**useCallback**

🔗 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)

## What is useCallback hook:

* The useCallback Hook returns **a memoized callback function**.
* It allow us **to cache a function definition** and it **does not get redefined on every render**.
* It will **not** **run on every render automatically**. This will **optimize and improve the overall performan**ce of your application.

## When to use the useCallback hook:

* When you need **to pass a function as props to a child component**.
* If you have a **function that is expensive to compute** and you need to call it in **multiple places**.
* When dealing with **functional components**.

## Benefits of using the useCallback hook:

* **Performance optimization**: This hook optimizes the performance of your application by **preventing a series of unnecessary re-rendering** in your components.
* **Restricting rendering of child components**: The useCallback hook in React allows us to **selectively render important child components in a parent component**. By using the useCallback hook, we can create memoized functions and pass them as **props** to child components. This ensures that only **the necessary child components are rendered** and updated when specific actions occur, resulting in improved performance.
* **Preventing memory leaks**: Since the hook returns the memoized function, it **prevents recreating functions**, which can lead to memory leaks.

## Drawbacks of the useCallback hook:

* **Complex code**: Only use the hook only when you need to **memoize an expensive function which needs to be passed down to children components as a prop**, otherwise, it will create a complex code structure too.
* **Excessive memory usage**: If you do not use the useCallbck hook properly, it can lead to excessive memory usage. For instance, if a memoized function holds onto **references to objects or variables that are no longer needed**, those resources may not be **freed up by garbage collection and could use more memory** than needed.

**The useCallback syntax:**

It takes **two arguments**: the **function you want to memoize**, and **the dependencies array**. i.e,useCallback(function, dependencies).

**Returns:**

On the **initial render**, useCallback returns **the function you have passed**.

During **subsequent renders**, it will either return an already **stored function** from the last render (if the dependencies haven’t changed), or return the **function you have passed during the current render**.

**Referential equality:**

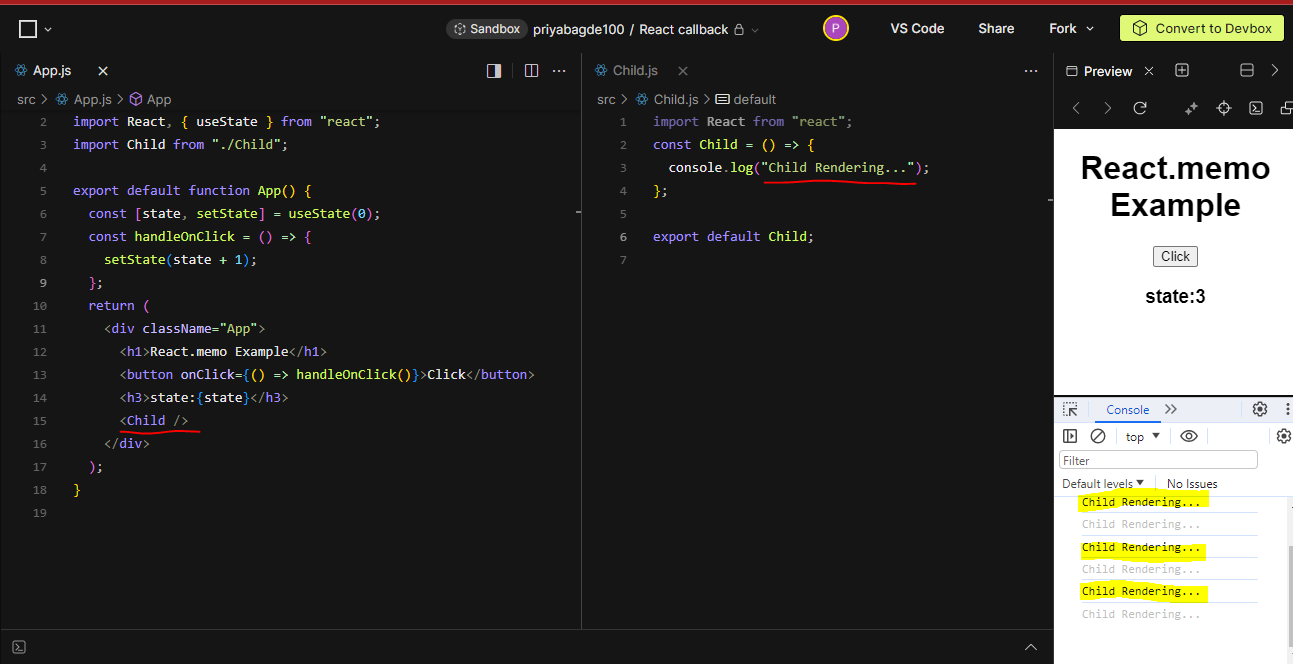
When a **component re-renders**, **every function (like handleOnClick/ handleOnChange) inside of that component is recreated** and therefore these functions’(i.e, objects) **references change between renders** and all the **deeply nested child components get call unnecessarily**. **Using useCallback, instead of recreating the function object on every re-render, we can use the same function object between renders**.

