What is AngularJS?

AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain. Following are the features of AngularJS framework.

- It is a powerful JavaScript based development framework to create RICH Internet Application (RIA).
- It provides developers options to write client side application (using JavaScript) in a clean MVC way.
- Application written in AngularJS is cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
- AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

What are the advantages of AngularJS?

- AngularJS provides capability to create Single Page Application in a very clean and maintainable way.
- AngularJS provides data binding capability to HTML thus giving user a rich and responsive experience.
- AngularJS code is unit testable.
- AngularJS uses dependency injection and make use of separation of concerns.
- AngularJS provides reusable components.
- With AngularJS, developer writes less code and gets more functionality.
- In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.
- AngularJS applications can run on all major browsers and smart phones including Android and iOS based phones/tablets.

What are the disadvantages of AngularJS?

Following are the disadvantages of AngularJS.

- **Not Secure** − Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
- **Not degradable** − If your application user disables JavaScript then user will just see the basic page and nothing more.

Explain AngularJS boot process.

When the page is loaded in the browser, following things happen:

- HTML document is loaded into the browser, and evaluated by the browser. AngularJS JavaScript file is loaded; the angular *global* object is created. Next, JavaScript which registers controller functions is executed.
- Next AngularJS scans through the HTML to look for AngularJS apps and views. Once view is located, it connects that view to the corresponding controller function.
- Next, AngularJS executes the controller functions. It then renders the views with data from the model populated by the controller. The page gets ready.

What is MVC?

Model **V**iew **C**ontroller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

- **Model** − It is the lowest level of the pattern responsible for maintaining data.
- **View** − It is responsible for displaying all or a portion of the data to the user.
- **Controller** − It is a software Code that controls the interactions between the Model and View.

How AngularJS integrates with HTML?

AngularJS being a pure javaScript based library integrates easily with HTML.

- **Step 1** − Include angularjs javascript libray in the html page
- **Step 2** − Point to AngularJS app

  Next we tell what part of the HTML contains the AngularJS app. This done by adding the *ng-app* attribute to the root HTML element of the AngularJS app. You can either add it to *html* element or *body* element as shown below:

  ```
  <body ng-app = "myapp"></body>
  ```

How angular.module works?

Angular.module is used to create angularjs modules along with its dependent modules. Consider the following example:

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

Here we've declared an application mainApp module using angular.module function. We've passed an empty array to it. This array generally contains dependent modules declared earlier.

What is data binding in AngularJS?

Data binding is the automatic synchronization of data between model and view components. Angular templates work differently. First the template (which is the uncompiled HTML along with any additional markup or directives) is compiled on the browser. The compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the

model are propagated to the view. The model is the single-source-of-truth for the application state, greatly simplifying the programming model for the developer. You can think of the view as simply an instant projection of your model. ng-model directive is used in data binding.

What is scope in AngularJS?

Scopes are objects that refer to the model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events. Scopes are objects that refer to the model. They act as glue between controller and view.

- Scope is the glue between application controller and the view. During the template linking phase the directives set up $watch expressions on the scope. The $watch allows the directives to be notified of property changes, which allows the directive to render the updated value to the DOM.
- Both controllers and directives have reference to the scope, but not to each other. This arrangement isolates the controller from the directive as well as from the DOM. This is an important point since it makes the controllers view agnostic, which greatly improves the testing story of the applications.
- Scopes provide APIs ($watch) to observe model mutations.
- Scopes provide APIs ($apply) to propagate any model changes through the system into the view from outside of the "Angular realm" (controllers, services, Angular event handlers).
- Each Angular application has exactly one root scope, but may have several child scopes.
- The application can have multiple scopes, because some directives create new child scopes (refer to directive documentation to see which directives create new scopes). When new scopes are created, they are added as children of their parent scope. This creates a tree structure which parallels the DOM where they're attached.
- Scopes can be nested to limit access to the properties of application components while providing access to shared model properties. Nested scopes are either "child scopes" or "isolate scopes". A "child scope" (prototypically) inherits properties from its parent scope. An "isolate scope" does not. See isolated scopes for more information.
- Scopes provide context against which expressions are evaluated. For example {{username}} expression is meaningless, unless it is evaluated against a specific scope which defines the username property.

What are the controllers in AngularJS?

Controllers are JavaScript functions that are bound to a particular scope. They are the prime actors in AngularJS framework and carry functions to operate on data and decide which view is to be updated to show the updated model based data.

Do not use controllers to:

- Manipulate DOM — Controllers should contain only business logic. Putting any presentation logic into Controllers significantly affects its testability. Angular has databinding for most cases and directives to encapsulate manual DOM manipulation.
- Format input — Use angular form controls instead.
- Filter output — Use angular filters instead.
- Share code or state across controllers — Use angular services instead.
- Manage the life-cycle of other components (for example, to create service instances).

