# Interview Questions and Answers

* Where do you keep your business logic?
  + An Angular service is a singleton, and is only instantiated when an application component (like a controller) depends on it. And since a service is a singleton, it is only instantiated one time. This makes it suitable for passing data across controllers, or even to other services. This also makes them reusable, testable, and overall much more manageable.
  + Singleton - There is only one object, but is injected into many places. (objects are passed by reference to a method)
* Providers in AngularJS
  + A provider is an object with a $get() method. The injector calls the $get method to create a new instance of a service. The Provider can have additional methods which would allow for configuration of the provider.
  + AngularJS uses *$provide* to register new providers. The providers basically create new instances, but only once for each provider.
  + Constant
    - A constant can be injected everywhere. A constant cannot be intercepted by a decorator, that means that the value of a constant can never be changed.
  + Value
    - A value is nothing more than a simple injectable value.
    - The value can be a string, number but also a function.
    - Value differs from constant in that value cannot be injected into configurations, but it can be intercepted by decorators.
  + Service
    - A service is an injectable constructor.
    - A service is a [singleton](http://en.wikipedia.org/wiki/Singleton_pattern) and will only be created once by AngularJS.
    - Services are a great way for communicating between controllers like sharing data.
  + Factories
    - A factory is an injectable function.
    - A factory is a lot like a service in the sense that it is a singleton and dependencies can be specified in the function.
      * The difference between a factory and a service is that a factory injects a plain function so AngularJS will call the function and a service injects a constructor.
      * A constructor creates a new object so *new* is called on a service and with a factory you can let the function return anything you want.
  + Decorator
    - A decorator can modify or encapsulate other providers except constant.
  + Providers
    - It allows you to have a complex creation function and configuration options.
    - A provider is actually a configurable factory.
    - The provider accepts an object or a constructor.
    - We should use a provider when we are creating a service for the entire application.
      * For example, when we are creating service to retrieve data from the API we need to set the API key once per application. We can set that in the config of app and pass that to the setter function of the provider.
      * Provider is accessible in the Configuration phase, while service and factory are not.
  + Provider vs Factory vs Service
    - A factory is a special case of a provider when all you need in your provider is a $get () function. It allows you to write it with less code. A service is a special case of a factory when you want to return an instance of a new object, with the same benefit of writing less code.
* Angular life cycle - Compile, load, Bootstrap
  + AngularJS Bootstrap Process includes from AngularJS initialization to compilation process. AngularJS initialization can be done in two ways, automatic initialization and manual initialization.
  + Automatic Bootstrap Process
    - Angular looks for the ng-app directive which is root of the AngularJS compilation process. If the ng-app directive is found then Angular will:
      * Load the module associated with the directive.
      * Create the application injector
      * Start compilation of DOM from ng-app directive, the root of the compilation.
  + Manual Bootstrap Process
    - Sometimes you may need to manual Initialize AngularJS in order to have more control over the initialization process. You can do that by calling angular.bootstrap () function inside angular.element(document).ready() function.
    - The angular.bootstrap () function takes two parameters, the document and module name injector as the second parameter to the angular.bootstrap function.
* The three phases of the life cycle of an AngularJS application happen each time a web page is loaded in the browser.
  + The first phase of the AngularJS life cycle is the **Bootstrap Phase**, which occurs when the AngularJS JavaScript library is downloaded to the browser.
    - AngularJS initializes its own necessary components and;
    - then initializes your module, which the ng-app directive points to.
    - The module is loaded, and any dependencies are injected into your module and made available to code within the module.
  + The second phase of the AngularJS life cycle is the HTML **Compilation Stage**. Initially when a web page is loaded, a static form of the DOM is loaded in the browser. During the compilation phase, the static DOM is replaced with a dynamic DOM that represents the AngularJS view.
    - Traversing the static DOM and collecting all the directives
    - Linking the directives to the appropriate JavaScript functionality in the AngularJS built-in library or custom directive code. The directives are combined with a scope to produce the dynamic or live view.
  + The final phase of the AngularJS application is the **Runtime Phase**, which exists until the user reloads or navigates away from a web page. At that point, any changes in the scope are reflected in the view, and any changes in the view are directly updated in the scope, making the scope the single source of data for the view.
* Configuration Block
  + Configuration blocks are meant for configuring your modules.
  + In configuration blocks, we can inject any provider.
  + AngularJS provides following helper functions to register providers, services and constants. Had these methods not been there on moduleInstance, then we would have needed configuration blocks.
