What is the Props Collection Pattern?

In this lesson, we'll have a quick overview of the props collection pattern!

WE'LL COVER THE FOLLOWING

Commonalities Across Component Usage

Regardless of the components, you build and who uses them, some things will remain the same for all users of the component.

How can we know this?

Well, consider the `Expandable` component we've worked so hard on.

Regardless of the UI solution the user decides to go for, they'll always need an `onClick` handler for the "toggle element".

```
const { toggle, expanded } = useExpanded()
// user 1.
<button onClick={toggle}>Click to view awesomeness...</button>
// user 2.
<div onClick={toggle}> Burn the place down 🔥</div>
// user 3.
<img onClick={toggle} alt="first image in carousel"/>
```

You get the point.

Regardless of the element they choose to render, every user still needs to handle the `onClick` callback.

If these users care about accessibility at all, they will do this:

```
const { toggle, expanded } = useExpanded()
```

```
// user 1.
<button onClick={toggle} aria-expanded={expanded}>Click to view awesomeness...</button>
// user 2.

<div onClick={toggle} aria-expanded={expanded}> Burn the place down 🔥</div>
// user 3.
<img onClick={toggle} aria-expanded={expanded} alt="first image in carousel"/>
```

Now, that's two props.

Depending on your use case, there could be more "common props" associated with your reusable hook or component.

Is there a way we can prevent the users from writing these props every single time? Can we expose some API from within your reusable custom hook or component?

These are the questions that the props collection pattern answers with a resounding yes!

> A **props collection** is a collection of "common props" associated with a custom hook or component.

Quick Quiz!

1 What's the main advantage of the props collection pattern?

COMPLETED 0%

1 of 2

In the next lesson, we'll look at this pattern in action!