**REACT – LIST AND HOOKS**

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

Ans:- In React, components are the building blocks of UI elements, and understanding their lifecycle is crucial for managing state, handling updates, and performing side effects. The lifecycle of a component refers to the series of phases it goes through from initialization to rendering, updating, and finally unmounting.

**Class Component Lifecycle:**

1. Mounting Phase:

- constructor(): This is the first method called when a component is initialized. It's used for setting initial state and binding event handlers.

- render(): This method renders the component's UI based on its current state and props.

- componentDidMount(): Invoked after the component is mounted (inserted into the tree). It's commonly used for fetching data from APIs, initializing libraries, or setting up subscriptions.

2. Updating Phase:

- shouldComponentUpdate(nextProps, nextState): This method is called before rendering when new props or state are received. It allows optimization by determining if the component needs to update.

- render(): Re-renders the component's UI.

- componentDidUpdate(prevProps, prevState): Invoked after an update has occurred. It's used for performing actions after the component re-renders, like updating the DOM or fetching new data.

3. Unmounting Phase:

- componentWillUnmount(): Invoked just before the component is unmounted and destroyed. It's used for cleanup tasks like cancelling network requests or unsubscribing from event listeners.

**Functional Component with Hooks Lifecycle:**

1. Mounting Phase:

- useState(): Initializes state variables.

- useEffect(() => {}, []): Similar to `componentDidMount`. Runs after the first render and is used for data fetching, setting up subscriptions, or manually changing the DOM. The empty dependency array ensures it only runs once.

2. Updating Phase:

- useState(): Updates state variables.

- useEffect(() => {}): Runs after every render if any dependency in the array changes. It's equivalent to `componentDidUpdate`.

- useEffect(() => {}, []): Runs only once, equivalent to `componentDidMount`.

3. Unmounting Phase:

- useEffect(() => { return () => {} }, []): Cleanup function within `useEffect` runs before the component unmounts. It's equivalent to `componentWillUnmount`.

**Key Differences:**

- Class Component:

- Lifecycle methods are separate, leading to clearer separation of concerns.

- More verbose syntax.

- Class-based, relying on `this`.

- Historical approach, familiar to developers with pre-Hooks experience.

- Functional Component with Hooks:

- Logic is grouped together based on functionality rather than lifecycle phase.

- Cleaner and more concise syntax.

- Functional approach, no `this`.

In summary, while class components and functional components with hooks achieve similar results, hooks provide a more concise and expressive way of managing component lifecycles and state.