3. React Hooks

Last updated by | Subramanya Dixit | May 5, 2025 at 7:59 PM GMT+5:30

📘 Why Hooks Are Important

Hooks allow you to use state and lifecycle features in functional components. They simplify component logic, improve code reuse, and make React code cleaner and easier to understand—especially important as we avoid class-based components.

🧠 Key Concepts

✅ useState

Allows you to add state to functional components.

```
import { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

✅ useEffect

Runs side effects (e.g., API calls, timers) in functional components.

```
import { useEffect } from 'react';

useEffect(() => {
  console.log('Component mounted');
}, []); // Empty array = run only once
```

✅ useContext

Provides access to shared values like theme or user data without prop drilling.

```
const MyContext = React.createContext();

function App() {
  return (
    <MyContext.Provider value="Hello">
      <Child />
    </MyContext.Provider>
  );
}

function Child() {
  const value = useContext(MyContext);
  return <div>{value}</div>;
}
```

✅ useRef

Gives you a mutable ref object whose `.current` property persists across renders.

```
const inputRef = useRef();
<input ref={inputRef} />
```

✅ useMemo

Memoizes expensive calculations to improve performance.

```
const result = useMemo(() => heavyCalculation(num), [num]);
```

✅ useCallback

Returns a memoized callback to avoid unnecessary re-renders.

```
const memoizedCallback = useCallback(() => {
  doSomething();
}, [dependency]);
```

🧩 **Advanced Hooks**

🔁 useReducer

Alternative to `useState` for managing complex state logic.

```
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 (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({ type: 'increment' })}>+</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
    </>
  );
}
```

🧱 Custom Hooks

Reuse logic across multiple components.

```
function useToggle(initialValue = false) {
  const [value, setValue] = useState(initialValue);
  const toggle = () => setValue((v) => !v);
  return [value, toggle];
}

function ToggleComponent() {
  const [isToggled, toggle] = useToggle();
  return <button onClick={toggle}>{isToggled ? 'ON' : 'OFF'}</button>;
}
```

🧠 Visual Flowchart (React Hooks Lifecycle)

```
flowchart TD
  A[Component Renders] --> B[useState initializes state]
  B --> C[useEffect runs side effects]
  C --> D[User Interaction]
  D --> E[setState triggers re-render]
  E --> A
```

💡 Guidelines

- Always call Hooks at the top level of the component.
- Only call Hooks from React functions, not regular JS functions.
- Break complex state into multiple useState/useReducer calls.
- Use useEffect for data fetching, subscriptions, or timers.

📝 Practice Exercises

1. Create a counter component using `useState` .
2. Add `useEffect` to log to the console when the component mounts.
3. Create a context and share data between parent and child components using `useContext` .
4. Use `useRef` to focus an input when a button is clicked.
5. Create a `useToggle` custom hook and use it in a component.
6. Replace `useState` with `useReducer` in a counter app.

❓ Quiz Questions

1. What is the purpose of `useState` ?

a) To fetch data
✅ **b) To add state to functional components**
c) To perform side effects
d) To share data globally

2. Which Hook is used for side effects like API calls?

a) useMemo

b) useState

c) useRef

✅ **d) useEffect**

---

3. What will `useContext` help you avoid?

a) useEffect

✅ **b) Prop drilling**

c) Re-renders

d) useState

---

4. When does `useEffect(() => {...}, [])` run?

a) On every render

✅ **b) Only on mount**

c) Only on updates

d) Never

---

5. Why would you use `useReducer` instead of `useState`?

a) To avoid props

b) To create reusable components

✅ **c) To handle complex state transitions**

d) To fetch data

---

6. What is a custom Hook in React?

a) A built-in React Hook

b) A lifecycle method

✅ **c) A reusable function that uses Hooks**

d) An external dependency
