Q: What is Angular.js?
A: AngularJS is a javascript framework used for creating single web page applications. It allows you to use HTML as your template language and enables you to extend HTML's syntax to express your application's components clearly

Q: Explain what are the key features of Angular.js ?
A: The key features of angular.js are
Scope
Controller
Model
View
Services
Data Binding
Directives
Filters
Testable

Q: Explain what is scope in Angular.js ?
A: Scope refers to the application model, it acts like glue between application controller and the view. Scopes are arranged in hierarchical structure and impersonate the DOM (Document Object Model) structure of the application. It can watch expressions and propagate events.

Q: Explain what is services in Angular.js ?
A: In angular.js services are the singleton objects or functions that are used for carrying out specific tasks. It holds some business logic and these function can be called as controllers, directive, filters and so on.

Q: Explain what is Angular Expression? Explain what is key difference between angular expressions and JavaScript expressions?
A: Like JavaScript, Angular expressions are code snippets that are usually placed in binding such as {{ expression }}
The key difference between the JavaScript expressions and Angular expressions
Context : In Angular, the expressions are evaluated against a scope object, while the Javascript expressions are evaluated against the global window
Forgiving: In Angular expression evaluation is forgiving to null and undefined, while in Javascript undefined properties generates TypeError or ReferenceError
No Control Flow Statements: Loops, conditionals or exceptions cannot be used in an angular expression
Filters: To format data before displaying it you can use filters

2014-08-13_16-53-00

Q: With options on page load how you can initialize a select box ?
A: You can initialize a select box with options on page load by using ng-init directive

<div ng-controller = " apps/dashboard/account " ng-switch
On = "! ! accounts" ng-init = " loadData ( ) ">

Q: Explain what are directives ? Mention some of the most commonly used directives in Angular.js application ?
A: A directive is something that introduces new syntax, they are like markers on DOM element which attaches a special behavior to it. In any Angular.js application, directives are the most important components.
Some of the commonly used directives are ng-model, ng-App, ng-bind, ng-repeat , ng-show etc.

Q: Mention what are the advantages of using Angular.js ?
A: Angular.js has several advantages in web development.

Angular.js supports MVS pattern
Can do two ways data binding using Angular.js

It has per-defined form validations
It supports both client server communication
It supports animations

Q: Explain what Angular JS routes does ?
A: Angular js routes enable you to create different URLs for different content in your application. Different URLs for different content enables user to bookmark URLs to specific content. Each such bookmarkable URL in Angular.js is called a route
A value in Angular JS is a simple object. It can be a number, string or JavaScript object. Values are typically used as configuration injected into factories, services or controllers. A value should be belong to an Angular.js module. Injecting a value into an Angular.js controller function is done by adding a parameter with the same name as the value

Q: Explain what is data binding in Angular.js ?
A: Automatic synchronization of data between the model and view components is referred as data binding in Angular.js. There are two ways for data binding

Data mining in classical template systems
Data binding in angular templates

Q: What makes angular.js better ?
A: Registering Callbacks: There is no need to register callbacks . This makes your code simple and easy to debug.
Control HTML DOM programmatically: All the application that are created using Angular never have to manipulate the DOM although it can be done if it is required.
Transfer data to and from the UI: Angular.js helps to eliminate almost all of the boiler plate like validating the form, displaying validation errors, returning to an internal model and so on which occurs due to flow of marshalling data.
No initilization code: With angular.js you can bootstrap your app easily using services, which auto-injected into your application in Guice like dependency injection style

Q: Explain what is string interpolation in angular.js ?
A: In angular.js the compiler during the compilation process matches text and attributes using interpolate service to see if they contains embedded expressions. As part of normal digest cycle these expressions are updated and registered as watches.

Q: Mention the steps for the compilation process of HTML happens?
A: Compilation of HTML process occurs in following ways
Using the standard browser API, first the HTML is parsed into DOM
By using the call to the $compile () method, compilation of the DOM is performed. The method traverses the DOM and matches the directives.
Link the template with scope by calling the linking function returned from the previous step

Q: Explain what is directive and Mention what are the different types of Directive?
A: During compilation process when specific HTML constructs are encountered a behaviour or function is triggered, this function is referred as directive. It is executed when the compiler encounters it in the DOM.
Different types of directives are
Element directives
Attribute directives
CSS class directives
Comment directives

Q: Explain what is linking function and type of linking function?
A: Link combines the directives with a scope and produce a live view. For registering DOM listeners as well as updating the DOM, link function is responsible. After the template is cloned it is executed.
Pre-linking function: Pre-linking function is executed before the child elements are linked. It is not considered as the safe way for DOM transformation.

Post linking function: Post linking function is executed after the child elements are linked. It is safe to do DOM transformation by post-linking function

Q: Explain what is injector?
A: An injector is a service locator. It is used to retrieve object instances as defined by provider, instantiate types, invoke methods and load modules. There is a single injector per Angular application, it helps to look up an object instance by its name.

Q: Explain what is the difference between link and compile in angular.js?
A: Compile function: It is used for template DOM Manipulation and collect all of the directives.
Link function: It is used for registering DOM listeners as well as instance DOM manipulation. It is executed once the template has been cloned.

Q: Explain what is factory method in angular.js?
A: For creating the directive, factory method is used. It is invoked only once, when compiler matches the directive for the first time. By using $injector.invoke the factory method is invoked.

Q: Mention what are the styling form that ngModel adds to CSS classes ?
A: ngModel adds these CSS classes to allow styling of form as well as control
ng- valid
ng- invalid
ng-pristine
ng-dirty

Q: Mention what are the characteristics of "Sc ope"?
A: To observer model mutations scopes provide APIs ($watch)
To propagate any model changes through the system into the view from outside of the Angular realm
A scope inherits properties from its parent scope, while providing access to shared model properties, scopes can be nested to isolate application components
Scope provides context against which expressions are evaluated

Q: Explain what is DI (Dependency Injection) and how an object or function can get a hold of its dependencies ?
A: DI or Dependency Injection is a software design pattern that deals with how code gets hold of its dependencies. In order to retrieve elements of the application which is required to be configured when module gets loaded , the operation "config" uses dependency injection.
These are the ways that object uses to hold of its dependencies
Typically using the new operator, dependency can be created
By referring to a global variable, dependency can be looked up
Dependency can be passed into where it is required

Q: Mention what are the advantages of using Angular.js framework ?
A: Advantages of using Angular.js as framework are
Supports two way data-binding
Supports MVC pattern
Support static template and angular template
Can add custom directive
Supports REST full services
Supports form validations
Support both client and server communication
Support dependency injection
Applying Animations
Event Handlers

Q: Explain the concept of scope hierarchy? How many scope can an application have?

A: Each angular application consist of one root scope but may have several child scopes. As child controllers and some directives create new child scopes, application can have multiple scopes. When new scopes are formed or created they are added as a children of their parent scope. Similar to DOM, they also creates a hierarchical structure.

