Creating strongly-typed event handlers

Interactive components need to handle events. In this lesson, we'll learn how to implement event handlers in a function component that has strongly-typed parameters. What we will learn also applies to class components as well.

WE'LL COVER THE FOLLOWING

Creating an inline anonymous event handler

We are going to finish the implementation of a `Searchbox` component to learn how to strongly-type event handlers. Click the link below to open the exercise in CodeSandbox:

CodeSandbox project

The app contains an `input` that will eventually allow a user to enter some search criteria. At the moment, users can't type anything into the `input`.

Let's add an `onChange` event handler inline on the `input` that updates the `criteria` state:

```
<input
  type="text"
  value={criteria}
  onChange={e => setCriteria(e.currentTarget.value)}
/>
```

Notice that the editor provides IntelliSense on the `e` parameter. What is the type of `e`?

Show Answer

So, the inline event handler is strongly-typed because TypeScript has cleverly inferred the type. Nice!

With the event handler in place, users can now enter criteria into the `input`.

Creating a named event handler

Let’s refactor the implementation to use a named function for the event handler:

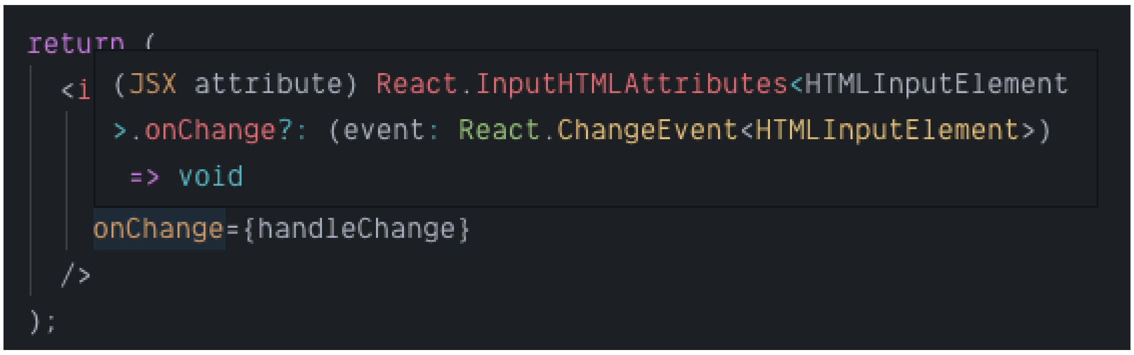
```
function Searchbox() {
  const [criteria, setCriteria] = React.useState("");

  const handleChange = e => setCriteria(e.currentTarget.value);

  return (
    <input
      type="text"
      value={criteria}
      onChange={handleChange}
    />
  );
}
```

What is the type of `e` now?

Show Answer

So, our named function event handler is not strongly-typed. Not so nice!

We can use a type annotation to define the type of `e` explicitly. What type should `e` be set to? We may have remembered what the type of `e` should be from the inline event handler implementation. If we have forgotten, we can hover over the event handler prop to find out:



```
return (
  <i (JSX attribute) React.InputHTMLAttributes<HTMLInputElement
     >.onChange?: (event: React.ChangeEvent<HTMLInputElement>)
      => void
    onChange={handleChange}
  />
);
```

So, the type is `React.ChangeEvent<HTMLInputElement>`. Let's add this to `handleChange`:

```
const handleChange = (e: React.ChangeEvent<HTMLInputElement>) =>
  setCriteria(e.currentTarget.value);
```

Wrap up

Excellent! The parameter type for inline event handlers is always inferred correctly. If we want to create a named event handler function, we can hover over the event handler prop to discover what the function parameter type should be.

In the next lesson, we'll learn how to create strongly-typed event props.