Hooks

10. Hooks(useState, useEffect, useReducer, useMemo, useRef, useCallback)

1. What are React Hooks? How do useState() and useEffect() work?

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

- **useState()**: Used to manage state in a functional component.

const [count, setCount] = useState(0);

  - count stores the value
  - setCount updates the value

- **useEffect()**: Runs side effects like data fetching, event listeners, etc.

useEffect(() => {

  console.log("Component mounted or updated");

}, [count]); // Runs when `count` changes

---

2. What problems did hooks solve? Why are hooks important?

Problems Solved:

- Removed the need for class components.
- Simplified state and lifecycle management in functional components.
- Made code reusable with **custom hooks**.

Importance:

- Cleaner and more readable code.
- Better performance with optimized re-renders.
- Easier state sharing across components.

---

3. What is useReducer()? How to use it in React?

useReducer() is an alternative to useState() for managing **complex state logic**.

- **Example:**

```
const reducer = (state, action) => {
  switch (action.type) {
    case "increment": return { count: state.count + 1 };
    case "decrement": return { count: state.count - 1 };
    default: return state;
  }
};

const [state, dispatch] = useReducer(reducer, { count: 0 });

<button onClick={() => dispatch({ type: "increment" })}>+</button>
```

- dispatch triggers actions to update state.
- Best for managing **multiple state changes** efficiently.

---

4. Purpose of useCallback() & useMemo() Hooks

- **useCallback()**: Memorizes functions to **prevent unnecessary re-creation**.
- **useMemo()**: Memorizes computed values to **avoid re-computation**.
- Example:

```
const memoizedFunction = useCallback(() => {
  console.log("Only created when `count` changes");
}, [count]);

const memoizedValue = useMemo(() => count * 2, [count]);
```

---

5. Difference between useCallback() & useMemo()

- **useCallback(fn, deps)** → Returns **a memorized function**.
- **useMemo(() => value, deps)** → Returns **a memorized value**.

- useCallback is used for **functions**, while useMemo is used for **values/computed results**.

---

6. What is useRef()? How does it work?

useRef() is used to reference **DOM elements** or store **mutable values** without causing re-renders.

- **Example (Accessing DOM):**

```
const inputRef = useRef(null);


const focusInput = () => {
  inputRef.current.focus();
};


<input ref={inputRef} />
<button onClick={focusInput}>Focus Input</button>
```

- **Doesn't cause re-renders** when the value changes.
- Useful for **storing previous values or persisting state between renders**.