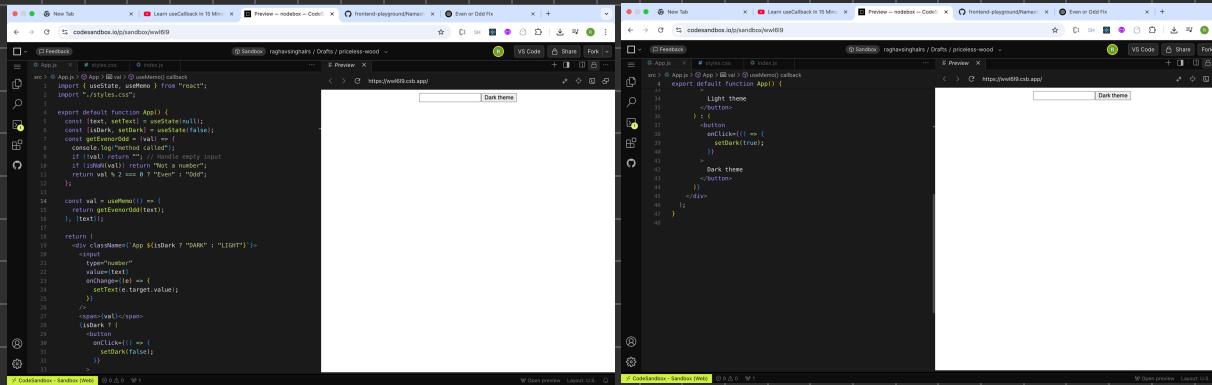




## 1. Use memo

As how many state variables we are using whenever it updates, whole component re-renders so using we memo we can establish a situation where whenever input changes then the calculation rehappens otherwise uses the old cache.



The screenshot shows two browser windows side-by-side. Both windows have the URL <https://www123.csb.app/>. The left window displays a light theme with a white background and black text. The right window displays a dark theme with a black background and white text. The code being run is a React component named App.js, which contains logic for determining if a number is even or odd and styling buttons based on the result.

```
App.js
1 // App.js # App > # App > val > useState callback
2 // App.js # App > # App > useState from "react"
3 // import {useState} from "react";
4
5 export default function App() {
6   const [text, setText] = useState("null");
7   const getEvenOdd = (val) => {
8     console.log("method called");
9     if (val === null) return "Please input a number";
10    if (!Number(val)) return "Not a number";
11    return val % 2 === 0 ? "Even" : "Odd";
12  };
13
14 const val = useState(() => {
15   return getEvenOdd(text);
16 }, [text]);
17
18 return (
19   <div className="App" style={isDark ? "DARK" : "LIGHT"}>
20     <input type="number" value={text}>
21     <button onClick={() => {
22       setText(target.value);
23     }}>
24       Set
25     </button>
26     <span>{val}</span>
27     <button onClick={() => {
28       setDark(!isDark);
29     }}>
30       Dark these
31     </button>
32   </div>
33 )
34
35 <Light theme>
36   <button>
37     Click me!
38   <button onClick={() => {
39     setDark(true);
40   }}>
41     Set dark
42   </button>
43 </Light theme>
44 <Dark theme>
45   <button>
46     Click me!
47   <button onClick={() => {
48     setDark(false);
49   }}>
50     Set light
51   </button>
52 </Dark theme>
53 )
```

## 2. Use useCallback

It is used to memoize a function so that its reference stays the same between renders unless dependencies change.

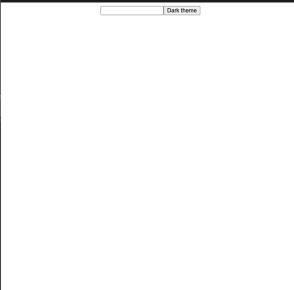
Some usecases:

When you pass a function as a prop to child components and want to avoid recreating it on every render.

Summary →  
useCallback  
useMemo

getEvenOdd  
getEvenOdd(text)

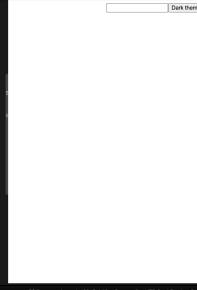
Keeps function ref stable  
Avoids recalculating unless needed.



```
const [isDark, setIsDark] = useState(false);

function App() {
  return (
    <div>
      <input type="checkbox" onClick={() => setIsDark(!isDark)} />
      <span>Dark theme</span>
    </div>
  );
}

export default App;
```



```
const [isDark, setIsDark] = useState(true);

function App() {
  return (
    <div>
      <input type="checkbox" onClick={() => setIsDark(!isDark)} />
      <span>Light theme</span>
    </div>
  );
}

export default App;
```

### 3. UseReducer

- It is used for state management.
- Alternative of useState hook.
- Preferable for complex state management logic, when state transitions depend on actions.

Syntax

```
const [state, dispatch] = useReducer(reducer, initialState);
```

reducer(currentState, action)

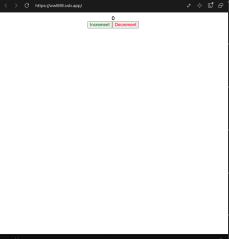
When use useReducer →

- \* When you have multiple related state updates.
- \* When the next state depends on the previous state.

Basic terminology :-

Reducer function → a function that decides how to update the state based on an action.

Dispatch function  $\Rightarrow$  used to send ("dispatch") an action to the reducer.



```
4 https://stackblitz.com/edit/react-1vqjwz?file=src%2Findex.js
T stackblitz.com https://stackblitz.com/edit/react-1vqjwz?file=src%2Findex.js
src/index.js
import React from 'react';
import ReactDOM from 'react-dom';
const App = () => {
  const [state, dispatch] = useState(0);
  const reducer = (state, action) => {
    switch (action.type) {
      case 'increment':
        return state + 1;
      case 'decrement':
        return state - 1;
      default:
        return state;
    }
  };
  if (state === 0) {
    const count = dispatch(useReducer(reducer, initialstate));
    return (
      <div>
        <h1>{state}</h1>
        <button onClick={()=> dispatch('increment')}>increment</button>
        <button onClick={()=> dispatch('decrement')}>decrement</button>
      </div>
    );
  } else {
    const count = dispatch(useReducer(reducer, initialstate));
    return (
      <div>
        <h1>{state}</h1>
        <button onClick={()=> dispatch('increment')}>increment</button>
        <button onClick={()=> dispatch('decrement')}>decrement</button>
      </div>
    );
  }
}
document
<div><App/></div>
```



```
5 https://stackblitz.com/edit/react-1vqjwz?file=src%2Findex.js
T stackblitz.com https://stackblitz.com/edit/react-1vqjwz?file=src%2Findex.js
src/index.js
import React from 'react';
import ReactDOM from 'react-dom';
const App = () => {
  const [state, dispatch] = useState(0);
  const reducer = (state, action) => {
    switch (action.type) {
      case 'increment':
        return state + 1;
      case 'decrement':
        return state - 1;
      default:
        return state;
    }
  };
  if (state === 0) {
    const count = dispatch(useReducer(reducer, initialstate));
    return (
      <div>
        <h1>{state}</h1>
        <button>increment</button>
        <button>decrement</button>
      </div>
    );
  } else {
    const count = dispatch(useReducer(reducer, initialstate));
    return (
      <div>
        <h1>{state}</h1>
        <button>increment</button>
        <button>decrement</button>
      </div>
    );
  }
}
document
<div><App/></div>
```