1.Explain Life cycle in Class Component and functional component with Hooks

.

**Class Component Lifecycle:**

Class components have several lifecycle methods that you can override to execute code at specific points in a component's lifecycle. The lifecycle of a class component can be divided into three main phases: Mounting, Updating, and Unmounting.

1. **Mounting:**
   * constructor(): The constructor is called before a component is mounted. It is used to initialize state and bind event handlers.
   * render(): The render method is responsible for rendering the component.
   * componentDidMount(): This method is called after the component has been rendered to the DOM. It is often used for network requests or subscriptions.
2. **Updating:**
   * shouldComponentUpdate(nextProps, nextState): This method is called before rendering when new props or state are received. It allows you to control whether the component should update or not.
   * render(): Re-render the component.
   * componentDidUpdate(prevProps, prevState): This method is called after the component has been updated in the DOM.
3. **Unmounting:**
   * componentWillUnmount(): This method is called just before the component is unmounted or destroyed. It is used for cleanup operations, such as canceling network requests or clearing subscriptions.

**Functional Component with Hooks:**

Functional components can now use Hooks to manage state and lifecycle events. The primary hooks for lifecycle events are useEffect and useState.

1. **Mounting and Updating:**
   * useEffect(() => {}, []): This hook combines the functionality of componentDidMount, componentDidUpdate, and componentWillUnmount. The function passed to useEffect runs after every render and can perform side effects. The empty array ([]) as the second argument ensures that the effect runs only once (on mount) or when specific dependencies change.
2. **Unmounting:**
   * The cleanup functionality in functional components is handled inside the useEffect function. If you return a function from the effect, React will run it when the component is unmounted.

Here's an example of a functional component with hooks:

jsx

import React, { useEffect, useState } from 'react';

function FunctionalComponent() {

const [data, setData] = useState([]);

useEffect(() => {

// componentDidMount and componentDidUpdate

// Fetch data or perform other side effects

return () => {

// componentWillUnmount

// Cleanup operations (unsubscribe, clear intervals, etc.)

};

}, [/\* dependencies \*/]);

return (

// JSX rendering

);

}

In summary, while class components have distinct lifecycle methods, functional components with hooks can achieve similar behavior using the useEffect hook. The key difference is that the logic for mounting, updating, and unmounting is encapsulated within the useEffect hook.