React Hooks Example: Explanation and Code

Functional Component: `ExampleComponent`

The `ExampleComponent` demonstrates the usage of various React hooks such as `useState`, `useEffect`, `useMemo`, `useCallback`, and `useRef`.

Code Example

```
function ExampleComponent() {
  // **State Hook (`useState`)**:
  const [count, setCount] = useState(0);

  // **Effect Hook (`useEffect`)**:
  useEffect(() => {
    console.log("Component rendered or count changed:", count);

    return () => {
      console.log("Cleanup before count change or unmount");
    };
  }, [count]);

  // **Memoization Hook (`useMemo`)**:
  const expensiveValue = useMemo(() => {
    console.log("Expensive calculation is running...");
    return count * 1000;
  }, [count]);

  // **Callback Memoization (`useCallback`)**:
  const increment = useCallback(() => {
    setCount(count + 1);
  }, [count]);

  // **Reference Hook (`useRef`)**:
  const inputRef = useRef(null);

  const focusInput = () => {
    inputRef.current.focus();
  };

  // JSX Returned
  return (
    <div>
      <p>Expensive Calculation: {expensiveValue}</p>
      <p>Current Count: {count}</p>
      <button onClick={increment}>Increment Count</button>
      <input ref={inputRef} placeholder="Click the button to focus me!" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
}
export default ExampleComponent;
```

Detailed Explanation of Hooks

1. `useState`

- Manages local state and triggers re-renders when updated.
- Example:

  ```
  const [count, setCount] = useState(0);
  ```

2. `useEffect`

- Mimics lifecycle methods like `componentDidMount` and `componentDidUpdate`.
- Includes cleanup logic to prevent memory leaks.
- Example:

  ```
  useEffect(() => {
    console.log("Effect triggered");

    return () => {
      console.log("Cleanup logic");
    };
  }, [dependency]);
  ```

3. `useMemo`

- Optimizes performance by memoizing expensive calculations.
- Example:

  ```
  const value = useMemo(() => expensiveComputation(), [dependency]);
  ```

4. `useCallback`

- Memoizes functions to prevent unnecessary re-creations on every render.
- Example:

  ```
  const callback = useCallback(() => {
    // Logic here
  }, [dependency]);
  ```

5. `useRef`

- Stores references to DOM elements or values that persist across renders without causing re-renders.
- Example:

```
const inputRef = useRef(null);
inputRef.current.focus();
```

Key Concepts Covered:

1. **State Management**: Using `useState` to manage local state.
2. **Side Effects**: Using `useEffect` for lifecycle-like behaviors.
3. **Performance Optimization**: Using `useMemo` and `useCallback` to avoid unnecessary recalculations and re-creations.
4. **DOM Interaction**: Using `useRef` to interact directly with DOM elements.