Keeping Value Constant

WE'LL COVER THE FOLLOWING

Using `useRef` to Ensure Value Remains Constant

The solution to this problem is simple. We can use the `useRef` hook to ensure that a value stays the same throughout component's entire lifetime.

Here's how it works:

```
//correct implementation
  const componentJustMounted = useRef(true)
  useEffect(
    () => {
      if (!componentJustMounted.current) {
        onExpand(expanded)
      }
     componentJustMounted.current = false
    },
    [expanded]
  )
```

`useRef` returns a `ref` object, and the value stored in the object may be retrieved from the current property, `ref.current`

The signature for `useRef` looks like this: `useRef(initialValue)`.

This means that a `ref` object is stored initially in `componentJustMounted.current` with the `current` property set to `true`.

```
const componentJustMounted = useRef(true)
```

After invoking the user callback, we then update this value to `false`.

```
componentJustMounted.current = false
```

Now, whenever there's a state or prop change, the value in the ref object isn't tampered with. It remains the same.

With the current implementation, whenever the `expanded` state value is toggled, the user callback function `onExpanded` will be invoked with the current value of `expanded`.

Final Implementation of `Expandable`

Here's what the final implementation of the `Expandable` component looks like:

```
import React, { createContext, useState, useCallback, useRef, useEffect, useMemo } from 'reac

const ExpandableContext = createContext()
const { Provider } = ExpandableContext

const Expandable = ({ children, onExpand }) => {
  const [expanded, setExpanded] = useState(false)
  const toggle = useCallback(
    () => setExpanded(prevExpanded => !prevExpanded),
    []
  )
  const componentJustMounted = useRef(true)
  useEffect(
    () => {
    if (!componentJustMounted.current) {
        onExpand(expanded)
      }
     componentJustMounted.current = false
    },
    [expanded]
  )
  const value = useMemo(
   () => ({ expanded, toggle }),
   [expanded, toggle]
  )
  return (
    <Provider value={value}>
        {children}
    </Provider>
  )
}

export default Expandable
```

Quick Quiz!

Quiz yourself on what we've learned so far.

Q What is one problem with use `useEffect`?

COMPLETED 0%

1 of 1

If you've followed along so far, that's great! We've broken down the most complex component in the bunch. Now, let's move on to the child components.