**Batch: T4**

**Practical No. 5**

**Title of Assignment: React Js**

**Student Name: Daivik Liladhar Karbhari**

**Student PRN: 22510111**

**Perform following problem statements using ReactJs**

**Problem Statement 0: Basics of ReactJs**

* What is React and what problem does it solve?
* React is a JavaScript library used for building user interfaces, primarily for single-page applications (SPAs). It allows developers to create reusable UI components and efficiently update and render the right components when data changes. React solves problems like:
  + Efficient updates through a virtual DOM, which reduces the number of DOM manipulations.
  + Reusability of components for faster development.
  + Easier management of dynamic data in UI.
* What are React components and how are they used?
* React components are reusable building blocks of a React application. Each component encapsulates its structure (HTML-like JSX), styles, and behavior (logic). Components can be functional (stateless) or class-based (stateful).
* **Functional Components** are simpler and are functions that return JSX.
* **Class Components** contain state and lifecycle methods.
* What is JSX in React?
* JSX (JavaScript XML) is a syntax extension of JavaScript used in React to describe what the UI should look like. It allows you to write HTML-like code within JavaScript, making it easier to visualize the component structure.
* Example: const element = <h1>Hello, World!</h1>;
* Under the hood, JSX is transformed into JavaScript (e.g., React.createElement).
* What are props in React and how do they differ from state?
* **Props** (short for "properties") are used to pass data from a parent component to a child component. They are immutable and help configure components. Props are read-only, meaning a component can only receive data through props but cannot modify it.
* **State** is an object managed within a component (usually in class components or through hooks like useState in functional components). State is mutable and allows components to handle dynamic data. Changes to the state trigger re-renders of the component.
* What is state in React and how does it work?
* State is a local data store within a component that holds information that can change over time. When the state changes, React re-renders the component to reflect the updated state in the UI. You modify state using the setState method in class components or the useState hook in functional components.
* Example with functional component: const [count, setCount] = useState(0);
* Here, count is the current state, and setCount is a function to update it.
* What are React lifecycle methods, and why are they important?
* Lifecycle methods are special methods in class components that get called at different stages of a component's life, from mounting to updating and unmounting. They are important for managing side effects like fetching data, setting timers, or cleaning up resources. Common lifecycle methods include:
  + componentDidMount: Called after the component is first rendered.
  + componentDidUpdate: Called when the component’s state or props change.
  + componentWillUnmount: Called right before the component is removed from the DOM.
* Elaborate following with respect to ReactJs

**a. Event Handling**

React uses synthetic events to handle user input like clicks, key presses, etc. Synthetic events are wrappers around native browser events and work the same across different browsers.

Example: <button onClick={handleClick}>Click me</button>

**b. Conditional Rendering**

Conditional rendering in React allows you to render different UI elements based on certain conditions.

Example: {isLoggedIn ? <LogoutButton /> : <LoginButton />}

**c. Lists and Keys**

When rendering lists of elements in React, each element needs a unique "key" to help React efficiently update and render only the changed items.

Example: {items.map(item => <li key={item.id}>{item.name}</li>)}

**d. Forms**

Handling forms in React involves controlling the form input using state. You typically create controlled components where form inputs reflect the state.

Example: <input type="text" value={this.state.value} onChange={this.handleChange} />

**e. Hooks**

Hooks are functions that allow functional components to use state and other React features like lifecycle methods without using class components. Common hooks include:

* useState: Manages state in functional components.
* useEffect: Handles side effects (replaces lifecycle methods).
* useContext: Allows accessing context values directly.

**f. React Router**

React Router is a library for managing navigation and routing in a React application. It helps in building a multi-page experience in a single-page app by mapping URLs to components. Example: <Route path="/about" component={About} />

**g. State Management**

State management is crucial for managing the dynamic data across components. While React has local component state, tools like Redux or Context API allow managing global state across the entire application.

**h. React Context API**

Context API allows you to share state globally without passing props down through multiple levels. It is useful for managing global data like user authentication status, theme settings, etc.

Example: const ThemeContext = React.createContext();

**i. How can you optimize the performance of a React application?**

* **Memoization**: Use React.memo or useMemo to prevent unnecessary re-renders.
* **Lazy Loading**: Split code and load components dynamically using React.lazy and Suspense.
* **Use useCallback**: Prevent re-creation of functions on each render.
* **Avoid unnecessary re-renders**: Implement shouldComponentUpdate or React.PureComponent for class components.
* **Virtualization**: Render only visible parts of large lists using libraries like react-window or react-virtualized.
* **Efficient state updates**: Avoid deep state updates to reduce unnecessary re-rendering.

These practices help ensure smoother performance, especially for large applications.