**MODULE: 4 (List and Hooks)**

**Q1.** **Explain Life cycle in Class Component and functional component with Hooks**

**Ans1**. **React Component Lifecycle**

The React component lifecycle refers to a series of methods that are invoked during different stages of a component’s existence. These stages include:

**Initialization Phase:**

In this phase, the developer defines the component’s initial state and props. This typically occurs in the constructor of a class component.

Example of initialization in a class component:

class Clock extends React.Component {

constructor(props) {

super(props);

this.state = { date: new Date() };

}

}

**Mounting Phase:**

Mounting is the stage when the component is rendered for the first time and mounted on the DOM.

Key methods during mounting:

constructor: Initializes state and binds methods. Executed before the component is mounted.

render(): Returns JSX to be rendered.

componentDidMount(): Invoked after the component is mounted. Useful for side effects (e.g., data fetching).

Example of mounting in a class component:

class MyComponent extends React.Component {

componentDidMount() {

// Perform actions after component is mounted

}

render() {

return <div>My Component</div>;

}

}

**Updating Phase:**

Occurs when the component’s state changes or receives new props.

Key methods during updating:

shouldComponentUpdate(nextProps, nextState): Determines whether the component should re-render.

componentDidUpdate(prevProps, prevState): Invoked after the component updates.

Example of updating in a class component:

class Counter extends React.Component {

componentDidUpdate(prevProps, prevState) {

// Perform actions after component updates

}

render() {

return <div>{this.props.count}</div>;

}

}

**Unmounting Phase:**

The final step in the component lifecycle, where the component is removed from the page.

Key method during unmounting:

componentWillUnmount(): Invoked just before the component is unmounted. Useful for cleanup (e.g., removing event listeners).

Example of unmounting in a class component:

class MyComponent extends React.Component {

componentWillUnmount() {

// Clean up resources before unmounting

}

render() {

return <div>My Component</div>;

}

}

**Functional Components with Hooks**

React Hooks provide a powerful way to manage state and lifecycle events in functional components. Hooks allow you to use React state and lifecycle features without using class components. Here are some key hooks:

**useState**: Allows functional components to manage state.

**useEffec**t: Replaces lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.

Other hooks like useContext, useReducer, and useCallback provide additional functionality.

Example of using useState and useEffect in a functional component:

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

function MyFunctionalComponent() {

const [count, setCount] = useState(0);

useEffect(() => {

// Perform side effects (e.g., data fetching) after component renders

document.title = `Count: ${count}`;

}, [count]);

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}