- Explain Life cycle in Class Component and functional component with Hooks

Life Cycle in Class Components:

1. Mounting Phase:

- ‘constructor()’: This is the first method called when an instance of a class component is created. It is used for initializing state and binding event handlers.

- ‘componentWillMount()’: This method is deprecated and not recommended for use.

- ‘render()’: This method is responsible for returning the JSX (the component's UI) that will be rendered to the DOM.

- ‘componentDidMount()’: This method is called immediately after the component is mounted to the DOM. It is commonly used for making API calls or initializing third-party libraries.

2. Updating Phase:

- ‘componentWillReceiveProps(nextProps)’: This method is deprecated and not recommended for use.

- ‘shouldComponentUpdate(nextProps, nextState)’: This method is used to optimize rendering performance by determining if the component should update or not. It compares the current props and state with the next props and state and returns a boolean value.

- ‘componentWillUpdate(nextProps, nextState)’: This method is deprecated and not recommended for use.

- ‘render()’: This method is called again to re-render the component when an update is triggered.

- ‘componentDidUpdate(prevProps, prevState)’: This method is called after the component is re-rendered due to an update. It is commonly used for performing side effects like updating the DOM or making API calls based on the updated props or state.

3. Unmounting Phase:

- ‘componentWillUnmount()’: This method is called right before the component is unmounted from the DOM. It is used for cleaning up any resources such as timers, subscriptions, or event listeners created in ‘componentDidMount()’.

Now let's move on to the life cycle of a functional component with hooks.

Life Cycle in Functional Components with Hooks:

React introduced Hooks as a way to add state and lifecycle features to functional components without using classes. The most commonly used hooks are ‘useState’ and ‘useEffect’.

1. **Mounting Phase:**

- ‘useState()’: This hook is used to add state to a functional component. It returns an array with two elements: the current state value and a function to update the state.

- ‘useEffect(callback, dependencies)’: This hook is used to perform side effects in functional components. It takes a callback function as the first parameter and an optional array of dependencies as the second parameter. The callback function is executed after the component is rendered, and it can be used to perform tasks like fetching data, subscribing to events, or updating the DOM. The dependencies array is used to specify values that, when changed, should trigger the callback function to re-run.

2. Updating Phase:

- ‘useState()’: You can use the same `useState` hook to manage state updates.

- ‘useEffect()’: The `useEffect` hook is also used for handling updates. The callback function passed to `useEffect` is re-run whenever any value in the dependencies array changes.

3. Unmounting Phase:

- ‘useEffect()’: You can utilize the ‘useEffect’ hook with a cleanup function to handle the unmounting phase. By returning a cleanup function from the callback, it will be executed when the component is unmounted.

In functional components with hooks, the ‘useState’ hook replaces the need for a constructor, and the ‘useEffect’ hook covers both mounting and updating phases. The cleanup logic in the unmounting phase can be achieved by returning a cleanup function in the ‘useEffect’ callback.