1️⃣ What is useCallback?

Definition: `useCallback` is a React hook that memoizes a function. - Returns a stable function reference between renders as long as dependencies don't change. - Useful for avoiding unnecessary re-renders of child components and stale closures.

Syntax:

```
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

- Returns a memoized function - Recreates the function only when dependencies change

2️⃣ Why useCallback Exists

- Functions in React are recreated on every render:

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

- On every render, `handleClick` is a new function → <Child> may re-render unnecessarily
- `useCallback` returns a **stable function reference**, preventing unnecessary renders and stale closures

3️⃣ Internal Working

- Stored in the fiber hook list, similar to `useMemo`
- On render:
- React compares dependency array with previous render
- If dependencies didn't change → returns same function reference
- If dependencies changed → creates a new function
- Equivalent to:

  ```
  useCallback(fn, deps) === useMemo(() => fn, deps)
  ```

4️⃣ When to Use useCallback

1️⃣ Prevent unnecessary child re-renders

```
const handleClick = useCallback(() => {
  setCount(prev => prev + 1);
}, []);
<Child onClick={handleClick} /> // stable reference
```

- Works with `React.memo(Child)` → avoids re-render unless props change

2️⃣ Prevent stale closures in effects / timers

```
const savedCallback = useRef();
useEffect(() => { savedCallback.current = callback; }, [callback]);
useEffect(() => {
  const tick = () => savedCallback.current();
  const id = setInterval(tick, delay);
  return () => clearInterval(id);
}, [delay]);
```

3️⃣ **Dynamic event handlers in large components** - Memoizing callbacks prevents recreating hundreds of handlers each render → improves performance

---

5️⃣ Difference Between useCallback and useMemo

Feature	useCallback	useMemo
Returns	Function	Value
Purpose	Stable function reference	Memoized value
Usage	Passing to children, event handlers, effect dependencies	Expensive computation, stable object/array
Dependency	[deps]	[deps]

Trick: `useCallback(fn, deps)` is equivalent to `useMemo(() => fn, deps)`

6️⃣ Gotchas

- Overusing `useCallback` increases complexity

- Only useful for child component re-render prevention or stale closures
- Dependencies must be correct:

```
const callback = useCallback(() => console.log(a, b), [a]); // ❌
missing b
```

- Does **not** prevent re-render by itself → combine with `React.memo` for child optimization
- Reference equality matters:

```
const memoizedFn = useCallback(() => {}, []);
<Child onClick={memoizedFn} /> // stable
```

- Without `useCallback`, new function every render → child re-renders

7️⃣Production Patterns

1️⃣Child component optimization

```
const Child = React.memo(({ onClick }) => {
  console.log("Child rendered");
  return <button onClick={onClick}>Click</button>;
});

function Parent() {
  const [count, setCount] = useState(0);
  const handleClick = useCallback(() => {
    setCount(prev => prev + 1);
  }, []);
  return <Child onClick={handleClick} />;
}
```

- Child only re-renders if props change

2️⃣Memoized callbacks for dynamic subscriptions

```
function useInterval(callback, delay) {
  const savedCallback = useRef();
  useEffect(() => { savedCallback.current = callback }, [callback]);
  useEffect(() => {
    const tick = () => savedCallback.current();
    const id = setInterval(tick, delay);
    return () => clearInterval(id);
  }, [delay]);
}
```

- Avoids stale closure problem in intervals

3️⃣Avoid inline functions in props

```
<Child onClick={() => setCount(count + 1)} /> // ❌ new function every render
<Child onClick={handleClick} /> // ✅ stable reference with useCallback
```

8️⃣Senior-Level Mental Model

- `useCallback` = stable function reference between renders
- Dependencies ensure correct value inside function
- Works best for:
- Prevent unnecessary child re-renders
- Avoid stale closures in effects / timers
- Stable function references for memoized props
- Not needed for trivial inline handlers or small apps

9️⃣Summary Table

Use Case	Without useCallback	With useCallback
Passing function to React.memo child	Child re-renders every parent render	Child re-renders only when deps change
Timer / interval	May use stale closure	Stable callback with current state
Inline handler in JSX	New function every render	Stable reference
Optimization	❌	✅for large apps / expensive components

✅**Key Takeaways:** - `useCallback` memoizes functions - Use with `React.memo` for child render optimization - Correct dependencies = essential - Avoid overuse for trivial cases - Solves stale closure issues in effects and intervals - Works hand-in-hand with `useMemo` and `useRef` in advanced patterns