Lists and Hooks

Q . Explain Life cycle in Class Component and functional component with Hooks?

Ans: The React component lifecycle is a collection of methods that are called at different stages of a component's lifecycle. These methods allow you to perform specific tasks at each stage of the lifecycle, such as initializing the component, updating the component, and unmounting the component.

* Class components:

In class components, you can use lifecycle methods to perform various tasks at different stages of a component's lifecycle. The most commonly used lifecycle methods are:

1. componentDidMount: componentDidMount(): This method is called after the component is inserted into the DOM. It's commonly used for fetching data, setting up timers, or performing any side effects.
2. componentDidUpdate(): The componentDidUpdate() method is called whenever the component's state or props change. It is a good place to update the DOM based on the new state or props.
3. componentWillUnmount(): This method is called just before the component is removed from the DOM. It's typically used for cleaning up resources like event listeners.

Ex: import React, { Component } from 'react';

class ExampleClassComponent extends Component {

componentDidMount() {

// Fetch data or set up timers

}

componentDidUpdate(prevProps, prevState) {

// Perform actions after an update

}

componentWillUnmount() {

// Clean up resources

}

render() {

return <div>Class Component with Lifecycles</div>;

}

}

* Functional components with hooks

Functional components can also manage lifecycles using hooks. The most commonly used hooks for lifecycle management are:

1. useState: Allows functional components to manage state. Used for updating state variables.

useEffect(() => { return () => {} }, []): The cleanup function in useEffect runs when the component is about to unmount. This is equivalent to componentWillUnmount.

1. useEffect(() => {}, []): Invoked after the component is rendered. It can perform side effects like data fetching or subscriptions. The empty dependency array ensures it runs only once after the initial render.

runs when the component is about to unmount. This is equivalent to componentWillUnmount.

Ex:

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

const ExampleComponent = () => {

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

useEffect(() => {

// This runs after the component renders

fetchData();

return () => {

};

}, []); // Empty dependency array means this effect runs only once on mount

return (

<div>

{/\* Component rendering logic \*/}

</div>

);

};