Creating strongly-typed function component states with useState

In this lesson, we are going to learn how to create strongly-typed states in a function component with the 'useState' hook.

WE'LL COVER THE FOLLOWING

- Inferred useState type
- Specifying the useState type
- Wrap up

Inferred useState type

The useState hook allows us to create a single piece of state. Click the following link to open a CodeSandbox project that makes use of useState: CodeSandbox useState project 1

This is a React and TypeScript project containing a Counter component with a piece of state called count.

What type is the **count** state?



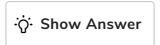
So, state with the useState hook is automatically strongly-typed by inferring the type from the initial value passed into the function. This is great in lots of cases because we don't need to do anything to get the benefits of strongly-typed state.

Specifying the useState type

Click the following link to open a different CodeSandbox project:

CodeSandbox useState project 2

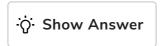
What is the type of the **count** state in this 2nd project?



So, there are times when TypeScript doesn't infer the type we want for state in the useState hook. How do we specify the type we want to use for the state? Well, useState is a generic function. So, we can pass the type in as a generic parameter:

```
const [...] = React.useState<StateType>(...);
```

Use the useState 's generic parameter in the second CodeSandbox project we have open to make the count state more strongly-typed.



Wrap up

Great stuff! In some cases, TypeScript can infer the type of the state returned from a useState hook from the initial value. For the instances when TypeScriptss inference isn't required, we can explicitly pass the state type into the useState generic parameter.

Now that we know how to strongly-type state with useState, we are going to learn how to strongly-type more complex state with useReducer in the next lesson.