***Angular***

Angular is a **JavaScript (a TypeScript based open-source full-stack web application)** **framework** which makes you able to create reactive **Single Page Applications** (SPAs). This is a leading front-end development framework which is regularly updated by Angular team of **Google** engineers. Angular+2 is completely based on **components**. It consists of several components forming a tree structure with **parent** and **child** components.

**Here is the difference between AngularJS and Angular:**

|  |  |
| --- | --- |
| **AngularJS** | **Angular** |
| AngularJS is common and popular name of the first version of **Angular1.0.** | Angular is common and popular name of the Angular’s version **beyond 2+** |
| AngularJS is a **JavaScript-based** open-source front-end web framework.  **MVC** based(Model-View-Controller) | Angular is a **TypeScript-based** open-source full-stack web application framework. **DOM** based(Document Object Model) |
| AngularJS uses the concept of **scope** or **controller, directories**. | Instead of scope and controller, Angular uses **hierarchy of components** as its primary architectural characteristic. |
| AngularJS has a **simple syntax** and used on HTML pages along with the source location. | Angular uses the different expression syntax. It uses **"[ ]"** for **property** **binding**, and **"( )"** for **event** **binding**. |
| AngularJS is a simple **JavaScript** file which is used with HTML pages and doesn't support the features of a **server-side** programming language. | Angular uses of Microsoft's **TypeScript** language, which provides Class-based **Object-Oriented Programming**, **Static Typing, Generics** etc. which are the features of a server-side programming language. |
| AngularJS **doesn't** support dynamic loading of the page. | Angular 2+ **supports** dynamic loading of the page. |

**Main building blocks except components in angular:**

Components, Modules, Directives, Decorators, Pipes, Data Binding, Templates, Metadata

**What is Angular ivy?**

Ivy is the **code name for Angular's next-generation compilation and rendering pipeline**. With the version 9 release of Angular, the new compiler and runtime instructions are used by default instead of the older compiler and runtime, known as View Engine.

**Features & Updates of Angular 10**

Warnings on CommonJS Imports, Optional Stricter Settings, New Date Range Picker, Enhanced Community Engagement, Boost in ngcc Performance, Async Locking Timeout, Removals and Corrections, Compiler Update.

**Features & Updates of Angular 9**

IVY Presented as the Default Compiler, More Consistent ng-update, API Extractor Updates, Fresh and new alternatives for 'providedIn', IDE and Language Service Enhancements, Typescript Updates, Component Harness, Phantom Template Variable Menace.

**Component** is used to break up the application into smaller components. But **Directive** is used to design re-usable components, which is more behavior-oriented. That is why components are widely used in later versions of Angular to make things easy and build a total component-based model.

**How to install Angular CLI:**

* To install angular, we need **NodeJS**, it manages **npm** dependencies which support some browsers when loading pages. It provides required **libraries to run Angular project**. Node.js serves your **run-time environment** as your localhost.
* Node.js is primarily used for **non-blocking**, **event-driven servers**, due to its single-threaded nature.
* After node is installed, Open the command prompt, in that use npm to install Angular CLI. Run the below Angular CLI command to install Angular CLI.

**---> npm install -g @angular/cli@latest**

* After the successful installation next, we must create AngularApp workspace file.

**ng new AngularApp** //it will create workspace with the name AngularApp

**cd AngularApp** //we are going inside workspace folder called AngularApp

**ng serve -o (OR) ng serve --port 4200** //to run the application

**ng test** //this will run the unit test cases files

The **ng serve** command launches the server, watches our files, and rebuilds the app as we make changes to those files. The --open (or just -o) option automatically opens our browser to <http://localhost:4200/>

**Introduction to Angular**

An **NgModule** is a class marked by the @**NgModule** decorator. @**NgModule** takes a metadata object that describes how to compile a component's template and how to create an injector at runtime. An app always has at least a **root module** that enables **bootstrapping**, and typically has many more feature modules.

**Components** define **views**, which are sets of **screen elements** that Angular can choose among and modify according to your program logic and data.  
**Components** are the main building block for **Angular** applications. Each component consists of: An HTML template that declares what renders on the page. A Typescript class that defines behavior. A CSS selector that defines how the component is used in a template.

Components use **services**, which provide specific **functionality** **not directly** related to views. Service providers can be injected into components as **dependencies**, making your code **modular**, **reusable**, and **efficient**.

