**Question 1: What are React hooks? How do useState() and useEffect() hooks work in functional components?**

**React Hooks** are built-in functions introduced in React 16.8 that let you use state and other React features in functional components.

**useState():**

* Adds state to functional components.
* Syntax: const [state, setState] = useState(initialValue);

const Counter = () => {

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

return <button onClick={() => setCount(count + 1)}>Count: {count}</button>;

};

**useEffect():**

* Performs side effects like fetching data, setting up subscriptions, etc.
* Syntax: useEffect(() => { ... }, [dependencies]);

useEffect(() => {

console.log("Component mounted or count changed");

}, [count]);

**Question 2: What problems did hooks solve in React development? Why are hooks considered an important addition to React?**

**Problems solved by hooks:**

* **Avoid class components:** Hooks let you use state and lifecycle methods in functional components.
* **Reusability:** Hooks allow logic reuse via custom hooks instead of higher-order components or render props.
* **Better organization:** They help in separating concerns and organizing code more cleanly.

**Why important?**

* They simplified component logic.
* Improved performance and code readability.
* Made functional components more powerful and standard in React development.

**Question 3: What is useReducer? How is it used in a React app?**

* useReducer is a hook used for managing **complex state logic** (like multiple related state updates).
* Similar to Redux reducer but local to a component.

const initialState = { count: 0 };

function reducer(state, action) {

switch (action.type) {

case "increment":

return { count: state.count + 1 };

case "decrement":

return { count: state.count - 1 };

default:

return state;

}

}

const Counter = () => {

const [state, dispatch] = useReducer(reducer, initialState);

return (

<>

<p>Count: {state.count}</p>

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

<button onClick={() => dispatch({ type: "decrement" })}>-</button>

</>

);

};

**Question 4: What is the purpose of useCallback & useMemo Hooks?**

* **useCallback**: Caches a function so that it doesn’t get recreated on every render unless dependencies change.
* **useMemo**: Caches the result of an expensive computation to avoid recalculating it on every render.

**Question 5: What’s the difference between the useCallback & useMemo Hooks?**

| **Feature** | **useCallback** | **useMemo** |
| --- | --- | --- |
| Returns | A **memoized function** | A **memoized value (result)** |
| Use case | Prevents unnecessary re-creations of functions | Caches results of expensive calculations |
| Syntax | useCallback(fn, deps) | useMemo(() => compute, deps) |

**Question 6: What is useRef? How does it work in a React app?**

* useRef creates a **mutable reference** that persists across renders.
* It can be used to:
  + Access DOM elements directly
  + Store values without causing re-renders

const InputFocus = () => {

const inputRef = useRef();

const focusInput = () => {

inputRef.current.focus();

};

return (

<>

<input ref={inputRef} type="text" />

<button onClick={focusInput}>Focus Input</button>

</>

);

};