**MODULE: 10 List and Hooks**

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

**Ans:**

**Class Component Lifecycle:**

1. **Mounting Phase:**

* constructor(): Called when the component is created. It's used for setting initial state and binding event handlers.
* componentWillMount(): Deprecated and rarely used. Called just before the component is inserted into the DOM.
* render(): Required method to render the component's UI.
* componentDidMount(): Called after the component has been inserted into the DOM. It's often used for making AJAX requests or setting up timers.

1. **Updating Phase:**

* shouldComponentUpdate(nextProps, nextState): Allows you to control whether the component should re-render based on changes in props or state. Returning false prevents re-rendering.
* componentWillUpdate(nextProps, nextState): Deprecated and rarely used. Called just before the component is about to re-render.
* render(): Required method to render the updated UI.
* componentDidUpdate(prevProps, prevState): Called after the component has re-rendered due to state or prop changes. It's often used for performing side effects.

1. **Unmounting Phase:**

* componentWillUnmount(): Called just before the component is removed from the DOM. It's used for cleanup tasks like clearing timers or event listeners.
* Functional Component with Hooks:
* React introduced Hooks to allow functional components to manage state and side effects similar to class components. While they don't directly mirror the class component lifecycle, Hooks enable you to achieve similar functionality:
* useState: Allows you to manage state within a functional component.
* useEffect: Enables you to perform side effects (e.g., data fetching, DOM manipulation) in response to component updates.
* Here's how Hooks correspond to the lifecycle phases:
* Mounting Phase:
* Functional components with Hooks don't have a constructor or componentWillMount. Instead, you use the useState Hook to initialize state and perform any setup inside the functional component's body.
* You can perform side effects after the component has rendered using useEffect. By default, useEffect runs after every render, similar to componentDidMount.
* Updating Phase:
* You can control when useEffect runs by passing dependencies as the second argument. This mimics the behavior of shouldComponentUpdate. If you provide specific dependencies, useEffect will only run when those dependencies change.
* Unmounting Phase:
* Cleanup logic, similar to componentWillUnmount, can be performed inside the cleanup function returned by useEffect.
* In summary, functional components with Hooks provide a more concise and readable way to manage component lifecycle and state compared to class components. However, the core concepts of initialization, updates, and cleanup remain similar between the two approaches.