Q: Explain what is the difference between angular.js and backbone.js?
A: Angular.js combines the functionalities of most of the 3rd party libraries, it supports individual functionalities required to develop HTML5 Apps. While Backbone.js do their jobs individually.

Q: Who created Angular JS ?
A: Intially it was developed by Misko Hevery and Adam Abrons. Currently it is being developed by Google.

Q: What is AngularJS?
A: AngularJS is open source client side MV* (Model – View – Whatever) framework for creating dynamic web applications. It gives life to your static HTML and makes it dynamic with its magic. It extends HTML using directives, expression and data binding techniques to define a powerful HTML template.

Q: Is AngularJS a framework, library or a plugin?
A: The suitable answer is that its a framework. As its lightweight so people also get confused between library and framework.AngularJS is open source client side MVC framework for creating dynamic web applications.

Q: Is it same as jQuery?
A: NO. jQuery is great library for manipulating the DOM, providing better user experience with animations and effects. You can create website using jQuery but not a web application. jQuery is just a library to play around with HTML, where as AngularJS is a framework to build a dynamic web app as it supports two data binding, MVC, allow testability, templates and many more. Its like AngularJS like a toolbox and jQuery is just a tool. You can read more here.

Q: Does Angular use the jQuery library?
A: YES, Angular can use jQuery if it's present in your app when the application is being bootstrapped. If jQuery is not present in your script path, Angular falls back to its own implementation of the subset of jQuery that we call jQLite.

Q: Why AngularJS?
A: AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.

- MVC implementation is done right.
- It extends HTML using directives, expression and data binding techniques to define a powerful HTML template.
- Two way data-binding, form validations, routing supports, inbuilt services.
- REST friendly.
- Dependency injection support.
- It helps you to structure and test your JavaScript code.

Q: What are the key features/concepts of Angular.js?
A: When you start learning AngularJS, you will come across following terms and these are the features/concept of AngularJS.

- Scope
- Directives
- Expression
- Filters
- Data Bindings
- Model
- View
- Controller
- Modules
- Services
- Dependency Injection

Q: Is AngularJS is compatible with all modern browsers?
A: YES. AngularJS team run extensive test suite against the following browsers: Safari, Chrome, Firefox, Opera 15, IE9 and mobile browsers (Android, Chrome Mobile, iOS Safari).

Q: What is the basic need to start with AngularJS?
A: To start with AngularJS, one need to make reference of angular.js. The latest version of AngularJS can be downloaded from AngularJS.com. You can either download the required js file and then host them locally or you can also use google CDN for referencing it. Here is the link for google CDN url for referencing AngularJS.

Q: How does the AngularJS framework initialize itself?
A: The AngularJS has a self-executing anonymous function present in angular.js code, which does the initialization of AngularJS. Here is how below it looks,

```
(function(window, document, undefined) {
<!--
here goes entire AngularJS code
including functions, services, providers etc related code goes here
-->
if (window.angular.bootstrap) {
   //AngularJS is already loaded, so we can return here...
   console.log('WARNING: Tried to load angular more than once.');
   return;
 }

 //try to bind to jquery now so that one can write angular.element().read()
 //but we will rebind on bootstrap again.
 bindJQuery();

 publishExternalAPI(angular);

 jqLite(document).ready(function() {
   angularInit(document, bootstrap);
 });

})(window, document);
```

Above function first check if Angular has already been setup. If it has, we return here to prevent Angular from trying to initialize itself a second time. And then it binds jQuery (if present) and publish external API. And finally angularInit() method does the trick for initialization of AngularJS.

Following articles are recommended to know in detail.
How AngularJS Works – Explained with Angular Code
Dissecting Angular: Initializing An App

Q: What is the bootstrapping in AngularJS?
A: Bootstrapping in AngularJS is nothing but just initializing, or starting, your Angular app. AngularJS supports automatic bootstrapping as well as manual way as well.

Q: What are templates in AngularJS?
A: In Angular, templates are written with HTML that contains Angular-specific elements and attributes. Angular combines the template with information from the model and controller to render the dynamic view that a user sees in the browser. In other words, if your HTML page is having some Angular specific elements/attributes it becomes a template in AngularJS.

Q: What are directives in AngularJS?
A: Directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell

AngularJS to attach a specified behavior to that DOM element or even transform the DOM element and its children. When AngularJS finds the directive at the time of rendering then it attaches the requested behavior to the DOM element. Angular comes with a set of these directives built-in, like ngBind, ngModel, and ngClass.

Q: Can we create our own directives?
A: YES. AngularJS allows us to create our own custom directive.

Q: What are different type or classification of directives?
A: AngularJS directives can be classified in 4 different types .

Element directives

```
<ng-directive></ng-directive>
```

Attribute directives

```
<span ng-directive></span>
```

CSS class directives

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

Comment directives

```
<!-- directive: ng-directive expression -->
```

Q: What is the name of directive is used to bootstrap an angular app?
A: ng-app directive is used to auto-bootstrap an AngularJS application. The ng-app directive defines the root element of the application and is typically present in the root element of the page - e.g. on the <body> or <html> tags.

Q: Can AngularJS have multiple ng-app directives in a single page?
A: The answer is NO. Only one AngularJS application can be auto-bootstrapped per HTML document. The first ngApp found in the document will be used to define the root element to auto-bootstrap as an application. If you have placed another ng-app directive then it will not be processed by AngularJS. You need to manually bootstrap the second app, instead of using second ng-app directive. Read

Q: Can angular applications (ng-app) be nested within each other?
A: NO. AngularJS applications cannot be nested within each other.

Q: Can you bootstrap multiple angular applications on same element?
A: NO. If you try to do that then it will show an error "App Already Bootstrapped with this Element". This usually happens when you accidentally use both ng-app and angular.bootstrap to bootstrap an application. You can also get this error if you accidentally load AngularJS itself more than once.

Q: In how many different ways, you can define a directive and what is the best practice?
A: Angular generally prefers camelCase for directives. But since HTML is not case-sensitive so it refers to directive in DOM in lower case form, delimited by dash (eg. ng-app). But when Angular complies then it normalize the directives. Below are example of valid directive declaration.

ng-model
ngModel
ng:model
ng_model
data-ng-model
x-ng-model

The normalization process is as follows:
1. Strip x- and data- from the front of the element/attributes.
2. Convert the :, -, or _-delimited name to camelCase.

The best practice to use dash-delimited (ng-model) or directly camelCase form (ngModel). If you are using HTML validation tool, then it is advised to use data- prefixed version. And it also answers another question which is "Difference between ng-* and data-ng-*".