What are the filters in AngularJS?

Filters format the value of an expression for display to the user. They can be used in view templates, controllers or services. Angular comes with a collection of built-in filters, but it is easy to define your own as well.

AngularJS Filters

AngularJS provides filters to transform data:

- currency Format a number to a currency format.
- date Format a date to a specified format.
- filter Select a subset of items from an array.
- json Format an object to a JSON string.
- limitTo Limits an array/string, into a specified number of elements/characters.
- lowercase Format a string to lower case.
- number Format a number to a string.
- orderBy Orders an array by an expression.
- uppercase Format a string to upper case.

Explain directives in AngularJS.

Directives are markers on DOM elements (such as elements, attributes, css, and more). These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives (ng-bind, ng-model, etc) to perform most of the task that developers have to do.

Explain templates in AngularJS.

Templates are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials".

What is routing in AngularJS?

It is concept of switching views. AngularJS based controller decides which view to render based on the business logic.

What is deep linking in AngularJS?

Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.

Which are the core directives of AngularJS?

Following are the three core directives of AngularJS.

- **ng-app** − This directive defines and links an AngularJS application to HTML.
- **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** − This directive binds the AngularJS Application data to HTML tags.

Explain following directive.

- **ng-app** directive defines and links an AngularJS application to HTML. It also indicate the start of the application.
- **ng-model** directive binds the values of AngularJS application data to HTML input controls. It creates a model variable which can be used with the html page and within the container control(for example, div) having ng-app directive.
- **ng-bind** directive binds the AngularJS Application data to HTML tags. ng-bind updates the model created by ng-model directive to be displayed in the html tag whenever user input something in the control or updates the html control's data when model data is updated by controller.
- **ng-controller** directive tells AngularJS what controller to use with this view. AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control.
- **ng-init** directive initializes an AngularJS Application data. It is used to put values to the variables to be used in the application.
- **ng-repeat** directive repeats html elements for each item in a collection.
- **ng-disabled** directive disables a given control.
- ng-show directive shows a given control.
- ng-hide directive hides a given control.
- ng-click directive represents a AngularJS click event.

On which types of component can we create a custom directive?

AngularJS provides support to create custom directives for following type of elements.

- ***Element d****irectives* − Directive activates when a matching element is encountered.
- **Attribute** − Directive activates when a matching attribute is encountered.
- ***CSS*** − Directive activates when a matching css style is encountered.
- **Comment** − Directive activates when a matching comment is encountered.

What are AngularJS expressions?

Expressions are used to bind application data to html. Expressions are written inside double braces like {{ expression}}. Expressions behave in same way as ng-bind directives. AngularJS application expressions are pure JavaScript expressions and outputs the data where they are used.

How to validate data in AngularJS?

AngularJS enriches form filling and validation. We can use $dirty and $invalid flags to do the validations in seamless way. Use novalidate with a form declaration to disable any browser specific validation.

Following can be used to track error.

- $dirty − states that value has been changed.
- $invalid − states that value entered is invalid.
- $error − states the exact error.

Explain ng-include directive.

Using AngularJS, we can embed HTML pages within a HTML page using ng-include directive.

```
<div ng-app = "" ng-controller = "studentController">
  <div ng-include = "'main.htm'"></div>
  <div ng-include = "'subjects.htm'"></div>
```

```
</div>
```

How to make an ajax call using Angular JS?

AngularJS provides $https: control which works as a service to make ajax call to read data from the server. The server makes a database call to get the desired records. AngularJS needs data in JSON format. Once the data is ready, $https: can be used to get the data from server in the following manner:

```
function studentController($scope,$https:) {
  var url = "data.txt";
  $https:.get(url).success( function(response) {
    $scope.students = response;
  });
}
```

What is use of $routeProvider in AngularJS?

$routeProvider is the key service which set the configuration of urls, maps them with the corresponding html page or ng-template, and attaches a controller with the same.

What is $rootScope?

Scope is a special JavaScript object which plays the role of joining controller with the views. Scope contains the model data. In controllers, model data is accessed via $scope object. $rootScope is the parent of all of the scope variables.

What is scope hierarchy in AngularJS?

Scopes are controllers specific. If we define nested controllers then child controller will inherit the scope of its parent controller.

```
<script>
    var mainApp = angular.module("mainApp", []);
    mainApp.controller("shapeController", function($scope) {
      $scope.message = "In shape controller";
      $scope.type = "Shape";
    });
    mainApp.controller("circleController", function($scope) {
      $scope.message = "In circle controller";
    });
</script>
```

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

- We've set values to models in shapeController.
- We've overridden message in child controller circleController. When "message" is used within module of controller circleController, the overridden message will be used.

What is a service?

