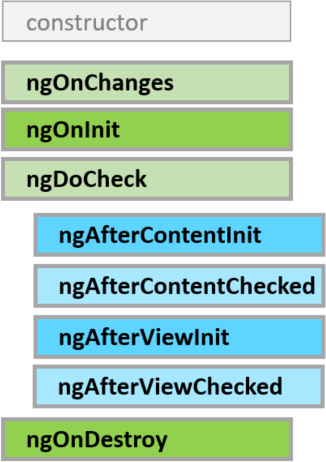
# Angular Component Life Cycle

*Angular calls lifecycle hook methods on directives and components as it creates, changes, and destroys them.*



A component has a lifecycle managed by Angular.

Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.

Angular offers **lifecycle hooks** that provide visibility into these key life moments and the ability to act when they occur.

8 Lifecycle Hooks

* ngOnChanges()
* Used in pretty much any component that has an input.
* Called whenever an input value changes
* Is called the first time before ngOnInit
* ngOnInit()
* Used to initialize data in a component.
* Called after input values are set when a component is initialized.
* Added to every component by default by the Angular CLI.
* Called only once
* ngDoCheck()
* Called during all change detection runs
* A run through the view by Angular to update/detect changes
* ngAfterContentInit()
* Called only once after first ngDoCheck()
* Called after the first run through of initializing content
* ngAfterContentChecked()
* Called after every ngDoCheck()
* Waits till after ngAfterContentInit() on first run through
* ngAfterViewInit()
* Called after Angular initializes component and child component content.
* Called only once after view is initialized
* ngAfterViewChecked()
* Called after all the content is initialized and checked. (Component and child components).
* First call is after ngAfterViewInit()
* Called after every ngAfterContentChecked() call is completed
* ngOnDestroy()
* Used to clean up any necessary code when a component is removed from the DOM.
* Fairly often used to unsubscribe from things like services.
* Called only once just before component is removed from the DOM.

In my experience as an Angular developer, I primarily use only four of these hooks. Mostly because I don’t want to do something to a component **after**the content has already been checked.

**ngOnChanges()**

**ngOnInit()**

**ngAfterViewInit()**

**ngOnDestory()**

The first two I use fairly frequently. They are very useful when dealing with input values or setting your component state based on outside data. The other two are very use case specific. If for some reason you need to do something after your component content has been set, use ngAfterViewInit. As I stated above, clean up your component with ngOnDestory(). Next I am going to set up an example module to demonstrate the different hooks.