1. what is one-way data binding?

One-way data binding in AngularJS refers to the mechanism where data flows in only one direction, typically from the model (data source) to the view (UI). This means that changes in the model automatically update the view, but changes in the view do not affect the model directly.

In other words, when you update data in your JavaScript code (the model), AngularJS will automatically update the corresponding parts of your HTML (the view) to reflect those changes. However, if a user interacts with the UI and changes something, it won't automatically update the model. To achieve this, you would typically use two-way data binding in AngularJS, which allows data to flow in both directions, enabling changes in the view to update the model as well.

2. Explain two-way data binding?

Two-way data binding in AngularJS is a feature that allows data to flow in both directions, between the model (data source) and the view (UI). This means that changes in the model automatically update the view, and conversely, changes in the view also affect the model. It keeps the model and the view synchronized.

In AngularJS, two-way data binding is often achieved using directives like ng-model, which is commonly used with input fields. When you use ng-model in an input element, any changes made by the user in the input field are automatically reflected in the model, and changes in the model are immediately reflected in the input field. This bidirectional data flow simplifies the development of interactive and dynamic web applications as you don't have to write a lot of boilerplate code to manually update the view or model when changes occur.

Here's a simple example of two-way data binding in AngularJS:

htmlCopy code

```
<!DOCTYPE html>
<html ng-app="myApp">
 <head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
 </head>
 <body ng-controller="myController">
  <input type="text" ng-model="message" />
  <p>{{message}}</p>
 </body>
 <script>
  var app = angular.module("myApp", []);

  app.controller("myController", function ($scope) {
   $scope.message = "Hello, AngularJS!";
  });
```

```
 </script>
</html>
```

3. what is service? list out some inbuild services.

In AngularJS, a service is a reusable component that provides specific functionality or data to different parts of your application. Services are typically used for tasks such as fetching data from a server, handling business logic, or sharing common functionality across multiple controllers, directives, or other parts of your application. Services are singletons, which means that they are instantiated only once and shared throughout the application.

AngularJS provides several built-in services that cover a wide range of functionality. Here are some of the commonly used built-in services:

1. **$http:** This service is used for making HTTP requests, such as fetching data from a remote server or sending data to a server.
2. **$rootScope:** The $rootScope service is a parent scope for all other scopes in the AngularJS application. It's often used to store global data that needs to be accessed from various parts of the application.
3. **$location:** This service helps with interacting with the browser's URL. It can be used to read and manipulate the URL, making it useful for building single-page applications.
4. **$timeout and $interval:** These services allow you to schedule code execution after a specified time delay or at regular intervals, respectively.
5. **$q and $timeout:** These services are used for managing asynchronous operations and promises. They are essential for handling asynchronous tasks, such as making multiple HTTP requests.
6. **$log:** This service provides a simple logging mechanism for debugging and error reporting.
7. **$filter:** The $filter service is used for formatting and filtering data in the view.
8. **$route and $routeParams:** These services are used for building and managing routes in AngularJS applications. They allow you to create single-page applications with different views.
9. **$translate:** If you need to implement internationalization (i18n) and localization (l10n) in your application, the $translate service is used for managing translations.
10. **$cacheFactory:** This service allows you to create and manage an in-memory cache for frequently used data, improving the performance of your application.

4. What is dependency injection? what are the different ways to implement it?

Dependency injection (DI) is a design pattern used in software development, including AngularJS, to achieve the following goals:

1. **Decoupling:** It helps in reducing the coupling between components in your application. In a well-designed system, components should depend on abstractions rather than concrete implementations. Dependency injection makes it easier to achieve this.
2. **Testing:** It simplifies unit testing. By injecting dependencies into a component, you can easily substitute real dependencies with mock or fake objects during testing. This allows you to isolate the component being tested and focus on its behavior without involving the actual dependencies.

In AngularJS, dependency injection is a core concept, and it's used to provide components with the objects or services they depend on. The framework manages the instantiation and injection of these dependencies into your components.

