**React Debugging Instructions - Step by Step**

**Prerequisites: Install React Developer Tools**

1. Install the **React Developer Tools** browser extension:
   * Chrome: [React Developer Tools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi)
   * Firefox: Search "React Developer Tools" in Firefox Add-ons
2. Open your browser's Developer Tools (F12 or right-click → Inspect)
3. You should see "⚛️ Components" and "⚛️ Profiler" tabs

**General Debugging Workflow**

**Step 1: Open Developer Tools**

* **Windows/Linux**: F12 or Ctrl+Shift+I
* **Mac**: Cmd+Option+I
* **Right-click method**: Right-click on page → "Inspect" or "Inspect Element”

**Step 2: Check Console First**

* Click the **Console** tab
* Look for red error messages
* Look for yellow warnings
* Check for any console.log output

**Step 3: Use React Developer Tools**

* Click the ⚛️ **Components** tab
* Navigate the component tree on the left
* Inspect props and state on the right
* Use the search box to find specific components

**Exercise-Specific Debugging Instructions**

**Exercise 1: Component Not Rendering**

**Follow These Follow These Debugging Steps:**

1. **Console Tab**: Look for syntax errors or warnings
2. **Elements Tab**: Check if the component's div exists in DOM but is empty
3. **Components Tab**: Find the BuggyComponent and see if it's rendered
4. **Sources Tab**:
   * Find the component file
   * Set a breakpoint on the line with JSX
   * Notice execution never reaches the JSX line

**You'll Notice:**

* Empty div in Elements tab
* No console errors (tricky!)
* Component appears in React DevTools but renders nothing

**Debugging Technique:** Add console.log('Component rendering') at the start of the component to verify it's being called.

**Exercise 2: Props Not Passing Through**

**Follow These Debugging Steps:**

1. **Components Tab**:
   * Find ChildComponent in the component tree
   * Look at the props panel on the right
   * Notice user prop exists, but name and age don't
2. **Console Tab**: Look for "Cannot read property of undefined" warnings

**You'll Notice:**

* Props panel shows: user: {name: "John", age: 25}
* But component expects name and age directly
* Text shows "undefined" values

**Debugging Technique:** Right-click the child component → Inspect → check props structure vs. what component expects.

**Exercise 3: Event Handler Issues**

**Follow These Debugging Steps:**

1. **Console Tab**: Look for immediate function call errors
2. **Elements Tab**:
   * Find the button element
   * Look at the onclick attribute in HTML
   * Notice it might show the result of the function call, not the function
3. **Components Tab**: Check if state updates when button is clicked

**You'll Notice:**

* Console error: "Cannot read property 'call' of undefined" or similar
* Button's onClick shows result instead of function reference

**Debugging Technique:** Add console.log('Button clicked') inside a proper arrow function to test event handling.

**Exercise 4: State Not Updating**

**Follow These Debugging Steps:**

1. **Components Tab**:
   * Find the component with items state
   * Click the button and watch the state panel
   * Notice the state reference doesn't change (same array object)

**Console Tab**: Add debugging:   
  
const addItem = () => {

console.log('Before:', items);

items.push('orange');

console.log('After:', items);

console.log('Same reference?', items === items);

setItems(items);

};

**You'll Notice:**

* State panel in React DevTools shows same array reference
* Console shows array contents change but React doesn't re-render
* Array reference stays the same

**Debugging Technique:** React DevTools will highlight when state actually changes. If it doesn't highlight, React didn't detect the change.

**Exercise 5: Infinite Re-renders**

**Follow These Debugging Steps:**

1. **Console Tab**:
   * You'll see hundreds/thousands of console messages
   * Error: "Too many re-renders. React limits the number of renders..."
2. **Components Tab**:
   * Component will be constantly updating
   * State keeps changing rapidly
3. **Profiler Tab**:
   * Click "Start profiling"
   * See constant re-renders happening

**You'll Notice:**

* Console flooded with messages
* Browser may become unresponsive
* React error about render limits

**Debugging Technique:** Add console.log('Render') at component start and console.log('Effect') in useEffect to see the loop.

