Handling User Triggered Events

With JavaScript, we've already talked about several events to handle user interaction (`click`, `mouseenter`, etc). With React, you can still use all of the same events, but the implementation is slightly different. Let's take a look at a simple implementation both with traditional DOM manipulation as well as with React. First, let's assume we have a button that when clicked, we want to log the button's value to the console. Then we'll see how to keep track of user input and use it in some way.

Button Click Event

Traditional DOM

With traditional methods, there are technically two ways to do it. The first method it would look something like this:

```html
<button id="some-button" value="1">Click Me</button>
<script>
  document
    .getElementById("some-button")
    .addEventListener("click", clickHandler);

  function clickHandler(e) {
    console.log(e.target.value);
  }
</script>
```

That is what we're used to doing, but there is another way to do it:

```html
<button id="some-button" value="1" onclick="clickHandler(e)">
  Click Me
</button>
<script>
  function clickHandler(e) {
    console.log(e.target.value);
  }
</script>
```

As you can see with the second option, it allows you to bypass the `addEventListener` in favor of adding it directly to the element itself. While this isn't the way you typically want to do it, that functionality is what you will use for React.

React

Using the above as a template, you can combine both the function creation AND the implementation in one fell swoop.

```
// If you declared the clickHandler function as above, it would simply be:
<button id="some-button" value="1" onClick={(e) => clickHandler(e)}>
  Click Me
</button>
// Or, if you don't need to pass in 'e' or any other value:
<button id="some-button" value="1" onClick={clickHandler}>
  Click Me
</button>
// Or, if you didn't declare a function, but want to run some simple code:
<button id="some-button" value="1" onClick={(e) => console.log(e.target.value)}>
  Click Me
</button>
```

As you can see it's very straightforward. The only major difference is that any event is written in camel case and then the function is wrapped in `{}`.

Input value binding

So let's say you had some variable that you wanted to update with the new value from an input box. In React it's pretty straight forward. You could simply follow the pattern above and utilize the `onChange` event in the same way you would use the `onClick` from above. One thing to note though is that this is simply reading the input field and is not actually updating any values that input might be tied to. This is something we'll be getting into more detail with later on.