**Thursday May 2nd, 2019**

8:46 — Let’s get a quick 20 minutes in to start the day.

8:49 — When operating in JSX it’s possible that we may sometimes want to use JavaScript. The problem is that JSX by default seeks to interpret things in HTML. While we may have predefined variables, if we attempt to use them in our JSX they will be interpreted literally. For example <p> Welcome firstName </p> will literally output “Welcome firstName”.

8:52 — In order for our JavaScript to be properly interpreted in the context of JSX, we need to surround any code we want interpreted as JavaScript with curly braces { }.

8:55 — The way inline-styles are interpreted in HTML and JSX is slightly different. While HTML expects our inline-style to be given using a string, JSX expects a JavaScript object to give the style. One important nuance to note is that while JavaScript objects are wrapped in curly braces, JSX also demands all of our objects to be wrapped in curly braces. The end result? We end up with a double set of curly braces.

8:58 — Apparently, another thing we need to note is that while our inline-styles may normally require a dash… background-color for example, JavaScript does not permit a dash in the middle of a property name. The solution? Convert the dash to CamelCase.

9:01 — Another difference between HTML and JSX comes when defining unit sizes. For example while HTML would use font-size: 30px, JSX uses fontSize: 200 or fontSize “200px”. We can also use quotes when defining another types of units such as ems or rems.

9:04 — The course instructor says that one reason we would want to use inline-styles in this fashion is that doing so would allow us to use JavaScript to dynamically determine which styles should be applied. We may for example use a function to determine what time it is. With that time, we may display a different background color depending on whether it is morning, afternoon, or evening.

10:35 — I’m back. Let’s get another 25 minutes or so in.

10:54 — I’ve spent the last 15 minutes or so creating a component and learning how arrow functions work in React. To be honest, stylistically I prefer traditionally defined functions over arrow functions. With that being said, I do understand that arrow functions have some advantages over traditional functions and that while working with others I may often come across arrow functions in React. For that reason, I thought it was important I learn more about how arrow functions work in React.

10:59 — I accidentally started the video so I had to go back and rewrite my arrow function. This time I was able to do everything much more quickly though. Here’s a code sample from my TodoItem.js file:

import React from "react"

const TodoItem = () => (

<div>

<input type="checkbox" />

<p>Placeholder text here</p>

<input type="checkbox" />

<p>Placeholder text here</p>

<input type="checkbox" />

<p>Placeholder text here</p>

<input type="checkbox" />

<p>Placeholder text here</p>

</div>

)

export default TodoItem

11:00 — I’m getting a bit tired, but I’m going to style things in CSS before taking a break.

11:10 — Cool, I’m getting more comfortable with CSS, inline-styles and JSX. Here’s some sample code from ToDoItem.js:

import React from "react"

const currentTask = {

backgroundColor: "yellow",

fontSize: "2em"

}

const TodoItem = () => (

<div>

<input type="checkbox" />

<p style = {currentTask}>Placeholder text here</p>

<input type="checkbox" />

<p>Placeholder text here</p>

<input type="checkbox" />

<p>Placeholder text here</p>

<input type="checkbox" />

<p>Placeholder text here</p>

</div>

)

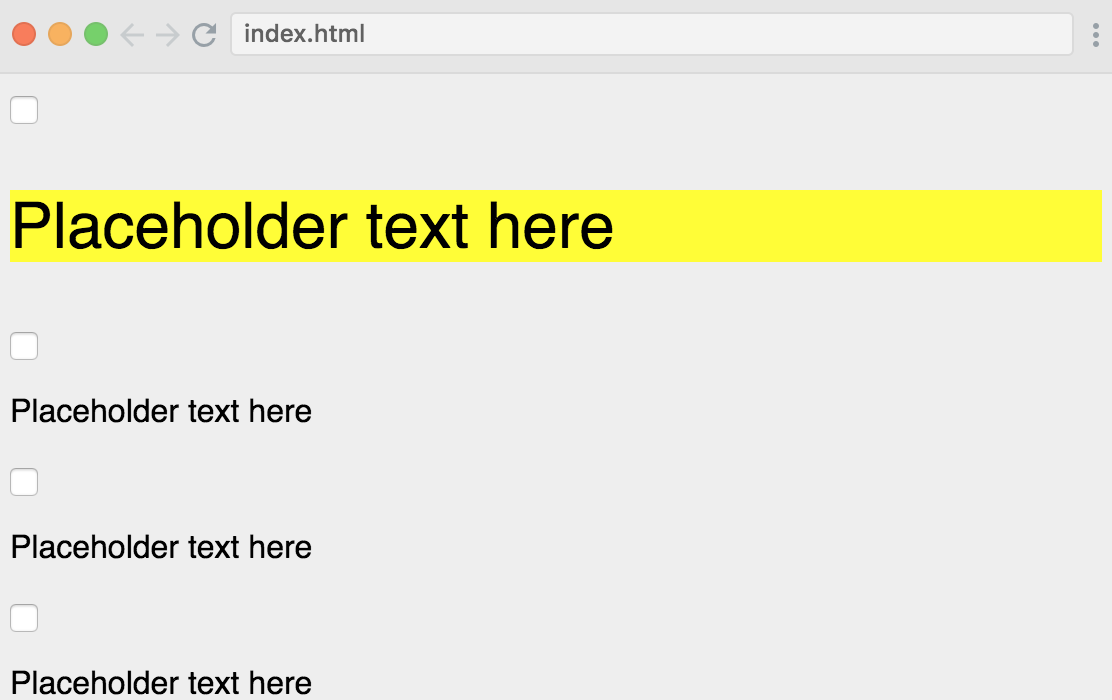
export default TodoItem

and style.css:

body {

background-color: #EEE

}

Here’s what outputs to the page:

Is it pretty? Not really, but we’re just focused on baby steps. Getting a little bit better during each pomodoro session and I’ve definitely seen notable improvements today already.

11:13 — Time to go enjoy the first meal of the day!

**Total time spent coding today: N/A**

**Total time spent coding in May 2019: N/A**