# ANGULAR

# CICLO DE VIDA:

El ciclo de vida de un componente se compone de una serie de eventos que ocurren desde la creación hasta la destrucción del componente

# ANGULAR

# CICLOS DE VIDA:

1. ngOnChanges: Se dispara cuando los datos de entrada (@Input) del componente cambian.
2. ngOnInit: Ocurre después de que Angular ha inicializado todas las propiedades del componente
3. ngDoCheck: Se ejecuta durante cada detección de cambios y permite realizar acciones de verificación personalizadas.
4. ngAfterContentInit: Ocurre después de que Angular haya proyectado el contenido en el componente.
5. ngAfterContentChecked: Se ejecuta después de cada verificación del contenido proyectado.
6. ngAfterViewInit: Ocurre después de que Angular haya inicializado las vistas del componente.
7. ngAfterViewChecked: Se ejecuta después de cada verificación de las vistas del componente.
8. ngOnDestroy: Se dispara justo antes de que Angular destruya el componente

# ANGULAR

# CICLOS DE VIDA:

## ngOnChanges

Se dispara cuando los datos de entrada (@Input) del componente cambian.

```typescript
import { Component, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-mi-componente',
  template: '<p>¡Hola, {{ nombre }}!</p>',
})
export class MiComponente implements OnChanges {

  // Propiedad de entrada
  @Input() nombreInput: string;

  // Propiedad del componente
  nombre: string;

  // Método ngOnChanges se llama cuando hay cambios en las propiedades de entrada
  ngOnChanges(changes: SimpleChanges): void {
    // Verifica si la propiedad de entrada 'nombreInput' ha cambiado
    if (changes.nombreInput) {
      this.nombre = changes.nombreInput.currentValue;
      console.log(`Se ha cambiado el valor de nombreInput a: ${this.nombre}`);
    }
  }
}
```

En este ejemplo, el componente tiene una propiedad de entrada llamada **nombreInput**. Cuando esta propiedad cambia desde el componente padre, el método **ngOnChanges** se activa.

Dentro de **ngOnChanges**, puedes acceder a los cambios a través del parámetro changes y realizar acciones en respuesta a esos cambios.

# ANGULAR

# CICLOS DE VIDA:

## ngOnInit

Ocurre después de que Angular ha inicializado todas las propiedades del componente.
Es un buen lugar para realizar inicializaciones adicionales.

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

@Component({
  selector: 'app-mi-componente',
  template: '<p>¡Hola, mundo!</p>',
})
export class MiComponente implements OnInit {

  // Propiedad de ejemplo
  nombre: string;

  // Constructor del componente
  constructor() {
    // Puedes inicializar propiedades aquí, pero es una buena práctica hacerlo en ngOnInit
  }

  // Método ngOnInit se llama después de que Angular ha inicializado todas las propiedades del componente
  ngOnInit(): void {
    this.nombre = 'Usuario';
    console.log(`Hola, ${this.nombre}! El componente se ha inicializado.`);
  }
}
```

En este ejemplo, **ngOnInit** se implementa en la clase del componente y se utiliza para realizar acciones después de que Angular haya inicializado todas las propiedades del componente. En este caso, se establece el valor de la propiedad nombre y se imprime un mensaje en la consola.

Recuerda que **ngOnInit** es un buen lugar para realizar inicializaciones adicionales, como la obtención de datos de un servicio o la configuración de suscripciones a **observables**.

# ANGULAR

# CICLOS DE VIDA:

## ngDoCheck

Se ejecuta durante cada detección de cambios y permite realizar acciones de verificación personalizadas.

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

@Component({
  selector: 'app-mi-componente',
  template: '<p>¡Hola, {{ nombre }}!</p>',
})
export class MiComponente implements DoCheck {

  nombre: string = 'Usuario';

  // Método ngDoCheck se llama durante cada detección de cambios
  ngDoCheck(): void {
    console.log('Se está ejecutando ngDoCheck');
    // Puedes realizar acciones de verificación personalizadas aquí
  }
}
```

En este ejemplo, el método **ngDoCheck** se llama durante cada detección de cambios en Angular. Puedes realizar acciones de verificación personalizadas dentro de este método. Ten en cuenta que **ngDoCheck** se ejecuta con frecuencia, por lo que es importante no realizar operaciones costosas en términos de rendimiento dentro de este método.

# ANGULAR

# CICLOS DE VIDA:

## ngAfterContentInit

Ocurre después de que Angular haya proyectado el contenido en el componente.

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

@Component({
  selector: 'app-mi-componente',
  template: '<ng-content></ng-content>',
})
export class MiComponente implements AfterContentInit {

  // ContentChild para acceder a un elemento dentro del contenido proyectado
  @ContentChild('nombreElemento') nombreElemento: ElementRef;

  // Método ngAfterContentInit se llama después de que Angular haya proyectado el contenido
  ngAfterContentInit(): void {
    // Realizar acciones después de que el contenido haya sido inicializado
    if (this.nombreElemento) {
      console.log(`Se ha encontrado un elemento con el nombre 'nombreElemento'.`);
    }
  }
}
```

En este ejemplo, **ngAfterContentInit** se utiliza para realizar acciones después de que Angular haya proyectado el contenido en el componente. También se usa **ContentChild** para acceder a un elemento específico dentro del contenido proyectado. Si el elemento con el nombre 'nombreElemento' está presente, se imprime un mensaje en la consola.

# ANGULAR CICLOS DE VIDA:

## ngAfterContentChecked

Se ejecuta después de cada verificación del contenido proyectado.

