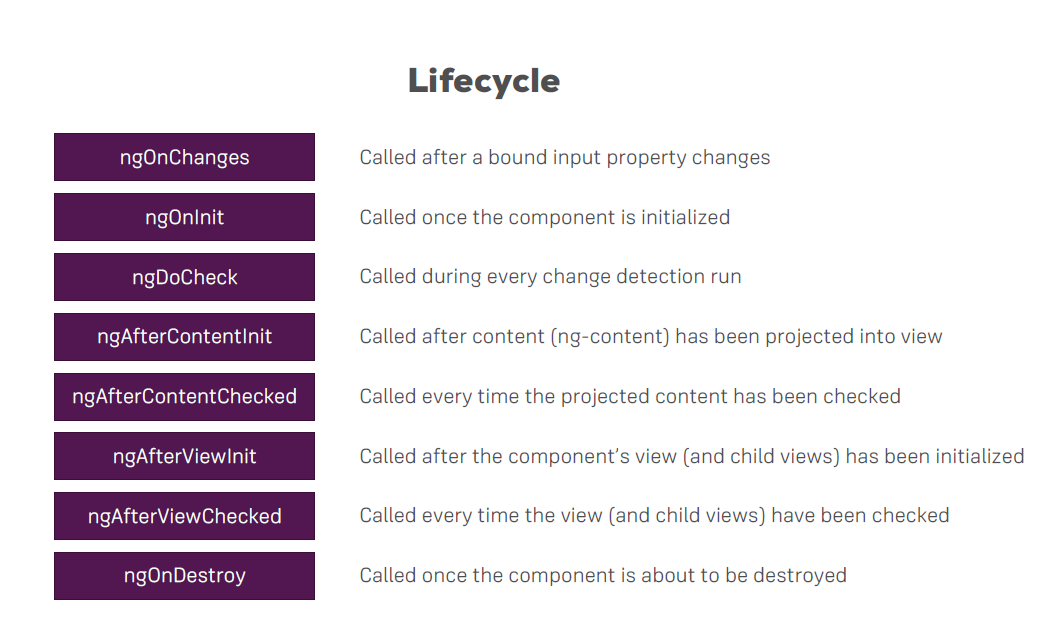
**77. Understanding the Component Lifecycle**

* -: Woo! We're nearing the end of the section, but there's one thing you might have recognized before, where you weren't sure what it does.
* What's up with this ngOnInit method we have in new components created through the CLI.
* What's it doing? NgOnInit is a lifecycle hook and Angular supports a couple of lifecycle hooks.
* Let's take a closer look.
* If a new component is created in Angular, and of course Angular is responsible for creating these components, when it finds one of our selectors for example, it will instantiate a new version of that component and add it into the DOM.
* So once a new component is instantiated, Angular goes through a couple of different phases in this creation process.
* And it will actually give us a chance to hook into these phases and execute some code.
* We can hook into these phases by implementing some methods Angular will call, if they are present.



1. **ngOnChanges**

* The first phase, the first hook we can hook into is ngOnChanges.
* And this may actually be executed multiple times.
* It's executed right at the start, when a new component is created.
* But thereafter, it's also always called, whenever one of our bound input properties changes.
* And with that, I mean, properties decorated with "@input", so whenever these properties receive new values.

1. **ngOnInit**

* Now the second hook is ngOnInit.
* This method gets executed once the component has been initialized.
* This does not mean that we can see it.
* It has not been added to the DOM, yet, so to say, it has not been displayed yet, but Angular finished the basic initialization.
* Our properties can now be accessed and initialized, for example.
* So the object was created, you could say.
* And if you are interested, ngOnInit will run after the constructor.

1. **ngDoCheck**

* Then, we have ngDoCheck.
* That will also run multiple times.
* Actually, this method will be executed a lot, because this will run whenever change detection runs.
* Now change detection simply is the system by which Angular determines whether something changed on the template of a component, or inside of a component I should say.
* So whether it needs to change something in the template.
* So whether some property value change from one to two, let's say, and that property's output in the template.
* Well, of course Angular needs to re-render that part of the template.
* And ngDoCheck is a hook executed on every check Angular makes.
* Now important on every check.
* So not just if something changed.
* A lot of times ngDoCheck will run because you clicked some button, which doesn't change anything, but still it's an event, and on events, Angular has to check if something changed, because how else would it know? You don't tell it, right? So it has to check on certain triggering events, like you clicked somewhere, or a timer fired, or an observable was resolved.
* And on these occasions, it will check your code and ngDoCheck will be executed.
* Now, whilst this might sound very inefficient, Angular does this in a very efficient way.
* So change detection, Angular works pretty great and doesn't cost a lot of performance.
* NgDoCheck is a great method to use, if you want to do something on every change detection cycle, like maybe, manually inform Angular about some change it would not be able to detect otherwise, though that is a very advanced use case.

1. **AfterContentInit.**

* Well then, we reach AfterContentInit.
* This is called whenever the content which is projected via ng-content has been initialized.
* So not the view of the component itself, but instead you could say, the view of the parent component, especially a part which will get added to our component through ng-content.
* An ngAfterContentInit check is executed whenever change detection checked this content we're projecting into our component.

1. **NgAfterViewInit**

* NgAfterViewInit is then reached, once the view of our own component has been finished initializing, so once our view has been rendered, you could say.

1. **ngAfterViewChecked**

* And the same, AfterViewChecked, well that is called whenever our view has been checked.
* So once we are sure that, well either all changes which had to be done were displayed in the view, or no changes were detected by Angular.

**7) ngOnDestroy**

* And finally, if you destroy a component, for example if you placed ngIf on it and this gets then set to false and therefore it removes it from the DOM, ngOnDestroy is called.
* And here's a great place to do some clean up work because this is called right before the object itself will be destroyed by Angular.
* These are the hooks.
* Nice to see them in theory.
* Let's see them in practice in the next lecture.