  + Eg: Suppose that $routeProvider is called in the config phase. This will result in an error ($routeProviderProvider not available) since Angular will look for a service ($route) which uses this provider. At configuration phase, we have only providers available to us and not the services that use these providers. Also, we cannot use these providers in the controllers, as configuration needs to happen before it gets injected into anything we want to use it in.
* Run Block
  + Run blocks are the closest thing in Angular to the main method and are used to kickstart the application.
  + Gets executed after the injector is created
  + Only instances and constants can be injected into run blocks. This is to prevent further system configuration during application run time
  + Run blocks typically contain code which is hard to unit-test
* MVC in Angular
  + Software pattern for developing web applications
  + Model – Lowest level of the pattern responsible for maintaining data
  + View – Responsible for displaying all or portion of data to the user
  + Controller - Code that controls the interaction between the View and the Model
  + MVC is popular because it isolates the application logic from the user interface layer and supports separation of concerns.
    - The controller receives all requests for the application and then works with the model to prepare any data needed by the view.
    - The view then uses the data prepared by the controller to generate a final presentable response.
* Directives in Angular
  + Helps you extend basic HTML elements/attributes and create *reusable* and *testable* code.
  + restrict is for defining the directive type, and it can be **A**ttribute, **C**lass, **E**lement, and co**M**ment.
  + scope is for managing the directive scope.
  + templateUrl is used for calling a view in order to render specific content by using data transferred to the directive scope.
  + use template and provide HTML code directly
  + link function: Function that you can use to perform directive-specific operations.
    - The directive is not only rendering some HTML code by providing some inputs.
    - You can also bind functions to the directive element, call a service and update the directive value, get directive attributes if it is an Etype directive, etc.
  + Isolated Scope
    - **"@" Scope:** This type of scope is used for passing value to the directive scope.
    - **"=" Scope:**In this scope type, scope variables are passed instead of the values, which means that we will not pass {{message}}, we will pass message instead.
      * The reason behind this feature is constructing two-way data binding between the directive and the page elements or controllers.
    - **"&" Scope:** In this scope type, we will have a look at how to pass expressions to the directive. In real-life cases, you may need to pass a specific function (expression) to directives in order to prevent coupling.
  + Directive Inheritance
    - Child directives use the require keyword to use the parent directive controller.
    - And one more important point is the fourth argument of the link function in the child directives. This argument refers to the controller attribute of the parent directive that means the child directive can use the controller function inside the controller.
    - Child directives are like a property of the parent directive.
  + [Binding to controllers with bindToController](https://blog.thoughtram.io/angularjs/2015/01/02/exploring-angular-1.3-bindToController.html#binding-to-controllers-with-bindtocontroller)
    - Angular 1.3 introduces a new property to the directive definition object called bindToController, which does exactly what it says. When set to true in a directive with isolated scope that uses controllerAs, the component’s properties are bound to the controller rather than to the scope.
    - In version 1.4, bindToController gets even more powerful. When having an isolated scope with properties to be bound to a controller, we always define those properties on the scope definition and bindToController is set to true. In 1.4 however, we can move all our property binding definitions to bindToController and make it an object literal.
  + The compile () and link () Functions
    - The compile () and link () functions define how the directive is to modify the HTML that matched the directive.
    - The compile () function is called once for each occurrence of the directive in the HTML page. The compile () function can then do any one-time configuration needed of the element containing the directive.
    - The link () function is called every time the element is to be bound to data in the $scope object.
    - The compile () function takes two parameters: The element and attributes parameters.
    - The link () function takes three parameters: The $scope parameter, the element parameter and the attributes parameter. The element and attributes parameter is the same as passed to the compile () function. The $scope parameter is the normal scope object, or an isolate scope in case you have specified one in the directive definition object.
    - A better name for the compile () function would have been something like create (), init () or configure (). Something that signals that this function is only called once.
    - A better name for the link () function would have been something like bind () or render (), which signals that this function is called whenever the directive needs to bind data to it, or to re-render it.
    - pre-link functions are called parent-then-child, whereas post-link functions are called child-then-parent.
    - When the post-link function is called, all previous steps have taken place – binding, transclusion, etc.
    - **The** pre-link **function is guaranteed to run on an element instance before any** post-link **function of its child elements has run.**
    - <https://www.jvandemo.com/the-nitty-gritty-of-compile-and-link-functions-inside-angularjs-directives/>
  + Compile function
    - Use the compile function to change the original DOM (template element) before AngularJS creates an instance of it and before a scope is created.
  + Pre-link function
    - Use the pre-link function to implement logic that runs when AngularJS has already compiled the child elements, but before any of the child element's post-link functions have been called.
