**🧩 1️⃣ What is a “Lifecycle” in React?**

Every React component — whether **class** or **functional** —  
**goes through a life cycle**:

🍼 It’s *created* → 💼 *updated* when data changes → ⚰️ *removed* when no longer needed.

So we call these stages:

1. **Mounting** — component is created and added to the DOM
2. **Updating** — when props or state change
3. **Unmounting** — when the component is removed from the DOM

**🧠 2️⃣ In Class Components (Old Way)**

Before hooks existed, React handled lifecycle using **special methods**:

| **Phase** | **Method** | **Description** |
| --- | --- | --- |
| Mounting | componentDidMount() | Runs once after the component appears on screen. (✅ like calling API once) |
| Updating | componentDidUpdate() | Runs whenever props or state change. |
| Unmounting | componentWillUnmount() | Runs when component is removed. (❌ cleanups) |

**📦 Example (Class Component)**

class ProductList extends React.Component {

state = { products: [] };

componentDidMount() {

// Mounting - API call

console.log("Mounted");

this.fetchData();

}

componentDidUpdate(prevProps, prevState) {

// Updating - State or prop change

console.log("Updated");

}

componentWillUnmount() {

// Unmounting - Cleanup

console.log("Unmounted");

}

fetchData() { /\* API logic \*/ }

render() {

return <div>Products go here...</div>;

}

}

**⚙️ 3️⃣ In Functional Components (Modern Way)**

Now in modern React (using **Hooks**),  
we use **useEffect** to represent *all lifecycle stages in one place*.

Let’s map them side-by-side 👇

| **Lifecycle Phase** | **Class Component** | **Functional Component** |
| --- | --- | --- |
| Mount (first render) | componentDidMount | useEffect(() => { ... }, []) |
| Update (state/prop change) | componentDidUpdate | useEffect(() => { ... }, [dep]) |
| Unmount (cleanup) | componentWillUnmount | useEffect(() => { return () => {...}; }, []) |

**🧩 Example (Functional Way)**

import React, { useState, useEffect } from "react";

function ProductList() {

const [products, setProducts] = useState([]);

// 🟢 Mounting + Updating

useEffect(() => {

console.log("Mounted or Updated");

fetchProducts();

// 🔴 Unmounting (cleanup)

return () => {

console.log("Component Unmounted");

};

}, []); // empty [] means only on mount/unmount

const fetchProducts = async () => {

const res = await fetch("/api/products");

const data = await res.json();

setProducts(data);

};

return <div>{products.length} products loaded</div>;

}

**🔍 4️⃣ How to Understand it in Real Life**

Let’s say you’re building a **chat app** 💬  
Each chat window (component) has a lifecycle:

| **Stage** | **What Happens** | **Hook** |
| --- | --- | --- |
| Mount | Connect to WebSocket (start listening) | useEffect(() => connect(), []) |
| Update | New message arrives or user types | React automatically re-renders |
| Unmount | Disconnect from WebSocket (cleanup) | return () => disconnect() |

So useEffect helps manage what happens in each stage.

**🧘‍♀️ 5️⃣ Key Points to Remember**

✅ React lifecycle = Mount → Update → Unmount  
✅ Functional components use useEffect for all lifecycle logic  
✅ Always use cleanup functions to stop timers, listeners, or API subscriptions  
✅ Class lifecycle methods are older — you’ll still see them in legacy codebases

**💬 In one simple line:**

The React component lifecycle is like a human life — it’s born (mounted), it changes (updated), and it dies (unmounted).  
useEffect is how you tell React *what to do* at each of these stages.