Modules, components and services are classes that use **decorators**. These decorators mark their type and provide **metadata** that tells Angular how to use them.

The metadata for a component class associates it with a **template** that defines a view. A template combines ordinary **HTML** with Angular directives and **binding** markup that allow Angular to modify the HTML before rendering it for display.

The metadata for a service class provides the information Angular needs to make it available to components through **dependency** **injection** (DI).

An app's components typically define many views, arranged hierarchically. Angular provides the [**Router**](https://angular.io/api/router/Router) service to help you define **navigation** paths among views. The router provides sophisticated **in-browser navigational** capabilities.

**Files used in AngularApp folder**

AngularApp files which are mainly used in your project are given below:

* **src folder:** This is the folder which contains the **main code** files related to our angular application.
* **app folder:** The app folder contains the **files**, we have created for app components.
* **app.component.css:** This file contains the **cascading style sheets code** for our app component.
* **app.component.html:** This file contains the **html file** related to app component. This is the template file which is used by angular to do the **data binding**.
* **app.component.spec.ts:** This file is a **unit testing file** related to app component. This file is used along with other unit tests. It is run from Angular CLI by the command **ng test**.
* **app.component.ts:** This is the most important **typescript file** which includes the view logic behind the component.
* **app.module.ts:** This is also a typescript file which includes all the **dependencies** for the website. This file is used to define the **needed modules** to be imported, the **components** to be declared and the main component to be **bootstrapped**.

**ng g c tutorial** //g-generate, c-component, tutorial-name of the component folder

**ng g s tutorial** //g-generate, s-service, tutorial-name of the service file

**Module**

* Module is a class with @NgModule metadata.
* Every angular app has at least one root module
* Encapsulation of different similar functionalities.

**Metadata:**

* Metadata describes how to process the class.
* Decorator is used to attach metadata

**Router**

If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the RouterModule.

The RouterModule routes your application to different pages without reloading the entire application.

const routes: Routes = [

  {

      path: “”,

      redirectTo: “admin”,

      pathMatch: 'full',

    },

    {

      path: 'admin',

      component: AdminComponent

    },

];

**Data Binding**

**Interpolation/Expression/Template expression- {{}}**

Interpolation refers to embedding expressions into marked up text.

One-way from **component** to **HTML**

**Event binding- ()**

To bind to an event you use the Angular event binding syntax. This syntax consists of a target event name within parentheses to the left of an **equal sign**, and a quoted template statement to the right. In the following example, the target event name is **click** and the template statement is **onSave().**

One way from **template** to **component** binding handlers for the events

**<button (click)="onSave()">Save</button>**

**Property Binding- []**

Property binding in Angular helps you **set values** for properties of HTML elements or directives. With property binding, you can do things such as **toggle button** functionality, set paths programmatically, and share values between components. Enclose it in square brackets, **[]**, which identifies the property as a target property. A target property is the DOM property to which you want to assign a value. For example, the target property in the following code is the image element's **src** property.

One way binding **component** property to **HTML** DOM property

**<img [src]="itemImageUrl">**

**Two-way binding - [()]**

Two-way binding gives components in your application a way to share data. Use two-way binding to listen for events and update values simultaneously between **parent** and **child** components.

Angular's two-way binding syntax is a combination of square brackets and parentheses, **[()]**. The [()] syntax combines the brackets of property binding, [], with the parentheses of event binding, ()

<app-sizer [(size)]="fontSizePx"></app-sizer>

**Directives**

Directives are instructions in the DOM. They specify how to place your components and business logic in the Angular.

Directives are js class and declared as @directive. There are 3 types of directives in angular.

1. Component directives
2. Structural directives
3. Attribute directives

**Component Directives**

These are used in main class. They contain the details of how the component should be processed, instantiated and used at runtime.

**Structural Directives (Built-in)**

There are start with a \* sign. These directives are used to manipulate and change the structure of the DOM elements.

Ex: **\*ngIf, \*ngFor, \*ngSwitchWhen, \*ngSwitchDefault**

**Attribute directives**

These are used to change the look and behavior of the DOM elements.

Ex: **ngClass, ngStyle**

**Pipes**

In Angular1, filters are used which are later called Pipes onwards Angular2.

Pipes are used to transform the data. It is denoted by symbol |

Syntax: **{{title | uppercase}}**

Pipe takes integers, strings, arrays and date as input separated with |. It transforms the data in the format as required and displays the same in the browser.