Services are JavaScript functions and are responsible to do specific tasks only. Each service is responsible for a specific task for example, $https: is used to make ajax call to get the server data. $route is used to define the routing information and so on. Inbuilt services are always prefixed with $ symbol.

What are the services in AngularJS?

Angular services are substitutable objects that are wired together using dependency injection (DI). You can use services to organize and share code across your app. Angular services are:

- *Lazily instantiated – Angular only instantiates a service when an application component depends on it.*
- *Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory.*

AngularJS come with several built-in services. For example $https: service is used to make XMLHttpRequests (Ajax calls). Services are singleton objects which are instantiated only once in app.

What is service method?

Using service method, we define a service and then assign method to it. We've also injected an already available service to it.

```
mainApp.service('CalcService', function(MathService){
  this.square = function(a) {      return MathService.multiply(a,a);
}});
```

What is factory method?

Using factory method, we first define a factory and then assign method to it.

```
var mainApp = angular.module("mainApp", []);
mainApp.factory('MathService', function() {
  var factory = {};
  factory.multiply = function(a, b) {     return a * b    }
  return factory;
});
```

What are the differences between service and factory methods?

Factory method is used to define a factory which can later be used to create services as and when required whereas service method is used to create a service whose purpose is to do some defined task.

Providers

1. Each web application you build is composed of objects that collaborate to get stuff done. These objects need to be instantiated and wired together for the app to work. In Angular apps most of these objects are instantiated and wired together automatically by the injector service.
2. The injector creates two types of objects, **services** and **specialized objects**.
3. Services are objects whose API is defined by the developer writing the service.
4. Specialized objects conform to a specific Angular framework API. These objects are one of controllers, directives, filters or animations.
5. The injector needs to know how to create these objects. You tell it by registering a "recipe" for creating your object with the injector. There are five recipe types.
6. The most verbose, but also the most comprehensive one is a Provider recipe. The remaining four recipe types — Value, Factory, Service and Constant — are just syntactic sugar on top of a provider recipe.

Which components can be injected as a dependency in AngularJS?

AngularJS provides a supreme Dependency Injection mechanism. It provides following core components which can be injected into each other as dependencies.

- value
- factory
- service
- provider
- constant

What is provider?

Provider is used by AngularJS internally to create services, factory etc. during config phase(phase during which AngularJS bootstraps itself). Below mention script can be used to create MathService that we've created earlier. Provider is a special factory method with a method get() which is used to return the value/service/factory.

```
var mainApp = angular.module("mainApp", []);
//create a service using provider which defines a method square to return square of a number.
mainApp.config(function($provide) {
  $provide.provider('MathService', function() {
        this.$get = function() { var factory = {};  factory.multiply = function(a, b) { return a * b;  }  return factory; };
 });});
```

What is constant?

Constants are used to pass values at config phase considering the fact that value cannot be used to be passed during config phase.

```
mainApp.constant("configParam", "constant value");
```

Is AngularJS extensible?

- Yes! In AngularJS we can create custom directive to extend AngularJS existing functionalities.
- Custom directives are used in AngularJS to extend the functionality of HTML. Custom directives are defined using "directive" function. A custom directive simply replaces the element for which it is activated. AngularJS application during bootstrap finds the matching elements and do one time activity using its compile() method of the custom directive then process the element using link() method of the custom directive based on the scope of the directive.

What is internationalization?

Internationalization is a way to show locale specific information on a website. For example, display content of a website in English language in United States and in Danish in France.

How to implement internationalization in AngularJS?

AngularJS supports inbuilt internationalization for three types of filters currency, date and numbers. We only need to incorporate corresponding js according to locale of the country. By default it handles the locale of the browser. For example, to use Danish locale, use following script
<script src = "https://code.angularjs.org/1.2.5/i18n/angular-locale_da-dk.js"></script>

What are the basic steps to unit test an AngularJS filter?

1. Inject the module that contains the filter.
2. Provide any mocks that the filter relies on.
3. Get an instance of the filter using $filter('yourFilterName').
4. Assert your expectations.

Dependency injection is a powerful software design pattern that Angular employs to compose responsibilities through an intrinsic interface. However, for those new to the process, it can be puzzling where you need to configure and mock these dependencies when creating your isolated unit tests. The open-source project "Angular Test Patterns" is a free resource that is focused on dispelling such confusion through high-quality examples.

What should be the maximum number of concurrent "watches"? Bonus: How would you keep an eye on that number?