  + Post-link function
    - Use the post-link function to execute logic, knowing that all child elements have been compiled and all pre-link and post-link functions of child elements have been executed.
  + Transclusion
    - In AngularJS, transclusion is the mechanism that allows you to grab the content of the DOM element of your directive and include it anywhere in the directive's template.
    - If transclude is set true in case of nested directives, the child directive’s compile will be called before the parent compile functions.
      * In AngularJS, transclusion is the inclusion of the directive's DOM element content into the directive's template
      * We can deduce that AngularJS needs to process the element's DOM content before it can make it available inside the template.
      * However, the element's child elements can also contain directives that apply transclusion themselves.
      * So, AngularJS has to recursively traverse the DOM first to check if transclusion is enabled in child elements and then compile the DOM backwards to make sure all DOM changes correctly "bubble up" again to the top before the processed DOM is ready to be added to the original directive's template.
* Custom Filter
  + A filter is very similar to a factory or service in many regards but has the added advantage of behaving on a global scope once created.
  + You can invoke a filter on both the data binding in your html
  + directly inside of your controller or directive by using the $filter service.
* $apply and $digest Explored
  + Angular sets up a watcher on the scope model, which in turn updates the view whenever the model changes.
  + It’s the $digest cycle where the watchers are fired. When a watcher is fired, AngularJS evaluates the scope model, and if it has changed then the corresponding listener function is called.
  + The $digest cycle starts as a result of a call to $scope.$digest(). Assume that you change a scope model in a handler function through the ng-click directive. In that case AngularJS automatically triggers a $digest cycle by calling $digest().
  + When the $digest cycle starts, it fires each of the watchers. These watchers check if the current value of the scope model is different from last calculated value. If yes, then the corresponding listener function executes.
  + Angular doesn’t directly call $digest(). Instead, it calls $scope.$apply(), which in turn calls $rootScope.$digest(). As a result of this, a digest cycle starts at the $rootScope, and subsequently visits all the child scopes calling the watchers along the way.
  + $apply() function comes in two flavors.
    - The first one takes a function as an argument, evaluates it, and triggers a $digest cycle.
    - The second version does not take any arguments and just starts a $digest cycle when called.
    - you change any model outside of the Angular context, then you need to inform Angular of the changes by calling $apply() manually. It’s like telling Angular that you are changing some models and it should fire the watchers so that your changes propagate properly.
    - You should always use the version of $apply() that accepts a function argument. This is because when you pass a function to $apply (), the function call is wrapped inside a try...catch block, and any exceptions that occur will be passed to the $exceptionHandler service.
    - The $digest loop doesn’t run just once. At the end of the current loop, it starts all over again to check if any of the models have changed. This is basically dirty checking, and is done to account for any model changes that might have been done by listener functions. So, the $digest cycle keeps looping until there are no more model changes, or it hits the max loop count of 10.
* Watch expressions
  + Watch a function
  + watch for changes to collections of objects:  $scope.$watchCollection():
  + watcher that listens for changes to a group of expressions: $scope.$watchGroup() (Arrays)
* UI Router
  + Supports Ng-Route
  + ui-router allows for [nested views](https://github.com/angular-ui/ui-router/wiki/Nested-States-%26-Nested-Views) and [multiple named views](https://github.com/angular-ui/ui-router/wiki/Multiple-Named-Views). This is very useful with larger app where you may have pages that inherit from other sections
  + allows for you to have strong-type linking between states based on state names. Change the url in one place will update every link to that state when you build your links with [ui-sref](http://angular-ui.github.io/ui-router/site/#/api/ui.router.state.directive:ui-sref)
  + There is also the concept of the [decorator](http://angular-ui.github.io/ui-router/site/#/api/ui.router.state.$stateProvider#methods_decorator) which could be used to allow your routes to be dynamically created based on the URL that is trying to be accessed.
  + [states](https://github.com/angular-ui/ui-router/wiki#state-manager) allow you to map and access different information about different states and you can easily pass information between states via [$stateParams](https://github.com/angular-ui/ui-router/wiki/URL-Routing#stateparams-service)
  + You can easily determine if you are in a state or parent of a state to adjust UI element (highlighting the navigation of the current state) within your templates via [$state](http://angular-ui.github.io/ui-router/site/#/api/ui.router.state.$state) provided by ui-router which you can expose via setting it in $rootScope on run.
* 404 error at central place in angular.
  + We can write an Http interceptor and decorate the promise rejection with a property handled to indicate whether it's handled the error.