**Template driven Form**

ngForm, ngSubmit, ngModel, ngModelGroup

[**Model driven Form or Relative Form or Dynamic Form**](https://www.youtube.com/watch?v=G3MrNLfL2Ss)

Form Group, Form Control, Form Array, Form Builder

**Differences between Template-driven and Reactive Forms**

* Template-driven forms make use of the "**FormsModule**", while reactive forms are based on "**ReactiveFormsModule**".
* Template-driven forms are **asynchronous** in nature, whereas Reactive forms are mostly **synchronous**.
* In a template-driven approach, most of the logic is driven from the **template**, whereas in reactive-driven approach, the logic resides mainly in the **component** or **typescript** **code**.

**Difference between UI and UX:**

* The “**UI**” in UI design stands for “**user interface**.” The user interface is the **graphical layout** of an application. It consists of the button’s users click on, the text they read, the images, sliders, text entry fields, and all the rest of the items the user interacts with. This includes screen layout, transitions, interface animations and every single micro-interaction. Any sort of visual element, interaction, or animation must all be designed. This job falls to UI designers. They decide what the application is going to look like. They must choose color schemes and button shapes — the width of lines and the fonts used for text. **UI designers create the look and feel of an application’s user interface.**
* **User experience** is determined by how **easy or difficult** it is to **interact** with the **user interface elements** that the UI designers have created. They determine the **structure** of the interface and the **functionality**. How it’s organized and how all the parts relate to one another. They design how the interface works. If it works well and feels seamless, the user will have a good experience. But if navigation is **complicated** or unintuitive, then a lousy user experience is likely. UX designers work to avoid the second scenario.
* **UI designers** are tasked with deciding how the user interface will look, **UX designers** are in charge of determining how the user interface operates.

**What is component lifecycle hooks in angular?**

A component instance has a lifecycle that starts when Angular **initiates** the component **class** and **renders** the component **view** along with its **child views**. The lifecycle continues with change detection, as Angular checks to see when data-bound **properties** change, and **updates** both the **view** and the **component** as needed. The lifecycle **ends** when Angular **destroys** the component instance and **removes** its rendered template from the **DOM.** Directives have a similar **lifecycle**, as **Angular** creates, updates, and destroys instances in the course of execution. Eg: ngOnChanges(), ngOnInit(),ngDoCheck(), ngAfterContentInit(), ngAfterContentChecked(), ngAfterViewInit(), ngAfterViewChecked(), ngOnDestroy().

**How to integrate component in angular?**

To define a component, you always import the Component symbol. The @Component decorator provides the Angular metadata for the component.

**Data communication between components in angular:**

There are five ways to share data between components:

* Parent to child component
* Child to parent component
* Sharing data between sibling components
* Sharing data using **ViewChild** property
* Sharing data between not related components

**NgClass** & **NgStyle** are Angular Directives that allow us to conditionally apply one-to-many classes/styles to an element.

**What are the basic rules of Decorators?**

A Class **Decorator** is declared just before a class declaration. The class **decorator** is applied to the constructor of the class and can be used to observe, modify, or replace a class definition. A class **decorator** cannot be used in a declaration file, or in any other context.

**What is bootstrapping in angular?**

Angular. bootstrap is a function component in the core ng module that is used for **starting** up the angular application **manually**, which gives you more control over how you initialize your application.

[**How do you provide a component in service?**](https://www.youtube.com/watch?v=cznobRuS0UQ)

**Inject a service into a component**

* In the **service**: marking/un-marking the service with @Injectable.
* In **bootstrapping**: Adding/removing MyService into the bootstrapping list of providers.
* In **component**. specifying the service as a provider in @Component({providers: [MyService]})

**Angular Generic Features:**

* Cross-Platform. With Angular, you can develop progressive web applications (PWA).
* High Speed & Optimum Performance.
* MVC Architecture.
* Efficient Two-Way Data Binding.
* Less Code Framework.
* Angular CLI (Command Line Interface)

**Difference between onClick() and ng-Click**

**ng-click** will allow your click event to directly call a function on the $scope where the element is in, **onclick** will not directly call the function on the $scope.

**Dependency injection**

Dependencies are services or objects that a class needs to perform its **function**. Dependency injection isa **design pattern** in which a class requests dependencies from external sources rather than creating them.

