1. ¿Qué son los hooks del ciclo de vida en Angular? Explique algunos hooks del ciclo de vida

Los hooks del ciclo de vida en Angular son métodos que se ejecutan cuando el componente alcanza un estado determinado.
Los métodos que pertenecen a esta clasificación son los siguientes:

-ngOnInit(): Es el **primer método** que se realiza en un componente, justo después del constructor. Es el método que se ejecuta al crear el componente, y normalmente se utilizará para hacer llamadas a la API, o establecer el estado inicial del componente.

-ngOnChanges(): Este método se ejecuta cuando un elemento padre le pasa algo a un elemento hijo. Es decir, el hijo tiene un **input** y va a recibir información de una variable del padre. Cuando esto ocurre, se lanza el ngOnChanges(), permitiendo interceptar ese cambio y ejecutar acciones con él.

-**ngAfterViewInit()**: Este método se lanza **después** de que se haya cargado el **DOM**. Se suele utilizar para realizar acciones que tengan que ver con elementos del DOM, como por ejemplo, seleccionar un elemento con el viewChild.

-**ngAfterViewChecked()**: Este método se ejecuta cuando hay algún **cambio en el DOM**, después de su inicialización.

-**ngAfterContentInit()**: Este método se ejecuta en los componentes que tienen un **ng-content**. Se ejecuta cuando el **componente hijo** se ha **inicializado**. Sirve para que el padre pueda interactuar con el hijo

-**ngAfterContentChecked()**: Este método se lanza en los componentes que tienen un **ng-content** cuando hay un **cambio** en el **componente hijo una vez se ha inicializado**.

-**ngDoCkeck()**: Se lanza cuando **Angular** realiza una **detección de cambios**. Sirve para establecer tu propia lógica con respecto a esta detección de cambios.

-**ngOnDestroy()**: El **último método** en lanzarse antes de que se destruya el componente. Se suele utilizar para cerrar suscripciones abiertas por el componente.

2. -¿Qué es la compilación AOT (Ahead-Of-Time)? ¿Cuáles son sus ventajas? ¿Conoces otra?

La compilación AOT es la que se realiza para **convertir** los **elementos de Angular** en archivos que un servidor web puede entender y consumir **al "construir" la aplicación** (hacer **build**), ya que no se puede implementar, por ejemplo, typescript o scss directamente en un servidor, se tienen que convertir **a javascript y css** respectivamente.

Las **principales ventajas** son: que permite el **desarrollo** en este tipo de super sets de los lenguajes mencionados y aprovechar todas sus ventajas **sin** que se tenga que **compilar**

cada archivo a mano, que **influye mucho en el rendimiento**, ya que todo el código ya está compilado antes de ejecutarse en el navegador y que **optimiza el código** lo máximo posible.

Otra compilación sería la **JIT, Just In Time**, que es la que **compila typescript a javascript en el tiempo de ejecución, en el navegador**, lo que radica en un **rendimiento más lento** y un **código menos optimizado**.

3. ¿En qué se diferencian los Observables de las Promesas?

Los **Observables** son elementos de la librería **RXJS**, que crean un **canal** al cual te puedes **suscribir** desde otro punto de la aplicación, para que cuando se ha realizado un cambio en ese observable, **todos los elementos** que están **suscritos** al elemento **reciban la actualización**. Además, los observables **pueden tener más de un valor** durante su vida útil.

Por otra parte, las **Promesas** son un elemento **nativo de javascript**, que sirve para gestionar la **asincronía**. Es decir, si un elemento va a tardar en realizar una función y quieres que la ejecución se espere a que haya acabado, **se envuelve** ese elemento en una promesa y , ya sea con un .then o con un async await, **una vez** se haya **resuelto** la promesa recibirías la información y el código se seguiría ejecutando. Esto sólamente pasará una vez. Es decir, **una promesa no puede tener más de un valor**.

4. ¿Qué es ViewEncapsulation y cuántas formas hay de hacerlo en Angular?

El **ViewEncapsulation** es la forma que tiene Angular de **encapsular el código css** para que no afecte entre componentes. Es decir, si yo en un componente tengo un elemento que tiene la clase title, y en otro componente también tengo un elemento con la clase tite, el css del primer elemento no interferirá con el css del segundo elemento y viceversa.

Hay **3 formas** de hacerlo en Angular:

- **Emulated**: La opción **por defecto**. Angular utiliza ciertos **atributos** codificados para que solamente le afecte su propio css al componente.

- **ShadowDom**: Es el **navegador** el que gestiona el encapsular el css del componente mediante el ShadowDom.

- **None**: **No se encapsula el código css** y todo css afecta a todos los componentes.