Q: Mention some angularJS directives and their purpose?
A: The beauty of AngularJS directives is that they are self explainatory. By just looking at directive name, you will get the idea about purpose and use of directive. Below are some mostly used
directives.
ng-app : Initializes application.
ng-model : Binds HTML controls to application data.
ng-Controller : Attaches a controller class to view.
ng-repeat : Bind repeated data HTML elements. Its like a for loop.
ng-if : Bind HTML elements with condition.
ng-show : Used to show the HTML elements.
ng-hide : Used to hide the HTML elements.
ng-class : Used to assign CSS class.
ng-src : Used to pass the URL image etc.

Event Listeners
ng-click : Click event to bind on HTML elements.
ng-dbl-click
Mouse event listeners
ng-mousedown
ng-mouseup
ng-mouseenter
ng-mouseleave
ng-mousemove
ng-mouseover
Keyboard event listeners
ng-keydown
ng-keyup
ng-keypress
ng-change

Q: What is Angular Expression?
A: Angular expressions are JavaScript-like code snippets that are usually placed in bindings such as {{ expression }}. For example, these are valid expressions in Angular:

```
{{ 3+4 }}
{{ a+b }}
{{ user.name }}
{{ items[index] }}
```

Q: How Angular expressions are different from JavaScript expressions?
A: Angular expressions are like JavaScript expressions but there are few differences.
Context : In Angular, the expressions are evaluated against a scope object, while the JavaScript expressions are evaluated against the global window object.
Forgiving: In Angular expression evaluation is forgiving to null and undefined, while in JavaScript undefined properties generates TypeError or ReferenceError.
No Control Flow Statements: Loops, conditionals or exceptions cannot be used in an Angular expression.
No Comma And Void Operators: You cannot use , (comma) or void in an Angular expression. And You cannot create regular expressions in an Angular expression.

Q: What is compilation process in Angular?
A: Once you have the markup, the AngularJS needs to attach the functionality. This process is called "compilation" in Angular. Compiling includes rendering of markup, replacing directives, attaching events to directives and creating a scope. The AngularJS has compiler service which traverses the DOM looking for attributes. The compilation process happens in two phases.

Compilation : traverse the DOM and collect all of the directives and creation of the linking function.

Linking: combine the directives with a scope and produce a live view. The linking function allows for the attaching of events and handling of scope. Any changes in the scope model are reflected in the view, and any user interactions with the view are reflected in the scope model.

When you create a new directive, you can write compile and/or linking functions for it to attach your custom behavior. To understand the compilation process of Angular, must read "The nitty-gritty of compile and link functions inside AngularJS directives".

Q: What is scope in AngularJS?
A: A scope is an object that ties a view (a DOM element) to the controller. In the MVC framework, scope object is your model. In other words, it is just a JavaScript object, which is used for communication between controller and view.

Q: What is $rootscope in AngularJS?
A: 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. Since $rootScope is a global, which means that anything you add here, automatically becomes available in $scope in all controller. To add something in $rootScope, you need to use app.run function which ensures that it will run prior to the rest of the app. You may say that "run" function is like "main" method of angular app.

```
app.run(function($rootScope) {
  $rootScope.name = "AngularJS";
});
```

And then you can use in your view. (via expression)

```
<body ng-app="myApp">
  <h1>Hello {{ name }}!</h1>
</body>
```

Q: What are controllers in AngularJS?
A: In Angular, a Controller is a JavaScript constructor function. When a Controller is attached to the DOM via the ng-controller directive, Angular will instantiate a new Controller object, using the specified Controller's constructor function. The job of a controller is to pass data from the model, to the view or the view can asks for something from the controller, and the controller turns to the model and takes that thing, and hands it back to the view.

```
var myApp = angular.module('myApp', []);
myApp.controller('MyController', ['$scope', function($scope) {
   $scope.website = 'jquerybyexample.net';
  }
]);
```

And then in your HTML using ng-controller directive,

```
<div ng-controller="MyController">
  <h1>My website address is {{ website }}!</h1>
</div>
```

Q: What is the difference between $scope and scope in AngularJS?
A: $scope is used while defining a controller (see previous question) where scope is used while creating a link function for custom directive. The common part is that they both refers to scope object in AngularJS, but the difference is that $scope uses dependency injection where scope doesn't. When the arguments are passed-in via dependency injection, their position doesn't matter. So for example, a controller defined as ($scope as first parameter)

```
myApp.controller('MyController', ['$scope', function($scope, $http) {
```

OR ($scope is second parameter)

```
myApp.controller('MyController', ['$scope', function($http, $scope) {
```

In both the case, the postion of $scope doesn't matter to AngularJS. Because AngularJS uses the argument name to get something out of the dependency-injection container and later use it.

But in case of link function, the position of scope does matter because it doesn't uses DI. The very first parameter has to

be scope and that's what AngularJS expects.

```
app.directive("myDirective", function() {
  return {
    scope: {};
    link: function(scope, element, attrs) {
      // code goes here.
    }
  };
});
```

However you can name it anything of your choice. In below code, foo is your scope object.

```
link: function(foo, bar, baz) {
  // code goes here.
}
```

Q: What is MVC Architecture in AngularJS?

A: In AngularJS, scope objects are treated as Model. The View is display of model that is your data. And the model gets initialized within a JavaScript constructor function, called Controller in AngularJS. Let take a look at below code to understand it better.

```
<!DOCTYPE html>
<html>
<head>
  <script data-require="angular.js@*" data-semver="1.3.6" src="https://code.angularjs.org/1.3.6/angular.js">
</script>
  <link rel="stylesheet" href="style.css" />
  <script>
    var myApp = angular.module('myApp', []);
    myApp.controller('MyController', ['$scope',
      function($scope) {
        $scope.website = 'jquerybyexample.net';
      }
    ]);
  </script>
</head>

<body ng-app="myApp">
  <div ng-controller="MyController">
    <h1>My website address is {{ website }}</h1>;
  </div>
</body>
</html>
```

Here MyController is a Controller and $scope (Model) is populated within Controller. And later on in div element $scope object data is displayed (View).

Q: Can we have nested controllers in AngularJS?

A: YES. We can create nested controllers in AngularJS. Nested controller are defined in hierarchical manner while using in View. Take a look at below code. Here the hierarchy is "MainCtrl -> SubCtrl -> SubCtrl1".

```
<div ng-controller="MainCtrl">
  <p>{{message}} {{name}}!</p>

  <div ng-controller="SubCtrl">
    <p>Hello {{name}}!</p>

    <div ng-controller="SubCtrl2">
      <p>{{message}} {{name}}! Your username is  {{username}}.</p>
```

```
    </div>
  </div>
</div>
```

Q: In case of nested controllers, does the $scope object shared across all controllers?
A: YES. The $scope object is shared across all controllers and it happens due to scope inheritance. Since the ng-controller directive creates a new child scope, we get a hierarchy of scopes that inherit from each other. So if we define a property on a parent scope, the child scope can manipulate the property. And if the property is not present in child scope, then parent scope property's value is used. Lets consider, the previous question HTML where there are 3 controllers. And here is the AngularJS code to define these controllers.

```
var myApp = angular.module('myApp', []);
myApp.controller('MainCtrl', ['$scope',
  function($scope) {
    $scope.message = 'Welcome';
    $scope.name = 'Jon';
  }
]);
myApp.controller('SubCtrl', ['$scope',
  function($scope) {
    $scope.name = 'Adams';
  }
]);
myApp.controller('SubCtrl2', ['$scope',
  function($scope) {
    $scope.name = 'Ema';
    $scope.username = 'ema123';
  }
]);
```

You will see that the controller "SubCtrl2" doesn't have "message" property define but it is used in HTML. So in this case, the immediate parent scope property will be used. But immediate parent scope which is "SubCtrl" in this case, also doesn't have "message" property. So it again goes one level up and finds the property is present so it uses parent scope value for "message" property and displays it.

