■ College Answer Generator

Comprehensive Answers for Computer Science

Subject:	Computer Science
Answer Mode:	Exam Mode
Total Questions:	1
Generated On:	August 05, 2025 at 02:54 PM
Custom Instructions:	Yes

Concise, exam-focused answers with key points and formal academic language.

Question 1

Q1: What are Angular Directives? Explain with types and examples.

Answer:

Angular Directives are a fundamental concept in the Angular framework, serving as special markers on a DOM element that tell Angular's HTML compiler to attach a specific behavior to that element or transform the DOM structure. Essentially, directives allow developers to **extend HTML vocabulary** and functionality, enabling dynamic manipulation of the user interface based on data and application logic. They are crucial for building declarative, reusable, and maintainable user interfaces by encapsulating DOM manipulation logic and applying it declaratively in templates. Every Angular component is technically a directive, but with an associated template.

```
**Definition:** A **Directive** in Angular is a class that is
decorated with `@Directive()` and provides instructions to
Angular to render, transform, or add behavior to an element in
the DOM. They enable **declarative programming** for UI
manipulation, allowing developers to describe *what* should
happen to the DOM rather than *how* to do it imperatively.
Directives are instrumental in creating dynamic and interactive
user interfaces by allowing the modification of the DOM
structure, appearance, or behavior without directly interacting
with the DOM API.
```

Types of Directives: Angular categorizes directives into three main types, each serving a distinct purpose:

```
1. **Component Directives:** * **Definition:** These are the most
common type of directives and are fundamental building blocks of
Angular applications. A *Component* is a directive with a
template (`@Component` decorator), encapsulating a specific part
of the UI with its own logic, data, and visual representation. *
**Role:** To render a specific piece of UI, manage its state, and
handle user interactions within its encapsulated view. They drive
the application's visual structure. * **Syntax (Declaration):**
typescript import { Component } from '@angular/core';
```

```
@Component({ selector: 'app-my-component', // HTML tag to use
this component templateUrl: './my-component.html', styleUrls:
['./my-component.css'] }) export class MyComponent { // Component
logic and properties } * **Syntax (Usage):** ``
```

```
2. **Structural Directives:** * **Definition:** Structural
directives **change the DOM layout** by adding, removing, or
manipulating elements and their sub-trees. They directly affect
the structure of the DOM. These directives are always prefixed
with an asterisk (`*`) in the template, which is **syntactic
sugar** for a `` element that Angular expands internally. *
**Role:** To conditionally render elements or repeat elements
based on collection data. * **Common Examples:** * `*ngIf`:
Conditionally adds or removes an element from the DOM. *
`*ngFor`: Iterates over a collection and renders a template for
each item. * `*ngSwitch`: Renders one of several possible
templates based on a switch condition. * **Core Concepts for
Custom Structural Directives:** Require `TemplateRef` (to
represent the template) and `ViewContainerRef` (to create/destroy
views in the DOM).
```

```
3. **Attribute Directives:** * **Definition:** Attribute
directives **change the appearance or behavior** of an existing
DOM element, component, or another directive. Unlike structural
directives, they do not add or remove elements; instead, they
modify properties or attach event listeners to the host element.
They are applied as attributes on HTML elements. * **Role:** To
add styling (`[ngStyle]`, `[ngClass]`), modify behavior (e.g.,
`ngModel` for two-way binding), or attach specific event
listeners. * **Common Examples:** * `[ngClass]`: Dynamically adds
or removes CSS classes. * `[ngStyle]`: Dynamically applies inline
CSS styles. * `ngModel`: Facilitates two-way data binding on form
elements. * **Core Concepts for Custom Attribute Directives:**
Often utilize `ElementRef` (for direct access to the host
element) and `Renderer2` (a safer, platform-agnostic way to
manipulate DOM properties).
```

Syntax for Creating a Custom Attribute Directive:

```
typescript // highlight.directive.ts import { Directive,
ElementRef, Renderer2, HostListener, Input } from
'@angular/core';
```

```
@Directive({ selector: '[appHighlight]' // This directive can be
applied using the 'appHighlight' attribute }) export class
HighlightDirective { @Input('appHighlight') highlightColor:
string = 'yellow'; // Input property to set the highlight color
```

constructor(private el: ElementRef, private renderer: Renderer2) { }

```
// Listen for mouseenter event on the host element
@HostListener('mouseenter') onMouseEnter() {
this.highlight(this.highlightColor); }
```

```
// Listen for mouseleave event on the host element
@HostListener('mouseleave') onMouseLeave() {
this.highlight(null); // Remove highlight }
```

private highlight(color: string | null) { // Use Renderer2 for safe DOM manipulation this.renderer.setStyle(this.el.nativeElement, 'backgroundColor', color); } } ``` **Example Usage and Output:**

Example 1: Structural Directive (`*ngIf`)

HTML Template (`app.component.html`): ```html Welcome back, {{ username }}! Please log in to continue. ```

```
*TypeScript (`app.component.ts`):* typescript import { Component
} from '@angular/core';
```

```
@Component({ selector: 'app-root', templateUrl:
'./app.component.html' }) export class AppComponent {
userLoggedIn: boolean = true; username: string = 'Alice'; }
```

Output (if `userLoggedIn` is `true`): ```html Welcome back, Alice! ``` *Output (if `userLoggedIn` is `false`):* ```html Please log in to continue. ``` ***Explanation:*** The `*ngIf` directive ensures that only one of the `div` elements is present in the DOM at any given time, based on the `userLoggedIn` boolean value. It physically adds or removes the element, which is more performant than merely hiding it with CSS.

Example 2: Attribute Directive (`appHighlight`)

HTML Template (`app.component.html`): ```html Hover over this paragraph. Hover over this button. ```

```
*TypeScript (`app.component.ts`):* *(Assume `HighlightDirective`
from above is declared and imported in `AppModule`)* typescript
import { Component } from '@angular/core';
```

```
@Component({ selector: 'app-root', templateUrl:
'./app.component.html', styleUrls: ['./app.component.css'] })
export class AppComponent { // No specific logic needed here for
the directive's functionality }
```

```
*Output:* When the page loads, the paragraph will have a default
background (or transparent), and the button will have a yellow
background (default `highlightColor` when no value is bound).
When the mouse cursor hovers over the paragraph, its background
color changes to `lightblue`. When it hovers over the button, its
background changes to `yellow`. Upon mouse leave, the background
returns to its previous state (or transparent for the paragraph,
and yellow for the button if no initial style was set).
*Explanation:* The `appHighlight` directive modifies the
`backgroundColor` style of the host element (`` and ``) based on
mouse events, demonstrating how attribute directives alter an
element's appearance or behavior without changing its presence in
the DOM structure.
```