Styling JSX elements

You've observed that JSX is incredibly versatile, and can accept a combination of JavaScript, HTML and CSS. In this reading, you'll learn some approaches for styling JSX elements and doing so in a way that achieves both a functional and visual aspect within an app.

There are various ways to style JSX elements.

Probably the simplest way to do this is using the `link` HTML element in the head of the **index.html** file in which your React app will mount.

The `href` attribute loads some CSS styles, probably with some CSS classes, and then, inside the function component's declarations, you can access those CSS classes using the `className` attribute.

```
function Promo(props) {
    return (
        <div className="promo-section">
            <div>
                <h1>{props.heading}</h1>
            </div>
            <div>
                <h2>{props.promoSubHeading}</h2>
            </div>
        </div>
    );
}
```

In CSS:

```
.promo-section {
    font-weight: bold;
    line-height: 20px;
}
```

Another way to add CSS styles to components is using inline styles.

The syntax of inline styles in JSX is a bit custom.

Consider a starting `Promo` component, containing code that you encountered earlier:

```
function Promo(props) {
    return (
        <div className="promo-section">
            <div>
                <h1>{props.heading}</h1>
            </div>
            <div>
                <h2>{props.promoSubHeading}</h2>
            </div>
        </div>
```

```
    );
}

export default Promo;
```

Now you can add some inline styles to it:

```
function Promo(props) {
    return (
        <div className="promo-section">
            <div>
                <h1 style={{color:"tomato", fontSize:"40px", fontWeight:"bold"}}>
                    {props.heading}
                </h1>
            </div>
            <div>
                <h2>{props.promoSubHeading}</h2>
            </div>
        </div>
    );
}

export default Promo;
```

You can start updating the `Promo` component by adding the JavaScript expression syntax:

```
<h1 style={}>
```

As explained previously, this means that whatever code you add inside these opening and closing curly braces is to be parsed as regular JavaScript. Now let's add a **style object literal** inside of these curly braces:

```
<h1 style={{color:"tomato",fontSize:"40px"}}>
```

You can then re-write this object literal:

```
{
    color: "tomato",
    fontSize: "40px"
}
```

So, there's nothing special about this object, except for the fact that you've inlined it and placed it inside a pair of curly braces. Additionally, since it's just JavaScript, those CSS properties that would be hyphenated in plain CSS, such as, for example, `font-size:40px`, become camelCased, and the value is a string, making it look like this: `fontSize:"40px"`.

Besides inlining a *style object literal*, you can also save it in a variable, and then use that variable instead of passing an

object literal.

That gives you an updated `Promo` component, with the styles object saved as a JavaScript variable:

```
function Promo(props) {

const styles = {
    color: "tomato",
    fontSize: "40px"
}

return (
        <div className="promo-section">
            <div>
                <h1 style={styles}>
                    {props.heading}
                </h1>
            </div>
            <div>
                <h2>{props.promoSubHeading}</h2>
            </div>
        </div>
    );
}
```

Using this approach makes your components more self-contained, because they come with their own styles built-in, but it also makes them a bit harder to maintain.