Q: What are different ways of bootstrapping AngularJS?
A: is neat and superheroic JavaScript MVW (Model View Whatever) Framework which allows to extend HTML capabilities with Directives, expression, two way binding. In this post, we will see different ways of bootstrapping AngularJS. Bootstrapping in AngularJS is nothing but just the initialization of Angular app.
There are two ways of bootstrapping AngularJS. One is Automatic Initialization and other is Manually using Script.
When you add ng-app directive to the root of your application, typically on the <html> tag or <body> tag if you want angular to auto-bootstrap your application.

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

When angularJS finds ng-app directive, it loads the module associated with it and then compile the DOM.
Another way to bootstrapping is manually initializing using script. Manual initialization provides you more control on how and when to initialize angular App. It is useful when you want to perform any other operation before Angular wakes up and compile the page. Below is the script which allows to manually bootstrap angularJS.

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

If you have noticed, it is using document.ready which is great as it will make sure that before bootstrapping your DOM is ready. You don't need to inclued jQuery library reference here, as angularJS has within it. So angular.bootstrap function bootstrap angularJS to document. The second parameter is the name of module. You need to take care of following things while using manual process.

Remember angular.bootstrap function will not create modules on the go. So you need to have your modules define, before you manually bootstrap. So below is the correct approach. First define the module and then bootstrap.

```
<script>
  angular.module('myApp', [])
    .controller('MyController', ['$scope', function ($scope) {
      $scope.message= 'Hello World';
    }]);

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

You cannot add controllers, services, directives, etc after an application bootstraps.

AngularJS Interview Questions and Answers for Experienced

Q: What is AngularJS?
A: It is javasScript framework which is written in javascript. It is Best for Single Page Applications. It extend the html with new attributes which makes it more useful for UI Developer.

Q: In which language, AngularJS is written?
A: javaScript

Q: When First AngularJS was released?
A: 2009

Q: When latest AngularJS was released?
A: October 31, 2014

Q: What is latest version of AngularJS?
A: 1.3.1

Q: Who created AngularJS?
A: Misko Hevery started to work on AngularJS in 2009. He was employee of Google.

Q: Is it opensource?
A: Yes, It is free to use.

Q: Explain what are the key features of Angular.js?
A: Scope
Controller
Model
View
Services
Data Binding
Directives
Filters
Testable

Q: From where we can download the AngularJS File?
A: <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js"></script>

Q: What is controller in AngularJS?

A: Controller is constructor function in Angular Controller.
When a Controller is attached to the DOM with use the ng-controller, Angular will instantiate a new Controller object using constructor function.

Q: Explain what are directives?
A: Directives are used to add new attributes of HTML.

Q: What are the different types of Directive?
A: Different types of directives are
Element directives
Attribute directives
CSS class directives
Comment directives

Q: Explain what is injector?
A: An injector is a service locator, used to retrieve object instances.

Q: Explain what are factory method in angularJs?
A: Factory method are used to create the directive. It is invoked only once, when compiler matches the directive for the first time.

Q: Does Angular use the jQuery library?
A: Yes, Angular can use jQuery if you have included the jQuery library.
IF Not, Angular falls back to its own implementation of the subset of jQuery that we call jQLite.

Q: What is ng-app, ng-init and ng-model?
A: ng-app - To initialize the Angular Application.
ng-init - To initialize the Angular Application data.
ng-model - To bind the html tags (input, select, textarea) to Angular Application Data.

Q: What is Data Binding in Angular JS?
A: It is synchronization of data between the model(Angular Application variable) and view components (display with {{}}).

Q: Give an Example of Data-Binding in AngularJS?
A:
Total Cost: 50

Q: What is Looping in AngularJs and Give an Example?
A: It is used to display the data in loop same as foreach in PHP
Example:

```
<div data-ng-app="" data-ng-init="names=['Web','Technology','Experts','Notes']">
<b>Loop Example:</b>
<ul>
<li data-ng-repeat="x in names">
   {{ x }}
  </li>
</ul>
</div>
```

Q: How to Write Expression in AngularJS?
A:

```
<div ng-app="">
        <b>Expression: {{ 15 + 55 }}</b>
</div>
```

Q: How to initiate variable in AngularJS?
A:
Total Cost: {{ quantity * cost }}

OR

Total Cost:

Q: Example of AngularJS Strings?
A:
Full String is : {{ Str1 + " " + Str2 }}

Q: Example of AngularJS Object?
A:
String Display: {{ myobj.Str2 }}

Q: What is Angular Controllers & give an Example?
A: Controller is constructor function in Angular Controller.
When a Controller is attached to the DOM with use the ng-controller, Angular will instantiate a new Controller object using constructor function.
Example:

```
<div ng-app="" ng-controller="StrControllerExample">
        String 1: <input ng-model="str1" type="text">
        String 2: <input ng-model="str2" type="text">
        Full String <b> {{fullString()}}</b>
</div>