**Exercise 6: Stale State in Callbacks**

**Follow These Debugging Steps:**

**Console Tab**: Add debugging to the interval:   
  
const interval = setInterval(() => {

console.log('Current count in closure:', count);

setCount(count + 1);

}, 1000);

1. **Components Tab**: Watch the count state - it gets stuck at 1

**You'll Notice:**

* Console shows "Current count in closure: 0" repeatedly
* State stops updating after first increment
* Timer continues running but with stale value

**Debugging Technique:** The closure captures the initial count value (0) and never gets updated.

**Exercise 7: Unnecessary Re-renders**

**Follow These Debugging Steps:**

1. **Console Tab**:
   * Click "Increment Parent" button
   * See "ExpensiveChild rendered" in console even though child didn't change
2. **Profiler Tab**:
   * Click "Start profiling"
   * Click parent button
   * See both parent AND child components in the flame graph
3. **Components Tab**: Highlight updates feature:
   * Settings gear → "Highlight updates when components render"
   * Click parent button → both components flash

**You'll Notice:**

* Child component re-renders when parent state changes
* Console logs show unnecessary renders
* Profiler shows child in render tree

**Debugging Technique:** Use React DevTools "Highlight updates" to visually see which components re-render.

**Exercise 8: Memory Leaks**

**Follow These Debugging Steps:**

**Console Tab**: Add debugging to event handlers:   
  
useEffect(() => {

const handleResize = () => {

console.log('Resize event fired, width:', window.innerWidth);

setWindowWidth(window.innerWidth);

};

window.addEventListener('resize', handleResize);

console.log('Event listener added');

*// Missing cleanup!*

}, []);

1. **Memory Tab** (Advanced):
   * Take heap snapshots before/after toggling component
   * Look for increasing event listener counts

**You'll Notice:**

* Multiple "Event listener added" messages when toggling
* Resize events fire multiple times per resize
* Memory usage gradually increases

**Debugging Technique:** Toggle component multiple times, then resize window. Count console messages to see accumulating listeners.

**Exercise 9: Context Issues**

**Follow These Debugging Steps:**

1. **Components Tab**:
   * Find ThemedComponent in component tree
   * Look for Context providers in parent tree
   * Notice missing ThemeContext.Provider
2. **Console Tab**: May show context-related warnings

**You'll Notice:**

* Component shows "undefined" theme
* No Provider wrapper in component tree
* Context value is default (undefined)

**Debugging Technique:** Check component tree structure - context consumers must be inside providers.

**Exercise 10: Key Prop Problems**

**Follow These Debugging Steps:**

1. **Elements Tab**:
   * Find the list items
   * Notice they use index-based keys
2. **Components Tab**:
   * Type in input fields
   * Click shuffle
   * Observe that input values stay in same positions
   * Component keys change but DOM elements get reused incorrectly

**You'll Notice:**

* Input values don't follow their associated data
* React reuses DOM elements incorrectly
* User experience breaks

**Debugging Technique:** Type different text in each input, then shuffle. Values should move with their data but don't.

**More Debugging Techniques**

**Setting Breakpoints in React Components**

1. **Sources Tab** → Find your component file
2. Click line number to set breakpoint
3. Interact with component to trigger breakpoint
4. Inspect variables in scope panel
5. Use "Step over" (F10) to debug line by line

**Using React DevTools Profiler**

1. **Profiler Tab** → Click "Start profiling"
2. Interact with your app
3. **Stop profiling** → Analyze the flame graph
4. Look for:
   * Components that render unnecessarily
   * Long render times
   * Frequent renders

**Console Debugging Best Practices**

*// Group related logs*

console.group('Component Update');

console.log('Props:', props);

console.log('State:', state);

console.groupEnd();

*// Use console.table for objects*

console.table(arrayOfObjects);

*// Use console.trace to see call stack*

console.trace('How did we get here?');

*// Time expensive operations*

console.time('Expensive Operation');

*// ... expensive code*

console.timeEnd('Expensive Operation’);

**Network Tab Debugging**

* Check for failed API calls
* Look at request/response headers
* Monitor loading times
* Check for duplicate requests

**Common React Error Patterns to Look For**

1. **White Screen of Death**: Check console for JavaScript errors
2. **Components not updating**: Check if you're mutating state
3. **Performance issues**: Use Profiler to find expensive renders
4. **Memory leaks**: Look for unremoved event listeners
5. **Context not working**: Verify Provider/Consumer relationship
6. **Props not passing**: Use React DevTools to trace prop flow

**Checklist for Quick Debugging**

* Check browser console for errors
* Use React DevTools to inspect components
* Verify props are being passed correctly
* Check state updates in React DevTools
* Look for dependency array issues in useEffect
* Verify event handlers are properly bound
* Check for memory leaks (event listeners, timers)
* Use Profiler for performance issues
* Verify context providers wrap consumers
* Check key props for list items

Start with the console, then use React DevTools, and gradually dig deeper with breakpoints and profiling as needed.