**What is Can Activate and Load Activate?**

CanActivate, CanActivateChild, CanDeactivate, Resolve, CanLoad - **Router guards**

* It is an **interface** that a class can implement to be a **guard deciding** if a route can be activated. If all guards return **true**, navigation **continues**. If any guard returns **false**, navigation is **cancelled**. If any guard returns a **UrlTree**, the current navigation is **cancelled** and a **new** **navigation** begins to the **UrlTree** returned from the guard. We use this guard, when we **want to check on some condition**, before activating the component or showing it to the user. This allows us to **cancel the navigation**.

interface CanActivate { **canActivate**(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree }

* It is an **interface** that a class can implement to be a guard deciding if **children can be loaded**. If all guards return **true**, navigation **continues**. If any guard returns **false**, navigation is **cancelled**. If any guard returns a **UrlTree**, current navigation is **cancelled** and a new navigation starts to the UrlTree returned from the guard. C**anLoad** is used to prevent the application from **loading** entire modules lazily if the user is not authorized to child routes.

interface CanLoad { **canLoad**(route: Route, segments: UrlSegment[]):

Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree }

[**What is observable?**](https://www.youtube.com/watch?v=c2OG74Lz8Tk)

**Observables** provide support for passing messages between parts of your application. They are used as a technique for event handling, asynchronous programming, and handling multiple values.

**Observer** is used to track an observable and it consciously listens to an observable. We can start and end the listing of observable with observer.

**What is the difference between observable and promises?**

Both observables and promises help us work with **asynchronous** functionality in JavaScript. Promises deal with **one asynchronous** event at a time, while observables handle a **sequence of asynchronous** events over a period of time.

**What are pipes in angular?**

* They are a simple way to transform values or data on the flow in an **Angular** template. There are some built in **pipes**, but you can also build your own **pipes**. A **pipe** takes in a value or values and then returns a modified value. Ex: **DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, DecimalPipe, PercentPipe.**

**Pure and Impure pipes**

By default, pipes are defined as **pure** so that Angular executes the pipe only when it detects a pure change to the input value. But impure pipe executes **every time irrespective** of source has changed or not.

**Async** is an example of an **impure pipe**. It is always checking for new input data. Pure will be true if not specified. When the **pipe** is pure, it does not check the input to see whether or not to update the transformed data.

**Angular closure**

A closure is the combination of a function enclosed with references to its surrounding stated. In other words, a closure gives you **access** to an **outer function's scope** from an **inner function**. In JavaScript, closures are created every time a function is created, at function creation time.

**index**. **html page**

This **file** will generally contain your homepage content, that is, the text and images that people see when they first go to your site or default **page** shown on a website if no other **page** is specified when a visitor requests the site.

Working with **@ViewChildren** is similar to **@ViewChild**, but the difference between the two is @ViewChildren provides a **list of element references** rather than returning a single reference. It is used to reference multiple elements. We can then iterate the list of the element referenced by the variable.

The Hash Location Strategy use the **Hash style** routing, while Path Location Strategy uses the **HTML 5** routing.

1. **HashLocationStrategy**  
   Where URL looks like http://localhost:4200/#/product
2. **PathLocationStrategy** (default)  
   Where URL looks like http://localhost:4200/product

* **The Jelly Stylesheet Library (JSL)**

The JSL tag library implements an XSLT-like declarative XML based processing engine which allows dynamic stylesheets and a free mix and match of all Jelly tags within the script.

The JSL tag library relies on the XML tags in JSTL to do most of its work, so JSL just provides the declarative XML processing via <jsl:stylesheet> <jsl:template>and <jsl:applyTemplates>

