React Assignment List And Hooks

**1. What is JavaScript Output method?**

🡪 In React, components are the building blocks of a user interface, and they can be created in two primary ways: as Class Components or Functional Components with Hooks. Both types of components have a life cycle, which refers to the sequence of events that occur during the component's existence. These events allow you to manage component initialization, updates, and cleanup. Let's discuss the life cycle in both types of components:

**Life Cycle in Class Components:**

Class components have a more extensive life cycle and include several lifecycle methods that you can override to control the component's behavior. Here's a brief overview of the key lifecycle methods in class components:

1. **constructor:** This method is called when the com ponent is first initialized. It's used for setting up initial state and binding event handlers.
2. **componentDidMount:** This method is invoked immediately after the component is inserted into the DOM. It's a good place to perform network requests or set up timers.
3. **componentDidUpdate:** This method is called whenever the component's props or state change. It's useful for performing side effects when the component updates.
4. **componentWillUnmount:** This method is called just before the component is removed from the DOM. It's used for cleanup tasks such as canceling network requests or cleaning up timers.

Here's a simplified example of a class component with its lifecycle methods:

class MyComponent extends React.Component

{ constructor(props)

{ super(props);

this.state = { data: [] };

}

componentDidMount()

{ // Perform initial setup here, like fetching data. }

componentDidUpdate(prevProps, prevState)

{ // Handle updates to props or state. }

componentWillUnmount() { // Clean up resources before component unmounts. }

render() {

return <div>{/\* Component content goes here \*/}

</div>; } }

**Life Cycle in Functional Components with Hooks:**

Functional components with Hooks also have a life cycle, but it's more lightweight and flexible compared to class components. Hooks allow you to manage state and side effects in functional components. Here are some equivalent lifecycle scenarios using hooks:

1. **useState:** Use the **useState** hook to manage component state.
2. **useEffect:** The **useEffect** hook is used for performing side effects in functional components. It combines **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount** behaviors in one place.

Here's a simplified example of a functional component with hooks:

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

function MyComponent() {

const [data, setData] = useState([]);

useEffect(() =>

{ // This function runs after the component renders. // You can perform side effects here, like fetching data.

return () =>

{ // This function runs when the component unmounts (cleanup). }; }, [/\* Dependencies \*/]); // Render your component content here.

return <div>{/\* Component content goes here \*/}</div>; }

In functional components with hooks, you use the **useState** hook for managing state and the **useEffect** hook for handling side effects and lifecycle behavior. You can control when the **useEffect** code runs by specifying dependencies in the second argument array, which is similar to how you control **componentDidUpdate** in class components.