**Angular Lifecycle Hooks**

In Angular, components have a lifecycle that encompasses their creation, updates, and destruction.1 Lifecycle hooks are methods that are called at specific points during this lifecycle, allowing you to perform actions at key moments.2

**Key Lifecycle Hooks:**

1. **ngOnChanges:**
   * Called before any other lifecycle hook.
   * Invoked whenever the component's input properties change.3
   * Receives a SimpleChanges object containing the current and previous values of the changed properties.
2. **ngOnInit:**
   * Called once after the first ngOnChanges.
   * A good place to initialize the component, fetch data, or subscribe to observables.4
3. **ngDoCheck:**
   * Called repeatedly during change detection to check for changes that Angular's default change detection might miss.5
   * Can be expensive, so use it judiciously.
4. **ngAfterContentInit:**
   * Called after Angular projects external content into the component's view.6
5. **ngAfterContentChecked:**
   * Called after every check of projected content.
6. **ngAfterViewInit:**
   * Called after Angular initializes the component's view.7
7. **ngAfterViewChecked:**
   * Called after every check of the component's view.
8. **ngOnDestroy:**
   * Called before Angular destroys the component.8
   * A good place to clean up subscriptions, detach event handlers, or release resources.9

**Example:**

TypeScript

import { Component, OnInit, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({

selector: 'app-my-component',

template: `

<p>Name: {{ name }}</p>

`

})

export class MyComponent implements OnInit, OnChanges {

@Input() name: string;

constructor() {

console.log('Constructor');

}

ngOnChanges(changes: SimpleChanges) {

console.log('ngOnChanges', changes);

}

ngOnInit() {

console.log('ngOnInit');

}

// ... other lifecycle hooks

}

**Key Considerations:**

* Use lifecycle hooks judiciously to avoid performance issues, especially ngDoCheck.
* Understand the order of lifecycle hooks and when they are called.
* Use lifecycle hooks to manage subscriptions, perform data fetching, and handle side effects.10

By effectively utilizing lifecycle hooks, you can manage the lifecycle of your components more efficiently and create more robust and performant Angular applications.11