Lists & Hooks Assignment

(Module-4)

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

In React, components are the building blocks of a user interface, and they can be categorized into two main types:

Class components and Functional components.

In the latest version of React Hooks (Version 16.8), functional components gained the ability to manage state and side effects, making them more powerful and reducing the need for class components.

Life Cycle in Class Components:

1. **Mounting Phase:**
    - **constructor()**: This is the first method called when a component is created. It is used for initializing state and binding methods.
    - **render()**: It is responsible for rendering the JSX of the component.
    - **componentDidMount()**: This method is invoked immediately after a component is inserted into the DOM. It is often used for fetching data or setting up subscriptions.
2. **Updating Phase:**

- **shouldComponentUpdate(nextProps, nextState)**: This method is called before rendering when new props or state are received. It returns a boolean to determine if the component should re-render.
- **render()**: Renders the updated JSX.
- **componentDidUpdate(prevProps, prevState)**: Invoked immediately after updating occurs. It is often used for interacting with the DOM or making network requests based on the updated props or state.

3. **Unmounting Phase:**
    - **componentWillUnmount()**: This method is called just before the component is removed from the DOM. It is used for cleanup, such as canceling network requests or cleaning up subscriptions.

Life Cycle in Functional Components with Hooks:

1. **Mounting Phase:**
    - **useState()**: Allows functional components to have state.
    - **useEffect(() => {}, [])**: This hook replaces **componentDidMount**. The function inside **useEffect** runs after the initial render. The empty dependency array (**[]**) ensures that it only runs once.
2. **Updating Phase:**
    - **useState()**: Still used to manage state updates.
    - **useEffect(() => {})**: This hook replaces **componentDidUpdate**. The function inside **useEffect** runs after every render, but you can control when it runs by specifying dependencies.
3. **Unmounting Phase:**

- **useEffect(() => { return () => {} }, [])**: This hook replaces **componentWillUnmount**. The cleanup function inside **useEffect** is called when the component is unmounted.

EXAMPLE:

```
// Class Component
class ComponetLifeCycle extends Component {
    constructor() {
        super();
        this.state = {count:0};
    }
    componentDidMount()
    {
        console.log("Mounted...!")
    }
    handleClick = ()=>{
        this.setState({count:this.state.count +1})
    }
    render() {
    return (
      <div>
        <h4 style={{border:"1px solid"}}
onClick={this.handleClick}>{this.state.count}</h4>
      </div>
    )
  }
```

```
  shouldComponentUpdate()
  {
    console.log("should updated..?");
    return true;
  }
  componentDidUpdate()
  {
    console.log("updated..!")
  }
}


// Functional Component with Hooks
const FunctionalComponentExample = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    // Equivalent to componentDidMount
    // Fetch data or set up subscriptions

    return () => {
      // Equivalent to componentWillUnmount
      // Cleanup operations
    };
```

```
  }, []); // Empty dependency array means it only runs once after initial
render


  useEffect(() => {
    // Equivalent to componentDidUpdate
    // Perform actions based on updated props or state
  }, [data]); // Specify dependencies to control when this effect runs


  // Render JSX
};
```