<script>
        function StrControllerExample($scope) {
                $scope.str1 = "Web",
                $scope.str2 = "Technology",
                $scope.fullString = function() {
                        return $scope.str1+ " " + $scope.str2;
                }
        }
</script>
```

Q: What is AngularJS?
A: AngularJS is an open-source JavaScript framework, maintained by Google, that assists with running single-page applications. Its goal is to augment browser-based applications with model–view–controller capability, in an effort to make both development and testing easier.

Q: Can you please explain what is testability like in Angular?
A: Very testable and designed this way from ground up. It has an integrated dependency injection framework, provides mocks for many heavy dependencies (server-side communication).

Q: Why is this project called "AngularJS"? Why is the namespace called "ng"?
A: Because HTML has Angular brackets and "ng" sounds like "Angular".

Q: Tell me does Angular use the jQuery library?
A: Yes, Angular can use jQuery if it's present in your app when the application is being bootstrapped. If jQuery is not present in your script path, Angular falls back to its own implementation of the subset of jQuery that we call jQLite.

Q: Is AngularJS a library, framework, plugin or a browser extension?
A: AngularJS fits the definition of a framework the best, even though it's much more lightweight than a typical framework and that's why many confuse it with a library.
AngularJS is 100% JavaScript, 100% client side and compatible with both desktop and mobile browsers. So it's definitely not a plugin or some other native browser extension.

Q: Tell me can we use the open-source Closure Library with Angular?
A: Yes, you can use widgets from the Closure Library in Angular.

Q: Do you know what is Angulars performance like?
A: The startup time heavily depends on your network connection, state of the cache, browser used and available hardware, but typically we measure bootstrap time in tens or hundreds of milliseconds.
The runtime performance will vary depending on the number and complexity of bindings on the page as well as the speed of your backend (for apps that fetch data from the backend). Just for an illustration we typically build snappy apps with hundreds or thousands of active bindings.

Q: Tell me which browsers does Angular work with?
A: We run our extensive test suite against the following browsers: Safari, Chrome, Firefox, Opera, IE8, IE9 and mobile browsers (Android, Chrome Mobile, iOS Safari). See Internet Explorer Compatibility for more details in supporting legacy IE browsers.

Q: Why to choose Angular JS Javascript Framework for front-end web development?
A: AngularJS is quickly becoming the dominant JavaScript framework for professional web development. With the growth and strength of HTML5 and the increasing performance in modern browsers, many JavaScript frameworks have been created to help develop rich client applications. These frameworks/libraries have given developers a huge toolkit to build enterprise complexity into client-side applications. Server side frameworks are becoming a thing of the past and being replaced with applications written in Backbone, Ember, AngularJS, Knockout, etc.So why am I talking about AngularJS over frameworks/libraries like Backbone, Ember, or Knockout?
For me, the major points of separation in AngularJS's favor are the following:
1. Good documentation
2. Write less code to do more
3. Backed by Google
4. Good developer community
5. Simple Data-Binding
6. Small footprint

If you're looking for a robust, well-maintained framework for any sized project, I strongly recommend that you take a look at AngularJS. It can be downloaded for free at AngularJS.org, which also contains a wealth of information, including the full API documentation, as well as numerous examples and tutorials that cover every facet of front-end web development. Following are some reasons

1. Angular JS Framework is developed by Google
Angular is built and maintained by dedicated Google engineers. This one may seem obvious, but it's important to remember that many (not all) frameworks are made by hobbyists in the open source community. While passion and drive have forged frameworks, like Cappucino and Knockout, Angular is built and maintained by dedicated (and highly talented) Google engineers. This means you not only have a large open community to learn from, but you also have skilled, highly-available engineers tasked to help you get your Angular questions answered.
This isn't Google's first attempt at a JavaScript framework; they first developed their comprehensive Web Toolkit, which compiles Java down to JavaScript, and was used by the Google Wave team extensively. With the rise of HTML5, CSS3, and JavaScript, as both a front-end and back-end language, Google realized that the web was not meant to be written

purely in Java.
AngularJS came about to standardize web application structure and provide a future template for how client-side apps should be developed.
Angular JS is being used by a host of applications, ranging from hobby to commercial products. Adoption of AngularJS as a viable framework for client-side development is quickly becoming known to the entire web development community. Because AngularJS is built by Google, you can be sure that you're dealing with efficient and reliable code that will scale with your project. If you're looking for a framework with a solid foundation, Angular is your choice!

2. Angular JS is equipped with a lot of features
If you're familiar with projects, like QUnit, Mocha or Jasmine, then you'll have no trouble learning Angular's unit-testing API.
Angular, similar to Backbone or JavaScriptMVC, is a complete solution for rapid front-end development. No other plugins or frameworks are necessary to build a data-driven web application. Here's an overview of Angular's stand-out features:

A) REST Easy. RESTful actions are quickly becoming the standard for communicating from the server to the client. In one line of JavaScript, you can quickly talk to the server and get the data you need to interact with your web pages. AngularJS turns this into a simple JavaScript object, as Models, following the MVVM (Model View View-Model) pattern.
B) MVVM to the Rescue! Models talk to ViewModel objects (through something called the $scope object), which listen for changes to the Models. These can then be delivered and rendered by the Views, which is the HTML that expresses your code. Views can be routed using the $routeProvider object, so you can deep-link and organize your Views and Controllers, turning them into navigable URLs. AngularJS also provides stateless controllers, which initialize and control the $scope object.
C) Data Binding and Dependency Injection. Everything in the MVVM pattern is communicated automatically across the UI whenever anything changes. This eliminates the need for wrappers, getters/setters or class declarations. AngularJS handles all of this, so you can express your data as simply as with JavaScript primitives, like arrays, or as complex as you wish, through custom types. Since everything happens automatically, you can ask for your dependencies as parameters in AngularJS service functions, rather than one giant main() call to execute your code.
D) Extends HTML. Most websites built today are a giant series of tags with little semantic clarity. You need to create extensive and exhaustive CSS classes to express the intention of each object in the DOM. With Angular, you can operate your HTML like XML, giving you endless possibilities for tags and attributes. Angular accomplishes this, via its HTML compiler and the use of directives to trigger behaviors based on the newly-created syntax you write.
E) Makes HTML your Template. If you're used to Mustache or Hogan.js, then you can quckly grasp the bracket syntax of Angular's templating engine, because it's just HTML. Angular traverses the DOM for these templates, which house the directives mentioned above. The templates are then passed to the AngularJS compiler as DOM elements, which can be extended, executed or reused. This is key, because, now, you have raw DOM components, rather than strings, allowing for direct manipulation and extension of the DOM tree.
F) Enterprise-level Testing. As stated above, AngularJS requires no additional frameworks or plugins, including testing. If you're familiar with projects, like QUnit, Mocha or Jasmine, then you'll have no trouble learning Angular's unit-testing API and Scenario Runner, which guides you through executing your tests in as close to the actual state of your production application as possible.

These are the fundamental principles that guide AngularJS to creating an efficient, performance-driven, and maintainable front-end codebase. As long as you have a source for storing data, AngularJS can do all of the heavy lifting on the client, while providing a rich, fast experience for the end user.

3. You can learn Angular JS easily
Getting started with AngularJS is incredibly easy. With a few attributes added to your HTML, you can have a simple Angular app up in under 5 minutes!
1. Add the ng-app directive to the <html> tag so Angular knows to run on the page:
<html lang="en" ng-app>
2. Add the Angular <script> tag to the end of your <head> tag:
<head>
<script src="lib/angular/angular.js"></script>
...

