# useMemo hook in React

useMemo is a React hook that **memoizes** (remembers) the result of a **function** so that it's only recomputed when its dependencies change.

In React is used to **memoize**, **or cache**, the result of an expensive calculation or a value between re-renders of a functional component. It helps **optimize performance** by **avoiding unnecessary recalculations**.

**Syntax:**

```
const Value = useMemo(()=>computeFunction(),[dependencies]);
```

**Parameters:**

- **Callback Function** → A function whose return value will be memoized.
- **Dependency Array** → Values that trigger recomputation when they change.

## Returns:

Returns the **memoized** result of the function.

## Use Case:

- Heavy calculations (e.g., filtering, sorting large lists).
- Avoid recalculating derived data on each render.

## Benefits:

- Prevents **performance issues** by **memoizing expensive computations**.
- Reduces **unnecessary re-renders** in child components (when used with React.memo).

## When NOT to use:

- Don't use for **simple values** or **small computations**.
- Overusing useMemo may make code harder to read with **no real gain**.

## Advantages of useMemo Hook:

- **Improves Performance**
  - Avoids unnecessary recalculations of expensive operations (like filtering, sorting, or computation).
- **Optimizes Rendering**
  - Prevents child components from re-rendering if their props are derived from memoized values.
- **Efficient Memory Usage**
  - Stores computed values in memory only when needed (based on dependencies).
- **Works Well with React.memo**
  - Helps avoid re-rendering child components when memoized props haven’t changed.
- **Useful for Derived Data**
  - Ideal for computing values from state/props that don't need to be recalculated every time.

- useCallback returns a **memoized version of a function** that only changes if its dependencies change.
  Used to **prevent unnecessary function re-creation** on every render.
- Its primary purpose is to optimize the performance of React applications by preventing unnecessary re-creations of functions, particularly when those functions are passed as props to child components.

**Syntax**

```
const memoCallback = useCallback(() => {
                                    // Function logic
                                }, [dependencies]);
```

## Use Case:

- Pass memoized functions to child components (especially with React.memo) to avoid re-renders.
- Optimize performance when functions are used inside useEffect, useMemo, or passed as props.

## Advantages:

- Prevents **unnecessary re-renders** of child components.
- Useful when passing **functions as props** to memoized components.
- Helps with **performance optimization** in large apps.

| Hook | Purpose | Returns |
|---|---|---|
| useMemo | Memoize **value** | A **value** |
| useCallback | Memoize **function** | A **function** |