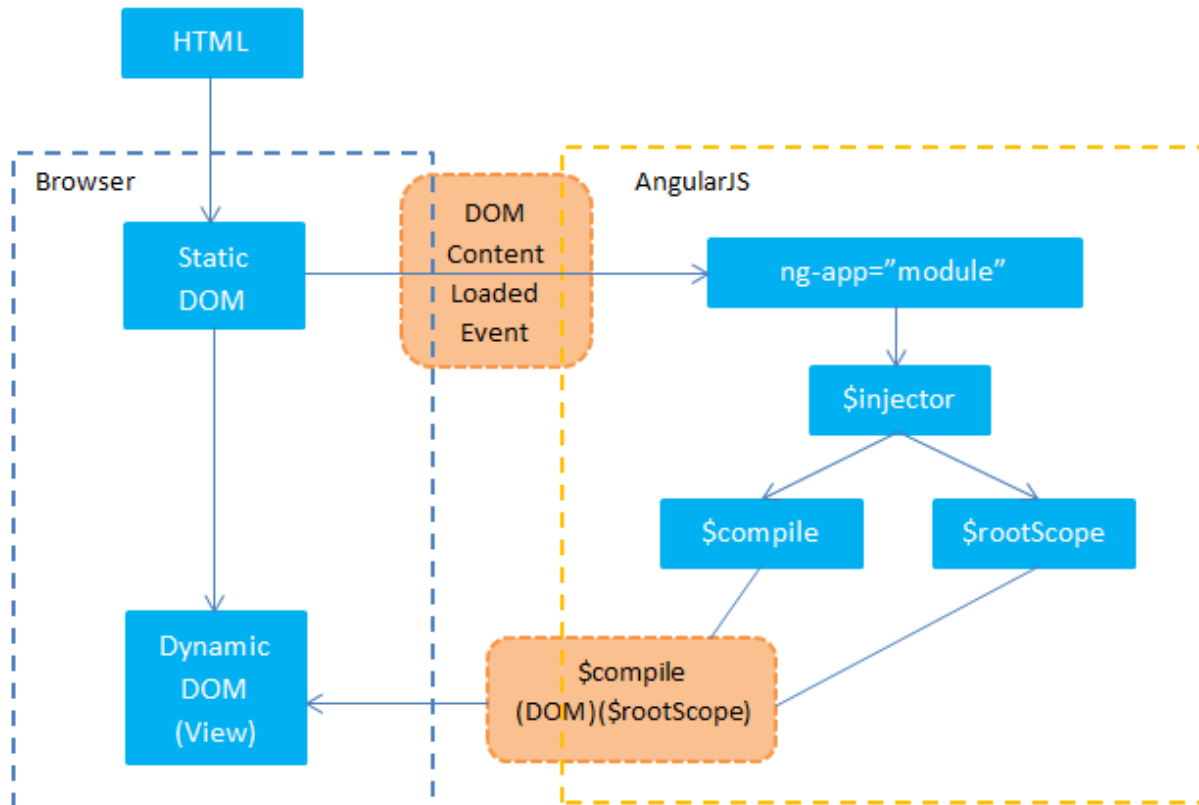
#### How AngularJS is initialized automatically?

**Ans.** Angular initializes automatically upon **DOMContentLoaded** event or when the angular.js script is downloaded to the browser and the *document.readyState* is set to **complete**. At this point AngularJS looks for the

**ng-app** directive which is the root of angular app compilation and tells about AngularJS part within DOM. When the **ng-app** directive is found then Angular will:

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

This process is called auto-bootstrapping.



### Example

```

<html>
<body ng-app="myApp">
  <div ng-controller="Ctrl">
    Hello {{msg}}!
  </div>

  <script src="lib/angular.js"></script>
  <script>
    var app = angular.module('myApp', []);
    app.controller('Ctrl', function ($scope) {
      $scope.msg = 'World';
    });
  </script>
</body>
</html>

```

**Q35. What is manual bootstrap process in AngularJS?****OR****How AngularJS is initialized manually?**

**Ans.** You can manually initialize your angular app by using *angular.bootstrap()* function. This function takes the modules as parameters and should be called within *angular.element(document).ready()* function. The *angular.element(document).ready()* function is fired when the DOM is ready for manipulation.

**Example**

```
<html>
<body>
  <div ng-controller="Ctrl">
    Hello {{msg}}!
  </div>
  <script src="lib/angular.js"></script>

  <script>
    var app = angular.module('myApp', []);
    app.controller('Ctrl', function ($scope) {
      $scope.msg = 'World';
    });

    //manual bootstrap process
    angular.element(document).ready(function () {
      angular.bootstrap(document, ['myApp']);
    });
  </script>
</body>
</html>
```

**Note**

- You should not use the **ng-app** directive when manually bootstrapping your app.
- You should not mix up the automatic and manual way of bootstrapping your app.
- Define modules, controller, services etc. before manually bootstrapping your app as defined in above example.

**Q36. How to bootstrap your angular app for multiple modules?**

**Ans.** AngularJS is automatically initialized for one module. But sometimes, it is required to bootstrap for multiple modules and it can be achieved by using two methods:

1. **Automatic bootstrap (by combining multiple modules into one module)** - You can combine multiple modules into single modules and your angular app will be automatically initialized for newly created module and other modules will act as dependent modules for newly created module.

**For example,** suppose you have two modules: module1 and model2, and you have to initialize your app automatically based on these two modules then you achieve this following way:



```

<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 = "Shailendra Chauhan";
    });

    //module2
    var app2 = angular.module("module2", []);
    app2.controller("Controller2", function ($scope) {
      $scope.name = "Deepak Chauhan";
    });

    //module3 dependent on module1 & module2
    angular.module("app", ["module1", "module2"]);
  </script>
</head>
<body>
  <!--angularjs auto bootstrap process-->
  <div ng-app="app">
    <h1>Multiple modules bootstrap</h1>
    <div ng-controller="Controller2">
      {{name}}
    </div>
    <div ng-controller="Controller1">
      {{name}}
    </div>
  </div>
</body>
</html>

```

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

The above example can be rewritten as for 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 = "Shailendra Chauhan";
    });

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

```

```

app2.controller("Controller2", function ($scope) {
    $scope.name = "Deepak Chauhan";
});

//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']);

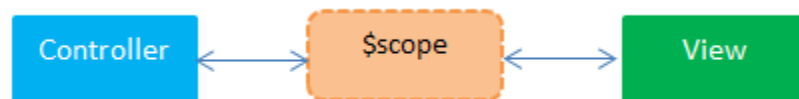
    //bootstrap div2 only for module1
    angular.bootstrap(div2, ['module1']);
});
</script>
</head>
<body>
    <!--angularjs manual bootstrap process-->
    <div id="div1">
        <h1>Multiple modules bootstrap</h1>
        <div ng-controller="Controller1">
            {{name}}
        </div>
        <div ng-controller="Controller2">
            {{name}}
        </div>
    </div>

    <div id="div2">
        <div ng-controller="Controller1">
            {{name}}
        </div>
    </div>
</body>
</html>

```

### Q37. What is scope in AngularJS?

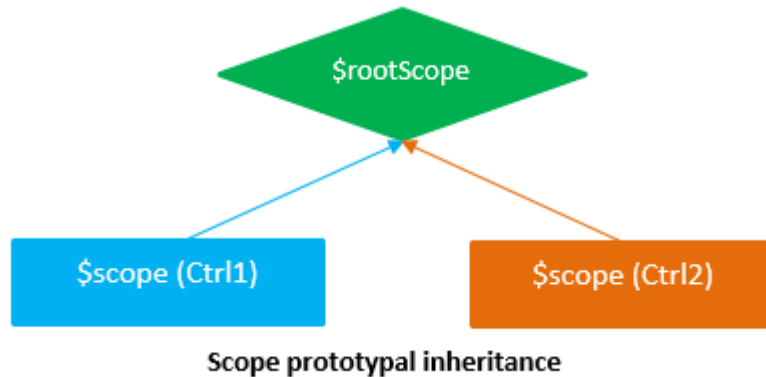
**Ans.** Scope is a JavaScript object that refers to the application model. It acts as a context for evaluating angular expressions. Basically, it acts as glue between controller and view.



Scopes are hierarchical in nature and follow the DOM structure of your AngularJS app. AngularJS has two scope objects: \$rootScope and \$scope.

### Q38. What is \$scope and \$rootScope?

**Ans.** **\$scope** - A \$scope is a JavaScript object which is used for communication between controller and view. Basically, \$scope binds a view (DOM element) to the model and functions defined in a controller.



**\$rootScope** - The \$rootScope is the top-most scope. An app can have only one \$rootScope which will be shared among all the components of an app. Hence it acts like a global variable. All other \$scopes are children of the \$rootScope.

**For example**, suppose you have two controllers: Ctrl1 and Ctrl2 as given below:

```

<!doctype html>
<html>
<body ng-app="myApp">
  <div ng-controller="Ctrl1" style="border:2px solid blue; padding:5px">
    Hello {{msg}}!
    <br />
    Hello {{name}}! (rootScope)
  </div>
  <br />
  <div ng-controller="Ctrl2" style="border:2px solid green; padding:5px">
    Hello {{msg}}!
    <br />
    Hey {{myName}}!
    <br />
    Hi {{name}}! (rootScope)
  </div>

