Assignment – list hooks

Question 1: How do you render a list of items in React? Why is it important to use keys when rendering lists?

In React, you render a list of items using the .map() function to iterate over the array and return JSX elements. Keys are important because they help React identify which items have changed, been added, or removed, optimizing re-renders and improving performance**.**

Question 2: What are keys in React, and what happens if you do not provide a unique key? answer in brief without code

Keys in React are unique identifiers assigned to list elements to help React track changes efficiently. If you do not provide a unique key, React may use the default array index, which can lead to incorrect UI updates, performance issues, and unexpected behavior when the list changes dynamically**.**

Question 3: How do you handle forms in React? Explain the concept of controlled components

Forms in React are handled using controlled components, where form inputs are tied to the component's state. The input values are updated via state and modified using event handlers, ensuring React fully controls the form elements.

Question 4: What is the difference between controlled and uncontrolled components in React?

Controlled components have their values managed by React state, ensuring a single source of truth. Uncontrolled components, on the other hand, let the DOM manage the input values using refs, making them useful for simple forms where direct DOM access is preferred. Controlled components provide better control, validation, and predictability.

Question 5: What are lifecycle methods in React class components? Describe the phases of a component’s lifecycle.

Lifecycle methods in React class components are special methods that run at different stages of a component’s life. The lifecycle has three phases:

1. **Mounting** – When the component is created and inserted into the DOM.
2. **Updating** – When the component re-renders due to state or prop changes.
3. **Unmounting** – When the component is removed from the DOM.

Question 6: Explain the purpose of componentDidMount(), componentDidUpdate(), and componentWillUnmount()

 **componentDidMount()**: Runs after the component is added to the DOM, often used for data fetching or setting up subscriptions.

 **componentDidUpdate()**: Runs after an update (state or props change), useful for side effects like fetching new data.

 **componentWillUnmount()**: Runs before the component is removed from the DOM, used for cleanup tasks like clearing timers or unsubscribing from events.

Question7: What are React hooks? How do useState() and useEffect() hooks work in functional components?

React Hooks are functions that allow functional components to use state and lifecycle features without needing class components.

* **useState()**: Manages state within a functional component.
* **useEffect()**: Handles side effects such as API calls, subscriptions, or DOM updates in functional components.

Question8: What problems did hooks solve in React development? Why are hooks considered an important addition to React?

Hooks solved issues with class components, such as complex state management and lifecycle logic reuse. They improved code reusability, readability, and made functional components more powerful. Hooks eliminate the need for class components, simplifying React development.

Question9: What is useReducer ? How we use in react app?

**useReducer** is an alternative to useState for managing complex state logic. It uses a reducer function to handle state transitions, making it useful for applications with multiple state updates. It is commonly used for state management in combination with the Context API.

Question10: What is the purpose of useCallback & useMemo Hooks?

 **useCallback**: Optimizes functions by memoizing them, preventing unnecessary re-creations on re-renders.

 **useMemo**: Memoizes the result of an expensive computation, recalculating only when dependencies change.

Question 11: What’s the Difference between the useCallback & useMemo Hooks?

 **useCallback** returns a memoized function, preventing unnecessary function re-creations.

 **useMemo** returns a memoized value, optimizing expensive calculations by caching results.

Question 12 : What is useRef ? How to work in react app?

**useRef** creates a reference to DOM elements or persists mutable values without causing re-renders. It is useful for accessing input fields, storing previous state values, or handling animations in React applications.