```
import { Component, AfterContentChecked, ContentChild, ElementRef } from '@angular/core';

@Component({
  selector: 'app-mi-componente',
  template: '<ng-content></ng-content>',
})
export class MiComponente implements AfterContentChecked {

  // ContentChild para acceder a un elemento dentro del contenido proyectado
  @ContentChild('nombreElemento') nombreElemento: ElementRef;

  // Método ngAfterContentChecked se llama después de cada verificación del contenido proyectado
  ngAfterContentChecked(): void {
    // Realizar acciones después de cada verificación del contenido
    if (this.nombreElemento) {
      console.log(`Se ha verificado el contenido y se ha encontrado un elemento con el nombre 'nombreElemento'.`);
    }
  }
}
```

En este ejemplo, **ngAfterContentChecked** se utiliza para realizar acciones después de cada verificación del contenido proyectado en el componente. También se usa **ContentChild** para acceder a un elemento específico dentro del contenido proyectado. Si el elemento con el nombre 'nombreElemento' está presente, se imprime un mensaje en la consola.

Recuerda que **ngAfterContentChecked** se ejecutará cada vez que Angular verifique el contenido proyectado, por lo que debes ser consciente de la eficiencia de las operaciones que realices en este método.

# ANGULAR

# CICLOS DE VIDA:

## ngAfterViewInit

Ocurre después de que Angular haya inicializado las vistas del componente.

```
import { Component, AfterViewInit, ViewChild, ElementRef } from '@angular/core';

@Component({
  selector: 'app-mi-componente',
  template: '<p #miParrafo>¡Hola, mundo!</p>',
})
export class MiComponente implements AfterViewInit {

  // ViewChild para acceder a un elemento en la vista del componente
  @ViewChild('miParrafo') miParrafo: ElementRef;

  // Método ngAfterViewInit se llama después de que Angular haya inicializado las vistas del componente
  ngAfterViewInit(): void {
    // Realizar acciones después de que la vista haya sido inicializada
    if (this.miParrafo) {
      console.log(`Se ha inicializado la vista y se ha encontrado un párrafo: ${this.miParrafo.nativeElement.textContent}`);
    }
  }
}
```

En este ejemplo, **ngAfterViewInit** se utiliza para realizar acciones después de que Angular haya inicializado las vistas del componente. Se usa **ViewChild** para acceder a un elemento específico en la vista del componente, en este caso, un párrafo con la referencia miParrafo. Si el párrafo está presente, se imprime su contenido en la consola.

# ANGULAR CICLOS DE VIDA:

## ngAfterViewChecked

Se ejecuta después de cada verificación de las vistas del componente.

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

@Component({
  selector: 'app-mi-componente',
  template: '<p #miParrafo>{{ mensaje }}</p>',
})
export class MiComponente implements AfterViewChecked {

  mensaje: string = '¡Hola, mundo!';

  // ViewChild para acceder a un elemento en la vista del componente
  @ViewChild('miParrafo') miParrafo: ElementRef;

  // Método ngAfterViewChecked se llama después de cada verificación de las vistas del componente
  ngAfterViewChecked(): void {
    // Realizar acciones después de cada verificación de las vistas
    if (this.miParrafo) {
      console.log(`Se ha verificado la vista y el contenido del párrafo es: ${this.miParrafo.nativeElement.textContent}`);
    }
  }
}
```

En este ejemplo, **ngAfterViewChecked** se utiliza para realizar acciones después de cada verificación de las vistas del componente. Se utiliza **ViewChild** para acceder a un elemento específico en la vista del componente (en este caso, un párrafo con la referencia miParrafo). Se imprime el contenido del párrafo en la consola después de cada verificación.

Este método se ejecuta después de que Angular ha realizado la verificación de las vistas del componente, por lo que es un buen lugar para realizar acciones que dependan de la vista actualizada.

# ANGULAR

# CICLOS DE VIDA:

## ngOnDestroy

Se dispara justo antes de que Angular destruya el componente.
Es un buen lugar para realizar limpieza, como la eliminación de suscripciones a observables.

```
import { Component, OnDestroy } from '@angular/core';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-mi-componente',
  template: '<p>¡Adiós, mundo!</p>',
})
export class MiComponente implements OnDestroy {

  // Ejemplo de suscripción a un observable
  private subscription: Subscription;

  constructor() {
    // Simulación de suscripción a un observable
    this.subscription = new Subscription();
  }

  // Método ngOnDestroy se llama justo antes de que Angular destruya el componente
  ngOnDestroy(): void {
    // Realizar limpieza, como desuscribirse de observables o liberar recursos
    if (this.subscription) {
      this.subscription.unsubscribe();
      console.log('Se ha desuscrito de la suscripción en ngOnDestroy.');
    }
  }
}
```

En este ejemplo, **ngOnDestroy** se utiliza para realizar acciones justo antes de que Angular destruya el componente. Se simula una suscripción a un observable en el constructor del componente, y en **ngOnDestroy**, se verifica y se desuscribe de la suscripción para evitar posibles pérdidas de memoria.

Este método es útil para realizar limpieza, como desuscribirse de observables, cancelar peticiones HTTP o liberar otros recursos cuando el componente está a punto de ser destruido.