There are several ways to implement dependency injection in AngularJS:

1. **Constructor Injection:** This is the most common way to perform dependency injection in AngularJS. You declare the dependencies as parameters in the constructor of your component (usually a controller or a service), and AngularJS takes care of injecting the correct instances. For example:

   javascriptCopy code
   ```
   myApp.controller('MyController', ['$scope', 'myService', function($scope, myService)
   {
     // $scope and myService are injected as dependencies
   }]);
   ```

2. **Service Annotation:** You can use the $inject property of a function to explicitly specify the dependencies. This is particularly useful when you need to minify your code since minification can break the automatic injection based on parameter names. For example:

   javascriptCopy code
   ```
   myApp.controller('MyController', MyController);
   MyController.$inject = ['$scope', 'myService'];

   function MyController($scope, myService) {
     // $scope and myService are injected as dependencies
   }
   ```

3. **Array Annotation:** This is similar to the service annotation but involves declaring dependencies as an array of strings. This approach is often used when the function is defined inline. For example:

   javascriptCopy code
   ```
   myApp.controller('MyController', ['$scope', 'myService', function($scope, myService)
   {
     // $scope and myService are injected as dependencies
   ```

```
}]);
```

4. **Implicit Annotation:** In earlier versions of AngularJS, you could rely on parameter names matching the service names. For example:

   ```
   javascriptCopy code
   myApp.controller('MyController', function($scope, myService) {
     // $scope and myService are injected as dependencies based on parameter names
   });
   ```

It's worth noting that in modern Angular versions (Angular 2+), the dependency injection system has evolved, and the implementation details are different. The core concept of dependency injection remains the same, but the syntax and mechanisms have been updated to align with TypeScript and ES6+ standards. The new Angular framework uses decorators and TypeScript to handle dependency injection more efficiently.

5. what is SPA ? List out its advantages.

SPA stands for Single Page Application. It is a type of web application or website that loads a single HTML page and dynamically updates the content as the user interacts with the application. In a traditional multi-page application, clicking on a link typically results in the entire page being reloaded from the server. In contrast, in a SPA, only the necessary parts of the page are updated, and transitions between different views are often smoother and faster.

Advantages of Single Page Applications (SPAs):

1. **Improved User Experience:** SPAs offer a more fluid and responsive user experience because they can update content without full page reloads. This leads to faster interactions and a more native app-like feel.
2. **Faster Load Times:** Initial loading of the SPA is often slower because it loads the entire application code upfront. However, subsequent interactions are faster since only data is exchanged with the server, reducing the amount of data to be transferred.
3. **Reduced Server Load:** SPAs can reduce the load on the server as they require fewer requests and less data transfer compared to traditional multi-page applications.
4. **Smoother Transitions:** SPAs can provide smooth transitions between views and sections of the application, creating a more seamless user experience.
5. **Better Caching:** SPAs can effectively cache data and assets, further reducing load times for returning users.
6. **Improved Development Workflow:** SPAs often use client-side routing, which makes it easier for developers to manage application state and create dynamic interfaces.
7. **Code Reusability:** SPAs often make use of component-based architecture, allowing developers to reuse components throughout the application, which can streamline development and maintenance.

8. **Enhanced Offline Functionality:** With the use of service workers and client-side caching, SPAs can continue to work partially or entirely offline, providing a robust experience for users with intermittent or unreliable internet connections.
9. **Better for Mobile:** SPAs are well-suited for mobile applications due to their fast and responsive nature. Many mobile apps are, in fact, SPAs wrapped in a native shell.
10. **Easier Debugging:** Debugging can be easier in SPAs because you can inspect and manipulate the application's state in the browser's developer tools.
11. **Dynamic Content Loading:** SPAs can load content on-demand, which can be useful for applications with a lot of data or different user roles, as it reduces the initial load time and keeps the interface simple.

6. write a simple program showing the use of ng-repeat.

Sure, here's a simple example of how to use ng-repeat in an AngularJS application. In this example, we'll create a list of items and use ng-repeat to display them in an HTML table.

HTML:

htmlCopy code

```
<!DOCTYPE html>
<html ng-app="myApp">
 <head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
 </head>
 <body ng-controller="myController">
  <h2>List of Items:</h2>
  <table>
   <tr>
    <th>ID</th>
    <th>Name</th>
   </tr>
   <tr ng-repeat="item in items">
    <td>{{ item.id }}</td>
    <td>{{ item.name }}</td>
   </tr>
  </table>
 </body>
 <script>
  var app = angular.module("myApp", []);

  app.controller("myController", function ($scope) {
   $scope.items = [
    { id: 1, name: "Item 1" },
    { id: 2, name: "Item 2" },
    { id: 3, name: "Item 3" },
    { id: 4, name: "Item 4" },
```

```
    ];
  });
 </script>
</html>
```

7. what is a filter explain any three simple filters.