  <script src="lib/angular.js"></script>
  <script>
    var app = angular.module('myApp', []);
    app.controller('Ctrl1', function ($scope, $rootScope) {
      $scope.msg = 'World';
      $rootScope.name = 'AngularJS';
    });

    app.controller('Ctrl2', function ($scope, $rootScope) {
      $scope.msg = 'Dot Net Tricks';
      $scope.myName = $rootScope.name;
    });
  </script>
</body>
</html>
  
```

### Output

Hello World!  
Hello AngularJS! (rootScope)

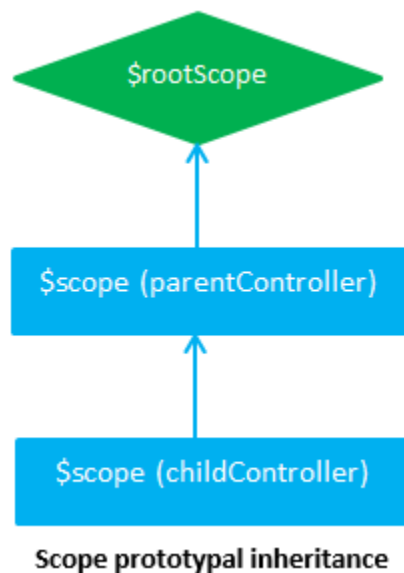
Hello Dot Net Tricks!  
Hey AngularJS!  
Hi AngularJS! (rootScope)

**Q39. What is scope hierarchy?**

**OR**

**What is scope inheritance?**

**Ans.** The \$scope object used by views in AngularJS are organized into a hierarchy. There is a root scope, and the \$rootScope can has one or more child scopes. Each controller has its own \$scope (which is a child of the \$rootScope), so whatever variables you create on \$scope within controller, these variables are accessible by the view based on this controller.



**For example,** suppose you have two controllers: ParentController and ChildController as given below:

```
<html>
<head>
  <script src="lib/angular.js"></script>
  <script>
    var app = angular.module('ScopeChain', []);
    app.controller("parentController", function ($scope) {
      $scope.managerName = 'Shailendra Chauhan';
      $scope.$parent.companyName = 'Dot Net Tricks'; //attached to $rootScope
    });
  </script>
</head>
</html>
```

```

    });
    app.controller("childController", function ($scope, $controller) {
        $scope.teamLeadName = 'Deepak Chauhan';
    });
</script>
</head>
<body ng-app="ScopeChain">
    <div ng-controller="parentController">
        <table style="border:2px solid #e37112">
            <caption>Parent Controller</caption>
            <tr>
                <td>Manager Name</td>
                <td>{{managerName}}</td>
            </tr>
            <tr>
                <td>Company Name</td>
                <td>{{companyName}}</td>
            </tr>
            <tr>
                <td>
                    <table ng-controller="childController" style="border:2px solid #428bca">
                        <caption>Child Controller</caption>
                        <tr>
                            <td>Team Lead Name</td>
                            <td>{{ teamLeadName }}</td>
                        </tr>
                        <tr>
                            <td>Reporting To</td>
                            <td>{{managerName}}</td>
                        </tr>
                        <tr>
                            <td>Company Name</td>
                            <td>{{companyName}}</td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </div>
</body>
</html>

```

**Output**

Parent Controller	
Manager Name	Shailendra Chauhan
Company Name	Dot Net Tricks
Child Controller	
Team Lead Name	Deepak Chauhan
Reporting To	Shailendra Chauhan
Company Name	Dot Net Tricks

**Q40. What is the difference between \$scope and scope?**

**Ans.** The module factory methods *like controller, directive, factory, filter, service, animation, config and run* receive arguments through dependency injection (DI). In case of DI, you inject the **scope object** with the dollar prefix i.e. **\$scope**. The reason is the **injected arguments** must match to the names of **injectable objects** followed by dollar (\$) prefix.

**For example**, you can inject the scope and element objects into a controller as given below:

```
module.controller('MyController', function ($scope, $element) { // injected arguments
});
```

When the methods like directive linker function don't receive arguments through dependency injection, you just pass the **scope object** without using dollar prefix i.e. **scope**. The reason is the passing arguments are received by its caller.

```
module.directive('myDirective', function () // injected arguments here
{
    return {
        // linker function does not use dependency injection
        link: function (scope, el, attrs) {
            // the calling function will pass the three arguments to the linker: scope,
            // element and attributes, in the same order
        }
    };
});
```

In the case of non-dependency injected arguments, you can give the name of injected objects as you wish. The above code can be re-written as:

```
module.directive("myDirective", function () {
    return {
        link: function (s, e, a) {
            // s == scope
            // e == element
            // a == attributes
        }
    };
});
```

In short, in case of DI the **scope object** is received as **\$scope** while in case of non-DI **scope object** is received as **scope** or with any name.

**Q41. How AngularJS is compiled?**

**Ans.** Angular's HTML compiler allows you to teach the browser new HTML syntax. The compiler allows you to attach new behaviors or attributes to any HTML element. Angular calls these behaviors as directives.

AngularJS compilation process takes place in the web browser; no server side or pre-compilation step is involved. Angular uses \$compiler service to compile your angular HTML page. The angular' compilation process begins after your HTML page (static DOM) is fully loaded. It happens in two phases:

1. **Compile** - It traverse the DOM and collect all of the directives. The result is a linking function.

2. **Link** - It combines the directives with a scope and produces a live view. Any changes in the scope model are reflected in the view, and any user interactions with the view are reflected in the scope model.

The concept of compile and link comes from C language, where you first compile the code and then link it to actually execute it. The process is very much similar in AngularJS as well.

## Q42. How AngularJS compilation is different from other JavaScript frameworks?

**Ans.** If you have worked on templates in other java script framework/library like backbone and jQuery, they process the template as a string and result as a string. You have to dump this result 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 your data.

## Q43. How Directives are compiled?

**Ans.** It is important to note that Angular operates on DOM nodes rather than strings. Usually, you don't notice this because when an html page loads, the web browser parses HTML into the DOM automatically.

HTML compilation happens in three phases:

1. The \$compile traverses the DOM and looks for directives. For each directive it finds, it adds it to a list of directives.
2. Once the entire DOM has been traversed, it will sort that list of directives by their priority. Then, each directive's own compile function is executed, giving each directive the chance to modify the DOM itself. Each compile function returns a linking function, which is then composed into a combined linking function and returned.
3. \$compile links the template with the scope by calling the combined linking function from the previous step. This in turn will call the linking function of the individual directives, registering listeners on the elements and setting up \$watch with the scope as each directive is configured to do.

The pseudo code for the above process is given below:

```
var $compile = ...; // injected into your code
var scope = ...;
var parent = ...; // DOM element where the compiled template can be appended

var html = '<div ng-bind="exp"></div>';

// Step 1: parse HTML into DOM element
var template = angular.element(html);

// Step 2: compile the template
var linkFn = $compile(template);

// Step 3: link the compiled template with the scope.
var element = linkFn(scope);

// Step 4: Append to DOM (optional)
parent.appendChild(element);
```





```

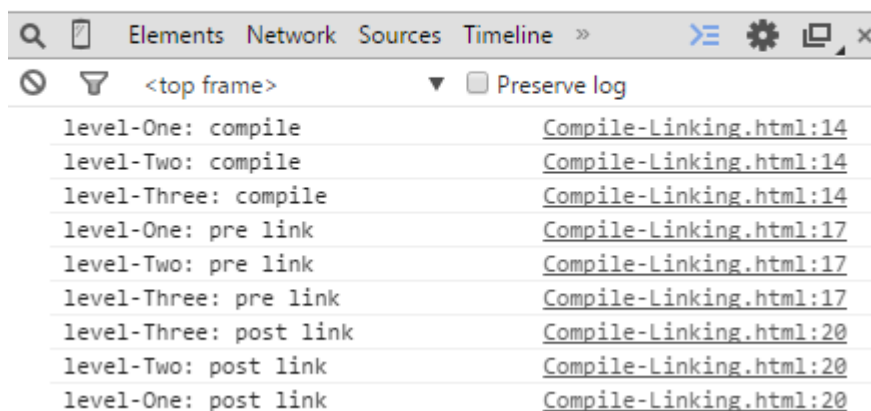
        post: function (scope, iElem, attrs) {
            console.log(name + ': post link');
        }
    }
}

app.directive('levelOne', createDirective('level-One'));
app.directive('levelTwo', createDirective('level-Two'));
app.directive('levelThree', createDirective('level-Three'));
</script>
</head>
<body ng-app="app">
    <level-one>
        <level-two>
            <level-three>
                Hello {{name}}
            </level-three>
        </level-two>
    </level-one>
</body>
</html>

```

**Output**

Hello



level-One: compile	Compile-Linking.html:14
level-Two: compile	Compile-Linking.html:14
level-Three: compile	Compile-Linking.html:14
level-One: pre link	Compile-Linking.html:17
level-Two: pre link	Compile-Linking.html:17
level-Three: pre link	Compile-Linking.html:17
level-Three: post link	Compile-Linking.html:20
level-Two: post link	Compile-Linking.html:20
level-One: post link	Compile-Linking.html:20

**Q45. What directives are used to show and hide HTML elements in AngularJS?**

**Ans.** **ng-show** and **ng-hide** directives are used to show and hide HTML elements in the AngularJS based on an expression. When the expression is true for ng-show or ng-hide, HTML element(s) are shown or hidden from the page. When the expression is false for ng-show or ng-hide, HTML element(s) are hidden or shown on the page.

```

<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>

```

#### Q46. Explain directives ng-if, ng-switch and ng-repeat?

**Ans. ng-if** – This directive can add / remove HTML elements from the DOM based on an expression. If the expression is true, it add HTML elements to DOM, otherwise HTML elements are removed 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>

```

**ng-switch** – This directive can add / remove HTML elements from the DOM conditionally based on scope expression.

```

<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>

```

**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 = ['Shailendra', 'Deepak', 'Kamal'];
});
</script>

```

## Q47. What are ng-repeat special variables?

**Ans.** The ng-repeat directive has a set of special variables which you are useful while iterating the collection. These variables are as follows:

- \$index
- \$first
- \$middle
- \$last

```

<html>
<head>
    <script src="lib/angular.js"></script>
    <script>
        var app = angular.module("app", []);

        app.controller("ctrl", function ($scope) {
            $scope.friends = [{ name: 'shailendra', gender: 'boy' },
                              { name: 'kiran', gender: 'girl' },
                              { name: 'deepak', gender: 'boy' },
                              { name: 'pooja', gender: 'girl' }
                              ];

        });
    </script>
</head>
<body ng-app="app">
    <div ng-controller="ctrl">
        <ul class="example-animate-container">

            <li ng-repeat="friend in friends">
                <div>
                    [{{$index + 1}}] {{friend.name}} is a {{friend.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>

```

### Output

- [1] shailendra is a boy. (first element found)
- [2] kiran is a girl. (middle element found)
- [3] deepak is a boy. (middle element found)
- [4] pooja is a girl. (last element found)

The \$index contains the index of the element being iterated. The \$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.

### **Q48. What are Templates in AngularJS?**

**Ans.** AngularJS templates are just plain old HTML that contains Angular-specific elements and attributes. AngularJS used these templates to show information from the model and controller.

#### Creating an AngularJS template

```
<html ng-app>
<!-- body tag with ngController directive -->
<body ng-controller="MyController">
  <input ng-model="txtName" value="shailendra"/>
  <!-- button tag with ng-click directive & string expression 'btnText' wrapped in "{{ }}"
markup -->
  <button ng-click="changeName()">{{btnText}}</button>
<script src="angular.js">
</body>
</html>
```

### **Q49. What is ng-include and when to use it?**

**Ans.** **ng-include** is a directive which is used to include external HTML fragments from other files into the view's HTML template.

**For example,** *index.html* file can be added inside the *div* element by using ng-include directive as an attribute.