- To reduce memory consumption and improve performance it is a good idea to limit the number of watches on a page to 2,000. A utility called ng-stats can help track your watch count and digest cycles.
- Jank happens when your application cannot keep up with the screen refresh rate. To achieve 60 frames-per-second, you only have about 16 milliseconds for your code to execute. It is crucial that the scope digest cycles are as short as possible for your application to be responsive and smooth. Memory use and digest cycle performance are directly affected by the number of active watches. Therefore, it is best to keep the number of watches below 2,000. The open-source utility ng-stats gives developers' insight into the number of watches Angular is managing, as well as the frequency and duration of digest cycles over time.
- Caution: Be wary of relying on a "single magic metric" as the golden rule to follow. You must take the context of your application into account. The number of watches is simply a basic health signal. If you have many thousands of watches, or worse, if you see that number continue to grow as you interact with your page. Those are strong indications that you should look under the hood and review your code.

How do you share data between controllers?

- Create an AngularJS service that will hold the data and inject it inside of the controllers.
- Using a service is the cleanest, fastest and easiest way to test. However, there are couple of other ways to implement data sharing between controllers, like:
    - Using events
    - Using $parent, nextSibling, controllerAs, etc. to directly access the controllers
    - Using the $rootScope to add the data on (not a good practice)
- The methods above are all correct, but are not the most efficient and easy to test.

What is the difference between ng-show/ng-hide and ng-if directives?

ng-show/ng-hide will always insert the DOM element, but will display/hide it based on the condition. ng-if will not insert the DOM element until the condition is not fulfilled. ng-if is better when we needed the DOM to be loaded conditionally, as it will help load page bit faster compared to ng-show/ng-hide. We only need to keep in mind what the difference between these directives is, so deciding which one to use totally depends on the task requirements.

What is a digest cycle in AngularJS?

In each digest cycle Angular compares the old and the new version of the scope model values. The digest cycle is triggered automatically. We can also use $apply() if we want to trigger the digest cycle manually.

Where should we implement the DOM manipulation in AngularJS?

In the directives. DOM Manipulations should not exist in controllers, services or anywhere else but in directives.

Is it a good or bad practice to use AngularJS together with jQuery?

- It is definitely a bad practice. We need to stay away from jQuery and try to realize the solution with an AngularJS approach. jQuery takes a traditional imperative approach to manipulating the DOM, and in an imperative approach, it is up to the programmer to express the individual steps leading up to the desired outcome.
- AngularJS, however, takes a declarative approach to DOM manipulation. Here, instead of worrying about all of the step by step details regarding how to do the desired outcome, we are just declaring what we want and AngularJS worries about the rest, taking care of everything for us.

If you were to migrate from Angular 1.4 to Angular 1.5, what is the main thing that would need refactoring?

Changing .directive to .component to adapt to the new Angular 1.5 components

How would you specify that a scope variable should have one-time binding only?

By using "::" in front of it. This allows the check if the candidate is aware of the available variable bindings in AngularJS.

What is the difference between one-way binding and two-way binding?

- One way binding implies that the scope variable in the html will be set to the first value its model is bound to (i.e. assigned to)
- Two way binding implies that the scope variable will change it's value everytime its model is assigned to a different value

Explain how $scope.$apply() works

$scope.$apply re-evaluates all the declared ng-models and applies the change to any that have been altered (i.e. assigned to a new value)
Explanation: $scope.$apply() is one of the core angular functions that should never be used explicitly, it forces the angular engine to run on all the watched variables and all external variables and apply the changes on their values

What directive would you use to hide elements from the HTML DOM by removing them from that DOM not changing their styling?

The ngIf Directive, when applied to an element, will remove that element from the DOM if it's condition is false.

What makes the angular.copy() method so powerful?

- It creates a deep copy of the variable.
- A deep copy of a variable means it doesn't point to the same memory reference as that variable. Usually assigning one variable to another creates a "shallow copy", which makes the two variables point to the same memory reference. Therefore if we change one, the other changes as well

How would you make an Angular service return a promise? Write a code snippet as an example

To add promise functionality to a service, we inject the "$q" dependency in the service, and then use it like so:

```
angular.factory('testService', function($q){
return {  getName: function(){  var deferred = $q.defer();
                    //API call here that returns data
                    testAPI.getName().then(function(name){deferred.resolve(name)          })
                    return deferred.promise;
          }}})
```

The $q library is a helper provider that implements promises and deferred objects to enable asynchronous functionality

What is the role of services in AngularJS and name any services made available by default?

- AngularJS Services are objects that provide separation of concerns to an AngularJS app.
- AngularJS Services can be created using a factory method or a service method.
- Services are singleton components. All components of the application (into which the service is injected) will work with single instance of the service.
- An AngularJS service allows developing of business logic without depending on the View logic which will work with it.
- AngularJS Services help create reusable components.
- A Service can be created either using the service() method or the factory() method.
- A typical service can be injected into another service or into an AngularJS Controller.

