**Angular Interview Questions**

**1)Angular Lifecycle hooks**

Angular lifecycle hooks are methods that allow you to tap into key moments in a component's lifecycle. Here's the typical execution sequence:

1. **constructor** - Runs when Angular instantiates the component.
2. **ngOnChanges** - Called when input properties change.
3. **ngOnInit** - Runs once after the component's inputs are initialized.
4. **ngDoCheck** - Invoked during every change detection cycle.
5. **ngAfterContentInit** - Runs once after projected content is initialized.
6. **ngAfterContentChecked** - Called after projected content is checked.
7. **ngAfterViewInit** - Runs once after the component's view is initialized.
8. **ngAfterViewChecked** - Called after the component's view is checked.
9. **ngOnDestroy** - Runs before the component is destroyed.

**2)View Encapsulation in Angular**

View Encapsulation in Angular controls how styles are applied to components, ensuring they don't unintentionally affect other parts of the application. Angular provides three encapsulation modes:

1. **ViewEncapsulation.Emulated** - Angular modifies CSS selectors so styles apply only to the component's view, mimicking Shadow DOM behavior.
2. **ViewEncapsulation**.**ShadowDom** - Uses the browser's native Shadow DOM API to encapsulate styles within the component.
3. **ViewEncapsulation**.**None** - No encapsulation; styles are globally applied and can affect any element in the application.

**3)How componenet share data or interact with each other**

Angular components can share data and interact with each other in several ways, depending on their relationship. Here are the common methods:

1. Parent to Child Communication
   * Using @**Input**() decorator: The parent component passes data to the child component via property binding.
   * Example: <child-component [data]="parentData"></child-component>
2. Child to Parent Communication
   * Using @**Output**() decorator with EventEmitter: The child component emits an event that the parent listens to.
   * Example: <child-component (event)="handleEvent($event)"></child-component>
3. Sibling Component Communication
   * Using a shared service: A service acts as a mediator, storing and sharing data between sibling components.
   * Example: A service with a **BehaviorSubject** or **Subject** can be used to notify components of changes.
4. Communication Without Direct Relationship
   * Using a shared service with RxJS observables: Components subscribe to a service that broadcasts data updates.
   * Using Angular's **ViewChild** or **ContentChild**: Allows a parent component to access a child component's properties and methods directly.

**4)Data Bindings in angular**

Data binding in Angular is a powerful mechanism that keeps the component and the view in sync. It allows data to flow between the template and the component efficiently. There are four types of data binding:

1. **Interpolation** ({{ }})
   * Used to bind component properties to the view.
   * Example: <h1>{{ title }}</h1>
2. **Property** **Binding** ([property] syntax)
   * Binds a property of an HTML element to a component property.
   * Example: <img [src]="imageUrl">
3. **Event** **Binding** ((event) syntax)
   * Allows the component to respond to user interactions.
   * Example: <button (click)="handleClick()">Click Me</button>
4. **Two**-**Way** **Binding** ([(ngModel)] syntax)
   * Synchronizes data between the component and the view.
   * Example: <input [(ngModel)]="userName">

5)**Event Binding in Angular**

Event binding in Angular allows components to listen for and respond to user interactions like clicks, key presses, and mouse movements. It uses a special syntax where the event name is enclosed in parentheses and linked to a method in the component.

**Example:**

<button (click)="handleClick()">Click Me</button>

In this example, when the button is clicked, the handleClick() method in the component is executed.

**Key Features:**

* **Binding to native events**: You can bind to standard events like click, **keyup**, **mouseover**, etc.
* **Passing event data**: Use $event to access event details.
* <input (keyup)="onKeyPress($event)">
* **Using event modifiers**: Angular allows filtering events using modifiers like .enter, .shift, etc.
* <input (keyup.enter)="submitForm()">

**6)Attribute Binding in angular**

Attribute binding in Angular allows you to set values for attributes dynamically in your templates. It is useful when dealing with attributes that do not have corresponding DOM properties, such as ARIA attributes or SVG attributes.

**Syntax:**

<p [attr.attribute-name]="expression"></p>

If the expression evaluates to null or undefined, Angular removes the attribute altogether.

**Common Use Cases:**

1. **Binding ARIA attributes** (for accessibility)
2. <button [attr.aria-label]="actionName">{{ actionName }}</button>
3. **Binding colspan in tables** (since colspan is an attribute, not a property)
4. <td [attr.colspan]="1 + 1">Merged Cells</td>
5. **Setting custom attributes dynamically**
6. <div [attr.data-custom]="customValue"></div>

7)**Class Binding in Angular**

Class binding in Angular allows you to dynamically add or remove CSS classes from an element based on component properties. It helps in styling elements conditionally.

**Syntax:**

<button [class.active]="isActive">Click Me</button>

In this example, the active class is applied only if isActive is true.

**Different Ways to Use Class Binding:**

1. **Binding a Single Class**
2. <div [class.highlight]="isHighlighted"></div>
   * Adds the highlight class when isHighlighted is true.
3. **Binding Multiple Classes Using Object Syntax**
4. <div [class]="{'bold': isBold, 'italic': isItalic}"></div>
   * Adds bold and italic classes based on the respective boolean values.
5. **Binding Multiple Classes Using String Syntax**
6. <div [class]="dynamicClasses"></div>
   * dynamicClasses can be a string like "class1 class2".

**7)Style binding in Angular**

Style binding in Angular allows you to dynamically set styles on elements based on component properties. It helps in applying styles conditionally and making UI elements more interactive.

Syntax:

<p [style.color]="textColor">Styled Text</p>

In this example, the color style is set based on the textColor property in the component.

Different Ways to Use Style Binding:

1. Binding a Single Style Property
2. <div [style.background-color]="bgColor"></div>
   * Sets the background color dynamically.
3. Binding Multiple Styles Using Object Syntax
4. <div [style]="{'width': widthValue, 'height': heightValue}"></div>
   * Allows multiple styles to be applied dynamically.
5. Adding Units to Style Values
6. <p [style.font-size.px]="fontSize">Dynamic Font Size</p>
   * .px, .em, or % can be used to specify units.

**🚀**