**useEffect**

🔗 Connect with Me:  
🌐 LinkedIn : https://www.linkedin.com/in/priya-bagde  
📂 GitHub : https://github.com/priya42bagde  
💻 LeetCode : https://leetcode.com/priya42bagde/  
🎥 YouTube Channel : https://youtube.com/channel/UCK1\_Op30\_pZ1zBs9l3HNyBw (Priya Frontend Vlogz)

* To handle **side effects in your functional components**. Side effects are **any operations that interact with the outside of the component scope**, such as **data fetching, subscriptions, timers, console logging, or manually changing the DOM**. We have handled **inside lifecycle methods in class components.**
* When we want **to perform something after each render of component** then we can use the useEffect() hook. By using this Hook, we tell React that our component **needs to do something after render by passing a function**. React remember the function we passed in useEffect() hook and call it later **after performing the DOM updates.**
* **NOTE:** It is not recommended to define a function outside and call it inside an effect.
* It’s a combination of 3 lifecycle methods i.e,

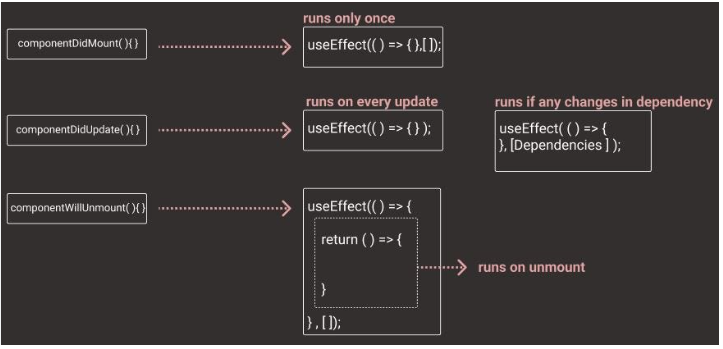
**after a component mounts (componentDidMount),**

**after its re-renders (componentDidUpdate),** and

**before it unmounts (componentWillUnmount).**

**Controlling side effects in useEffect:**

1. To run useEffect only once on the **first render** pass any **empty array** in the dependency.
2. To run useEffect on **every render** **do not pass** any dependency.
3. To run useEffect on **change of a particular value**. Pass the **state and props in the dependency array.**



**Use Cases of React useEffect Hooks:**

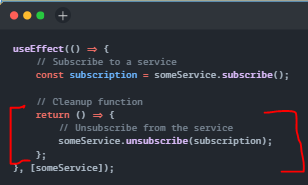
* To add a button's event listener
* To **fetch data when a component mount**
* To **run code when state changes or prop changes**
* To **set up timers or intervals.**
* To **clean up event listeners** during the time of the component **unmounts**

**Syntax / How does it work?**

1. We call **useEffect with a callback function** that **contains the side effect logic.**
2. By default, this function **runs after initial and every render of the component**.
3. You can **optionally provide** a second argument i.e, an array of dependencies values. After **rendering finishes**, useEffect will **check the list of dependency values** against the values from the last render and **will call effect function if any one of them has changed**.
4. The effect will **only run again if** any of the values in the dependency array change.
5. If **Cleanup Function provided**, runs **before the next effect** or **during component unmounting**, facilitating cleanup tasks such as unsubscribing.

**Cleanup Function in useEffect:** We need to **clean up some resources before the component unmounts** by returning a cleanup function from your effect. **When the component unmounts, or before the next time the effect runs, this cleanup function gets executed**. For example,

* If your effect **subscribes** to a service, you want to **unsubscribe** when the component unmounts to avoid memory leaks.
* If you have a **countdown timer** using the **setInterval** function, that interval will not stop unless we use the **clearInterval** function. A timer managed with setInterval() and clearInterval().
* An **event subscription** using **window.addEventListener()** and **window.removeEventListener().**



**How can I run an effect only on mount and unmount?**

To run an effect only when the component mounts and when it unmounts, you can **pass an empty array [] as the second argument** to useEffect. This signifies that the effect doesn't depend on any values and should only run on mount and cleanup on unmount.

**Why is it “unmounting” with every render?**

Well, the cleanup function you can (optionally) return from useEffect **isn’t only called when the component is unmounted. It’s called every time before that effect runs – to clean up from the last run.** This is actually more powerful than the componentWillUnmount lifecycle because it lets you **run a side effect before and after every render**, if you need to.