```
<div ng-controller="MyCtrl">
  <div ng-include="'index.html'"></div>
</div>
```

ng-include directive is limited to load HTML fragments file from same domain but it doesn't work for cross-domain i.e. it can't load the HTML fragments file from different domains.

### **Q50. What angular components can be defined within AngularJS templates?**

**Ans.** AngularJS templates can have following angular elements and attributes:

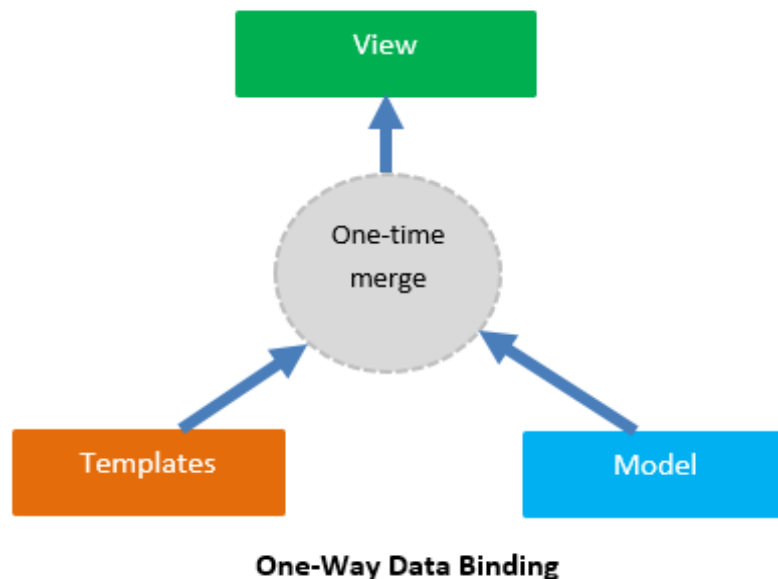
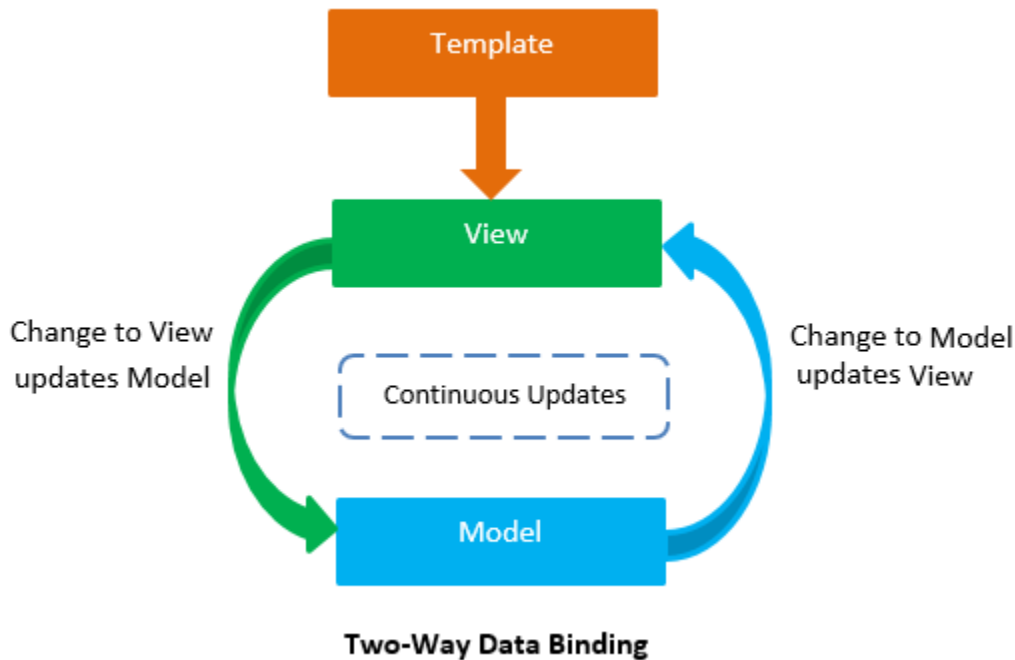
1. Directive
2. Angular Markup ('{{}}')
3. Filters
4. Form Controls

**Q51. What is data binding in AngularJS?**

**Ans.** AngularJS data-binding is the most useful feature which saves you from writing boilerplate code (i.e. *the sections of code which is included in many places with little or no alteration*). Now, developers are not responsible for manually manipulating the DOM elements and attributes to reflect model changes. AngularJS provides two-way data-binding to handle the synchronization of data between model and view.

**Q52. Explain Two-way and One-way data binding in AngularJS?**

**Ans.** **Two-way data binding** - It is used to synchronize the data between model and view. It means, any change in model will update the view and vice versa. **ng-model** directive is used for two-way data binding.



**One-way data binding** - This binding is introduced in **Angular 1.3**. An expression that starts with double colon (::), is considered a one-time expression i.e. one-way binding.

### Two-Way and One-Way data binding Example

```
<div ng-controller="MyCtrl">
  <label>Name (two-way binding): <input type="text" ng-model="name" /></label>
  <strong>Your name (one-way binding):</strong> {{::name}}<br />
  <strong>Your name (normal binding):</strong> {{name}}
</div>
<script>
  var app = angular.module('app', []);
  app.controller("MyCtrl", function ($scope) {
    $scope.name = "Shailendra Chauhan"
  })
</script>
```

**Q53. What is issue with two-way data binding?**

**OR**

**Why one-way data binding is introduced?**

**Ans.** In order to make data-binding possible, Angular uses \$watch APIs to observe model changes on the scope. Angular registered *watchers* for each variable on scope to observe the change in its value. If the value, of variable on scope is changed then the view gets updated automatically.

This automatic change happens because of \$digest cycle is triggered. Hence, Angular processes all registered *watchers* on the current scope and its children and checks for model changes and calls dedicated *watch listeners* until the model is stabilized and no more listeners are fired. Once the \$digest loop finishes the execution, the browser re-renders the DOM and reflects the changes.

By default, every variable on a scope is observed by the angular. In this way, unnecessary variable are also observed by the angular that is time consuming and as a result page is becoming slow.

Hence to avoid **unnecessary observing** of variables on scope object, angular introduced one-way data binding.

**Q54. How AngularJS handle data binding?**

**Ans.** AngularJS handle data-binding mechanism with the help of three powerful functions: \$watch(), \$digest() and \$apply(). Most of the time AngularJS will call the \$scope.\$watch() and \$scope.\$digest() functions for you, but in some cases you may have to call these functions yourself to update new values.

**Q55. What is the difference between \$watch, \$digest and \$apply?**

**Ans.** **\$watch()** - This function is used to observe changes in a variable on the \$scope. It accepts three parameters: *expression*, *listener* and *equality object*, where listener and equality object are optional parameters.

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

```
<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>

```

**\$digest()** - This function iterates through all the watches in the \$scope object, and its child \$scope objects (if it has any). When \$digest() iterates over the watches, it checks if the value of the expression has changed. If the value has changed, AngularJS calls the listener with the new value and the old value.

```
$digest()
```

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.

```

<html>
<head>
    <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>

```

```

    });
  });
</script>
</body>
</html>

```

**\$apply()** - Angular do auto-magically updates only those model changes which are inside AngularJS context. When you do change in any model outside of the Angular context (like browser DOM events, setTimeout, XHR or third party libraries), then you need to inform Angular of the changes by calling \$apply() manually. When the \$apply() function call finishes AngularJS calls \$digest() internally, so all data bindings are updated.

```
$apply([exp])
```

### Example

```

<html>
<head>
  <title>AngularJS Apply</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>

```

### Q56. Which one is fast between \$digest and \$apply?

**Ans.** \$digest() is faster than \$apply(), since \$apply() triggers watchers on the entire scope chain i.e. on the current scope and its parents or children (if it has) while \$digest() triggers watchers on the current scope and its children(if it has).



**Q57. Which one handles exception automatically between \$digest and \$apply?**

**Ans.** When error occurs in one of the watchers, \$digest() cannot handle errors via \$exceptionHandler service, In this case you have to handle exception yourself.

While \$apply() uses try catch block internally to handle errors and if error occurs in one of the watchers then it passes errors to \$exceptionHandler service.

**Pseudo-Code of \$apply()**

```
function $apply(expr) {
  try {
    return $eval(expr);
  } catch (e) {
    $exceptionHandler(e);
  } finally {
    $root.$digest();
  }
}
```

**Q58. Explain \$watch(), \$watchgroup() and \$watchCollection() functions of scope?**

**Ans.** **\$watch** - This function is used to observe changes in a variable on the \$scope. It accepts three parameters: *expression*, *listener* and *equality object*, where listener and equality object are optional parameters.

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

Here, watchExpression is the expression in the scope to watch. This expression is called on every \$digest() and returns the value that is being watched.

The listener defines a function that is called when the value of the watchExpression changes to a new value. If the watchExpression is not changed then listener will not be called.

The objectEquality is a boolean type which is used for comparing the objects for equality using angular.equals instead of comparing for reference equality.

```
scope.name = 'shailendra';
scope.counter = 0;

scope.$watch('name', function (newVal, oldVal) {
  scope.counter = scope.counter + 1;
});
```

**\$watchgroup** - This function is introduced in **Angular1.3**. This works the same as \$watch() function except that the first parameter is an array of expressions to watch.

