7. callBack, Memo, Option

Tuesday, 23 December 2025 11:24 AM

🧠 Simple Mental Model

- options.push() → **adds item to array**
- useCallback() → **don't recreate function**
- useMemo() → **don't recalculate value**

These three come from **different levels** of JavaScript / React. I'll explain them **clearly**, with **simple mental models**, **examples**, and **when to use / not use**.

1️⃣ options.push() (Plain JavaScript)

What it is

push() is an **Array method** in JavaScript.
It **adds an item to the end of an array** and **mutates the array**.

```
const options = ["A", "B"];
options.push("C");
console.log(options); // ["A", "B", "C"]
```

Important things to know

- ✅ Changes the **original array**
- ✅ Returns the **new length** of the array
- ❌ Not recommended directly in **React state**

```
const arr = [];
const length = arr.push(10);
console.log(length); // 1
console.log(arr);    // [10]
```

⚠️ In React (common mistake)

```
// ❌ WRONG (mutates state)
options.push("C");
setOptions(options);
```

✅ Correct React way

```
setOptions(prev => [...prev, "C"]);
```

📌 **Rule**:
In React → avoid .push() on state arrays

2️⃣ useCallback (React Hook)

What it is

useCallback **memoizes a function** so React does **not recreate it on every render**.

Why this exists

In React:

- Components re-render
- Functions are **re-created every render**
- This can cause **unnecessary child re-renders**

Without useCallback

```
function Parent() {
  const handleClick = () => {
    console.log("Clicked");
  };
return <Child onClick={handleClick} />;
}
```

}

👉 handleClick is **new every render**

With useCallback

```
const handleClick = useCallback(() => {
  console.log("Clicked");
}, []);
```

👉 Same function reference between renders

When to use useCallback

Use it **ONLY when**:

- Passing functions to React.memo components
- Functions are dependencies in useEffect
- Preventing unnecessary re-renders

❌ Don't overuse

```
// ❌ pointless
const value = useCallback(() => count + 1, [count]);
```

📌 **Rule**

useCallback = performance optimization, not default behavior

3️⃣ useMemo (React Hook)

What it is

useMemo **memoizes a computed value**, not a function.

Why this exists

- Expensive calculations
- Avoid recalculating on every render

Without useMemo

```
const expensiveValue = slowFunction(count);
```

👉 Runs on **every render**

With useMemo

```
const expensiveValue = useMemo(() => {
  return slowFunction(count);
}, [count]);
```

👉 Runs **only when count changes**

Example

```
const doubled = useMemo(() => count * 2, [count]);
```

When to use useMemo

Use it when:

- Heavy calculations
- Derived data
- Prevent unnecessary recalculations

❌ Don't overuse

```
// ❌ unnecessary
const name = useMemo(() => "John", []);
```

📌 **Rule**

useMemo saves CPU, not memory

🔥 useCallback vs useMemo (Very Important)

Feature	useCallback	useMemo
Memoizes	Function	Value
Returns	Function	Computed value
Use case	Prevent re-creation of functions	Prevent expensive calculations
Syntax	useCallback(fn, deps)	useMemo(() => value, deps)

Relationship

```
useCallback(fn, deps)
// is same as
useMemo(() => fn, deps)
```

✅ Common Interview Summary

- .push() mutates array → avoid in React state
- useCallback memoizes functions
- useMemo memoizes values
- Both are **performance optimizations**, not required everywhere

If you want, I can next:

- Explain **real-world use cases**
- Show **bad vs good performance examples**
- Explain **React.memo + useCallback combo**
- Explain **when NOT to use these hooks**

Just tell me 👍