<table>
<tr><td rowspan="4">Day 20</td><td colspan="2">Advanced Angular Concepts</td></tr>
<tr><td rowspan="3">Angular Directives and Pipes</td><td>Built-in directives (ngIf, ngFor, ngClass, ngStyle)</td></tr>
<tr><td>Custom directives and their use</td></tr>
<tr><td>Using Angular Pipes for data transformation

Input ======= OUTPUT
Transformation / formatting data directly within templates
 {{ value | Pipename}}
Built in pipes in Angular :
1. Date Pipe : Formats date according to locale or specified date format
 Todays date is : {{ todaysDate | date: ‘fulldate’}}
2 . Currency pipes: Formats number as currency
 Price: {{productPrice | currency:’USD’: ‘symbol’:’1.2-2’ }}
3. UpperCase/Lowercase:
 {{hello | uppercase}}
4. DecimalPipe/percentage pipe
 Value: {{ price | number: ‘1.2-2’}}
 Percentage :{{0.75 | percent }}
5. Slice Pipe: creates a new array or string containing a subset of the original
 First three letter : {{ Angular | slice: 0.3 }}

Pure pipes : (Default) Only re-execytes when their input value changes.
Impure pipes : re-executes in every change detection cycle, regardless of whether the input value has changed or not.</td></tr>
<tr><td>Practical exercises: Applying directives and pipes in a real-world example</td></tr>
<tr><td rowspan="2">Component Styling and Communication</td><td>In Angular, component styling determines how HTML elements look, styles can be applied in following ways :
Inline: - Defined directly in the component’s HTML using the style attributes or in the styles property of @Component.

External: Stored in separate .css/.scss file and linked via styleURls property.
Scoped styles: Styles in a component’s css file are encapsulated to that component by angular view encapsulation

Pro :
Inline: it is quick formal small changes, no file switching.
External : Clean and clear separation of style and logic, reusable.
Scoped : Prevents accidental style leakage, maintains consistency.

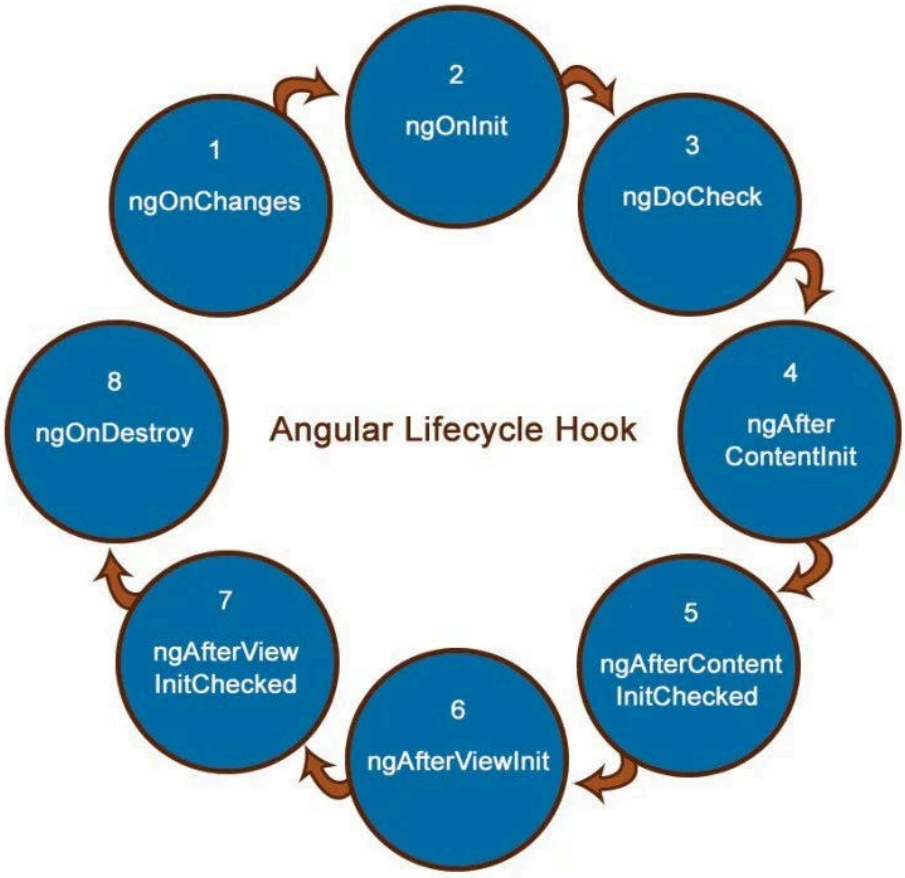
Cons:
Inline : Harder to maintain for large projects.
External : Slightly more setups, formal small tweaks.
Scope: Might require global styles for shared components.</td></tr>
<tr><td><u>Best practices for styling Angular components:</u>
1. Use external styles for maintainability</td></tr>
</table>

2. Reserve inline styles for temporary debugging a or minimal changes
3. Keep CSS selectors specific to prevent unintended overrides.
4. Use Angular encapsulation to avoid style leakage.

Component Lifecycle Methods

Angular components go through specific changes during their existence - from creation to destruction.

Below stages are handled by lifecycle hooks.



Understanding the lifecycle hooks (ngOnInit, ngOnChanges, etc.)

ngOnInit – Called once after the component's first render.

ngOnChanges – Called whenever input properties change.

ngOnDestroy – Called right before a component is removed.

Step 1: Create an application and add a component name lifecycle-demo

Step 2: Inside our typescript file implement oninit, ONchnages, ondestroy within the class .

Pros

<table>
<tr><td></td><td></td><td>

- Gives precise control over initialization and cleanup.
- Helps manage subscriptions, data fetching, and DOM updates.

Cons

- Misusing hooks can lead to performance issues.
- Complex logic in hooks can make code harder to maintain.

Best Practices

- Keep hook methods **short and focused**.
- Use **ngOnDestroy** to unsubscribe from Observables.
- Avoid heavy logic in ngOnChanges; instead, call separate methods.

</td></tr>
<tr><td></td><td></td><td>Practical exercises: Implementing lifecycle hooks for dynamic behavior</td></tr>
<tr><td>Day 21</td><td>Component Communication</td><td>Passing data between parent and child components
Component communication in angular is how they share data.

Parent -> child @input()
Child -> parent : Using @output with Even Emitter

Pro:
Keeps component modular and reusable
Ensures clear data flow.

Cons :
Overuse of input/output for deep component trees can get complex.
Requires additional handling for sibling communication

Step 1 : Create an application with two component ie
Parent and child
Step 2:
In Child component
We receive data from parent using @input
Send event to parent
Emit payload to parent via sendMessage()

INparent component - HTML
We will pass value to child using property binding
Event ding: listen to child event</td></tr>
</table>

		IN parent component (.TS) Hold last message from child Update local state -> view updates automatically
		Using Input and Output decorators
		Practical exercises: Building a communication system between components

Method	**How to Use**	**Pros**	**Cons**
Inline (HTML)	`<p style="...">`	Quick	Hard to maintain
Inline (Component)	`styles: [...]`	Better organization	Still not reusable
External CSS	`styleUrls: [...]`	Clean & reusable	Extra file
Scoped	Default in Angular	No leakage	Needs global styles sometimes