◇ WHAT ARE HOOKS?

❓ What are React Hooks?

👉 **Hooks are functions that let you use React features (state, lifecycle, context, etc.) inside functional components.**

Before Hooks:

- State ❌ only in class components
- Lifecycle ❌ only in class components

After Hooks:

- **Everything in functional components** ✅

❓ Why Hooks were introduced?

- Avoid **class components complexity**
- Avoid **this keyword confusion**
- Reuse logic easily
- Cleaner & readable code

📌 **Rule of Hooks:**

1. Call Hooks **only at the top level**
2. Call Hooks **only inside React function components or custom hooks**

◇ 1️⃣ useState Hook

❓ What is useState?

👉 useState is used to **create and manage state in a functional component**.

◇ Syntax

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

◇ Simple Example

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

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

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

◇ Why useState?

- Component re-render happens when state changes
- Used for counters, form inputs, toggles, etc.

❓ Questions

- **Does useState update immediately?** ❌ No (async)
- **Can we store objects/arrays?** ✅ Yes
- **Does changing normal variable re-render component?** ❌ No

📌 **One-liner:**

useState allows functional components to hold and update state and triggers re-render.

◇ 2️⃣ useEffect Hook

❓ What is useEffect?

👉 useEffect is used to **handle side effects** in React.

◇ Side Effects include:

- API calls
- Timers
- DOM manipulation
- Subscriptions

◇ Syntax

```
useEffect(() => {
  // side effect code
}, [dependencies]);
```

◇ Example: Run once (API call)

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

function Users() {
  useEffect(() => {
    console.log("Component Mounted");
  }, []);

  return <h1>Users</h1>;
}
```

◇ Dependency cases (VERY IMPORTANT)

Dependency	Meaning
[]	Runs once (on mount)
[state]	Runs when state changes
No array	Runs on every render

◇ Cleanup Example

```
useEffect(() => {
  const timer = setInterval(() => {
    console.log("Running...");
  }, 1000);

  return () => clearInterval(timer);
}, []);
```

📌 **One-liner:**

useEffect replaces lifecycle methods like componentDidMount, componentDidUpdate, componentWillUnmount.

◇ 3 useContext Hook

❓ Problem it solves?

👉 **Prop drilling** (passing props again and again)

◇ What is useContext?

👉 useContext allows components to **consume global data** without passing props.

◇ Example

```
import { createContext, useContext } from "react";

const ThemeContext = createContext();

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Header />
    </ThemeContext.Provider>
  );
}

function Header() {
  const theme = useContext(ThemeContext);
  return <h1>Theme: {theme}</h1>;
}
```

📌 **Used for:** theme, auth, user data

📌 **One-liner:**

useContext is used to access context values directly without prop drilling.

◇ 4 useReducer Hook

❓ Why useReducer?

👉 When state logic becomes **complex**

◇ Similar to:

- Redux pattern
- Action + reducer logic

◇ Example

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

const reducer = (state, action) => {
  if (action.type === "increment") {
    return { count: state.count + 1 };
  }
  return state;
};

function Counter() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <>
      <p>{state.count}</p>
      <button onClick={() => dispatch({ type: "increment" })}>

            Increment
      </button>
    </>
  );
}
```

📌 **useState vs useReducer**

- Simple state → useState
- Complex state → useReducer

📌 **One-liner:**

useReducer is used for managing complex state logic in a predictable way.

◇ 5️⃣ useMemo Hook

❓ What is useMemo?

👉 **Memoizes a value** to avoid expensive recalculations.

◇ Example

```
import { useMemo, useState } from "react";

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

  const double = useMemo(() => {

return count * 2;
  }, [count]);

  return (
    <>
      <p>{double}</p>
      <button onClick={() => setCount(count + 1)}>+</button>
    </>
  );
}
```

📌 **Used for performance optimization**

📌 **One-liner:**

useMemo memorizes computed values to improve performance.

◇ 6 useCallback Hook

❓ What is useCallback?

👉 **Memoizes a function**, not a value.

◇ Example

```
import { useCallback, useState } from "react";

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



const increment = useCallback(() => {
    setCount(prev => prev + 1);
  }, []);

  return <button onClick={increment}>+</button>;
}
```

📌 Difference

- useMemo → value
- useCallback → function

📌 One-liner:

useCallback prevents unnecessary function re-creation.

◇ 7️⃣ useRef Hook

❓ What is useRef?

👉 Used to **access DOM elements** or **persist values without re-rendering**.

◇ Example (Focus input)

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

function App() {
  const inputRef = useRef();

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

📌 **Important:** Updating ref does **NOT** re-render component

📌 **One-liner:**

useRef stores mutable values without causing re-render.

🔥 QUICK RAPID FIRE

Hook	Purpose

useState	State management
useEffect	Side effects
useContext	Avoid prop drilling
useReducer	Complex state
useMemo	Memoize value
useCallback	Memoize function
useRef	DOM & persistent values