```
$watchGroup(watchExpression, listener)
```

The listener is passed as an array with the new and old values for the watched variables. The listener is called whenever any expression 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';
});

```

**\$watchCollection** - This function is used to watch the properties of an object and fires whenever any of the properties change. It takes an object as the first parameter and watches the properties of the object.

```
$watchCollection(obj, listener)
```

The listener is called whenever anything within the obj has been changed.

```

$scope.names = ['shailendra', 'deepak', 'mohit', 'kapil'];
$scope.dataCount = 4;

$scope.$watchCollection('names', function (newVal, oldVal) {
    $scope.dataCount = newVal.length;
});

```

## Q59. Explain AngularJS scope life-cycle?

**Ans.** Scope data goes through a life cycle when the angular app is loaded into the browser. Understanding the scope life cycle will help you to understand the interaction between scope and other AngularJS components.

The scope data goes through the following life cycle phases:

1. **Creation** – This phase initialized the scope. The root scope is created by the \$injector when the application is bootstrapped. During template linking, some directives create new child scopes. A digest loop is also created in this phase that interacts with the browser event loop. This digest loop is responsible to update DOM elements with the changes made to the model as well as executing any registered watcher functions.
2. **Watcher registration** - This phase registers watches for values on the scope that are represented in the template. These watches propagate model changes automatically to the DOM elements.

You can also register your own watch functions on a scope value by using the *\$watch()* function.

3. **Model mutation** - This phase occurs when data in the scope changes. When you make the changes in your angular app code, the scope function *\$apply()* updates the model and calls the *\$digest()* function to update the DOM elements and registered watches.

When you do the changes to scope inside your angular code like within controllers or services, angular internally call *\$apply()* function for you. But when you do the changes to the scope outside the angular code, you have to call *\$apply()* function explicitly on the scope to force the model and DOM to be updated correctly.

4. **Mutation observation** – This phase occurs when the `$digest()` function is executed by the digest loop at the end of `$apply()` call. When `$digest()` function executes, it evaluates all watches for model changes. If a value has been changed, `$digest()` calls the `$watch` listener and updates the DOM elements.
5. **Scope destruction** – This phase occurs when child scopes are no longer needed and these scopes are removed from the browser's memory by using `$destroy()` function. It is the responsibility of the child scope creator to destroy them via `scope.$destroy()` API.

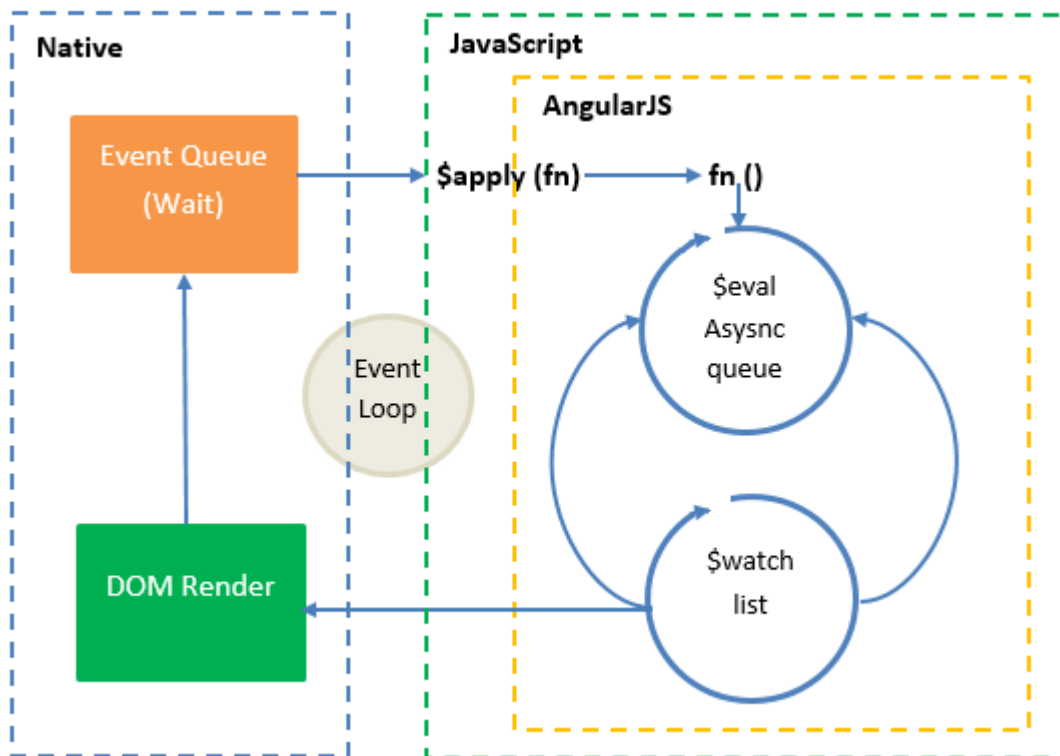
This stop propagation of `$digest` calls into the child scopes and allow the memory to be reclaimed by the browser garbage collector.

## Q60. Explain digest life-cycle in AngularJS?

**Ans.** The digest loop is responsible to update DOM elements with the changes made to the model as well as executing any registered watcher functions.

**The \$digest loop** is fired when the browser receives an event that can be managed by the angular context. This loop is made up of two smaller loops. One processes the `$evalAsync` queue and the other one processes the `$watch` list.

**The \$digest loop** keeps iterating until the `$evalAsync` queue is empty and the `$watch` list does not detect any changes in the model.



The **\$evalAsync queue** contains those tasks which are scheduled by *\$evalAsync()* function from a directive or controller.

The **\$watch list** contains watches correspondence to each DOM element which is bound to the *\$scope* object. These watches are resolved in the *\$digest* loop through a process called dirty checking. The dirty checking is a process which checks whether a value has changed, if the value has changed, it set the *\$scope* as dirty. If the *\$scope* is dirty, another *\$digest* loop is triggered.

When you do the changes to the scope outside the angular context, you have to call *\$apply()* function explicitly on the scope to trigger *\$digest* cycle immediately.

#### Q61. When to use *\$destroy()* function of scope?

**Ans.** *\$destroy()* - This function permanently detached the current scope with all of its children from the parent scope. This is required when child scopes are no longer needed. Hence, *\$destroy()* function is called to remove these scopes from the browser's memory.

When *\$destroy()* is called all the watchers and listeners get removed and the object which represented the scope becomes eligible for garbage collection.

#### Q62. What is difference between *\$evalAsync* and *\$timeout*?

**Ans.** *\$evalAsync* - This executes the expression on the current scope on later time. The *\$evalAsync* makes no guarantees as to when the expression will be executed, only that:

1. If code is queued using *\$evalAsync* from a directive, it will run after the DOM has been manipulated by Angular, but before the browser renders.
2. If code is queued using *\$evalAsync* from a controller, it will run before the DOM has been manipulated by Angular and before the browser renders.

*\$timeout* - This also executes the expression on the current scope on later time. When the code is queued using *\$timeout*, it will run after the DOM has been manipulated by Angular and after the browser renders which may cause flicker in some cases.

#### Q63. What is the difference between *\$watch* and *\$observe*?

**Ans.** *\$watch* is a method on the *scope* object which is used to watch expressions. The expression can be either strings or functions. It can be called wherever you have access to scope (a controller or a directive linking function).

*\$observe* is a method on the *attrs* object which is only used to observe the value change of a DOM attribute. It is only used inside directives.

**Note** - All *\$observes* and *\$watches* are checked on every digest cycle.

#### Q64. What is the difference between *\$parse* and *\$eval*?

**Ans.** *\$parse* and *\$eval* both operate on angular expressions i.e. `{{ expression }}`.

*\$eval* is a *scope* method which executes an expression on the current scope and returns the result.

```
scope.a = 1;
scope.b = 2;

scope.$eval('a+b'); // 3
```

**\$parse** is an Angular service which converts an expression into a function. Then function can be invoked and passed a context (usually scope) in order to retrieve the expression's value.

```
<div my-attr="obj.name" my-directive>
dotnet-tricks.com
</div>

<script type="text/javascript">
myApp.directive('myDirective', function( $parse, $log ) {
    return function( scope, el, attrs ) {

        // parse the "my-attr" attribute value into a function
        var myFn = $parse( attrs.myAttr );
        // "myFn" is now a function which can be invoked to get the expression's value;

        // the following line logs the value of obj.name on scope:
        $log.log(myFn(scope) ); // dotnet-tricks.com

        el.bind('click', function() {

            // "myFn.assign" is also a function; it can be invoked to
            // update the expression value
            myFn.assign(scope, "New name");

            scope.$apply();
        })
    }
});
</script>
```

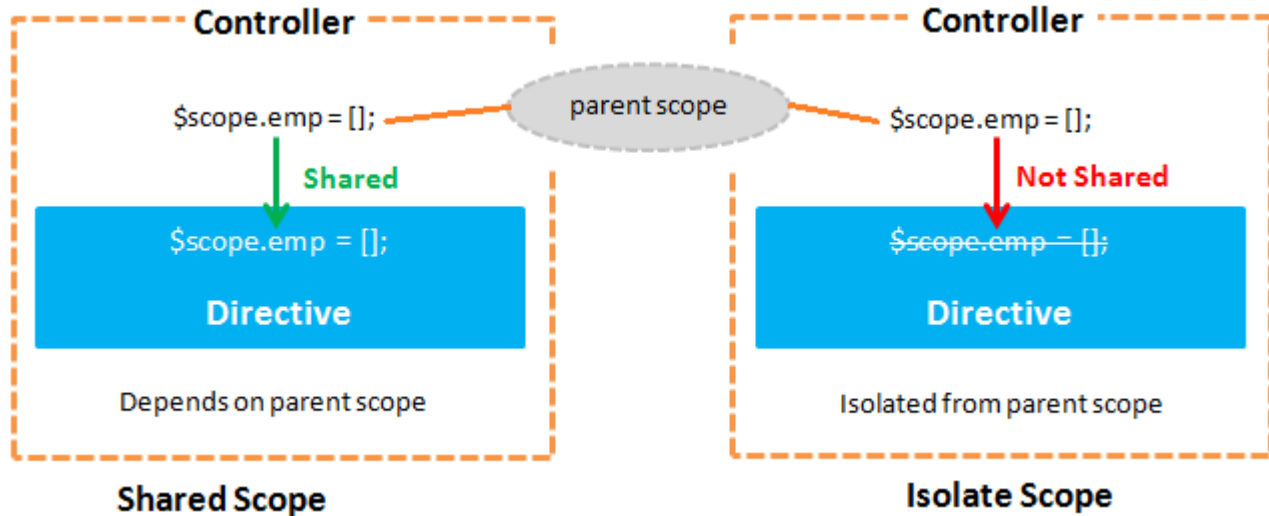
Also, if the expression is assignable then the returned function will have an assign property. The assign property is a function that can be used to change the expression's value on the given context.

