Custom Styling Options

We'll now learn how to override default styles!

WE'LL COVER THE FOLLOWING

Styling

So far, we've handled styling with the `className` prop. What about the option to override default styles by passing a `style` prop? As of right now, we can't do that

Let's fix that!

Instead of explicitly destructuring the `style` prop, we can pass any other prop passed by the user to the `button` component.

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Note the use of the rest parameter and spread syntax on **lines number 5** and **10**, respectively, in `Header.js`.

With this done, the `Header` component receives our default styles, while allowing for change via the `className` or `style` props. Here's how you would pass styles to the `Header` component.

```
// override style via className
<Expandable.Header className="my-class">
        React hooks
</Expandable.Header>

// override style via style prop
<Expandable.Header style={{color: "red"}}>
        React hooks
</Expandable.Header>
```

Let's try passing some styles! Let's override via `className` first. Have a look at `App.js`.

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Of course, this does not have any effect right now because we have not defined anything for this new `className`. Let's override `style` via style prop now.

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Styling `Body` and `Icon`

Now, I'll go ahead and do the same for the other child components, `Body` and `Icon`.

`Body` Component Before

```
// before
const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
```

`Body` Component After

Have a look at the `Body.js` file for changes to the body component. Note that we've created a `Body.css` file.

```
import React, { useContext } from 'react'
import { ExpandableContext } from './Expandable'
import './Body.css'

const Body = ({ children, className = '', ...otherProps }) => {
  const { expanded } = useContext(ExpandableContext)
  const combinedClassNames = ['Expandable-panel', className].join('')

  return expanded ? (
    <div className={combinedClassNames} {...otherProps}>
      {children}
    </div>
  ) : null
}

export default Body
```

The `Icon` Component Before

```
// before
const Icon = () => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? '-' : '+'
}
```

The `Icon` Component After

Here, we are doing the same for the `Icon` component. Note that we've created an `Icon.css` file as well.

```
import React, { useContext } from 'react'
import { ExpandableContext } from './Expandable'
import './Body.css'

const Body = ({ children, className = '', ...otherProps }) => {
  const { expanded } = useContext(ExpandableContext)
  const combinedClassNames = ['Expandable-panel', className].join('')

  return expanded ? (
    <div className={combinedClassNames} {...otherProps}>
      {children}
    </div>
  ) : null
}

export default Body
```

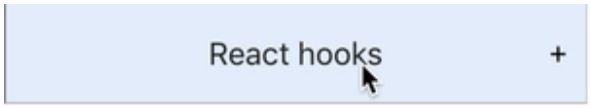
Notice the `+` and `-` move to the far right because of this styling!

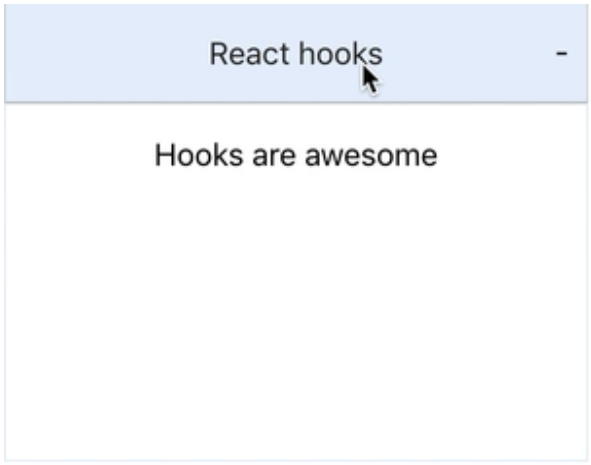
Styling `Expandable`

And finally, some styles for the parent component, `Expandable`.

```
.Expandable-panel {
    margin: 0;
    padding: 1em 1.5em;
    border: 1px solid hsl(216, 94%, 94%);;
    min-height: 150px;
  }
```

Now we've got a beautiful reusable component!





2 of 2

Quick Quiz!

1

What method of overriding is being used here?

```
<Expandable.Header style={{color: "red"}}>
```

COMPLETED 0%

1 of 3

This is great! Our component is stylable and is great for our use case. Let's look at how someone else might consume it for their usecase

look at how someone else might consume it for their usecase.