* Emit and Broadcast
  + The key thing to remember when using $scope to fire your events, is that they will communicate only with immediate parent or child scopes only! Scopes aren’t always child and parent. We might have sibling scopes. Using $scope to fire an event will miss out sibling scopes, and just carry on up! They do not go sideways!
  + $rootScope.$emit will fire an event for all $rootScope.$on listeners only.
  + The interesting part is that $rootScope.$broadcast will notify all $rootScope.$on as well as $scope.$on listeners
  + If you choose to use $emit, one of your other $scope listeners can cancel it, so prevent it bubbling further. Using $broadcast has the opposite effect in which it cannot be cancelled! Cancelling an event can be done with stopPropagation function.
  + When using $rootScope.$on, we need to unbind those listeners each time the $scope is destroyed. $scope.$on listeners are automatically unbound, but we’ll need to call the above closure manually on the $destroy event:
* Moment JS
  + Parse, Validate, Manipulate and display date and time in JS
  + Multiple Locale Support
  + It supports internationalization and time zone
* Underscore.js
  + A popular JavaScript library that contains 112 commonly needed helper functions.
* D3 JS
  + D3.js or just D3 for Data-Driven Documents is a JavaScript library for producing dynamic, interactive data visualizations in web browsers
  + written in JavaScript and uses a functional style which means you can reuse code and add specific functions to your heart's content
* TypeScript
  + Due to the static typing, code written in TypeScript is more predictable, and is generally easier to debug.
  + Makes it easier to organize the code base for very large and complicated apps thanks to modules, namespaces and strong OOP support.
  + TypeScript has a compilation step to JavaScript that catches all kinds of errors before they reach runtime and break something.
  + The upcoming Angular 2 framework is written in TypeScript and it's recommended that developers use the language in their projects as well.
  + A very distinctive feature of TypeScript is the support of static typing. This means that you can declare the types of variables, and the compiler will make sure that they aren't assigned the wrong types of values. If type declarations are omitted, they will be inferred automatically from your code.
  + Interfaces
    - Interfaces are used to type-check whether an object fits a certain structure. By defining an interface, we can name a specific combination of variables, making sure that they will always go together.
  + TypeScript offers a class system that is very similar to the one in these languages, including inheritance, abstract classes, interface implementations, setters/getters, and more.
  + Generics are templates that allow the same function to accept arguments of various different types. Creating reusable components using generics is better than using the any data type, as generics preserve the types of the variables that go in and out of them.
  + TypeScript introduces a syntax for exporting and importing modules, but cannot handle the actual wiring between files. To enable external modules TS relies on third-party libraries: [require.js](http://requirejs.org/) for browser apps and [CommonJS](https://en.wikipedia.org/wiki/CommonJS) for Node.js.
* Promises
  + Promises in AngularJS are provided by the built-in $q service.
  + They provide a way to execute asynchronous functions in series by registering them with a promise object.
  + A deferred object is simply an object that exposes a promise as well as the associated methods for resolving that promise.
    - It is constructed using the $q.deferred() function
    - exposes three main methods: resolve(), reject(), and notify().
    - The associated promise object can be accessed via the promise property.
    - The parameter of the resolve() function will be passed to the callback function made by then() function.
    - we can call the reject() method if something goes wrong
    - The second parameter of the then() method is an optional error handling callback function that'll be called if and only if the promise is rejected.
    - Using $q constructor: The $q service itself is also a function that allows you to quickly convert a callback based asynchronous function into a promise based solution.
    - The deferred object exposes the Promise instance and its APIs which are to be used for resolving the promise. A new instance of deferred is constructed by calling $q.defer(). It exposes three methods:
      * **resolve**: resolves a promise with a value.
      * **reject**: rejects a promise with a reason.
      * **notify**: provides status updates.
    - finally() method: to ensure a specific function executes regardless of the result of the promise
    - then(successCallback, errorCallback, notifyCallback)- The Regardless promise is resolved or rejected, it calls one of the success or error callback. Notification callback may be called zero or more than one time.
    - catch(errorCallback)- This is the shorthand for the promise.then(null, errorCallback) where success callback is not given.
    - finally(callback, notifyCallback)- This allows you to watch every single event of a promise. The callback is called once when either promise is resolved or rejected.
    - Since $http is an abstraction over $q, it has different callbacks. Instead of .then and .catch, it’s .success and .error and the arguments you get are different.
    - $resource service returns a promise. It can be retrieve by simply $resource.get(‘somedata’).$promise.
    - $q.all() is one of the method that i use more frequently. $q.all accepts array of promises as argument. Once all of the promises get completed. you will get the result in callback function as array of results.