## Q65. What is Isolate Scope and why it is required?

**Ans.** By default, directives have access to the parent scope in AngularJS apps. Hence, you can write your custom directive code based on parent scope. If the parent scope changes at all the directive is no longer useful.

```
angular.module('mydirective').directive('sharedScope', function () {
    return {
        template: 'Name: {{emp.name}} Address: {{emp.address}}'
    };
});
```

The shared scope allows the parent scope to flow down into the directive but the Isolate scope doesn't allow the parent scope to flow down into the directive.



### Creating Isolate Scope

You can isolate the scope in a directive by adding a scope property into the directive as given below:

```
angular.module('mydirective').directive('sharedScope', function () {
  return {
    scope: {},
    template: 'Name: {{emp.name}} Address: {{emp.address}}'
  };
});
```

### **Q66. Does AngularJS support MVC?**

**Ans.** AngularJS is a MVC framework. It does not implement MVC in the traditional way, but rather something closer to MVVM Model-View-ViewModel).

### **Q67. What is Model in AngularJS?**

**Ans.** Models are plain old JavaScript objects that represent data used by your app. Models are also used to represent your app's current state.

### **Q68. What is ViewModel in AngularJS?**

**Ans.** A viewmodel is an object that provides specific data and methods to maintain specific views. Basically, it is a \$scope object which lives within your AngularJS app's controller. A viewmodel is associated with a HTML element with the ng-model and ng-bind directives.

### **Q69. What is Controller in AngularJS?**

**Ans.** The controller defines the actual behavior of your app. It contains business logic for the view and connects the model to view with the help of \$scope. A controller is associated with a HTML element with the ng-controller directive.

### Creating Controller

```
<script type="text/javascript">
  //defining main controller
```

```

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

    //defining book viewmodel
    $scope.book =
    {
        id: 1,
        name: 'AngularJS Interview Questions and Answers',
        author: 'Shailendra Chauhan',
    };
});
</script>

```

### Using Controller

```

<div ng-controller="mainController">
    Id: <span ng-bind="book.id"></span>
    <br />
    Name:<input type="text" ng-model="book.name" />
    <br />
    Author: <input type="text" ng-model="book.author" />
</div>

```

## Q70. How to share information between controllers in AngularJS?

OR

### What are the ways to communicate between controllers in AngularJS?

**Ans.** There are various different ways to share data between controllers in an AngularJS app. The most commonly used are Scope, Service, Factory and Providers.

## Q71. What is \$emit, \$broadcast and \$on in AngularJS?

**Ans.** AngularJS provides \$on, \$emit, and \$broadcast services for event-based communication between controllers.

**\$emit** – It dispatches an event name upwards through the scope hierarchy and notify to the registered *\$rootScope.Scope* listeners. The event life cycle starts at the scope on which \$emit was called. The event traverses upwards toward the root scope and calls all registered listeners along the way. The event **will stop** propagating if one of the listeners cancels it.

```

<!DOCTYPE html>
<html>
<head>
    <title>Broadcasting</title>
    <script src="lib/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("firstCtrl", function ($scope) {
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
                console.log($scope.message);
            });
        });
    </script>

```

```

    });

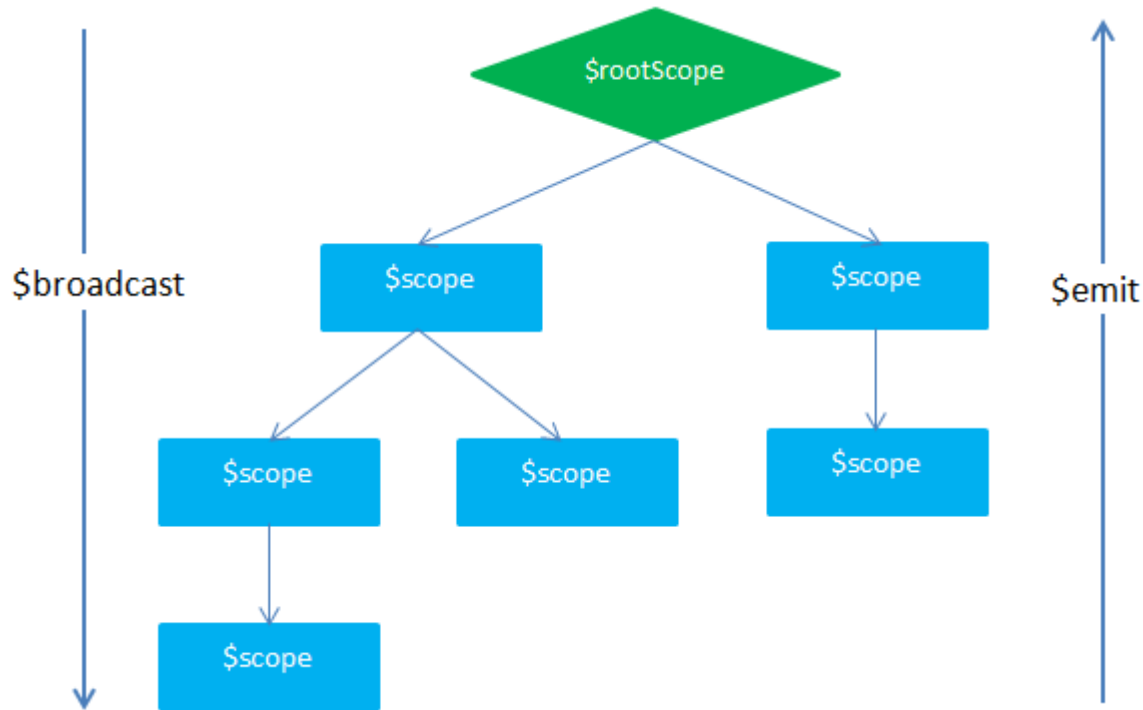
    app.controller("secondCtrl", function ($scope) {
        $scope.handleClick = function (msg) {
            $scope.$emit('eventName', { message: msg });
        };
    });

</script>
</head>
<body ng-app="app">
    <div ng-controller="firstCtrl" style="border:2px solid #E75D5C; padding:5px;">
        <h1>Parent Controller</h1>
        <p>Emit Message : {{message}}</p>
        <br />
        <div ng-controller="secondCtrl" style="border:2px solid #428bca; padding:5px;">
            <h1>Child Controller</h1>
            <input ng-model="msg">
            <button ng-click="handleClick(msg);">Emit</button>
        </div>
    </div>
</body>
</html>

```

**Output**





**\$broadcast** – It dispatches an event name downwards to all child scopes (and their children) and notify to the registered *\$rootScope.Scope* listeners. The event life cycle starts at the scope on which *\$broadcast* was called. All listeners for the event on this scope get notified. Afterwards, the event traverses downwards toward the child scopes and calls all registered listeners along the way. The event **cannot be canceled**.

```

<!DOCTYPE html>
<html>
<head>
  <title>Broadcasting</title>
  <script src="lib/angular.js"></script>
  <script>
    var app = angular.module('app', []);

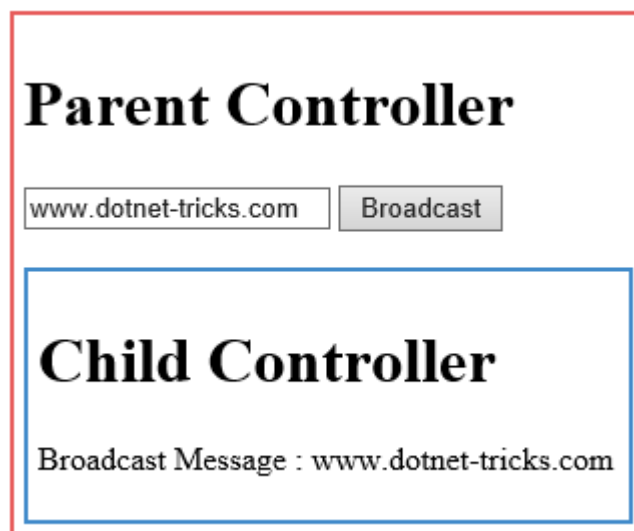
    app.controller("firstCtrl", function ($scope) {
      $scope.handleClick = function (msg) {
        $scope.$broadcast('eventName', { message: msg });
      };
    });

    app.controller("secondCtrl", function ($scope) {
      $scope.$on('eventName', function (event, args) {
        $scope.message = args.message;
        console.log($scope.message);
      });
    });

  </script>
</head>
<body ng-app="app">
  
```

```
<div ng-controller="firstCtrl" style="border:2px solid #E75D5C; padding:5px;">
  <h1>Parent Controller</h1>
  <input ng-model="msg">
  <button ng-click="handleClick(msg);">Broadcast</button>
  <br /><br />
  <div ng-controller="secondCtrl" style="border:2px solid #428bca;padding:5px;">
    <h1>Child Controller</h1>
    <p>Broadcast Message : {{message}}</p>
  </div>
</div>
</body>
</html>
```

### Output



**\$on** – It listen on events of a given type. It can catch the event dispatched by \$broadcast and \$emit.

**Note** - If there is no parent-child relation between your scopes you can inject \$rootScope into the controller and broadcast the event to all child scopes but you cannot emit your event. You can emit your event only when you have parent-child relation and event propagation is initiated by child. However, \$emit can fire an event only for all \$rootScope.\$on listeners.

### **Q72. What is View in AngularJS?**

**Ans.** The view is responsible for presenting your models data to end user. Typically it is the HTML markup which exists after AngularJS has parsed and compiled the HTML to include rendered markup and bindings.

