🔧 Introduction to **Hooks** in React

React **Hooks** are special functions that let you "hook into" React features like state and lifecycle methods — **without writing a class**.

Hooks were introduced in **React 16.8** to allow developers to use state and other React features in **functional components**.

🚀 Why Hooks?

Before hooks, only **class components** could manage state and use lifecycle methods like `componentDidMount`, `componentDidUpdate`, etc.

Hooks allow you to:

- Use **state**
- Use **lifecycle effects**
- **Reuse logic** across components
- Write **cleaner, simpler code**

🔑 Most Common Hooks

Hook	Purpose
`useState()`	Add state to a functional component
`useEffect()`	Perform side effects (like lifecycle methods)
`useContext()`	Access context directly
`useRef()`	Access or reference a DOM element or store mutable values
`useMemo()`	Optimize performance by memoizing calculations
`useCallback()`	Memoize functions to prevent unnecessary re-renders
`useReducer()`	For complex state logic (like Redux lite)

Hook	Purpose
`useLayoutEffect()`	Like `useEffect` but runs synchronously after DOM updates

🧪 Example: `useState` & `useEffect`

jsx

Copy Edit

```jsx
import React, { useState, useEffect } from 'react'; function Counter() { const [count, setCount] = useState(0); // state useEffect(() => { console.log(`Count is: ${count}`); // side-effect (like componentDidUpdate) }, [count]); // runs when 'count' changes return ( <div> <p>Count: {count}</p> <button onClick={() => setCount(count + 1)}>Increment</button> </div> ); }
```

📌 Rules of Hooks

1. ✅ Only call Hooks **at the top level** of your component (not inside loops or conditions).
2. ✅ Only call Hooks **from React functions** (functional components or custom hooks).

🧩 Custom Hooks

You can create your own hook using the `use` prefix:

js

Copy Edit

```js
function useCustomLogic() { const [data, setData] = useState(null); // your logic here return data; }
```