Few of the inbuilt services in AngularJS are:

- ***$http*** service: The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP
- ***$log*** service: Simple service for logging. Default implementation safely writes the message into the browser's console
- ***$anchorScroll***: it scrolls to the element related to the specified hash or (if omitted) to the current value of $location.hash()

When creating a directive, it can be used in several different ways in the view. Which ways for using a directive do you know? How do you define the way your directive will be used?

- When you create a directive, it can be used as an attribute, element or class name. To define which way to use, you need to set the restrict option in your directive declaration.
- The restrict option is typically set to:
  - 'A' – only matches attribute name
  - 'E' – only matches element name
  - 'C' – only matches class name
- These restrictions can all be combined as needed: 'AEC' – matches either attribute or element or class name

When should you use an attribute versus an element?

Use an element when you are creating a component that is in control of the template. Use an attribute when you are decorating an existing element with new functionality.

How do you reset a $timeout, $interval(), and disable a $watch()?

- To reset a timeout and/or $interval, assign the result of the function to a variable and then call the .cancel() function.

```
var customTimeout = $timeout(function () {  // arbitrary code}, 55);$timeout.cancel(customTimeout);
```

- To disable $watch(), we call its deregistration function. $watch() then returns a deregistration function that we store to a variable and that will be called for cleanup

```
var deregisterWatchFn = $scope.$on('$destroy', function () {    deregisterWatchFn();});
```

What are Directives?

Directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children. Angular comes with a set of these directives built-in, like ngBind, ngModel, and ngClass. Much like you create controllers and services, you can create your own directives for Angular to use. When Angular bootstraps your application, the HTML compiler traverses the DOM matching directives against the DOM elements.

What is DDO Directive Definition Object?

"DDO is an object used while creating a custom directive. A standard DDO object has following parameters.

```
var directiveDefinitionObject = {
        priority: 0,
        transclude: false,
        restrict: 'A',
        templateNamespace: 'html',
        scope: false,
        template: '<div></div>', // or  templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },
        controller: function($scope, $element, $attrs, $transclude, otherInjectables) { ... },
        controllerAs: 'stringIdentifier',
        bindToController: false,
   require: 'siblingDirectiveName', // or // ['^parentDirectiveName', '?optionalDirectiveName', '?^optionalParent'],
   compile: function compile(tElement, tAttrs, transclude) {
    return { pre: function preLink(scope, iElement, iAttrs, controller) { . },  post: function postLink(scope, iElement, iAttrs, controller) { ... } }
    // or     // return function postLink( ... ) { ... }   },    // or     // link: {   //  pre: function preLink(scope, iElement, iAttrs, controller) { ... },  //  post: function
postLink(scope, iElement, iAttrs, controller) { ... }   // }   // or    // link: function postLink( ... ) { ... } };"
```

What is a singleton pattern and where we can find it in Angularjs?

Is a great pattern that restricts the use of a class more than once. We can find singleton pattern in angular in dependency injection and in the services. In a sense, if you do 2 times 'new Object()' without this pattern, you will be alocating 2 pieces of memory for the same object. With singleton pattern, if the object exists, you reuse it.

What is an interceptor? What are common uses of it?

An interceptor is a middleware code where all the $http requests go through.

The interceptor is a factory that are registered in $httpProvider. You have 2 types of requests that go through the interceptor, request and response (with requestError and responseErrorrespectively). This piece of code is very useful for error handling, authentication or middleware in all the requests/responses.

How would you programmatically change or adapt the template of a directive before it is executed and transformed?

You would use the compile function. The compile function gives you access to the directive's template before transclusion occurs and templates are transformed, so changes can safely be made to DOM elements. This is very useful for cases where the DOM needs to be constructed based on runtime directive parameters.

How would you validate a text input field for a twitter username, including the @ symbol?

You would use the ngPattern directive to perform a regex match that matches Twitter usernames. The same principal can be applied to validating phone numbers, serial numbers, barcodes, zip codes and any other text input.

How would you implement application-wide exception handling in your Angular app?

Angular has a built-in error handler service called $exceptionHandler which can easily be overriden as seen below:

```
myApp.factory('$exceptionHandler', function($log, ErrorService) {    return function(exception, cause) { if (console) {
$log.error(exception);  $log.error(cause);   } ErrorService.send(exception, cause); };});
```

This is very useful for sending errors to third party error logging services or helpdesk applications. Errors trapped inside of event callbacks are not propagated to this handler, but can manually be relayed to this handler by calling $exceptionHandler(e) from within a try catch block.

How do you hide an HTML element via a button click in AngularJS?

You can do this by using the ng-hide directive in conjunction with a controller we can hide an HTML element on button click.

```
<div ng-controller="MyCtrl"> <button ng-click="hide()">Hide element</button>  <p ng-hide="isHide">Hello World!</p></div>
function MyCtrl($scope){  $scope.isHide = false; $scope.hide = function(){ $scope.isHide = true;}}
```

How would you react on model changes to trigger some further action? For instance, say you have an input text field called email and you want to trigger or execute some code as soon as a user starts to type in their email.