* **What is the RxJS in Angular?**

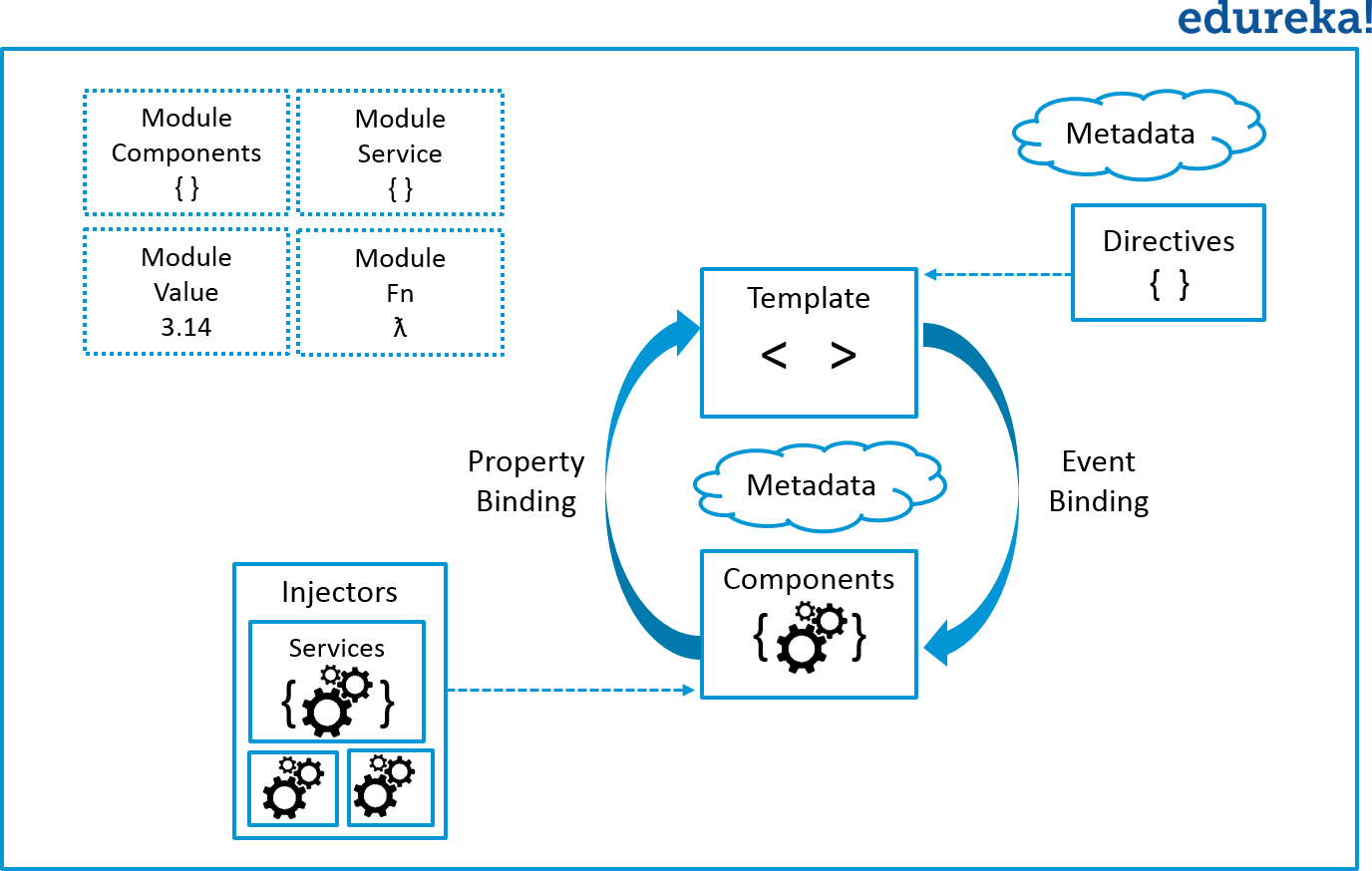
RxJS (**Reactive Extensions for JavaScript**) is a library for **reactive** programming using **observables** that makes it easier to compose **asynchronous** or callback-based code. It provides one core type, the [Observable](https://rxjs.dev/guide/observable), satellite types (Observer, Schedulers, Subjects) and operators (map, filter, reduce, every, etc.…) to allow handling asynchronous events as collections.

* **What is node Forge?**

The Forge software is a fully **native** implementation of the **TLS protocol** in JavaScript, a set of **cryptography** **utilities**, and a set of tools for **developing Web Apps** that utilize many network resources.

* **Just-in-Time (JIT),** compiles your app in the browser at **runtime**. **Ahead-of-Time (AOT)**, compiles your app at **build time** on the server. JIT compilation is the **default** when you run the **ng build** (build only) or **ng serve** (build and serve locally) CLI commands. For AOT compilation, include the **--aot** option with the ng build or ng serve command. JIT and AOT are **two** ways to compile code in an Angular project. We use **JIT** in **development** mode while **AOT** is for **production** mode.
* **What is angular material?**

Angular Material is a **UI component library** for **Angular JS** developers. Angular Material components help in constructing **attractive, consistent**, and **functional** web pages and web applications while adhering to modern web design principles like browser portability, device independence, and graceful degradation.



* **What is JIT and AOT compilation?**

In **JIT - (Just-in-Time)** compilation the application compiles inside the browser during runtime.

