**Module 10) List and Hooks**

**Lifecycle method in Class Component :**

In React, class components have a set of lifecycle methods that allow developers to define specific behavior at different stages of a component's existence. These methods can be categorized into three main phases: Mounting, Updating, and Unmounting. Each phase serves a different purpose and provides developers with hooks to execute code at specific points in the component's life.

**Mounting Phase:**

* constructor(props): This method is called when an instance of the component is being created. It is used to initialize the component's state and bind methods.
* static getDerivedStateFromProps(props, state): This method is invoked right before calling the render method. It allows the component to update its state based on changes in props.
* render(): This is a mandatory method that must be implemented. It returns the JSX representation of the component.
* componentDidMount(): This method is called after the component has been rendered to the DOM. It is often used for operations such as data fetching or setting up subscriptions.

**Updating Phase:**

* static getDerivedStateFromProps(props, state) : This method is also called during the updating phase, similar to the mounting phase.
* shouldComponentUpdate(nextProps, nextState) : This method is used to control whether the component should re-render or not. It can improve performance by preventing unnecessary renders.
* render(): Renders the updated JSX representation of the component.
* getSnapshotBeforeUpdate(prevProps, prevState) : This method is called right before the changes from the virtual DOM are reflected in the actual DOM. It is often used to capture information about the DOM before it gets updated.
* componentDidUpdate(prevProps, prevState, snapshot) : This method is invoked after the component has been updated. It can be used for tasks such as network requests or updating the DOM in response to prop or state changes.

**Unmounting Phase:**

* componentWillUnmount() : This method is called just before the component is unmounted and destroyed. It can be used to perform cleanup tasks such as cancelling network requests, clearing timers, or cleaning up subscriptions.

These lifecycle methods provide a way to manage the behavior of a React class component throughout its existence, allowing developers to execute specific logic at various stages. It's important to note that with the introduction of React Hooks in functional components, some of these lifecycle methods have equivalents, such as useEffect for side effects in functional components.

**Life cycle method in function component :**

In function components, you can achieve similar lifecycle behavior using the useEffect hook and other hooks introduced in React. The useEffect hook allows you to perform side effects in your function components, such as fetching data, subscribing to external events, or manually managing the component lifecycle

**Mounting Phase:**

* useEffect(() => {}, []): This hook is similar to componentDidMount in class components. It runs after the initial render and is used for performing actions like data fetching. The empty dependency array ([]) ensures that the effect runs only once after the initial render.

**Updating Phase:**

* useEffect(() => {}, [dependency]): By specifying dependencies in the array, you can control when the effect runs. This is similar to the combination of componentDidUpdate and shouldComponentUpdate in class components.

**Unmounting Phase:**

* useEffect(() => { return () => {} }, []): The cleanup function inside useEffect serves the same purpose as componentWillUnmount in class components. It is called when the component is about to unmount.

// Example :

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

function FunctionComponent() {

  // Mounting Phase (equivalent to componentDidMount)

  useEffect(() => {

    // Perform initial setup (e.g., data fetching)

    console.log('Component is mounted');

    // Cleanup function (equivalent to componentWillUnmount)

    return () => {

      // Perform cleanup tasks (e.g., clearing timers or subscriptions)

      console.log('Component will unmount');

    };

  }, []); // Empty dependency array means this effect runs only once after the initial render

  // Updating Phase (equivalent to componentDidUpdate and shouldComponentUpdate)

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

  useEffect(() => {

    // Perform actions when 'count' changes

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

  }, [count]);

  return (

    <div>

      <p>Count: {count}</p>

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

    </div>

  );

}

export default FunctionComponent;