**useMemo vs useCallback:**

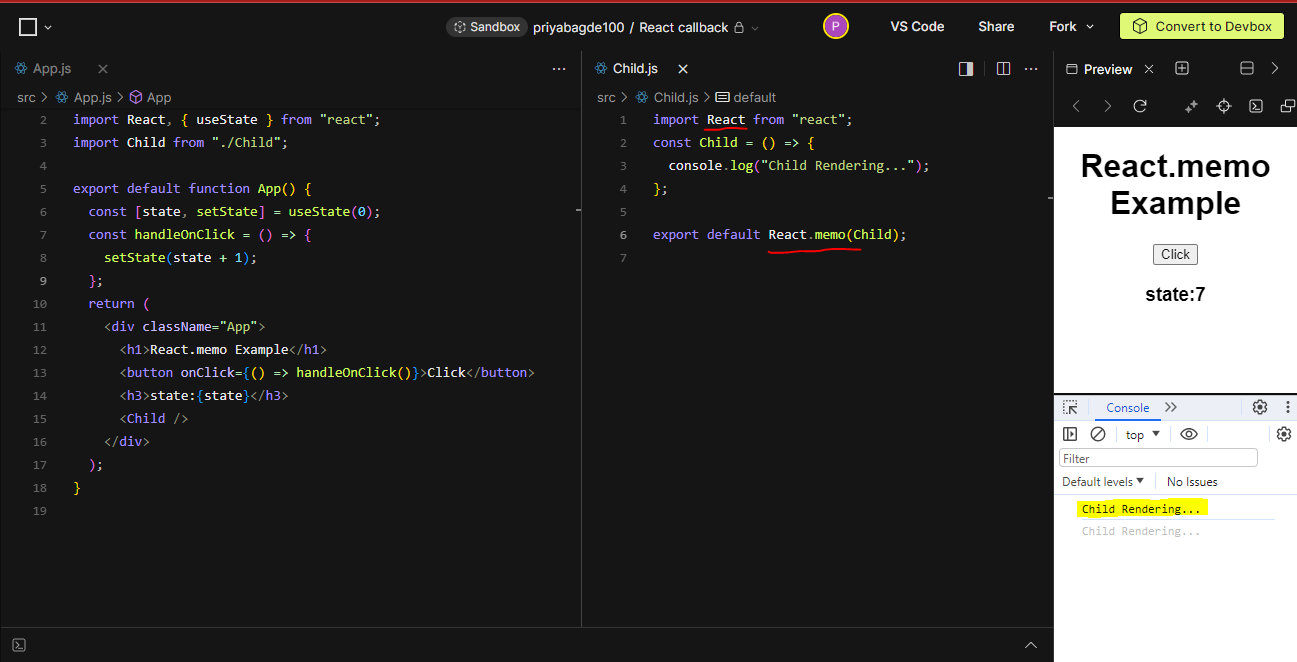
The main difference is that useMemo returns **a memoized value** and useCallback returns **a memoized function**.

**Example:**

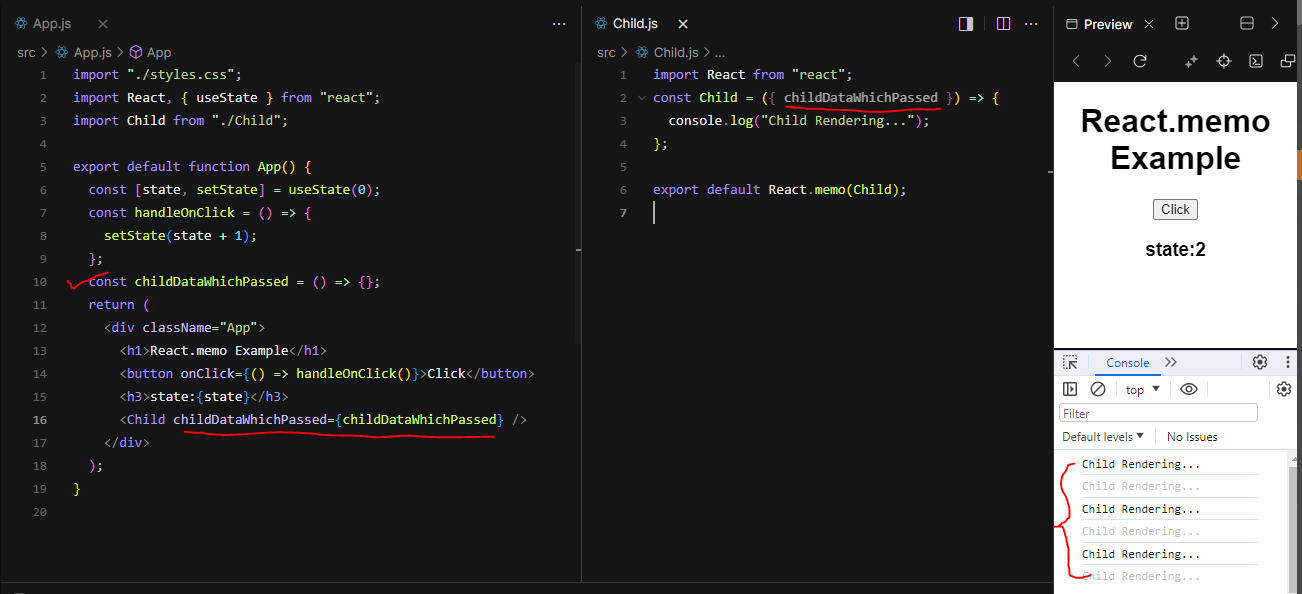
Suppose we have a **counter at parent component** and inside a **child component(we just have a console.log())**. When we **increment the counter at parent** as state is not used at child component so **why is the child component get re-render unnecessarily**. To avoid this case, for that we have **to wrap the child component by React.memo(Child).**



