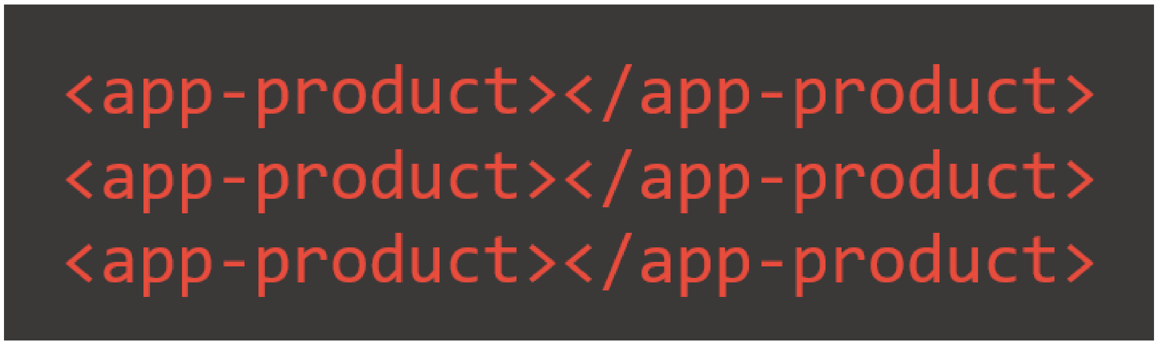
Angular Lifecycle Hooks

This cheat-sheet contains all the theoretical concept related to Angular life-cycle hook and how and when to use them.

“The life cycle hooks are the methods that angular invokes on the directives and components as it creates, changes, and destroys them.”

When a new component is created in angular, and when it finds one of a selector, it will instantiate the new version of that component and add it into the DOM.



```
<app-product></app-product>
<app-product></app-product>
<app-product></app-product>
```

So, once a new component is created, angular goes through different phases in this creation process. And it also gives us a chance to actually hook into these phases and execute some code.

We can hook into these phases by implementing some methods which angular will call if they are present in the component class.

What is Angular Component lifecycle hooks

When the angular application starts, it first creates and renders the **root** component. Then, it creates and renders its Children’s & their children. It forms a tree of components.

Once Angular loads the components, it starts the process of rendering the view. To do that, it needs to check the **input** properties, evaluate the data bindings & expressions, render the projected content etc. Angular also removes the component from the DOM, when it is no longer needs it.

Angular lets us know when these events happen using lifecycle hooks

For example,

- `ngOnInit` when Angular initializes the component for the first time.
- When a component’s input property change, Angular invokes `ngOnChanges`

- If the component is destroyed, Angular invokes `ngOnDestroy`

Let’s learn about each of these life cycle hooks one by one.

What is change detection cycle?

Before diving into the lifecycle hooks, we need to understand the change detection cycle.

Change detection is the mechanism by which angular keeps the template in sync with the component

Consider the following code.

```
<div>Hello {{name}}</div>
```

Angular updates the DOM, whenever the value of the `name` changes. And it does it instantly.

How does angular know when the value of this `name` property changes? It does so by running a **change detection cycle** on every **event** that may result in a change. It runs it on every **input change**, **DOM events**, **timer events** like `setTimeout()` and `setInterval()` , **http** requests etc.

During the change detection cycle angular checks each and every bound property in the template, with that of the component class. If it detects any changes, it updates the DOM.

Angular raises the life cycle hooks during the important stages of the change detection mechanism.

Constructor

Life Cycle of a component begins, when Angular creates the component class. First method that gets invoked is class **Constructor**.

Constructor is neither a life cycle hook nor it is specific to Angular. It is a JavaScript feature. It is a method which gets invoked, when a class is created.

Angular makes use of a constructor to inject dependencies.
At this point, none of the components **input** properties are available to use. Neither its child components are constructed. Projected contents are also not available.

Hence there is not much you can do in this method. And also it is recommend not to use it

Once Angular instantiates the class, it kick-start the first change detection cycle of the component.

ngOnChanges

The first phase or the first hook of a component creation is `ngOnChanges`

It is executed right at the start, when a new component is created, and it also gets executed whenever one of the bound input property changes.

The Angular invokes `ngOnChanges` life cycle hook whenever any data-bound input property of the component or directive changes.

Initializing the Input properties is the first task that angular carries during the change detection cycle.

And if it detects any change in input property, then it raises the `ngOnChanges` hook. It does so during every change detection cycle.

