# Series 2: Hooks & Lifecycle

---

## ✅ 1. What are Hooks in React?

💬 **Interview Answer:**
Hooks are special functions in React that let you use state and other React features inside functional components, without writing a class.

🧠 **Key Hook Names:** `useState`, `useEffect`, `useContext`, `useRef`, `useMemo`, `useCallback`

📌 **Example:**

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

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

🔁 **Analogy:**
Hooks are like plug-ins for power tools — they add superpowers to simple functions.

---

## ✅ 2. What is `useState` and how does it work?

💬 **Interview Answer:**
`useState` is a Hook that lets you add and manage local state in functional components. It returns a state variable and an updater function.

📌 **Example:**

```
const [name, setName] = useState('');
```

🔁 **Analogy:**
It's like a notebook with a pen — you write down current values (`state`) and can update them anytime.

---

## ✅ 3. What is `useEffect` and when is it used?

💬 **Interview Answer:**
`useEffect` runs side effects in React — like fetching data, setting timers, or manually updating the DOM. It replaces lifecycle methods like `componentDidMount`.

📌 **Example:**

```
useEffect(() => {
  console.log('Component mounted');
}, []);
```

🔁 **Analogy:**
Like setting an alarm clock after something happens (e.g., page loads).

---

### ✳️ Follow-Up Interview Questions:

- **When does `useEffect` run?**
  It runs after the render. With an empty dependency array `[]`, it runs once after the first render.

**How to clean up side effects?**
Return a cleanup function:

```
useEffect(() => {
  const timer = setInterval(...);
  return () => clearInterval(timer);
}, []);
```

- 

---

## ✅ 4. What is `useRef` used for?

💬 **Interview Answer:**
`useRef` returns a mutable object that persists across renders. It can reference DOM nodes or store values without causing re-renders.

📌 **Example:**

```
const inputRef = useRef();

<input ref={inputRef} />
<button onClick={() => inputRef.current.focus()}>Focus</button>
```

🔁 **Analogy:**
It's like bookmarking a page — you can return to it anytime without changing the book itself.

---

## ✅ 5. What is `useMemo` and why is it useful?

💬 **Interview Answer:**
`useMemo` memoizes expensive computations and only re-runs them when dependencies change — improving performance.

📌 **Example:**

```
const result = useMemo(() => expensiveCalc(value), [value]);
```

🔁 **Analogy:**
It's like saving a math result on a sticky note — reuse it until inputs change.

---

## ✅ 6. What is `useCallback` and how is it different from `useMemo`?

💬 **Interview Answer:**
`useCallback` memoizes a function definition, preventing it from being re-created unless dependencies change. Useful when passing functions to child components.

📌 **Example:**

```
const handleClick = useCallback(() => {
  console.log('Clicked!');
}, []);
```

🔁 **Analogy:**
Like saving a function in your phone — you don't rewrite it every time you want to use it.

---

## ✅ 7. What are Custom Hooks?

💬 **Interview Answer:**
Custom Hooks are reusable logic extractors. They're just functions that use React hooks internally and help keep components clean.

📌 **Example:**

```
function useCounter() {
  const [count, setCount] = useState(0);
  return { count, increment: () => setCount(count + 1) };
}
```

🔁 **Analogy:**
Like reusable formulas in Excel — same logic, different inputs.

---

## ✅ 8. What is the Component Lifecycle in Functional Components?

💬 **Interview Answer:**
In functional components, lifecycle phases like **mounting**, **updating**, and **unmounting** are handled with `useEffect`.

| Lifecycle Phase | Equivalent with `useEffect` |
| --- | --- |
| componentDidMount | `useEffect(..., [])` |
| componentDidUpdate | `useEffect(..., [deps])` |

| | |
|---|---|
| componentWillUnmount | return a cleanup function inside `useEffect` |

---

## ✅ 9. What are Hook Rules?

💬 **Interview Answer:**
Hooks must follow 2 main rules:

1. Only call Hooks at the top level (not inside loops or conditions)

2. Only call Hooks in React functions (components or custom hooks)

🔁 **Analogy:**
It's like obeying driving rules — break them and things crash (literally, in React).

---