In **AOT - (Ahead-of-Time)** the application compiles during the build time.

* **What is Eager Loading?**

This is the default module loading strategy. Feature modules under Eager Loading are loaded before the application starts.

* This is typically used for small size applications.
* **What is Lazy Loading?**

This is dynamically loading the feature modules when there is a demand. This makes the application faster.

* It is used for bigger applications where all the modules are not required at the start of the application.
* **What is Event capturing?**

This process starts with capturing the event of the outermost element and then propagating in to the innermost elements.

* **What is Event Bubbling?**

This process starts with capturing the event of the innermost element and then propagation in to the outermost element.

Q1) Explain the different directives in Angular JS.

**Directives** are markers on the DOM element which tell AngularJS to attach a specified behavior to that DOM element or even transform the DOM element with its children. Simple AngularJS allows extending HTML with new attributes called Directives. AngularJS has a set of built-in directives which offers functionality to the applications. It also defines its own directives. A directive can be defined using some functions which are: Element name, Attribute, Class, and Comment.

**Why use Directive in AngularJS?**

* It gives support to creating custom directives for different types of elements.
* A directive is activated when the same element or matching element is there in front.
* It is used to give more power to HTML by helping them with the new syntax.
* Directive classes, like component classes, can implement life-cycle hooks to influence their configuration and behavior.

Q2) How will you boot Angular JS?

Q3) What is jQLite? Write a small code to explain.

Q4) Explain $scope object in Angular JS. Explain with a small snippet.

Q5) What is a SPA in Angular JS?

 Single Page Applications are web applications that load a single HTML page and only a part of the page instead of the entire page gets updated with every click of the mouse. The page does not reload or transfer control to another page during the process. This ensures high performance and loading pages faster. Most modern applications use the concept of SPA. In the SPA, the whole data is sent to the client from the server at the beginning. As the client clicks certain parts on the webpage, only the required part of the information is fetched from the server and the page is rewritten dynamically. This results in a lesser load on the server and is cost-efficient. SPAs use AJAX and HTML5 to create a fluid and responsive Web applications and most of the work happens on the client-side. Popular applications such as Facebook, Gmail, Twitter, Google Drive, Netflix, and many more are examples of SPA.

Q6) What is dependency injection?

In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), **dependency injection** is a [design pattern](https://en.wikipedia.org/wiki/Software_design_pattern) in which an [object](https://en.wikipedia.org/wiki/Object_(computer_science)) or [function](https://en.wikipedia.org/wiki/Subroutine) receives other objects or functions that it depends on. A form of [inversion of control](https://en.wikipedia.org/wiki/Inversion_of_control), dependency injection aims to [separate the concerns](https://en.wikipedia.org/wiki/Separation_of_concerns) of constructing objects and using them, leading to [loosely](https://en.wikipedia.org/wiki/Loose_coupling) [coupled](https://en.wikipedia.org/wiki/Coupling_(computer_programming)) programs.[[1]](https://en.wikipedia.org/wiki/Dependency_injection#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Dependency_injection#cite_note-MarkSeeman2011P4-2)[[3]](https://en.wikipedia.org/wiki/Dependency_injection#cite_note-3) The pattern ensures that an object or function which wants to use a given [service](https://en.wikipedia.org/wiki/Service_(systems_architecture)) should not have to know how to construct those services. Instead, the receiving '[client](https://en.wikipedia.org/wiki/Client_(computing))' (object or function) is provided with its dependencies by external code (an 'injector'), which it is not aware of.[[4]](https://en.wikipedia.org/wiki/Dependency_injection#cite_note-HollywoodPrinciple.c2-4) Dependency injection helps by making implicit dependencies explicit and helps solve the following problems:[[5]](https://en.wikipedia.org/wiki/Dependency_injection" \l "cite_note-5)

Q7) What happens when a double click event is called?

Q8) Explain different ways by which a directive can be invoked.

You can invoke a directive by using:

* Element name
* Attribute
* Class
* Comment

Q9) What are filters?

The filter filter selects a subset of an array.

The filter filter can only be used on arrays, and it returns an array containing only the matching items.

Q10) What is the difference between prefix $ and $$?

The $ in AngularJs is a built-in object.It contains application data and methods.

The scope($) acts as a link between controller and view.

$$ in this are treated as private variables. We use $$ to avoid the internal variable conflicts and not to expose for external use.