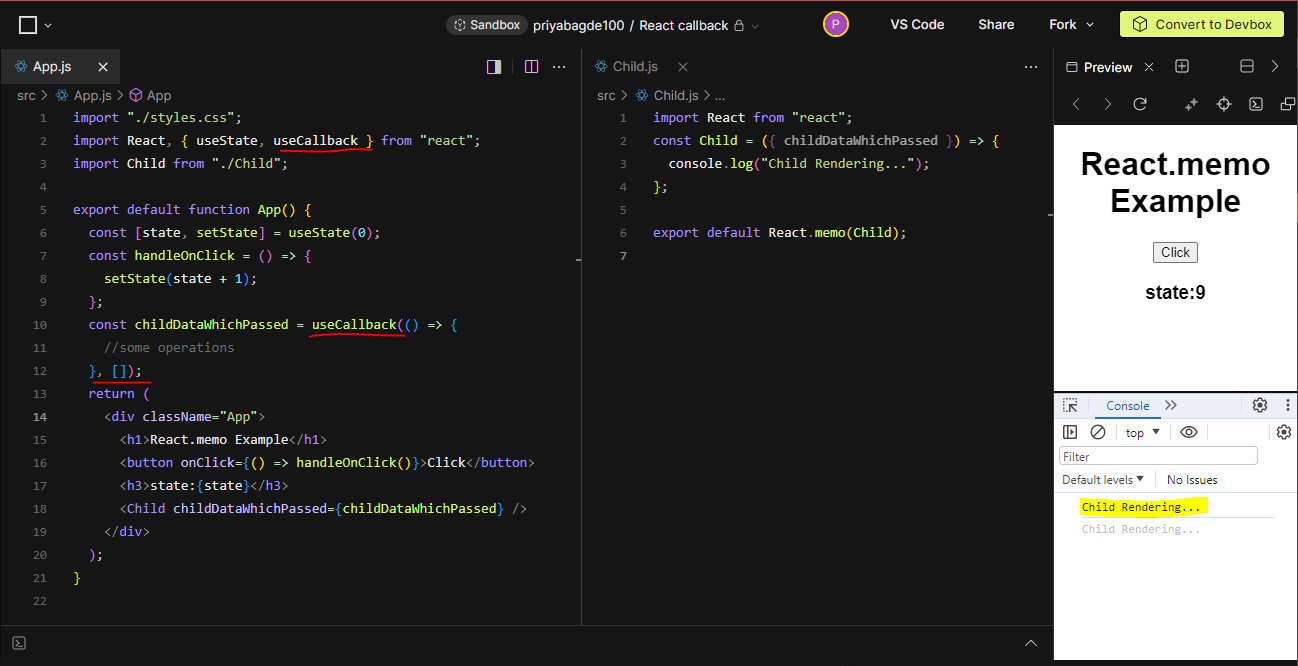
With React.memo(), it is **rendering at initial mounting phase that is as expected** but **you won’t see at subsequent rendering**.



When **we pass a props to the child component,** then **React.memo() won’t work and child component get started to re-render again**. This is due to **referential equality**. When the **component rerender** then **function is also recreated**, then the **child component think it’s recreated** means **something get change** so the **Child Component started to rerender**.



To avoid this, **we can use a useCallback**. To render at **one time so we passed an empty array** as a dependency:



When we want **to render the Child component based on certain conditions/dependencies** then you can pass it like below.