We can achieve this using $watch function in our controller.

```
function MyCtrl($scope) { $scope.email = ""; $scope.$watch("email", function(newValue, oldValue) { if ($scope.email.length > 0) {
    console.log("User has started writing into email"); }});}
```

How do you disable a button depending on a checkbox's state?

We can use the ng-disabled directive and bind its condition to the checkbox's state. `<body ng-app> <label><input type="checkbox" ng-model="checked"/>Disable Button</label><button ng-disabled="checked">Select me</button></body>`

Stateful filters

It is strongly discouraged to write filters that are stateful, because the execution of those can't be optimized by Angular, which often leads to performance issues. Many stateful filters can be converted into stateless filters just by exposing the hidden state as a model and turning it into an argument for the filter.

If you however do need to write a stateful filter, you have to mark the filter as $stateful, which means that it will be executed one or more times during the each $digest cycle.

```
angular.module('myStatefulFilterApp', [])
.filter('decorate', ['decoration', function(decoration) {  function decorateFilter(input) {   return decoration.symbol + input +   decoration.symbol;  }
decorateFilter stateful = true;  return decorateFilter;}])
.controller('MyController', ['$scope', 'decoration', function($scope, decoration) {  $scope.greeting = 'hello';  $scope.decoration = decoration;
}]).value('decoration', {symbol: '*'});

<div ng-controller="MyController">  Input: <input ng-model="greeting" type="text"><br>  Decoration: <input ng-
model="decoration.symbol" type="text"><br>  No filter: {{greeting}}<br>  Decorated: {{greeting | decorate}}<br>
</div>
```

Using filters in controllers, services, and directives

You can also use filters in controllers, services, and directives.For this, inject a dependency with the name <filterName>Filter into your controller/service/directive. E.g. a filter called number is injected by using the dependency numberFilter. The injected argument is a function that takes the value to format as first argument, and filter parameters starting with the second argument.

```
angular.module('FilterInControllerModule', []).controller('FilterController', ['filterFilter', function FilterController(filterFilter)
{this.array = [    {name: 'Tobias'},{name: 'Jeff'}, {name: 'Brian'}, {name: 'Igor'},{name:'James'}, {name: 'Brad'} ];
  this.filteredArray = filterFilter(this.array, 'a');}]);

<div ng-controller="FilterController as ctrl">  <div>All entries:    <span ng-repeat="entry in ctrl.array">{{entry.name}}
</span></div>
  <div>    Entries that contain an "a":    <span ng-repeat="entry in ctrl.filteredArray">{{entry.name}} </span>  </div></div>
```

Creating custom filters

Writing your own filter is very easy: just register a new filter factory function with your module. Internally, this uses the filterProvider. This factory function should return a new filter function which takes the input value as the first argument. Any filter arguments are passed in as additional arguments to the filter function.

The filter function should be a pure function, which means that it should always return the same result given the same input arguments and should not affect external state, for example, other Angular services. Angular relies on this contract and will by default execute a filter only when the inputs to the function change. Stateful filters are possible, but less performant.

```
angular.module('myReverseFilterApp', [])
.filter('reverse', function() {  return function(input, uppercase) { input = input || '';    var out = '';    for (var i = 0; i < input.length; i++) {   out =
input.charAt(i) + out;    }    if (uppercase) {     out = out.toUpperCase();    }    return out;
  };})
.controller('MyController', ['$scope', 'reverseFilter', function($scope, reverseFilter) { $scope.greeting = 'hello'; $scope.filteredGreeting =
reverseFilter($scope.greeting);
}]);
```

Dependency Injection

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies.
The Angular injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.

Using Dependency Injection

DI is pervasive throughout Angular. You can use it when defining components or when providing `run` and `config` blocks for a module.

- Components such as services, directives, filters, and animations are defined by an injectable factory method or constructor function. These components can be injected with "service" and "value" components as dependencies.
- Controllers are defined by a constructor function, which can be injected with any of the "service" and "value" components as dependencies, but they can also be provided with special dependencies. See Controllers below for a list of these special dependencies.
- The `run` method accepts a function, which can be injected with "service", "value" and "constant" components as dependencies. Note that you cannot inject "providers" into `run` blocks.
- The `config` method accepts a function, which can be injected with "provider" and "constant" components as dependencies. Note that you cannot inject "service" or "value" components into configuration.

