**MODULE: 10 List and Hooks**

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

**Ans:** In React, components are the building blocks of a user interface, and they can be created as class components or functional components. With the introduction of React Hooks, functional components can now also have state and lifecycle-like features, making them more powerful and comparable to class components.

**Class Component Lifecycle:**

**Mounting Phase:**

* **constructor():** Called when an instance of the component is being created.
* **render():** Renders the component UI.
* **componentDidMount():** Invoked after the component and its children have been rendered to the DOM. It is commonly used for AJAX requests and setting up subscriptions.

**Updating Phase:**

* **render():** Re-renders the component when state or props change.
* **componentDidUpdate(prevProps, prevState):** Called after the component updates, but not for the initial render. Useful for performing side effects when props or state change.

**Unmounting Phase:**

* **componentWillUnmount():** Invoked immediately before a component is unmounted and destroyed. Cleanup operations can be performed here.

**Functional Component with Hooks Lifecycle:**

**useState:**

* useState(initialState): Allows functional components to have local state.

**useEffect:**

* **useEffect(callback, dependencies):** Performs side effects in function components. It combines functionality from componentDidMount, componentDidUpdate, and componentWillUnmount in class components.
* The callback function is executed after each render.
* dependencies array specifies when the effect should run (similar to componentDidUpdate).
* If no dependencies are provided, the effect runs after every render.
* Here's an example of a functional component with hooks:

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

function FunctionalComponentWithHooks() {

  // useState

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

  // useEffect (componentDidMount)

  useEffect(() => {

    console.log('Component mounted');

    // useEffect cleanup (componentWillUnmount)

    return () => {

      console.log('Component will unmount');

    };

  }, []);

  // useEffect (componentDidUpdate)

  useEffect(() => {

    console.log('Count updated:', count);

  }, [count]);

  return (

    <div>

      <p>Count: {count}</p>

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

    </div>

  );

}

export default FunctionalComponentWithHooks;

export default FunctionalComponentWithHooks;

In this example, useState is used to manage local state, and useEffect handles side effects. The empty dependency array ([]) in the first useEffect call ensures it behaves like componentDidMount. The second useEffect runs whenever the count state changes, similar to componentDidUpdate.