In Angular, lifecycle hooks are methods that allow you to tap into specific points in the lifecycle of a component or directive. These hooks provide opportunities to perform actions at key moments during the component's **creation, rendering, and destruction phases**. Understanding and utilizing these lifecycle hooks can help you manage component initialization, data loading, and clean-up operations effectively. Here are the main lifecycle hooks available in Angular components:

1. **ngOnChanges()**:
   * This hook is called when Angular sets or resets data-bound input properties of a component. It receives a **SimpleChanges** object containing the previous and current values of the input properties.
   * You would typically use the ngOnChanges() hook **when you need to perform certain actions or updates in response to changes in input properties of a component.** **This hook is particularly useful for handling data synchronization between parent and child components** or for triggering side effects based on input changes.
2. **ngOnInit()**:
   * This hook is called after Angular has initialized the component's data-bound properties and set their initial values. It is commonly used for initialization logic such as fetching data from a backend service.
   * The ngOnInit() lifecycle hook in Angular is called once, after the component has been initialized and its data-bound properties have been initialized. It is commonly used for component initialization logic, such as fetching data from a backend service, initializing component properties, or setting up subscriptions.
3. **ngDoCheck()**:
   * **This hook is called during every change detection cycle**, allowing you to implement custom change detection logic. It is typically used for optimizing performance or detecting changes that Angular's default change detection mechanism may miss. **This hook is called frequently, so it's important to use it judiciously to avoid performance issues.**
4. **ngAfterContentInit()**:
   * This hook is called after Angular has projected external content (e.g., content passed through component projections or **<ng-content>**). It is commonly used for initialization that depends on projected content.
5. **ngAfterContentChecked()**:
   * This hook is called after Angular checks the projected content for changes. It is called each time the projected content is checked for changes.
6. **ngAfterViewInit()**:
   * This hook is called after Angular has initialized the component's views and child views. It is commonly used for initialization logic that depends on the component's view or DOM elements.
7. **ngAfterViewChecked()**:
   * This hook is called after Angular checks the component's views and child views for changes. It is called each time the views are checked for changes.
8. **ngOnDestroy()**:
   * This hook is called just before Angular destroys the component and cleans up resources such as subscriptions, event listeners, or timers. It is commonly used to perform clean-up operations to prevent memory leaks.