### **Q73. How to apply validation in AngularJS?**

**Ans.** AngularJS provides you built-in validation directives to validate form client side. This makes your life pretty easy to handle client-side form validations without adding a lot of extra effort. AngularJS form validations are based on the HTML5 form validators.

#### AngularJS directives for form validation

Here is a list of AngularJS directive which can be applied on an input field to validate its value.

```
<input type="text"
      ng-model="{ string }"
      [name="{ string }"]
      [ng-required="{ boolean }"]
      [ng-minlength="{ number }"]
      [ng-maxlength="{ number }"]
      [ng-pattern="{ string }"]
      [ng-change="{ string }"]>
</input>
```

## Q74. How to do custom form validation in AngularJS?

**Ans.** AngularJS allows you to create your own custom validation directives. **For example**, you have to compare password and confirm password fields. To achieve this task, you have to make a custom directive that will be fired whenever the password or confirm password changes.

### Creating custom directive to compare password and confirm password fields

```
//defining module
var myapp = angular.module('myapp', []);

//creating custom directive
myapp.directive('ngCompare', function () {
  return {
    require: 'ngModel',
    link: function (scope, currentEl, attrs, ctrl) {
      var comparefield = document.getElementsByName(attrs.ngCompare)[0]; //getting
      first element
      compareEl = angular.element(comparefield);

      //current field key up
      currentEl.on('keyup', function () {
        if (compareEl.val() != "") {
          var isMatch = currentEl.val() === compareEl.val();
          ctrl.$setValidity('compare', isMatch);
          scope.$digest();
        }
      });

      //Element to compare field key up
      compareEl.on('keyup', function () {
        if (currentEl.val() != "") {
          var isMatch = currentEl.val() === compareEl.val();
          ctrl.$setValidity('compare', isMatch);
          scope.$digest();
        }
      });
    }
  };
});
```

### Using above created custom directive

```
<form name="userForm" ng-submit="submitForm()" novalidate>
  <!-- Password -->
```

```

<div>
  <label>Password</label>
  <input type="Password" name="password" ng-required="true" ng-model="user.password">
  <p ng-show="userForm.password.$invalid">Your password is required.</p>
</div>

<!-- Confirm Password -->
<div>
  <label>Confirm Password</label>
  <input type="Password" name="confirmPassword" ng-compare="password" ng-
required="true" ng-model="user.confirmPassword" >
  <p ng-show="userForm.confirmPassword.$error.compare &&
!userForm.confirmPassword.$error.required">Confirm password doesn't match.</p>
</div>

<!-- Other code has been removed for clarity-->
</form>

```

## Q75. What are different Angular form properties?

**Ans.** Angular provides properties on form which help you to get information about a form or its inputs and to validate them.

**\$valid** - It is a boolean property that tells whether the form or its inputs are valid or not. If all containing form and controls are valid, then it will be true, otherwise it will be false.

### Syntax

```

formName.$valid
formName.inputFieldName.$valid

```

**\$invalid** - It is a boolean property that tells whether the form or its inputs are invalid or not. If at least one containing form and control is invalid then it will be true, otherwise it will be false.

### Syntax

```

formName.$invalid
formName.inputFieldName.$invalid

```

**\$pristine** - It is a boolean property that tells whether the form or its inputs are unmodified by the user or not. If the form or its inputs are unmodified by the user, then it will be true, otherwise it will be false.

### Syntax

```

formName.inputFieldName.$pristine

```

**\$dirty** - It is a boolean property that is actually just reverse of pristine i.e. it tells whether the form or its inputs are modified by the user or not. If the form or its inputs are modified by the user, then it will be true, otherwise it will be false.

### Syntax

```

formName.$dirty
formName.inputFieldName.$dirty

```

**\$error** - This is an object hash which contains references to all invalid controls or forms. It has all errors as keys: where keys are validation tokens (such as required, url or email) and values are arrays of controls or forms that are invalid with given error. For a control, if a validation fails then it will be true, otherwise it will be false.

#### Syntax

```
formName.$error  
formName.inputFieldName.$error
```

### Q76. What are different states of a form in AngularJS?

**Ans.** The AngularJS form goes to the following states, starting from the form rendering and when the user has finished the filling of form.

**State 1: pristine and invalid** - When the form is first time rendered and the user has not interacted with the form yet.

**State 2: dirty and invalid** - User has interacted with the form, but form validity has not been satisfied, yet.

**State 3: dirty and valid** - User has finished the filling of form and the entire form validations has been satisfied

### Q77. What is Service in AngularJS?

**Ans.** A service is a reusable singleton object which is used to organize and share code across your app. A service can be injected into controllers, filters, directives.

AngularJS offers several built-in services (like \$http, \$provide, \$resource, \$window, \$parse) which always start with \$ sign.

### Q78. What are different ways to create service in AngularJS?

**Ans.** There are five ways to create a service as given below:

1. Service
2. Factory
3. Provider
4. Value
5. Constant

### Q79. What is the difference between Factory, Service and Provider?

**Ans. Factory** - A factory is a simple function which allows you to add some logic before creating the object. It returns the created object.

#### Example

```
//define a factory using factory() function  
app.factory('MyFactory', function () {  
  
    var serviceObj = {};  
    serviceObj.function1 = function () {  
        //TO DO:  
    };  
});
```

```

    serviceObj.function2 = function () {
        //TO DO:
    };

    return serviceObj;
});

```

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

**Service** - A service is a constructor function which creates the object using new keyword. You can add properties and functions to a service object by using *this* keyword. Unlike factory, it doesn't return anything.

#### Example

```

//define a service using service() function
app.service('MyService', function () {
    this.function1 = function () {
        //TO DO:
    };

    this.function2 = function () {
        //TO DO:
    };
});

```

**When to use:** It is a singleton object. Use it when you need to share a single object across the application.

**For example,** authenticated user details.

**Provider** - A provider is used to create a configurable service object. It returns value by using *\$get()* function.

#### Example

```

//define a provider using provider() function
app.provider('configurable', function () {
    var privateName = '';
    this.setName = function (newName) {
        privateName = newName;
    };
    this.$get = function () {
        return {
            name: privateName
        };
    };
});

//configuring provider using config() function
app.config(function (configurableService) {
    configurableService.setName('www.dotnet-tricks.com');
});

```

**When to use:** When you need to provide module-wise configuration for your service object before making it available.

**Example**

```

<html>
<head>
  <title>AngularJS Service, Factory and Providers</title>
  <script src="lib/angular.js"></script>
</head>
<body>
  <div class="container" style="padding-top:20px;">
    <div ng-app="myApp" ng-controller="myController">
      <p>From Service: {{serviceName}}</p>
      <p>From Factory: {{factoryName}}</p>
      <p>From Provider: {{providerName}}</p>
    </div>
  </div>
<script>
  //defining module
  var app = angular.module('myApp', []);

  //defining service
  app.service('myService', function () {
    this.name = '';
    this.setName = function (newName) {
      this.name = newName;
      return this.name;
    };
  });

  //defining factory
  app.factory('myFactory', function () {
    var serviceObj = {};
    serviceObj.name = '';
    serviceObj.setName = function (newName) {
      serviceObj.name = newName;
    };
    return serviceObj;
  });

  //defining provider
  app.provider('configurable', function () {
    var privateName = '';
    this.setName = function (newName) {
      privateName = newName;
    };
    this.$get = function () {
      return {
        name: privateName
      };
    };
  });

  //configuring provider
  app.config(function (configurableProvider) {
    configurableProvider.setName("Saksham Chauhan");
  });

```

```
//defining controller
app.controller('myController', function ($scope, myService, myFactory,
configurable) {
    $scope.serviceName = myService.setName("Saksham Chauhan");

    myFactory.setName("Saksham Chauhan");
    $scope.factoryName = myFactory.name;

    $scope.providerName = configurable.name;
});
</script>
</body>
</html>
```

**Output**

From Service: Saksham Chauhan

From Factory: Saksham Chauhan

From Provider: Saksham Chauhan

**Q80. What is difference between value and constant?**

**Ans.** Value and Constant are simple objects which are used to share data globally with in a module.

**Value** - A value can be a number, string, date-time, array or object. You can also register a function as a value. Values are typically used as configuration which is injected into factories, services or controllers.

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

//define value
app.value("numberValue", 100);
app.value("stringValue", "dotnet-tricks.com");
app.value("objectValue", { name: "dotnet-tricks.com", totalUsers: 120000 });
```

**Constant** - A constant is like as value. The difference between a value and a constant service is that constant service can be injected into a module configuration function i.e. *config()* but value service cannot be.

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

//define constant
app.value("mynumberValue", 200);
app.value("mystringValue", "webgeekschool.com");

//module configuration function
app.config(function (mynumberValue) { //here value objects can't be injected
    console.log("Before:" + mynumberValue);
    mynumberValue = "New Angular Constant Service";
    console.log("After:" + mynumberValue);
});
```



### Q81. What is the difference between \$http and \$resource?

**Ans.** \$http service is a core Angular service which allows you to make AJAX requests by using GET, HEAD, POST, PUT, DELETE, JSONP and PATCH methods. It is very much like the \$.ajax() method in jQuery. It can be used with RESTful and Non-RESTful server-side data sources.

\$http is good for quick retrieval of server-side data that doesn't really need any specific structure or complex behaviors.

\$resource warps \$http and allows you to interact with RESTful server-side data sources. It requires the ngResource module to be installed which exist in *angular-resource.js*

\$http is good for retrieval of RESTful server-side data sources that might need any specific structure or complex behaviors.

### Q82. What methods \$http service support?

**Ans.** The \$http service supports the following methods:

1. \$http.get()
2. \$http.head()
3. \$http.post()
4. \$http.put()
5. \$http.delete()
6. \$http.jsonp()
7. \$http.patch()

### Q83. How to enable caching in \$http service?

**Ans.** You can enable caching in \$http service by setting configuration property *cache* to *true*. When cache is enabled, \$http service stores the response from the server in local cache. In this way, next time the response will be served from the cache without sending request to server.

```
$http.get("http://server/myserviceapi",{  
  cache:true  
}).success(function(){  
  //TO DO:  
});
```

### Q84. What methods \$resource service object support?

**Ans.** The \$resource service object supports the following methods:

1. get()
2. query()
3. save()
4. remove()
5. delete()

**Q85. What is \$q service and when to use it?**

**Ans.** \$q is a service that helps you to run functions asynchronously, and use their return values when they have done processing.

