**📌 Complete Example: useState Hook**

The useState Hook allows **functional components** to have their own **local state** without using class components. It provides a **simpler and more readable** way to manage state.

**✔️ Features in this Example**

✅ **Using useState to define state inside a function**  
✅ **Updating state dynamically on button clicks**  
✅ **Using multiple useState variables to manage different states**  
✅ **Demonstrating best practices for state updates**

**📝 Counter.js (Functional Component with useState)**

This component **tracks a counter value** and updates it on button clicks.

import React, { useState } from "react";

const Counter = () => {

// Step 1: Define state variables using useState

const [count, setCount] = useState(0); // Initial state value is 0

// Step 2: Function to increment count

const increment = () => {

setCount(prevCount => prevCount + 1); // Best practice: Using previous state

};

// Step 3: Function to decrement count

const decrement = () => {

setCount(prevCount => prevCount - 1);

};

return (

<div style={styles.container}>

<h2>Counter: {count}</h2>

<button style={styles.button} onClick={increment}>

➕ Increment

</button>

<button style={styles.button} onClick={decrement}>

➖ Decrement

</button>

</div>

);

};

// Inline styles

const styles = {

container: {

textAlign: "center",

padding: "20px",

border: "1px solid #ddd",

borderRadius: "8px",

width: "200px",

margin: "20px auto",

backgroundColor: "#f9f9f9",

},

button: {

margin: "5px",

padding: "10px",

fontSize: "16px",

cursor: "pointer",

},

};

export default Counter;

**📝 App.js (Parent Component)**

This component **renders multiple Counter components** to show that each one has independent state.

import React from "react";

import Counter from "./Counter"; // Importing functional Counter component

const App = () => {

return (

<div className="App">

<h1>useState Hook Example</h1>

<Counter /> {/\* Independent Counter instance \*/}

<Counter /> {/\* Another independent Counter \*/}

</div>

);

};

export default App;

**✔️ How It Works**

1. **State Initialization (useState)**
   * const [count, setCount] = useState(0); initializes count with 0.
2. **State Updates (setCount)**
   * setCount(prevCount + 1) updates count when clicking "Increment".
   * setCount(prevCount - 1) updates count when clicking "Decrement".
3. **Reactivity**
   * When state changes, **the component automatically re-renders** with the new value.
4. **Each Instance is Independent**
   * If App.js renders multiple <Counter />, each has its **own isolated state**.

**✔️ Why Use useState Hook?**

| **Feature** | **Benefit** |
| --- | --- |
| **Simpler Code** | Eliminates the need for this.state and this.setState(). |
| **Functional Approach** | No need for class components; better readability. |
| **Independent State** | Each component maintains its own state. |
| **Easier Maintenance** | Code is more reusable and modular. |

**🚀 Real-World Use Cases for useState**

* **Form handling**: Store and update input field values dynamically.
* **Theme toggles**: Switch between light and dark modes.
* **User interactions**: Show/hide elements dynamically (e.g., modals, tooltips).
* **Counters, like/unlike buttons**: Track user actions efficiently.

Would you like an **example with multiple useState variables** (e.g., a form with multiple inputs)? 😃