Expressions as props

You've already learned a bit about using expressions as props. These can be, among other things, ternary operators, function calls, or some arithmetic operations.

However, you can pass almost any kind of expression as a prop.

For example:

```
const bool = false;

function Example(props) {
    return (
        <h2>The value of the toggleBoolean prop is: {props.toggleBoolean.toString()}</h2>
    );
};

export default function App() {
    return (
        <div className="App">
            <Example toggleBoolean={!bool} />
        </div>
    );
};
```

In the example above, you're using the `!bool`, that is, the NOT operator, which evaluates to `true`, since `!false` is true.

Also, for the `toggleBoolean` prop to be rendered on the page, you're converting its boolean value to a string using the JavaScript's built-in toString method.

Here's an extension of the above code which shows more ways to work with expressions as props in React.

What is happening here is several props are being passed to the `Example` component, and rendering each of these props' values to the screen.

```
const bool = false;
const str1 = "just";

function Example(props) {
    return (
        <div>
            <h2>
                The value of the toggleBoolean prop is:{props.toggleBoolean.toString()}
            </h2>
            <p>The value of the math prop is: <em>{props.math}</em></p>
            <p>The value of the str prop is: <em>{props.str}</em></p>
        </div>
    );
};

export default function App() {
```

```
    return (
        <div className="App">
            <Example
                toggleBoolean={!bool}
                math={(10 + 20) / 3}
                str={str1 + ' another ' + 'string'}
            />
        </div>
    );
};
```

In this improvement to the **`Example`** component, three props are being passed to it: **`toggleBoolean`**, **`math`**, and **`str`**. The **`toggleBoolean`** is unchanged, and the **`math`** prop and the **`str`** prop have been added.

The **`math`** prop is there to show that you can add arithmetic operators and numbers inside JSX, and it will be evaluated just like it does in plain JavaScript.

The **`str`** prop is there to show that you can concatenate strings, as well as strings and variables – which is shown by adding string literals of " another " and "string" to the **`str1`** variable.

In summary, just like you can use expressions inside function components, you can also use them as prop values inside JSX elements, when rendering those function components.