Life Cycle Hooks

**1. OnChanges**

👉 **Definition:** Runs when the value of an @Input() property changes.  
Use it when your child component should do something after the parent sends new data.

export class ChildComponent implements OnChanges {

@Input() name: string = "";

ngOnChanges() {

console.log("Name changed to:", this.name);

}

}

➡️ If parent sends a new name = "Apsara", it prints:  
Name changed to: Apsara.

## 2. OnInit

👉 **Definition:** Runs **once**, when the component is first created.  
Use it for setup (like showing a welcome message or loading data).

export class AppComponent implements OnInit {

message: string = "";

ngOnInit() {

this.message = "Welcome to my app!";

console.log("Component initialized");

}

}

➡️ Prints only once when the component starts.

## 3. DoCheck

👉 **Definition:** Runs many times whenever Angular checks for changes.  
Use it if you want to **manually check something** again and again.

export class AppComponent implements DoCheck {

count: number = 0;

ngDoCheck() {

this.count++;

console.log("DoCheck run:", this.count);

}

}

➡️ It will keep printing: DoCheck run: 1, DoCheck run: 2, etc.

 **OnChanges** → Runs when input values change.

 **OnInit** → Runs once at the start.

 **DoCheck** → Runs again and again to check changes.

**Example 1 — OnInit changes UI**

export class AppComponent implements OnInit {

message: string = "Loading...";

ngOnInit() {

this.message = "Welcome Student!";

}

}

**Template (HTML):**

<p>{{ message }}</p>

**Example 2 — OnChanges updates UI**

export class ChildComponent implements OnChanges {

@Input() name: string = "";

message: string = "";

ngOnChanges() {

this.message = "Hello " + this.name;

}

}

**Template (HTML):**

<p>{{ message }}</p>

➡️ If parent sends name = "Anu" → it shows Hello Anu.

**Example 3 — DoCheck counts changes**

export class AppComponent implements DoCheck {

count: number = 0;

ngDoCheck() {

this.count++;

}

}

Template (HTML):

<p>Change detection ran {{ count }} times</p>

**1. AfterContentInit**

👉 **Definition:** Runs **once**, after Angular puts the **content from parent (ng-content)** inside the child.  
Use it if you want to **do something after projected content is ready**.

child.component.html

<p>Child works!</p>

<ng-content></ng-content> <!-- parent content will come here -->

child.component.ts

import { AfterContentInit, Component } from '@angular/core';

@Component({

selector: 'app-child',

templateUrl: './child.component.html'

})

export class ChildComponent implements AfterContentInit {

ngAfterContentInit() {

console.log("AfterContentInit: projected content is ready!");

}

}

parent.component.html

<app-child>

<p>This is projected from parent!</p>

</app-child>

Output on screen:

Child works!

This is projected from parent!

Console:  
AfterContentInit: projected content is ready!

**2. AfterViewChecked**

👉 **Definition:** Runs **many times**, after Angular updates the **view and child views**.  
Use it when you need to **react after Angular finishes rendering the DOM**.

import { AfterViewChecked, Component } from '@angular/core';

@Component({

selector: 'app-view-check',

template: `

<p>Counter: {{ count }}</p>

<button (click)="increase()">Increase</button>

`

})

export class ViewCheckComponent implements AfterViewChecked {

count = 0;

increase() {

this.count++;

}

ngAfterViewChecked() {

console.log("AfterViewChecked: view updated, count =", this.count);

}

}

Screen shows:

Counter: 0 [Increase button]

Click **Increase** → Counter updates, and in console:  
AfterViewChecked: view updated, count = 1

**What is a Pipe?**

👉 A **pipe** in Angular is like a **filter** that changes how data looks in the HTML.  
Example of **built-in pipe**:

<p>{{ today | date:'fullDate' }}</p>

If today = new Date(), this will show → **Monday, September 11, 2025**.

**🔹 Why Custom Pipes?**

Built-in pipes (like date, uppercase) are limited.  
If we need **our own transformation**, we create a **custom pipe**.

* Reusable
* Simple
* Keeps template clean

Simple Example — Custom Pipe

Step 1: Create a Pipe (Uppercase First Letter)

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'capitalizeFirst' })

export class CapitalizeFirstPipe implements PipeTransform {

transform(value: string): string {

if (!value) return '';

return value.charAt(0).toUpperCase() + value.slice(1);

}

}

 Pipes = **functions to format data in HTML**.

 Built-in: date, uppercase, currency, etc.

 **Custom Pipe** → When built-in is not enough, we create our own.

 Example: "hello" | capitalizeFirst → Hello

Step 2: Use in Template

<p>{{ 'hello world' | capitalizeFirst }}</p>

➡️ Output:

Hello world