\$q service is said to be inspired by *Chris Kowal's Q library* which allow users to monitor asynchronous progress by providing a "promise" as a return from a call.

It is good when you need to process a number of asynchronous activities simultaneously. The *\$q.all()* function lets you trigger several callbacks at the same time, and use a single then function to join them all together.

```
var first = $http.get("/app/data/first.json"),
    second = $http.get("/app/data/second.json"),
    third = $http.get("/app/data/third.json");

$q.all([first, second, third]).then(function (result) {
    var tmp = [];
    angular.forEach(result, function (response) {
        tmp.push(response.data);
    });
    return tmp;
}).then(function (tmpResult) {
    $scope.combinedResult = tmpResult.join(", ");
});
```

**Q86. What is the difference between Kris Kowal's Q and \$q?**

**Ans.** There are two main differences between Kris Kowal's Q and \$q:

1. \$q is integrated with the \$rootScope.Scope model observation mechanism in angular, which means faster propagation of resolution or rejection into your models and avoiding unnecessary browser repaints, which would result in flickering UI.
2. Q has many more features than \$q, but that comes at a cost of bytes. \$q is tiny, but contains all the important functionality needed for common async tasks.

**Q87. What is Restangular?**

**Ans.** Restangular is an Angular service specifically designed simply to fetch data from the rest of the world. To use Restangular you need to link the *restangular.js* file and include restangular resource as a dependency within your angular app.

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

app.controller('MainController', function ($scope, Restangular) {
    //TO DO:
});
```

**Q88. What are the advantages of Restangular over \$resource and \$http?**

**Ans.** Restangular has several features that distinguish it from \$resource:

1. It uses promises.
2. You can use this in \$routeProvider.resolve.

3. It doesn't have all those \$resource bugs like trailing slashes, additional: in the URL, escaping information, expecting only arrays for getting lists, etc.
4. It supports all HTTP methods.
5. It supports ETag out of the box.
6. It supports self-linking elements if you receive from the server some item that has a link to itself, you can use that to query the server instead of writing the URL manually.
7. You don't have to create one \$resource object per request.
8. You don't have to write or remember ANY URL like \$resource. You just write the name of the resource you want to fetch and that's it.
9. It supports nested RESTful resources.
10. Restangular lets you create your own methods.
11. Support for wrapped responses.

### Q89. What is difference between \$window and window in AngularJS?

**Ans.** **\$window** is an Angular service which reference to the browser's **window** object. The **window** object is globally available in JavaScript; it causes testability problems, because it is a global variable. Angular refer to it through the **\$window** service, so that it can be overridden, removed or mocked for testing.

```
<script>
  var app = angular.module('windowExample', []);
  app.controller('ExampleController',function ($scope, $window) {
    $scope.greeting = 'Hello, World!';
    $scope.doGreeting = function (greeting) {
      $window.alert(greeting);
    };
  });
</script>
<div ng-app="windowExample" ng-controller="ExampleController">
  <input type="text" ng-model="greeting" />
  <button ng-click="doGreeting(greeting)">ALERT</button>
</div>
```

### Q90. What is difference between \$document and window.document in AngularJS?

**Ans.** **\$document** is an Angular service which reference to the browser's **window.document** object.

```
<script>
  var app = angular.module('documentExample', []);
  app.controller('ExampleController', ['$scope', '$document', function ($scope,
$document) {
    $scope.title = $document[0].title;
    $scope.windowTitle = angular.element(window.document)[0].title;
  }]);
</script>

<div ng-app="documentExample" ng-controller="ExampleController">
  <p>$document title: <b ng-bind="title"></b></p>
  <p>>window.document title: <b ng-bind="windowTitle"></b></p>
</div>
```

**Q91. What is difference between \$timeout and window.setTimeout in AngularJS?**

**Ans.** **\$timeout** is an Angular service which wraps the browser's **window.setTimeout()** function into a try/catch block and delegates any exceptions to **\$exceptionHandler** service. It is used to call a JavaScript function after a given time delay. The **\$timeout** service only schedules a single call to the function.

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

app.controller("MyController", function ($scope, $timeout) {
    $timeout(callAtTimeout, 1000);
});

function callAtTimeout() {
    console.log("Timeout occurred");
}
```

**Q92. What is difference between \$interval and window. setInterval in AngularJS?**

**Ans.** **\$interval** is an Angular service which wraps the browser's **window. setInterval()** function. It is also used to call a JavaScript function repeatedly within a time interval.

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

app.controller("MyController", function ($scope, $interval) {
    $interval(callAtInterval, 3000);
});

function callAtInterval() {
    console.log("Interval occurred");
}
```

**Q93. What is Routing in AngularJS?**

**Ans.** AngularJS Routing helps you to divide your app into multiple views and bind different views to Controllers. The magic of Routing is taken care by an AngularJS service **\$routeProvider**. **\$routeProvider** service provides method **when()** and **otherwise()** to define the routes for your app. Routing has dependency on **ngRoute** module.

**Defining Route for your application**

```
<script type="text/javascript">
    var myApp = angular.module('myApp', ['ngRoute']);

    myApp.config(['$routeProvider',
        function ($routeProvider) {
            $routeProvider.
                when('/products', { //route
                    templateUrl: 'views/products.html',
                    controller: 'productController'
                }).
                when('/product/:productId', { //route with parameter
                    templateUrl: 'views/product.html',
                    controller: 'productController'
                }).
                otherwise({ //default route
                    redirectTo: '/index'
                });
        }
    ]);
```

```

    });
  }]);
</script>

```

#### Q94. What is AngularUI router and how it is different from ngRoute?

**Ans.** The UI-Router is a routing framework for AngularJS built by the *AngularUI* team. Unlike ngRoute, it changes your angular app views based on state of the app and not based on the route URL (ngRoute).

The ui-router helps you to create nested views, use multiple views on the same page, have multiple views that control a single view, and more.

To use it you need to include reference of *ui-router.js* file into your angular app.

#### Q95. What is \$injector and \$inject?

**Ans.** **\$injector** is a service which is used to invoke controller functions, service functions, filter functions, and any other function that might need dependencies as parameters. Angular creates only a single \$injector object when an application is bootstrapped and uses that object for invoking.

```

<script>
  var app = angular.module('app', []);
  app.service('s1', function () {
    this.value = 22;
  });

  app.controller("MyCtrl", function ($scope, $injector) {
    $scope.doSomething = function () {
      var s1 = $injector.get('s1')
      s1.value += 10
    }
    $scope.value = function () {
      var s1 = $injector.get('s1')
      return s1.value
    }
  });
</script>

<div ng-app="app" ng-controller="MyCtrl">
  <button ng-click="doSomething()">increment</button>
  {{value()}}
</div>

```

**\$inject** is property which is used to inject the dependencies of a function as an array of strings.

```

<script type="text/javascript">
  var MyController = function(renamed$scope, renamedGreeter) {
    // ...
  }

  MyController['$inject'] = ['$scope', 'greeter']; //inject dependencies as an array of
strings
</script>

```

## Q96. What is Dependency Injection in AngularJS?

**Ans.** Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies. AngularJS comes with a built-in dependency injection mechanism. You can divide your AngularJS app into multiple different types of components which AngularJS can inject into each other.

There are following three ways of injecting dependencies into your code:

1. Implicitly from the function parameter names

```
<script type="text/javascript">
    function MyController($scope, greeter) {
        // ...
    }
</script>
```

2. Using the \$inject property annotation

```
<script type="text/javascript">
    var MyController = function(renamed$scope, renamedGreeter) {
        // ...
    }

    MyController['$inject'] = ['$scope', 'greeter'];
</script>
```

3. Using the inline array annotation

```
<script type="text/javascript">
    someModule.factory('greeter', ['$window', function (renamed$window) {
        // ...
    }]);
</script>
```

## Q97. How to do Language Internationalization in AngularJS?

**Ans.** The **angular-translate** is an AngularJS module that brings **i18n** (internationalization) and **l10n** (localization) into your Angular app. It allows you to create a JSON file that represents translation data as per language. These languages specific JSON files can be lazy-loaded from the server only when necessary. The angular-translate library (*angular-translate.js*) also provides built-in directives and filters that make the process of internationalizing simple.

```
<!DOCTYPE html>
<html>
<head>
    <title>AngularJS Internalization</title>
    <script src="lib/angular.js"></script>
    <script src="lib/angular-translate.js"></script>
    <script>
        var app = angular.module('myApp', ['pascalprecht.translate']);
        app.config(function ($translateProvider) {
            $translateProvider.translations('en', {
                TITLE: 'Hello',
```

```

        PARA: 'This is a paragraph.',
        BUTTON_LANG_EN: 'english',
        BUTTON_LANG_DE: 'german'
    })
    .translations('de', {
        TITLE: 'Hallo',
        PARA: 'Dies ist ein Paragraph.',
        BUTTON_LANG_EN: 'englisch',
        BUTTON_LANG_DE: 'deutsch'
    });
    //setting default language
    $translateProvider.preferredLanguage('en');
});

app.controller('TranslateController', function ($translate, $scope) {
    $scope.changeLanguage = function (langKey) {
        $translate.use(langKey);
    };
});
</script>
</head>
<body ng-app="myApp">
    <div ng-controller="TranslateController">

        <h1>{{ 'TITLE' | translate }}</h1>
        <p>{{ 'PARA' | translate }}</p>

        <button ng-click="changeLanguage('en')" translate="BUTTON_LANG_EN"></button>
        <button ng-click="changeLanguage('de')" translate="BUTTON_LANG_DE"></button>

    </div>
</body>
</html>

```

### Q98. What is i18n and L10n?

**Ans.** **i18n** means *Internationalization*, where 18 stands for the number of letters in word *Internationalization* between the first i and last n. It is the process of developing products in such a way that they can be localized for languages and cultures easily.

**L10n** means *Localization*, where 10 stand for the number of letters in word *Localization* between the first L and last n. It is the process of adapting applications and text to enable their usability in a particular culture.

### Q99. What is \$locale service?

**Ans.** \$locale service provides localization rules for various Angular components. As of now the only public api is: id – {string}

**For example**, a locale id is formatted as: languageId-countryId (e.g. en-us)

### Q100. What is a locale ID?

**Ans.** A locale is a specific geographical, political, or cultural region. The most commonly used locale ID consists of two parts: language code and country code. For example, en-US, en-AU, hi-IN are all valid locale IDs that have both language codes and country codes.

**Q101. How to manage cookie in AngularJS?**

**Ans.** AngularJS provides *ngCookies* module for reading and writing browser cookies. To use it include the **angular-cookies.js** file and set *ngCookies* as a dependency in your angular app. This module provides two services for cookie management: *\$cookies* and *\$cookieStore*.

**Q102. What is difference between \$cookies and \$cookieStore service?**

**Ans.** **\$cookies** - This service provides read/write access to browser's cookies.

If you want to use existing cookie solution, say read/write cookies from your existing server session system then use *\$cookie*.

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

app.controller('ExampleController', function ($cookies) {
    // Retrieving a cookie
    var favoriteCookie = $cookies.myFavorite;
    // Setting a cookie
    $cookies.myFavorite = 'meal';
});
```

**\$cookiesStore** - *\$cookieStore* is a thin wrapper around *\$cookies*. It provides a key-value (string-object) storage that is backed by session cookies. The objects which are put or retrieved from this storage are automatically serialized or deserialized by angular to JSON and vice versa.

