

Specifying a type for a function prop

A prop can be a function. In this lesson, we'll learn how to strongly-type function props.

WE'LL COVER THE FOLLOWING ^

- Strongly-typing a render prop
- Wrap up

Strongly-typing a render prop

An example of a function prop is a prop that allows the consumer of a component to control the rendering of part of a component. This is called a **render prop**.

Open the CodeSandbox project we were working on in the last lesson. Let's add a render prop to our `Hello` component to optionally allow the consumer to control how the message is rendered. The prop will be called `renderMessage`, and the `Hello` component will be as follows:

```
const Hello = ({
  who,
  renderMessage,
  message = "How are you?"
}: Props) => (
  <React.Fragment>
    <p>
      Hello, {who.name}
      {who.friend && " my friend"}
    </p>
    {message && (renderMessage ? renderMessage(message) : <p>{message}</p>
  >)}
  </React.Fragment>
);
```

An example consumption of `Hello` is as follows:

```
<Hello  
  who={{ name: "Bob", friend: true }}  
  message="Hey, how are you?"  
  renderMessage={m => <i>{m}</i>}  
/>
```

So, what should the definition of the `Props` type be now with the new `renderMessage` prop?

 Show Answer

Can you think of another common use case for function props in React?

 Show Answer

Wrap up

Well done! We can now create strongly-typed function props using the following syntax:

```
(param1: Type1, param2: Type2, ...) => ReturnType;
```

That concludes this chapter on strong-typed props in function components. We now have the knowledge to give consumers of our components a great experience. Next up, let's check what we have learned with a quiz.