```
</head>
```
3. Add regular HTML. AngularJS directives are accessed through HTML attributes, while expressions are evaluated with double-bracket notation:

```
<body ng-controller="ActivitiesListCtrl">
  <h1>Today's activities</h1>
  <ul>
   <li ng-repeat="activity in activities">
    {{activity.name}}
   </li>
  </ul>
</body>
</html>
```

Q: What are the key features of AngularJS?

A: Scope

The job of the Scope is to detect changes to model objects and create an execution context for expressions. There is one root scope for the application (ng-app) with hierarchical children scopes. It marshals the model to the view and forwards events to the controller.

Controller

The Controller is responsible for construction of the model and connects it to the view (HTML). The scope sits between the controller and the view. Controllers should be straightforward and simply contain the business logic needed for a view. Generally you want thin controllers and rich services. Controllers can be nested and handle inheritance. The big difference in AngularJS from the other JavaScript frameworks is there is no DOM manipulation in controllers. It is something to unlearn when developing in AngularJS.

Model

In AngularJS, a Model is simply a JavaScript object. No need to extend anything or create any structure. This allows for nested models - something that Backbone doesn't do out-of-the-box.

View

The View is based on DOM objects, not on strings. The view is the HTML. HTML is declarative – well suited for UI design. The View should not contain any functional behavior. The flexibility here is to allow for multiple views per Controller.

Services

The Services in AngularJS are singletons that perform common tasks for web applications. If you need to share common functionality between Controllers, then use Services. Built-in AngularJS, Services start with a $. There are several ways to build a service: Service API, Factory API, or the $provide API.

Data Binding

Data Binding in AngularJS is a two-way binding between the View and the Model. Automatic synchronizing between views and data models makes this really easy (and straightforward) to use. Updating the model is reflected in View without any explicit JavaScript code to bind them together, or to add event listeners to reflect data changes.

Directives

Now this is cool. AngularJS allows you to use Directives to transform the DOM or to create new behavior. A directive allows you to extend the HTML vocabulary in a declarative fashion. The 'ng' prefix stands for built-in AngularJS directives. The App (ng-app), Model (ng-model), the Controller (ng-controller), etc. are built into the framework. AngularJS allows for building your own directives. Building directives is not extremely difficult, but not easy either. There are different things that can be done with them. Please check out AngularJS's documentation on directives.

Filters

The Filters in AngularJS perform data transformation. They can be used to do formatting (like I did in my Directives example with padding zeros), or they can be used to do filter results (think search).

Validation
AngularJS has some built-in validation around HTML5 input variables (text, number, URL, email, radio, checkbox) and some directives (required, pattern, minlength, maxlength, min, max). If you want to create your own validation, it is just as simple as creating a directive to perform your validation.

Testable
Testing is a big concern for enterprise applications. There are several different ways to write and run tests against JavaScript code, thus against AngularJS. The developers at AngularJS advocate using Jasmine tests ran using Testacular. I have found this method of testing very straightforward and, while writing tests may not be the most enjoyable, it is just as importable as any other piece of developing an application.

Q: What is a scope in AngularJS?
A: scope is an object that refers to the application model. It is the glue between application controller and the view. Both the controllers and directives have reference to the scope, but not with each other. It is an execution context for expressions and arranged in hierarchical structure. Scopes can watch expressions and propagate events.

Q: Can you explain the concept of scope hierarchy? How many scopes can an application have?
A: Each Angular application has exactly one root scope, but may have several child scopes. The application can have multiple scopes, because child controllers and some directives create new child scopes. When new scopes are created, they are added as children of their parent scope. This creates a hierarchical structure similar to the DOM where they're attached.

When Angular evaluates a bound variable like say {{firstName}}, it first looks at the scope associated with the given element for the firstName property. If no such property is found, it searches the parent scope and so on until the root scope is reached. In JavaScript this behaviour is known as prototypical inheritance, and child scopes prototypically inherit from their parents. The reverse is not true. i.e. the parent can't see it's children's bound properties.

Q: Is AngularJS a templating system?
A: At the highest level, Angular does look like a just another templating system. But there is one important reason why the Angular templating system is different, that makes it very good fit for application development: bidirectional data binding. The template is compiled in the browser and the compilation step produces a live view. This means you, the developers, don't need to write code to constantly sync the view with the model and the model with the view as in other templating systems.

Q: Do I need to worry about security holes in AngularJS?
A: Like any other technology, AngularJS is not impervious to attack. Angular does, however, provide built-in protection from basic security holes including cross-site scripting and HTML injection attacks. AngularJS does round-trip escaping on all strings for you and even offers XSRF protection for server-side communication.

AngularJS was 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 attack vectors and we highly recommended their use.

Q: What are the key differences between AngularJS and jQuery?
A: AngularJS and jQuery are the Javascript frameworks and are different with each other, so never mix up the AngularJS and jQuery code in your project. Use only one Javascript framework at a time. If you are starting a new project, must consider AngularJS over jQuery. If you are a experienced jQuery developer, then you have to invest some time to work in AngularJS way. There are a lot of difference between AngularJS and jQuery.

1. Web designing approach in jQuery and AngularJS

In jQuery, you design a page, and then you make it dynamic. This is because jQuery was designed for augmentation and has grown incredibly from that simple premise.

But in AngularJS, you must start from the ground up with your architecture in mind. Instead of starting by thinking "I have this piece of the DOM and I want to make it do X", you have to start with what you want to accomplish, then go about designing your application, and then finally go about designing your view.

2. Don't augment jQuery with AngularJS

Similarly, don't start with the idea that jQuery does X, Y, and Z, so I'll just add AngularJS on top of that for models and controllers. This is really tempting when you're just starting out, which is why I always recommend that new AngularJS developers don't use jQuery at all, at least until they get used to doing things the "Angular Way".

I've seen many developers here and on the mailing list create these elaborate solutions with jQuery plugins of 150 or 200 lines of code that they then glue into AngularJS with a collection of callbacks and $applys that are confusing and convoluted; but they eventually get it working! The problem is that in most cases that jQuery plugin could be rewritten in AngularJS in a fraction of the code, where suddenly everything becomes comprehensible and straightforward.

The bottom line is this: when solutioning, first "think in AngularJS"; if you can't think of a solution, ask the community; if after all of that there is no easy solution, then feel free to reach for the jQuery. But don't let jQuery become a crutch or you'll never master AngularJS.

3. Always think in terms of architecture

First know that single-page applications are applications. They're not webpages. So we need to think like a server-side developer in addition to thinking like a client-side developer. We have to think about how to divide our application into individual, extensible, testable components.

So then how do you do that? How do you "think in AngularJS"? Here are some general principles, contrasted with jQuery.

The view is the "official record"

In jQuery, we programmatically change the view. We could have a dropdown menu defined as a ul like so:

```
<ul class="main-menu">
  <li class="active">
    <a href="#/home">Home</a>
  </li>
  <li>
    <a href="#/menu1">Menu 1</a>
    <ul>
      <li><a href="#/sm1">Submenu 1</a></li>
      <li><a href="#/sm2">Submenu 2</a></li>
      <li><a href="#/sm3">Submenu 3</a></li>
    </ul>
  </li>
  <li>
    <a href="#/home">Menu 2</a>
  </li>
</ul>
```

In jQuery, in our application logic, we would activate it with something like:

```
$('.main-menu').dropdownMenu();
```

When we just look at the view, it's not immediately obvious that there is any functionality here. For small applications,

that's fine. But for non-trivial applications, things quickly get confusing and hard to maintain.

In AngularJS, though, the view is the official record of view-based functionality. Our ul declaration would look like this instead:

```
<ul class="main-menu" dropdown-menu>
  ...
</ul>
```

These two do the same thing, but in the AngularJS version anyone looking at the template knows what's supposed to happen. Whenever a new member of the development team comes on board, he can look at this and then know that there is a directive called dropdownMenu operating on it; he doesn't need to intuit the right answer or sift through any code. The view told us what was supposed to happen. Much cleaner.

Developers new to AngularJS often ask a question like: how do I find all links of a specific kind and add a directive onto them. The developer is always flabbergasted when we reply: you don't. But the reason you don't do that is that this is like half-jQuery, half-AngularJS, and no good. The problem here is that the developer is trying to "do jQuery" in the context of AngularJS. That's never going to work well. The view is the official record. Outside of a directive (more on this below), you never, ever, never change the DOM. And directives are applied in the view, so intent is clear.

Remember: don't design, and then mark up. You must architect, and then design.

Data binding
This is by far one of the most awesome features of AngularJS and cuts out a lot of the need to do the kinds of DOM manipulations I mentioned in the previous section. AngularJS will automatically update your view so you don't have to! In jQuery, we respond to events and then update content. Something like:

```
$.ajax({
  url: '/myEndpoint.json',
  success: function ( data, status ) {
    $('ul#log').append('<li>Data Received!</li>');
  }
});
```

For a view that looks like this:

```
<ul class="messages" id="log">
</ul>
```

Apart from mixing concerns, we also have the same problems of signifying intent that I mentioned before. But more importantly, we had to manually reference and update a DOM node. And if we want to delete a log entry, we have to code against the DOM for that too. How do we test the logic apart from the DOM? And what if we want to change the presentation?
This a little messy and a trifle frail. But in AngularJS, we can do this:

```
$http( '/myEndpoint.json' ).then( function ( response ) {
  $scope.log.push( { msg: 'Data Received!' } );
});
```

And our view can look like this:

```
<ul class="messages">
  <li ng-repeat="entry in log">{{ entry.msg }}</li>
</ul>
```

But for that matter, our view could look like this:

```
<div class="messages">
  <div class="alert" ng-repeat="entry in log">
    {{ entry.msg }}
  </div>
</div>
```

And now instead of using an unordered list, we're using Bootstrap alert boxes. And we never had to change the controller code! But more importantly, no matter where or how the log gets updated, the view will change too. Automatically. Neat!

Though I didn't show it here, the data binding is two-way. So those log messages could also be editable in the view just by doing this:
<input ng-model="entry.msg" />.
And there was much rejoicing.

Distinct model layer
In jQuery, the DOM is kind of like the model. But in AngularJS, we have a separate model layer that we can manage in any way we want, completely independently from the view. This helps for the above data binding, maintains separation of concerns, and introduces far greater testability. Other answers mentioned this point, so I'll just leave it at that.

Separation of concerns
And all of the above tie into this over-arching theme: keep your concerns separate. Your view acts as the official record of what is supposed to happen (for the most part); your model represents your data; you have a service layer to perform reusable tasks; you do DOM manipulation and augment your view with directives; and you glue it all together with controllers. This was also mentioned in other answers, and the only thing I would add pertains to testability, which I discuss in another section below.

Dependency injection
To help us out with separation of concerns is dependency injection (DI). If you come from a server-side language (from Java to PHP) you're probably familiar with this concept already, but if you're a client-side guy coming from jQuery, this concept can seem anything from silly to superfluous to hipster. But it's not.

From a broad perspective, DI means that you can declare components very freely and then from any other component, just ask for an instance of it and it will be granted. You don't have to know about loading order, or file locations, or anything like that. The power may not immediately be visible, but I'll provide just one (common) example: testing.

Let's say in our application, we require a service that implements server-side storage through a REST API and, depending on application state, local storage as well. When running tests on our controllers, we don't want to have to communicate with the server - we're testing the controller, after all. We can just add a mock service of the same name as our original component, and the injector will ensure that our controller gets the fake one automatically - our controller doesn't and needn't know the difference.

4. Test-driven development
This is really part of section 3 on architecture, but it's so important that I'm putting it as its own top-level section.
Out of all of the many jQuery plugins you've seen, used, or written, how many of them had an accompanying test suite? Not very many because jQuery isn't very amenable to that. But AngularJS is.
In jQuery, the only way to test is often to create the component independently with a sample/demo page against which our tests can perform DOM manipulation. So then we have to develop a component separately and then integrate it into our application. How inconvenient! So much of the time, when developing with jQuery, we opt for iterative instead of test-driven development. And who could blame us?
But because we have separation of concerns, we can do test-driven development iteratively in AngularJS! For example, let's say we want a super-simple directive to indicate in our menu what our current route is. We can declare what we want

in our view:
Hello

Okay, now we can write a test:

```
it( 'should add "active" when the route changes', inject(function() {
   var elm = $compile( '<a href="/hello" when-active>Hello</a>' )( $scope );

   $location.path('/not-matching');
   expect( elm.hasClass('active') ).toBeFalsey();

   $location.path( '/hello' );
   expect( elm.hasClass('active') ).toBeTruthy();
}));
```

We run our test and confirm that it fails. So now we can write our directive:

```
.directive( 'whenActive', function ( $location ) {
   return {
      scope: true,
      link: function ( scope, element, attrs ) {
         scope.$on( '$routeChangeSuccess', function () {
            if ( $location.path() == element.attr( 'href' ) ) {
               element.addClass( 'active' );
            }
            else {
               element.removeClass( 'active' );
            }
         });
      }
   };
});
```

Our test now passes and our menu performs as requested. Our development is both iterative and test-driven.

5. Conceptually, directives are not packaged jQuery

You'll often hear "only do DOM manipulation in a directive". This is a necessity. Treat it with due deference!

But let's dive a little deeper...

Some directives just decorate what's already in the view (think ngClass) and therefore sometimes do DOM manipulation straight away and then are basically done. But if a directive is like a "widget" and has a template, it should also respect separation of concerns. That is, the template too should remain largely independent from its implementation in the link and controller functions.

AngularJS comes with an entire set of tools to make this very easy; with ngClass we can dynamically update the class; ngBind allows two-way data binding; ngShow and ngHide programmatically show or hide an element; and many more - including the ones we write ourselves. In other words, we can do all kinds of awesomeness without DOM manipulation. The less DOM manipulation, the easier directives are to test, the easier they are to style, the easier they are to change in the future, and the more re-usable and distributable they are.

I see lots of developers new to AngularJS using directives as the place to throw a bunch of jQuery. In other words, they think "since I can't do DOM manipulation in the controller, I'll take that code put it in a directive". While that certainly is much better, it's often still wrong.

Think of the logger we programmed in section 3. Even if we put that in a directive, we still want to do it the "Angular Way". It still doesn't take any DOM manipulation! There are lots of times when DOM manipulation is necessary, but it's a lot rarer than you think! Before doing DOM manipulation anywhere in your application, ask yourself if you really need to. There

might be a better way.

Here's a quick example that shows the pattern I see most frequently. We want a toggleable button. (Note: this example is a little contrived and a skosh verbose to represent more complicated cases that are solved in exactly the same way.)