This hook is not raised if change detection does not detect any changes.

Input properties are those properties, which we define using the `@Input` decorator. It is one of the ways by which a parent component communicates with the child component.

In the following example, the child component declares the property `message` as the input property.

```
@Input() message:string
```

The parent can send the data to the child using the property binding as shown below.

```
<app-child [message]="message">
</app-child>
```

The change detector checks if such input properties of a component are changed by the parent component. If it is then it raises the `ngOnChanges` hook.

ngOnInit

The Angular raises the ngOnInit hook, after it creates the component and updates its input properties. This hook is raised after the `ngOnChanges` hook.

This hook is fired **only once** and immediately after its creation (during the first change detection).

This is a perfect place where you want to add any initialization logic for your component.

Here you have access to every **input** property of the component. You can use them in **http get** requests to get the data from the back-end server or run some initialization logic etc.

But remember that, none of child components or projected content are available at this point. Hence any properties we decorate with `@ViewChild`, `@ViewChildren`, `@ContentChild` & `@ContentChildren` will not be available to use.

ngDoCheck

The Angular invokes the `ngDoCheck` hook event during every change detection cycle. This hook is invoked even if there is no change in any of the properties.

For example, `ngDoCheck` will run if you clicked a button on the webpage which does not change anything. But still, it's an event.

And on events, angular has to check if something changed. And for that `ngDoCheck` method has to be called.

Angular invokes it after the `ngOnChanges` & `ngOnInit` hooks.

You can use this hook to Implement a custom change detection, whenever Angular fails to detect the changes made to Input properties.

`ngDoCheck` is also a great method to use, when you want to execute some code on every change detection cycle.

ngAfterContentInit

`ngAfterContentInit` Life cycle hook is called after the Component's projected content has been fully initialized.

Angular also updates the properties decorated with the `ContentChild` and `ContentChildren` before raising this hook. This hook is also raised, even if there is no content to project.

The content here refers to the external content injected from the parent component via **Content Projection**.

The Angular Components can include the `ng-content` element, which acts as a placeholder for the content from the parent.

```
<h2>Child Component</h2>
<ng-content></ng-content>  <!-- placehodler for content from parent -->
```

Parent injects the content between the opening & closing selector element. Angular passes this content to the child component

```
<h1>Parent Component</h1>
<app-child> <p>This content is injected from parent</p></app-child>
```

During the change detection cycle, Angular checks if the injected content has changed and updates the DOM.

This is a component only hook.

ngAfterContentChecked

`ngAfterContentChecked` Life cycle hook is called during every change detection cycle after Angular finishes checking of component's projected content.

Angular also updates the properties decorated with the `@ContentChild` and `@ContentChildren` before raising this hook. Angular calls this hook even if there is no projected content in the component.

This hook is very similar to the `ngAfterContentInit` hook.

Both are called after the external content is initialized, checked & updated.

Only difference is that `ngAfterContentChecked` is raised after every change detection cycle. While `ngAfterContentInit` during the first change detection cycle.

This is a component only hook.

ngAfterViewInit

`ngAfterViewInit` hook is called after the Component's View & all its child views are fully initialized. Angular also updates the properties decorated with the `@ViewChild` & `@ViewChildren` properties before raising this hook.

The View here refers to the view template of the current component and all its child components & directives.

This hook is called during the first change detection cycle, where angular initializes the view for the first time

At this point all the lifecycle hook methods & change detection of all child components & directives are processed & Component is completely ready.

This is a component only hook.

ngAfterViewChecked

The Angular fires this hook after it checks & updates the component's views and child views. This event is fired after the `ngAfterViewInit` and after that during every change detection cycle.

This hook is very similar to the `ngAfterViewInit` hook. Both are called after all the child components & directives are initialized and updated.

Only difference is that `ngAfterViewChecked` is raised during every change detection cycle. While `ngAfterViewInit` is raised during the first change detection cycle.

This is a component only hook.

ngOnDestroy

And finally, if you destroy a component, for example, when you placed `ngIf` on a component, and this `ngIf` then set to `false`, at that time, `ngIf` will remove that component from the DOM, at that time, `ngOnDestroy` is called.

This method is the great place to do some cleanup work, because this is called right before the objects will be destroyed itself by angular.

This is the correct place where you would like to Unsubscribe Observables and detach event handlers to avoid memory leaks.