In AngularJS, filters are used to format or transform data before displaying it in the view. They allow you to apply a variety of transformations to data, such as formatting numbers, dates, or strings. Here are three simple filters in AngularJS and their explanations:

1. **currency Filter:**
   - **Purpose:** The currency filter is used for formatting numbers as currency values.
   - **Usage:** You can use it to display numbers with a currency symbol, decimal places, and optional custom symbols.
   - **Example:**

     htmlCopy code
     ```
     <p>{{ amount | currency: "$" }}</p>
     <!-- Displays: $100.00 -->
     ```

2. **date Filter:**
   - **Purpose:** The date filter is used to format dates.
   - **Usage:** You can format a date in various ways, specifying the format you want to use.
   - **Example:**

     htmlCopy code
     ```
     <p>{{ today | date: "dd/MM/yyyy" }}</p>
     <!-- Displays: 28/10/2023 (assuming 'today' is a Date object) -->
     ```

3. **uppercase and lowercase Filters:**
   - **Purpose:** The uppercase and lowercase filters are used to transform strings to all uppercase or all lowercase characters, respectively.
   - **Usage:** They are helpful for ensuring consistent letter case in your data.
   - **Example:**

     htmlCopy code
     ```
     <p>{{ text | uppercase }}</p>
     <!-- Displays: HELLO, ANGULARJS -->
     ```
     htmlCopy code
     ```
     <p>{{ text | lowercase }}</p>
     <!-- Displays: hello, angularjs -->
     ```

These are just a few examples of the many filters available in AngularJS. You can also create custom filters to suit your specific formatting or data transformation needs. Filters are a powerful tool for presenting data in a user-friendly and readable format in your AngularJS applications.

8. write a program to create a service to calculate the area of the circle

To create an AngularJS service to calculate the area of a circle, you can follow these steps:

1. Define an AngularJS module and create a service within it.
2. In the service, define a method that calculates the area of a circle based on its radius.
3. Inject this service into a controller, and use the service method to calculate and display the area in your HTML.

Here's a simple example:

htmlCopy code

```
<!DOCTYPE html>
<html ng-app="myApp">
 <head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
 </head>
 <body ng-controller="CircleController">
  <h2>Circle Area Calculator</h2>
  <div>
   <label for="radius">Enter Radius: </label>
   <input type="number" id="radius" ng-model="radius" />
   <button ng-click="calculateArea()">Calculate Area</button>
  </div>
  <p ng-if="area">The area of the circle is: {{ area }}</p>
 </body>
 <script>
  var app = angular.module("myApp", []);

  app.service("CircleService", function () {
   this.calculateArea = function (radius) {
    return Math.PI * radius * radius;
   };
  });

  app.controller("CircleController", function ($scope, CircleService) {
   $scope.calculateArea = function () {
    if ($scope.radius) {
     $scope.area = CircleService.calculateArea($scope.radius);
    } else {
```

```
      $scope.area = null;
    }
  };
 });
</script>
</html>
```

9. Explain angular js MVC architecture

AngularJS follows a variation of the Model-View-Controller (MVC) architectural pattern known as MV* (Model-View-Whatever) or MVW (Model-View-Whatever) architecture. In AngularJS, this is more accurately described as the Model-View-ViewModel (MVVM) architecture. Here's an explanation of how this architecture works in AngularJS:

1. **Model (M):**
   - The Model represents the application's data and business logic. In AngularJS, the Model is often represented as JavaScript objects or arrays. These objects store the actual data that the application uses and manipulates. Models are typically defined as part of Angular services or factories.
2. **View (V):**
   - The View is the user interface that presents the data to the user. In AngularJS, the View is represented by HTML templates and is responsible for displaying the data and responding to user interactions. Directives are used to enhance HTML elements and make them interactive, while expressions and data binding are used to display dynamic content in the View.
3. **ViewModel (VM):**
   - In AngularJS, the ViewModel is represented by controllers. Controllers act as an intermediary between the Model and the View. They handle user input, update the Model, and control the flow of data to and from the View. Controllers are responsible for defining the application's behavior and logic.

AngularJS enhances the traditional MVC pattern with a two-way data binding mechanism, which automatically keeps the Model and View in sync. When the Model changes, the View is automatically updated, and when the user interacts with the View, the changes are reflected in the Model. This two-way data binding simplifies the development of dynamic and responsive web applications.

AngularJS also introduces the concept of services and dependency injection, which allows you to create reusable components for handling common tasks, such as making HTTP requests or managing shared data.