Factory Methods

The way you define a directive, service, or filter is with a factory function. The factory methods are registered with modules. The recommended way of declaring factories is:

```
angular.module('myModule', []).factory('serviceId', ['depService', function(depService) {  // ... }])
.directive('directiveName', ['depService', function(depService) { // ... }])
.filter('filterName', ['depService', function(depService) { // ... }]);
```

Module Methods

We can specify functions to run at configuration and run time for a module by calling the `config` and `run` methods. These functions are injectable with dependencies just like the factory functions above.

```
angular.module('myModule', []).config(['depProvider', function(depProvider) {  // ...}])
.run(['depService', function(depService) {  // ...}]);
```

Controllers

Controllers are "classes" or "constructor functions" that are responsible for providing the application behavior that supports the declarative markup in the template. The recommended way of declaring Controllers is using the array notation:

```
someModule.controller('MyController', ['$scope', 'dep1', 'dep2', function($scope, dep1, dep2) {  ...
  $scope.aMethod = function() {    ...  }  ...}]);
```

Unlike services, there can be many instances of the same type of controller in an application.
Moreover, additional dependencies are made available to Controllers:

- `$scope`: Controllers are associated with an element in the DOM and so are provided with access to the scope. Other components (like services) only have access to the `$rootScope` service.
- resolves: If a controller is instantiated as part of a route, then any values that are resolved as part of the route are made available for injection into the controller.

Dependency Annotation

Angular invokes certain functions (like service factories and controllers) via the injector. You need to annotate these functions so that the injector knows what services to inject into the function. There are three ways of annotating your code with service name information:

- Using the inline array annotation (preferred)
- Using the `$inject` property annotation
- Implicitly from the function parameter names (has caveats)

Inline Array Annotation

- This is the preferred way to annotate application components. This is how the examples in the documentation are written.
- For example: `someModule.controller('MyController', ['$scope', 'greeter', function($scope, greeter) {  // ...}]);`
- Here we pass an array whose elements consist of a list of strings (the names of the dependencies) followed by the function itself.
- When using this type of annotation, take care to keep the annotation array in sync with the parameters in the function declaration.

`$inject` Property Annotation

To allow the minifiers to rename the function parameters and still be able to inject the right services, the function needs to be annotated with the `$inject` property. The `$inject` property is an array of service names to inject.

```
var MyController = function($scope, greeter) {  // ...}
MyController.$inject = ['$scope', 'greeter'];
```

```
someModule.controller('MyController', MyController);
```

In this scenario the ordering of the values in the $inject array must match the ordering of the parameters in MyController.
Just like with the array annotation, you'll need to take care to keep the $inject in sync with the parameters in the function declaration.

Implicit Annotation

Careful: If you plan to minify your code, your service names will get renamed and break your app.

- The simplest way to get hold of the dependencies is to assume that the function parameter names are the names of the dependencies.

```
someModule.controller('MyController', function($scope, greeter) {  // ...});
```

- Given a function, the injector can infer the names of the services to inject by examining the function declaration and extracting the parameter names. In the above example, $scope and greeter are two services which need to be injected into the function.
- One advantage of this approach is that there's no array of names to keep in sync with the function parameters. You can also freely reorder dependencies.
- However this method will not work with JavaScript minifiers/obfuscators because of how they rename parameters.
- Tools like ng-annotate let you use implicit dependency annotations in your app and automatically add inline array annotations prior to minifying. If you decide to take this approach, you probably want to use ng-strict-di.
- Because of these caveats, we recommend avoiding this style of annotation.

Using Strict Dependency Injection

- You can add an ng-strict-di directive on the same element as ng-app to opt into strict DI mode:
  <!doctype html> <html ng-app="myApp" ng-strict-di><body> I can add: {{ 1 + 2 }}. <script src="angular.js"></script>
  </body></html>
- Strict mode throws an error whenever a service tries to use implicit annotations.
- Consider this module, which includes a willBreak service that uses implicit DI:

```
angular.module('myApp', [])
.factory('willBreak', function($rootScope) {  // $rootScope is implicitly injected})
.run(['willBreak', function(willBreak) {// Angular will throw when this runs}]);
```

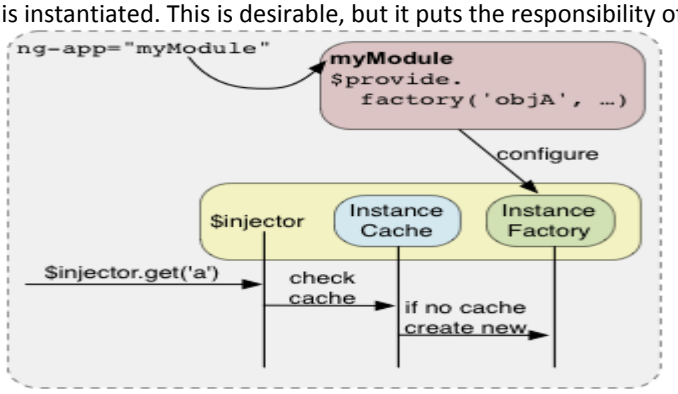
- When the willBreak service is instantiated, Angular will throw an error because of strict mode. This is useful when using a tool like ng-annotate to ensure that all of your application components have annotations.
- If you're using manual bootstrapping, you can also use strict DI by providing strictDi: true in the optional config argument:
  angular.bootstrap(document, ['myApp'], { strictDi: true});

Why Dependency Injection?

