REACTJS ASSIGNMENT

MODULE – 4 REACT JS - LISTS AND HOOKS

Submitted to :-

Mr. Raj Nagar

Submitted by :-

Janvi Panchal

MODULE – 4

REACTJS - LISTS AND HOOKS

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

Ans:-

The React component lifecycle refers to the series of phases that a React component goes through, from its creation and rendering to updates and eventual removal from the DOM. While the traditional lifecycle methods are associated with class components, the introduction of hooks has provided a more versatile way to manage component behaviour in functional components.

❖ There are 3 phases in the React Component Life Cycle:

1. Mounting Phase
2. Updating Phase
3. Un-mounting Phase

i. Mounting Phase :

- During the mounting phase, a functional component is being created and added to the DOM. In this phase, you typically initialise state and perform any setup that's needed when the component is first rendered.

  - UseState :

    The **useState** hook allows you to add state to your functional components. It replaces the need for a constructor and **this.state** in class components. You can initialise state and retrieve the current value and a function to update it.

- UseEffect :

  The **useEffect** hook with an empty dependency array simulates the **componentDidMount** lifecycle method. It runs the provided function after the component is first rendered. This is a good place to perform data fetching or initial setup.

ii. Updating Phase :

- In the updating phase, the functional component is re-rendered due to changes in its props or state. You can use the **useEffect** hook without an empty dependency array to achieve behaviour similar to **componentDidUpdate**.

- UseEffect :

  By using the **useEffect** hook without a dependency array, you can simulate the behaviour of **componentDidUpdate**. The provided function will run on every render.

iii. Unmounting Phase:

- In the unmounting phase, the functional component is being removed from the DOM. The cleanup function in the **useEffect** hook simulates the behaviour of **componentWillUnmount**.

- UseEffect : By returning a function from the **useEffect** hook, you can specify cleanup operations to be performed when the component is unmounted.

Conclusion:-

- Always use the appropriate hook for the intended behaviour to keep your code clean and maintainable.

- Functional components with hooks offer a more concise and intuitive way to manage component behaviour compared to class components and lifecycle methods.

- Understanding these phases and using hooks effectively will enhance your ability to build efficient and responsive React applications.