**Exercise 1: Implement componentDidMount**

Create a React component that fetches data from an API when the component is first mounted and displays the data in a list format.

**Exercise 2: Use componentDidUpdate**

Create a counter component that logs a message to the console whenever the count value is updated.

**Exercise 3: Cleanup with componentWillUnmount**

Create a component that sets up an interval to update a timestamp every second and clears the interval when the component is unmounted.

**Exercise 4: Use getDerivedStateFromProps**

Create a component that updates its state based on changes in its props.

**Exercise 5: Use shouldComponentUpdate**

Create a component that re-renders only if the value of the text prop changes.

**Exercise 6: Advanced Exercise: Combine Lifecycle Methods**

Create a component that:

1. Fetches data when mounted.
2. Logs updates whenever the props or state change.
3. Cleans up any subscriptions when unmounted.

**Exercise 7: Fetch Data with useEffect (Equivalent to componentDidMount)**

Create a functional component that fetches data when it mounts using the useEffect hook.

**Exercise 8: Cleanup with useEffect (Equivalent to componentWillUnmount)**

Create a functional component that sets up an interval to update a timestamp and cleans it up when the component is unmounted.

**Exercise 9: Conditional Re-Rendering with Hooks**

Create a functional component that only updates the title of the page (document.title) when a counter is incremented.

**Exercise 10: Simulating shouldComponentUpdate with React.memo**

Create a functional component that prevents unnecessary re-renders using React.memo.

**Exercise 11: Combining Lifecycle Methods with Hooks**

Create a functional component that:

1. Fetches data on mount.
2. Logs updates to the state.
3. Cleans up subscriptions when unmounted.

**Exercise 12: Implementing getSnapshotBeforeUpdate**

Create a component that displays a chat box. When new messages are added, the component scrolls to the bottom to display the latest message.

**Exercise 13: Using componentDidCatch for Error Boundaries**

Create a parent component that catches errors in a child component and displays a fallback UI.

**Exercise 14: Prevent Re-Renders with shouldComponentUpdate**

Create a component that displays user data but only re-renders when the user ID changes.

**Exercise 15: Using componentWillUnmount for Cleanup**

Create a countdown timer component that clears its interval when the component is unmounted.

**Exercise 16: Fetch Data with componentDidMount and Update UI**

Create a component that fetches a list of posts from a simulated API and displays them. Add a button to refresh the list.

**Exercise 17: Conditionally Fetch Data with componentDidUpdate**

Create a component that fetches data based on a selected category and updates the display when the category changes. Ensure the component does not fetch data unnecessarily.

**Exercise 18: Timer with componentWillUnmount for Cleanup**

Create a stopwatch component that starts counting when mounted and stops counting when unmounted. The parent component should control the mounting and unmounting of the stopwatch.

**Exercise 19: Dynamic Theme Changer**

Create a component that changes its background color when the theme (dark or light) is toggled. Use componentDidUpdate to log theme changes to the console.

**Exercise 20: Lazy Loading with componentDidMount**

Create a component that simulates lazy-loading a large dataset, showing a "Loading..." message until the data is loaded.

**Exercise 21: shouldComponentUpdate Optimization**

Create a counter component that only re-renders when the count is divisible by 5.

**Exercise 22: Chat Application Simulation**

Create a chat application component that simulates receiving new messages every 5 seconds. Use componentDidMount to start the simulation and componentWillUnmount to stop it.

**Exercise 23: Dynamic Form Builder**

Create a component that builds a dynamic form based on a configuration object. Use componentDidUpdate to log changes when the form configuration updates.

**Exercise 24: Data Synchronization with External API**

Simulate a component that fetches and synchronizes data with an external API every 10 seconds. Use componentDidMount to start the sync process and componentWillUnmount to stop it.

**Exercise 25: Scroll Position Saver**

Create a component that saves the scroll position of the page and restores it when the component is remounted. Use componentDidMount and componentWillUnmount for this.

**Exercise 26: E-Commerce Cart Tracker**

Build a shopping cart component that tracks items added or removed. Log changes to the cart whenever the state is updated using componentDidUpdate.

**Exercise 27: Real-Time Stock Tracker**

Create a stock tracker application that fetches stock prices from an API every 10 seconds. Use componentDidMount to start fetching data, componentDidUpdate to compare changes, and componentWillUnmount to clean up.

**Exercise 28: Multi-Page Form with Step Tracking**

Build a multi-page form with three steps. Track the current step in the component's state and use componentDidUpdate to log each step transition.

**Exercise 29: Notification Manager**

Build a notification manager component that displays notifications fetched from an API and automatically dismisses them after 5 seconds.

**Exercise 30: Timer-Based Quiz Application**

Create a quiz component that presents one question at a time, moving to the next question every 15 seconds. If the user answers, it immediately moves to the next question.