**Day 18: Custom Hooks – Reusable Logic in React**

### **1. Introduction**

React provides **Hooks** (like useState, useEffect, etc.) to manage state and side effects. However, when multiple components share the same logic, repeating code in each component becomes messy.

To solve this, React gives us **Custom Hooks** — a way to **reuse logic** across components.

### **2. What are Custom Hooks?**

A **Custom Hook** is just a **JavaScript function** whose name starts with **“use”** and allows you to reuse stateful logic.

✅ **Key Points:** - Custom hooks are reusable. - They can use other hooks inside (like useState, useEffect, etc.). - They **don’t return JSX**; they return data, functions, or objects.

### **3. Why We Use Custom Hooks**

| Problem | Solution |
| --- | --- |
| Repeated logic in multiple components (e.g., fetching data, handling forms) | Write it once in a custom hook and reuse |
| Complex components become hard to read | Move logic out to a custom hook |
| Need separation of concerns | Keep logic and UI separate |

### **4. When to Use Custom Hooks**

You use them when: - The same logic (like fetching data, tracking window size, managing input state, etc.) appears in multiple places. - You want to make code cleaner, modular, and easier to maintain.

### **5. Syntax of a Custom Hook**

// Example: useCounter.js  
import { useState } from 'react';  
  
function useCounter(initialValue = 0) {  
 const [count, setCount] = useState(initialValue);  
  
 const increment = () => setCount(count + 1);  
 const decrement = () => setCount(count - 1);  
 const reset = () => setCount(0);  
  
 return { count, increment, decrement, reset }; // ✅ Returning values and functions  
}  
  
export default useCounter;

### **6. Using Custom Hook in a Component**

// CounterApp.js  
import useCounter from './useCounter';  
  
function CounterApp() {  
 const { count, increment, decrement, reset } = useCounter(0); // ✅ Using custom hook  
  
 return (  
 <div style={{ textAlign: 'center' }}>  
 <h1>Count: {count}</h1>  
 <button onClick={increment}>Increase</button>  
 <button onClick={decrement}>Decrease</button>  
 <button onClick={reset}>Reset</button>  
 </div>  
 );  
}  
  
export default CounterApp;

✅ **Explanation:** - The logic for counting is now reusable in any component. - We can use the same hook in 10 different components without rewriting the code.

### **7. Example 2: Custom Hook for Fetching Data**

// useFetch.js  
import { useState, useEffect } from 'react';  
  
function useFetch(url) {  
 const [data, setData] = useState(null);  
 const [loading, setLoading] = useState(true);  
 const [error, setError] = useState(null);  
  
 useEffect(() => {  
 fetch(url)  
 .then((res) => {  
 if (!res.ok) throw new Error('Network error');  
 return res.json();  
 })  
 .then((data) => setData(data))  
 .catch((err) => setError(err.message))  
 .finally(() => setLoading(false));  
 }, [url]);  
  
 return { data, loading, error }; // ✅ Return state  
}  
  
export default useFetch;

// App.js  
import useFetch from './useFetch';  
  
function App() {  
 const { data, loading, error } = useFetch('https://jsonplaceholder.typicode.com/users');  
  
 if (loading) return <h3>Loading...</h3>;  
 if (error) return <h3>Error: {error}</h3>;  
  
 return (  
 <ul>  
 {data.map((user) => (  
 <li key={user.id}>{user.name}</li>  
 ))}  
 </ul>  
 );  
}  
  
export default App;

✅ **Explanation:** - The logic for fetching data is isolated inside useFetch. - You can reuse useFetch() anywhere in your project just by passing a different URL.

### **8. Rules of Custom Hooks**

* Must start with **use** (like useForm, useFetch, etc.).
* Can call other hooks inside.
* Must be used **inside a React component or another hook** (not inside loops, conditions, or nested functions).

### **9. Advantages of Custom Hooks**

* Code reusability
* Better readability
* Cleaner separation between logic and UI
* Easier testing and debugging

### **10. Exercise for Day 18**

🧠 **Task:** Create a **custom hook** called useLocalStorage that: - Saves data to localStorage. - Retrieves the data when the app loads. - Automatically updates localStorage when data changes.

**Example Output:**

const [name, setName] = useLocalStorage('username', 'Rehan');  
  
return (  
 <div>  
 <input value={name} onChange={(e) => setName(e.target.value)} />  
 <p>Stored Name: {name}</p>  
 </div>  
);

✅ **Hint:** You will use useState, useEffect, and localStorage.getItem() and setItem() inside your hook.

### **Summary:**

* Custom hooks = reusable logic.
* They make components cleaner.
* You can use them for counters, forms, data fetching, localStorage, etc.
* Follow hook naming conventions and React rules.