1) There are only three ways a component (object or function) can get a hold of its dependencies:
   a) The component can create the dependency, typically using the new operator.
   b) The component can look up the dependency, by referring to a global variable.
   c) The component can have the dependency passed to it where it is needed.
2) The first two options of creating or looking up dependencies are not optimal because they hard code the dependency to the component. This makes it difficult, if not impossible, to modify the dependencies. This is especially problematic in tests, where it is often desirable to provide mock dependencies for test isolation.
3) The third option is the most viable, since it removes the responsibility of locating the dependency from the component. The dependency is simply handed to the component.

```
function SomeClass(greeter) {  this.greeter = greeter;}
SomeClass.prototype.doSomething = function(name) {  this.greeter.greet(name);}
```

In the above example SomeClass is not concerned with creating or locating the greeter dependency, it is simply handed the greeterwhen it is instantiated. This is desirable, but it puts the responsibility of getting hold of the dependency on the code that constructs SomeClass.



To manage the responsibility of dependency creation, each Angular application has an injector. The injector is a service locator that is responsible for construction and lookup of dependencies.Here is an example of using the injector service:

1) `// Provide the wiring information in a module var myModule = angular.module('myModule', []);`
2) Teach the injector how to build a `greeter` service. Notice that `greeter` is dependent on the `$window` service. The `greeter` service is an object that contains a `greet` method.

```
myModule.factory('greeter', function($window) {  return {    greet: function(text) {     $window.alert(text);    } };});
```

3) Create a new injector that can provide components defined in our `myModule` module and request our `greeter` service from the injector. (This is usually done automatically by angular bootstrap).

```
var injector = angular.injector(['ng', 'myModule']); var greeter = injector.get('greeter');
```

4) Asking for dependencies solves the issue of hard coding, but it also means that the injector needs to be passed throughout the application. Passing the injector breaks the Law of Demeter. To remedy this, we use a declarative notation in our HTML templates, to hand the responsibility of creating components over to the injector, as in this example:

```
<div ng-controller="MyController">  <button ng-click="sayHello()">Hello</button></div>
function MyController($scope, greeter) {  $scope.sayHello = function() {    greeter.greet('Hello World');  };}
```

5) When Angular compiles the HTML, it processes the `ng-controller` directive, which in turn asks the injector to create an instance of the controller and its dependencies.

```
injector.instantiate(MyController);
```

6) This is all done behind the scenes. Notice that by having the `ng-controller` ask the injector to instantiate the class, it can satisfy all of the dependencies of `MyController` without the controller ever knowing about the injector.
7) This is the best outcome. The application code simply declares the dependencies it needs, without having to deal with the injector. This setup does not break the Law of Demeter.
8) **Note:** Angular uses **constructor injection**.

Angular Expressions vs. JavaScript Expressions

Angular expressions are like JavaScript expressions with the following differences:

- **Context:** JavaScript expressions are evaluated against the global `window`. In Angular, expressions are evaluated against a `scope`object.
- **Forgiving:** In JavaScript, trying to evaluate undefined properties generates `ReferenceError` or `TypeError`. In Angular, expression evaluation is forgiving to `undefined` and `null`.
- **Filters:** You can use filters within expressions to format data before displaying it.
- **No Control Flow Statements:** You cannot use the following in an Angular expression: conditionals, loops, or exceptions.
- **No Function Declarations:** You cannot declare functions in an Angular expression, even inside `ng-init` directive.
- **No RegExp Creation With Literal Notation:** You cannot create regular expressions in an Angular expression.
- **No Object Creation With New Operator:** You cannot use `new` operator in an Angular expression.
- **No Bitwise, Comma, And Void Operators:** You cannot use Bitwise, , or `void` operators in an Angular expression.

If you want to run more complex JavaScript code, you should make it a controller method and call the method from your view. If you want to `eval()` an Angular expression yourself, use the `$eval()` method.

Context

- Angular does not use JavaScript's `eval()` to evaluate expressions. Instead Angular's $parse service processes these expressions.
- Angular expressions do not have direct access to global variables like `window`, `document` or `location`. This restriction is intentional. It prevents accidental access to the global state – a common source of subtle bugs.
- Instead use services like `$window` and `$location` in functions on controllers, which are then called from expressions. Such services provide mockable access to globals.
- It is possible to access the context object using the identifier `this` and the locals object using the identifier `$locals`.