```
.directive( 'myDirective', function () {
  return {
    template: '<a class="btn">Toggle me!</a>',
    link: function ( scope, element, attrs ) {
      var on = false;

      $(element).click( function () {
        if ( on ) {
          $(element).removeClass( 'active' );
        }
        else {
          $(element).addClass( 'active' );
        }

        on = !on;
      });
    }
  };
});
```

There are a few things wrong with this. First, jQuery was never necessary. There's nothing we did here that needed jQuery at all! Second, even if we already have jQuery on our page, there's no reason to use it here; we can simply use angular.element and our component will still work when dropped into a project that doesn't have jQuery. Third, even assuming jQuery was required for this directive to work, jqLite (angular.element) will always use jQuery if it was loaded! So we needn't use the $ - we can just use angular.element. Fourth, closely related to the third, is that jqLite elements needn't be wrapped in $ - the element that is passed to the link function would already be a jQuery element! And fifth, which we've mentioned in previous sections, why are we mixing template stuff into our logic?
This directive can be rewritten (even for very complicated cases!) much more simply like so:

```
.directive( 'myDirective', function () {
  return {
    scope: true,
    template: '<a class="btn" ng-class="{active: on}" ng-click="toggle()">Toggle me!</a>',
    link: function ( scope, element, attrs ) {
      scope.on = false;

      scope.toggle = function () {
        scope.on = !scope.on;
      };
    }
  };
});
```

Again, the template stuff is in the template, so you (or your users) can easily swap it out for one that meets any style necessary, and the logic never had to be touched.

And there are still all those other benefits, like testing - it's easy! No matter what's in the template, the directive's internal API is never touched, so refactoring is easy. You can change the template as much as you want without touching the directive. And no matter what you change, your tests still pass.

So if directives aren't just collections of jQuery-like functions, what are they? Directives are actually extensions of HTML. If HTML doesn't do something you need it to do, you write a directive to do it for you, and then use it just as if it was part of HTML.

Put another way, if AngularJS doesn't do something out of the box, think how the team would accomplish it to fit right in with ngClick, ngClass, et al.

Q: How will you display different images based on the status being red, amber, or green?
A: Use the ng-switch and ng-switch-when directives as shown below.

```
<div ng-switch on="account.status">
 <div ng-switch-when="AMBER">
  <img class="statusIcon"
   src='apps/dashboard/amber-dot.jpg' />
 </div>
 <div ng-switch-when="GREEN">
  <img class="statusIcon"
   src='apps/dashboard/green-dot.jpg' />
 </div>
 <div ng-switch-when="RED">
  <img class="statusIcon"
   src='apps/dashboard/red-dot.jpg' />
 </div>
</div>
```

Q: How will you initialize a select box with options on page load?
A: Use the ng-init directive.

```
<div ng-controller="apps/dashboard/account" ng-switch
 on="!!accounts" ng-init="loadData()">
```

Q: How will you show/hide buttons and enable/disable buttons conditionally?
A: Using the ng-show and ng-disabled directives.

```
<div class="dataControlPanel" ng-show="accounts.releasePortfolios">
    <div class="dataControlButtons">
     <button class="btn btn-primary btn-small" ng-click="saveComments()" ng-
disabled="disableSaveButton">Save</button>
     <button class="btn btn-primary btn-small" ng-click="releaseRun()" ng-
disabled="disableReleaseButton">Release</button>
    </div>
</div>
```

Q: How will you loop through a collection and list each item?
A: Using the ng-repeat directive.

```
<table class="table table-bordered table-striped table-hover table-fixed-head portal-data-table">
  <thead>
   <tr>
    <th>account</th>
    <th>Difference</th>
    <th>Status</th>
   </tr>
  </thead>
  <tbody>
   <tr ng-repeat="account in acounts">
    <td width="40%">{{account.accountCode}}</td>
    <td width="30%" style="text-align: right">{{account.difference | currency: ""}}</td>
```

```
   <td width="30%">
    <div ng-switch on="account.status">
     <div ng-switch-when="AMBER">
      <img class="statusIcon" src='apps/dashboard/amber-dot.jpg' />
     </div>
     <div ng-switch-when="GREEN">
      <img class="statusIcon" src='apps/dashboard/green-dot.jpg' />
     </div>
     <div ng-switch-when="RED">
      <img class="statusIcon" src='apps/dashboard/red-dot.jpg' />
     </div>
    </div>
   </td>
  </tr>
 </tbody>
</table>
```

Q: How will you add options to a select box?
A: Using the ng-options and ng-model directives.

```
<fieldset>
 <dl class="control-group">
  <dt>
   <label for="cientId">
    <h4>Client Id:</h4>
   </label>
  </dt>
  <dd>
   <select id="cientId" class="input-xlarge" ng-model="clientId"
    ng-options="reportClient.clientId as reportClient.clientId  for reportClient in reportClients "
    ng-click="getReportParams()" ng-change="getValuationDates()" />
  </dd>
 </dl>
 <dl class="control-group">
  <dt>
   <label for="valuationDate">
    <h4>
     Valuation Date <small>(dd/mm/yyyy)</small>
    </h4>
   </label>
  </dt>
  <dd>
   <select id="valuationDate" class="input-xlarge"
    ng-model="valuationDate"
    ng-options="reportdate for reportdate in reportDates" />
  </dd>
 </dl>
</fieldset>
```

Q: How will you display inprogress revolving image to indicate that RESTful data is bing loaded?
A:

```
<div ng-show="loading">
        <img class="loading" src="portal/images/loading_32.gif" />
</div>
```

```
$scope.loadReportData = function($http) {
```

```
$scope.loading = true;  // start spinng the image
$http(
 {
  method : 'GET',
  url : propertiesService.get('restPath')
    + '/myapp/portfolio/'
    + $scope.clientId
    + '/'
    + dateService
      .userToRest($scope.reportDate),
  cacheBreaker : true
 }).success(
 function(data, config) {
  $scope.reportData = data;
  portal.log('reportData: ',
    $scope.reportData);
  $scope.loading = false;   // stop spinning the image
 }).error(
 function(data, status, headers, config) {
  if(data.errorMsg != null) {
   $scope.httpError = data.errorMsg;
  } else {
  $scope.httpError = "Error retrieving data from " + errorService
    .getApacheErrorTitleMessage(status,
      data, config);
    }
  $scope.loading = false;  // stop spinning the image
 });
};
```