Q1. What are React hooks? How do useState() and useEffect() hooks work in functional components?

Ans: React Hooks are special functions introduced in React 16.8 that allow developers to use state and other React features in functional components, without writing class components.

Hooks make code:

- Simpler and cleaner
- Easier to reuse logic
- Easier to read and maintain

Commonly used hooks are useState(), useEffect(), useContext(), etc.

useState() Hook

The useState() hook is used to add state (data) to a functional component.

How it works:

- It returns two values:
    1. Current state value
    2. A function to update the state

Syntax:

const [state, setState] = useState(initialValue);

useEffect() Hook

The useEffect() hook is used to handle side effects in functional components.

Side effects include:

- Fetching data from API
- DOM manipulation
- Timers (setTimeout, setInterval)
- Subscriptions

Syntax:

useEffect(() => {

```
  // side effect code
}, [dependencies]);
```

Q2. What problems did hooks solve in React development? Why are hooks considered an important addition to React?

Ans: 1. Use State and Lifecycle in Functional Components

Hooks allow functional components to:

- Use state
- Use lifecycle features
  without writing class components.

2. Cleaner and Readable Code

Hooks remove:

- Class syntax
- Constructor
- this keyword

This makes components shorter and easier to read.

3. Better Logic Reusability

Custom hooks allow developers to:

- Extract reusable logic
- Share logic between components
  without changing component structure.

4. Improved Separation of Concerns

Hooks group related logic together instead of splitting it across lifecycle methods, making code more organized.

Why Hooks Are an Important Addition to React

- Simplify React development
- Encourage functional programming
- Improve code readability and maintenance
- Reduce bugs related to this

- Make state management and side effects easier

Q3. What is useReducer ? How we use in react app?

Ans: useReducer is a React Hook used for state management in functional components. It is an alternative to useState() and is especially useful when:

- State logic is complex
- State depends on previous state
- Multiple related values need to be updated together

useReducer works on the reducer pattern, similar to Redux.

Ex:

```
import { useReducer } from "react";

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;
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: "INCREMENT" })}>+</button>
```

```
    <button onClick={() => dispatch({ type: "DECREMENT" })}>-</button>
   </div>
  );
}
export default Counter;
```

Q4. What is the purpose of useCallback & useMemo Hooks?

Ans: Both useCallback and useMemo are React Hooks used for performance optimization. They help prevent unnecessary re-renders and re-calculations in React applications.

useCallback Hook:

useCallback is used to memoize a function, so the same function reference is reused unless its dependencies change.

Purpose:

- Prevents re-creation of functions on every render
- Improves performance when passing functions as props to child components

Syntax:

```
const memoizedFunction = useCallback(() => {
  // function logic
}, [dependencies]);
```

useMemo Hook:

useMemo is used to memoize a calculated value, so the value is recomputed only when its dependencies change.

Purpose:

- Avoids expensive calculations on every render
- Improves performance in complex computations

Syntax:

```
const memoizedValue = useMemo(() => {
  return expensiveCalculation();
}, [dependencies]);
```

Q5. What's the Difference between the useCallback & useMemo Hooks?

Ans: Both useCallback and useMemo are React Hooks used for performance optimization, but they serve different purposes.

Feature	useCallback	useMemo
Memoizes	Function	Value
Returns	Same function reference	Cached result
Use case	Passing callbacks	Heavy calculations
Performance goal	Avoid re-renders	Avoid re-calculation

Q6. What is useRef ? How to work in react app?

Ans: useRef is a React Hook used to create a mutable reference that persists across component renders without causing re-render when its value changes.

It is commonly used to:

- Access DOM elements directly
- Store previous values
- Hold mutable data (like timers, intervals)

How useRef Works

- useRef() returns an object with a single property: current
- Changing ref.current does NOT re-render the component
- The value remains the same between renders

Syntax:

const ref = useRef(initialValue);

How We Use useRef in a React App

1. Import useRef
2. Create ref using useRef()
3. Attach ref to JSX element OR store value
4. Access value using .current