If you are creating a new solution which will persist cookies based on key/value pairs, use *\$cookieStore*.

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

app.controller('ExampleController',function ($cookieStore) {
    // Put cookie
    $cookieStore.put('myFavorite', 'meal');
    // Get cookie
    var favoriteCookie = $cookieStore.get('myFavorite');
    // Removing a cookie
    $cookieStore.remove('myFavorite');
});
```

**Q103. How to handle mobile browsers/devices events in AngularJS?**

**Ans.** Mobile browsers/devices deal with events differently than desktop browsers. The AngularJS provide **ngTouch library** (*angular-touch.js*) to detect mobile browsers/devices events.

**For example**, Mobile browsers detect a tap event and then wait for second event about 300 ms if any. So if we're double-tapping the device then after this delay the browser fires a click event.

In this way this delay can make our apps unresponsive. Hence, instead of dealing with the click event, we can detect touch event using *ngTouch library*. It handles touch detection for us through the *ng-click* directive. Hence it will take care of calling the correct click event for mobile.

```
<button ng-click="save()">Save</button>
```



#### Q104. How to detect swipe event in mobile browsers/devices in AngularJS?

**Ans.** The **ngTouch** library provides **swipe** directives to capture user swipes, either left or right across the screen. These events are useful when the user want to swipe to the next photo gallery photo or to a new portion of our app.

The **ngSwipeLeft** directive detects when an HTML element is swiped from the right to the left and the **ngSwipeRight** directive detects when an HTML element is swiped from the left to the right.

#### Q105. How to do animation with the help of AngularJS?

**Ans.** AngularJS 1.2 comes with animation support via *ngAnimate* module. To enable animations within your angular app, you need to link the **angular-animate.js** file and include *ngAnimate* module as a dependency within your angular app.

#### Q106. What directives support animations?

**Ans.** The following directives support animations:

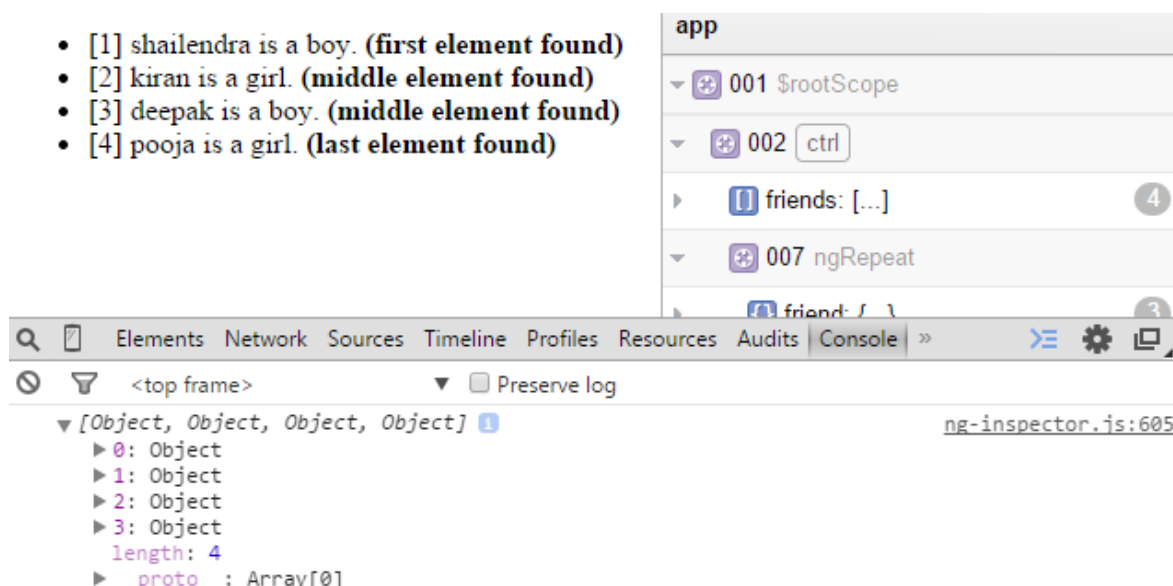
1. **ngRepeat** – It supports enter, leave and move animations.
2. **ngInclude** – It supports enter and leave animations.
3. **ngView** – It supports enter and leave animations.
4. **ngIf** – It supports enter and leave animations.
5. **ngSwitch** – It supports enter and leave animations.
6. **ngClass** – It supports addClass and removeClass animations.
7. **ngShow & ngHide** – These support addClass and removeClass animations.

#### Q107. How to debug AngularJS app in browser?

**Ans.** **AngularJS Batarang** and **ng-inspector** are browser extensions for Chrome that adds an inspector pane into browser to debug and understand your AngularJS app. ng-inspector also works with safari browser.

##### ng-inspector Extension:

- [1] shailendra is a boy. (first element found)
- [2] kiran is a girl. (middle element found)
- [3] deepak is a boy. (middle element found)
- [4] pooja is a girl. (last element found)



### Q108. How to securely parse and manipulate your HTML data in AngularJS?

**Ans.** AngularJS provides *ngSanitize* module to securely parse and manipulate HTML data in your application. To use it include the **angular-sanitize.js** file and set *ngSanitize* as a dependency in your angular app.

```
var app = angular.module('sanitizeExample', ['ngSanitize']);
app.controller('ExampleController', function ($scope, $sce) {

    var snippet = '<p style="color:blue">an html\n' +
        '<em onmouseover="this.textContent=\'PWN3D!\'">click here</em>\n snippet</p>';

    $scope.trustedSnippet = $sce.trustAsHtml(snippet); //sce=Strict Contextual Escaping
});
```

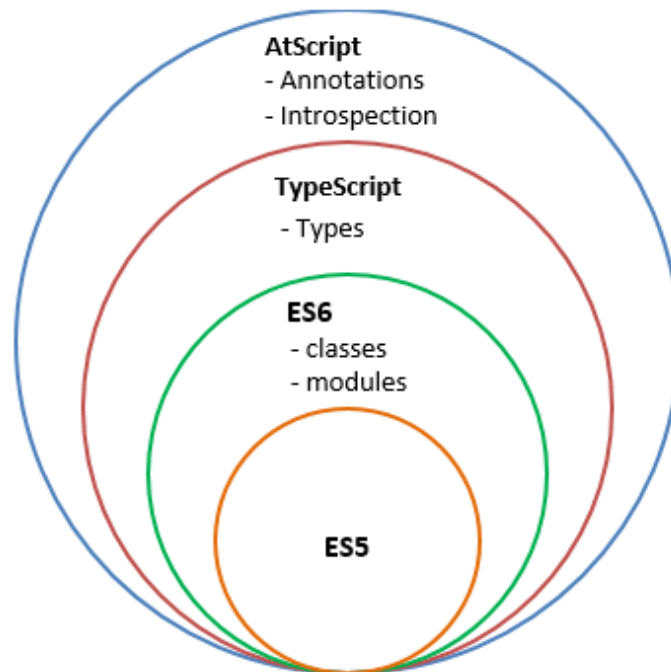
The snippet may contain HTML, CSS, URLs and JavaScript code which you want to safely render in your app.

### Q109. What is Angular 2.0?

**Ans.** Angular 2.0 is being written with AtScript, but that doesn't mean you have to write your application code with AtScript or know anything about AtScript to use Angular 2.0. You can easily write with TypeScript, ES6, ES5, CoffeeScript etc. whatever you like. It is still in development phase (at the time of writing this book).

### Q110. What is AtScript?

**Ans.** AtScript is Google's new superset for JavaScript. It enhances JavaScript with new features to make it more robust. It is not only designed to run on top of ECMAScript 5 and ECMAScript 6, but on the top of Microsoft's superset TypeScript language as well.



The aim of AtScript is to make type annotation data available at runtime to enhance JavaScript with type, field and metadata annotations.

# Others Free Interview Books

As the author, I am absolutely delighted to share my others interview books as well. I want these books to reach as many techy people as possible.

You can also download these books from my blog [www.dotnet-tricks.com](http://www.dotnet-tricks.com) in PDF format absolutely free.



## ASP.NET MVC Interview Questions & Answers

This book is appropriate for novice as well as for senior level professionals who wants to strengthen their skills before appearing for an interview on ASP.NET MVC. This book is equally helpful to sharpen their programming skills and understanding ASP.NET MVC in a short time.

This book is not only the ASP.NET MVC interview book but it is more than that. This book helps you to get the depth knowledge of ASP.NET MVC with a simple and elegant way.

Download



## LINQ Interview Questions & Answers

This book is appropriate for novice as well as for senior level professionals who want to understand what LINQ does, how it does in .NET languages like C# and VB. This book is equally helpful to show you the best of using LINQ with the help of many practical ways to make your daily programming life easier and more productive.

This book is not only help you to learn LINQ but it also be helpful to learn Entity Framework. This book helps you to do hands on LINQ as well as preparing yourself for an interview on LINQ.

Download