What is the Prop Getters Pattern?

Here's a quick overview of the prop getters pattern!

WE'LL COVER THE FOLLOWING

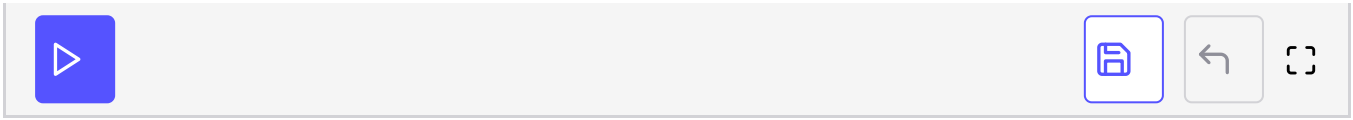
Props Collection vs. Prop Getters Pattern

In JavaScript, all functions are objects. However, functions are built to be more customizable and reusable.

Consider the following:

```
const obj = {name: "React hooks"}

console.log(obj)
```



If you wanted to create the same object but allow for some level of customization, you could do this:

```
const objCreator = n => ({name:n})
```

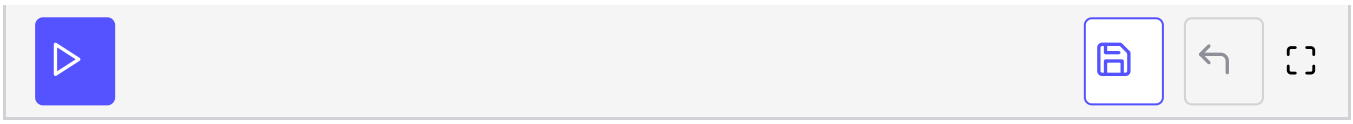
Now, this `objCreator` function can be called multiple times to create different objects as shown below:

```
const objCreator = n => ({name:n})
```

```
const obj1 = objCreator("React hooks")
const obj2 = objCreator("React hooks mastery")

const obj3 = objCreator("React hooks advanced patterns")

console.log(obj1)
console.log(obj2)
console.log(obj3)
```



Why have I chosen to explain this?

Because this is the difference between the props collection pattern and prop getters pattern.

While props collection relies on providing an object, prop getters expose functions that can be invoked to create a collection of props.

Refactoring the Previous Solution

Because functions can be customized, using a prop getter allows for more interesting use cases.

Let's start by refactoring the previous solution to use a props getter.

```
// before
  const togglerProps = useMemo(
    () => ({
      onClick: toggle,
      'aria-expanded': expanded
    }),
    [toggle, expanded]
  )

// now
  const getTogglerProps = useCallback(
    () => ({
      onClick: toggle,
      'aria-expanded': expanded
    }),
    [toggle, expanded]
  )
```

Instead of `togglerProps`, we now create a memoized function, `getTogglerProps`, that returns the same props collection.

We'll expose this via the returned `value` variable within the `useExpanded` hook

as shown below:

```
export default function useExpanded () {
  ...
  const getTogglerProps = useCallback(
    () => ({
      onClick: toggle,
      'aria-expanded': expanded
    }),
    [toggle, expanded]
  )
  // look here 👇
  const value = useMemo(() => ({ expanded, toggle, getTogglerProps }), [
    expanded,
    toggle,
    getTogglerProps
  ])
  return value
}
```

Now, we need to update how the props collection is consumed.

```
// before
<button {...togglerProps}>Click to view awesomeness...</button>
// now
<button {...getTogglerProps()}>Click to view awesomeness...</button>
```

The user would do this in their app.

Putting It All Together

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

And that's it! The user's app works just like before.

Quick Quiz

Let's take a quick quiz before moving on!

1 What is the main difference between the prop getters pattern and the props collection pattern?

I know what you're thinking. If we're not going to pass any arguments to the `getTogglerProps` function, why bother with the refactor?

That's a great question and we'll pass in some arguments soon.