# Preventing Unnecessary Renders

Let's optimize our code!

WE'LL COVER THE FOLLOWING



## Memoizing The Callback #

`toggle` acts as a callback function and it'll eventually be invoked by `Expandable.Header`. Let's prevent any future performance issues by memoizing the callback.

```
import React, { createContext, useState, useCallback } from 'react'

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

const Expandable = ({children}) => {
  const [expanded, setExpanded] = useState(false)
  // look here 👇
  const toggle = useCallback(
    () => setExpanded(prevExpanded => !prevExpanded),
    []
  )
  return <Provider>{children}</Provider>
}

export default Expandable
```

Not sure how `useCallback` works? Have a look at this cheatsheet.

Once we have both `expanded` and `toggle` created, we can expose these via the `Provider`'s value prop.

```
import React, { createContext, useState, useCallback } from 'react'

const ExpandableContext = createContext()
```

```
const { Provider } = ExpandableContext

const Expandable = ({children}) => {

  const [expanded, setExpanded] = useState(false)
  // look here 👇
  const toggle = useCallback(
    () => setExpanded(prevExpanded => !prevExpanded),
    []
  )
   // look here 👇
  const value = { expanded, toggle }
   // and here 👇
  return <Provider value={value}>{children}</Provider>
  }

export default Expandable
```

This works, but the `value` reference will be different on every re-render causing the `Provider` to re-render its children.

Let's memoize the `value`.

```
import React, { createContext, useState, useCallback } from 'react'

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

const Expandable = ({children}) => {
  const [expanded, setExpanded] = useState(false)
  // look here 👇
  const value = useMemo(
        () => ({ expanded, toggle }),
        [expanded, toggle]
  )
  return <Provider value={value}>{children}</Provider>
  }

export default Expandable
```

`useMemo` takes a callback that returns the object value `{expanded, toggle}` and we pass an array dependency `[expanded, toggle]`. This means that the memoized value remains the same unless the dependencies change.

We've done a great job so far!

## Quick Quiz! #

Before we move on, let's take a quick quiz on what we've covered so far!

1 What does memoization do in React?

COMPLETED 0%

1 of 2

In the next lesson, we'll discuss an approach for avoiding unnecessary state callbacks.