In summary, AngularJS's MVVM architecture can be described as follows:

- **Model:** Represents the application's data and business logic.
- **View:** Represents the user interface and is responsible for rendering data.

- **ViewModel (Controller):** Acts as an intermediary between the Model and the View, controlling the application's behavior and logic. AngularJS introduces two-way data binding and dependency injection to enhance the traditional MVC pattern.

10. What is directive? explain various directives

In AngularJS, directives are a fundamental part of the framework that allows you to extend HTML with new behavior and attributes, making your HTML more dynamic and interactive. Directives are markers on HTML elements that tell AngularJS to do something to the element or its contents. They are a way to teach the browser new HTML syntax and functionality.

AngularJS provides several built-in directives, and you can also create custom directives to encapsulate and reuse complex behavior or UI components. Here are some commonly used built-in directives in AngularJS:

1. **ng-app:** This directive defines the root element of the AngularJS application. It indicates where the Angular application starts. For example:

   htmlCopy code
   ```
   <div ng-app="myApp">
     <!-- The AngularJS application starts here -->
   </div>
   ```

2. **ng-controller:** This directive defines the controller for a specific section of the HTML. It connects a controller to the HTML element, allowing you to access controller functions and properties within that section. For example:

   htmlCopy code
   ```
   <div ng-controller="MyController">
     <!-- Controller functions and properties are accessible here -->
   </div>
   ```

3. **ng-model:** The ng-model directive is used for two-way data binding. It binds the value of HTML controls (input, select, textarea) to application data, allowing changes in the UI to automatically update the model, and vice versa. For example:

   htmlCopy code
   ```
   <input type="text" ng-model="myVariable" />
   ```

4. **ng-repeat:** This directive is used to iterate over a collection (array or object) and generate HTML elements for each item in the collection. It allows you to create dynamic lists and tables. For example:

   htmlCopy code
   ```
   <ul>
     <li ng-repeat="item in items">{{ item.name }}</li>
   ```

```
</ul>
```

5. **ng-show and ng-hide:** These directives conditionally show or hide elements in the view based on a given expression. For example:

   ```
   htmlCopy code
   <div ng-show="showElement">This is shown when showElement is true</div>
   <div ng-hide="hideElement">This is hidden when hideElement is true</div>
   ```

6. **ng-click:** This directive adds a click event handler to an element, allowing you to call a function in response to a user's click. For example:

   ```
   htmlCopy code
   <button ng-click="doSomething()">Click me</button>
   ```

7. **ng-if:** The ng-if directive removes or recreates a portion of the DOM based on a condition. If the condition is true, the element is included in the DOM; if false, it is removed. For example:

   ```
   htmlCopy code
   <div ng-if="shouldShow">This is included if shouldShow is true</div>
   ```

These are just a few of the many directives available in AngularJS. You can create custom directives to encapsulate complex functionality and promote code reusability in your application. Directives are a powerful feature of AngularJS that enable you to build dynamic and interactive web applications.

11. Write angular js program to display name qualification and address using MVC architecture

Certainly, here's a simple AngularJS program that follows the MVC architecture to display a person's name, qualification, and address.

HTML (index.html):

```
htmlCopy code
<!DOCTYPE html>
<html ng-app="myApp">
 <head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
 </head>
 <body>
  <div ng-controller="PersonController">
   <h2>Person Details</h2>
   <p>Name: {{ person.name }}</p>
   <p>Qualification: {{ person.qualification }}</p>
```

```
    <p>Address: {{ person.address }}</p>
  </div>
 </body>
 <script>
  var app = angular.module("myApp", []);

  app.controller("PersonController", function ($scope) {
   // Model (M)
   $scope.person = {
    name: "John Doe",
    qualification: "Bachelor's Degree",
    address: "123 Main St, City, Country",
   };
  });
 </script>
</html>
```

In this program:

- We create an AngularJS module "myApp" and define a controller "PersonController" to control the behavior of a specific section of the HTML.
- Inside the controller, we define the Model (M) using the $scope variable. The model represents the person's information, including their name, qualification, and address.
- In the HTML, we use expressions (e.g., {{ person.name }}) to display the values of the model properties. The View (V) is responsible for rendering this data.

This program follows the MVC architecture, where:

- Model (M) is represented by the data stored in $scope.person.
- View (V) is represented by the HTML elements that display the person's information.
- Controller (C) is represented by the "PersonController," which controls the behavior and logic associated with the person's data.

When you open this HTML in a browser with AngularJS included, it will display the person's name, qualification, and address on the webpage.