Applying conditional rendering

State is all the data your app is currently working with. With this in mind, you can decide to conditionally render specific components in your app, based on whether specific state data has specific values. To make this possible, React works with the readily available JavaScript syntax and concepts.

Consider a minimalistic productivity app.

The app takes the client computer's current datetime, and based on the data, displays one of two messages on the screen:

1. For workdays, the message is: "Get it done"
2. For weekends, the message is: "Get some rest"

There are a few ways you can achieve this in React.

One approach would include setting a component for each of the possible messages, which means you'd have two components. Let's name them `Workdays` and `Weekends`.

Then, you'd have a `CurrentMessage` component, which would render the appropriate component based on the value returned from the `getDay()` function call.

Here's a simplified `CurrentMessage` component:

```
function CurrentMessage() {
    const day = new Date().getDay();
    if (day >= 1 && day <= 5) {
        return <Workdays />
    }
    return <Weekends />
}
```

Instead of calculating it directly, you could use some historical data instead, and perhaps get that data from a user via an input, from a parent component.

In that case, the `CurrentMessage` component might look like this:

```
function CurrentMessage(props) {
    if (props.day >= 1 && props.day <= 5) {
        return <Workdays />
    }
    return <Weekends />
}
```

Conditional rendering with the help of element variables

To further improve your `CurrentMessage` component, you might want to use element variables. This is useful in

some cases, where you want to streamline your render code - that is, when you want to separate the conditional logic from the code to render your UI.

Here's an example of doing this with the `CurrentMessage` component:

```
function CurrentMessage({day}) {
    const weekday = (day >= 1 && day <= 5);
    const weekend = (day >= 6 && day <= 7);
    let message;

    if (weekday) {
        message = <Workdays />
    } else if (weekend) {
        message = <Weekends />
    } else {
        message = <ErrorComponent />
    }

    return (
        <div>
            {message}
        </div>
    )
}
```

The output of the `CurrentMessage` component will depend on what the received value of the day variable is. On the condition of the day variable having the value of any number between 1 and 5 (inclusive), the output will be the contents of the `Workdays` component. Otherwise, on the condition of the day variable having the value of either 6 or 7, the output will be the contents of the `Weekends` component.

Conditional rendering using the logical AND operator

Another interesting approach in conditional rendering is the use of the logical `AND` operator `&&`.

In the following component, here's how the `&&` operator is used to achieve conditional rendering:

```
function LogicalAndExample() {
    const val = prompt('Anything but a 0')

    return (
        <div>
            <h1>Please don't type in a zero</h1>
            {val &&
                <h2>Yay, no 0 was typed in!</h2>
            }
        </div>
    )
}
```

There are a few things to unpack here, so here is the explanation of the `LogicalAndExample` component, top to bottom:

1. First, you ask the user to type into the prompt, specifying that you require anything other than a zero character; and you save the input into the `val` value.
2. In the return statement, an `h1` heading is wrapped inside a `div` element, and then curly braces are used to include a JSX expression. Inside this JSX expression is a single `&&` operator, which is surrounded by some code both on its left and on its right sides; on the left side, the val value is provided, and on the right, a piece of JSX is provided.

To understand what will be output on screen, consider the following example in standard JavaScript:

```
true && console.log('This will show')
```

If you ran this command in the browser's console, the text 'This will show' will be output.

On the flip side, consider the following example:

```
false && console.log('This will never show')
```

If you ran *this* command, the output will just be the boolean value of `false`.

In other words, if a prop gets evaluated to `true`, using the `&&` operator, you can render whatever JSX elements you want to the right of the `&&` operator.