**Sunday May 12th, 2019 Daily Coding Journal**

12:14 — Here we goooooo. Let’s review the material from yesterday. Maybe it will make more sense now that I’m a bit more well rested.

12:27 — I’ve been trying to follow along with the React tutorial but like yesterday I’m not really understanding a lot of the logic about why certain things are happening. I’m going to watch a few videos on events and event handling in React to see if I can get a somewhat better understanding of what’s going on under the hood.

12:34 — I watched a video about event handling and learned a neat trick about how I could use this.props.children to display the children of some prop. Other than that though, everything continues to feel pretty foggy thus far.

12:35 — I’m going to get something to eat and come back for another study session later.

17:21 — I’m back again.

17:23 — One quick thing I learned is that we can log multiple things inside of console.log method. For example we could write:

console.log(“message”, variableName)

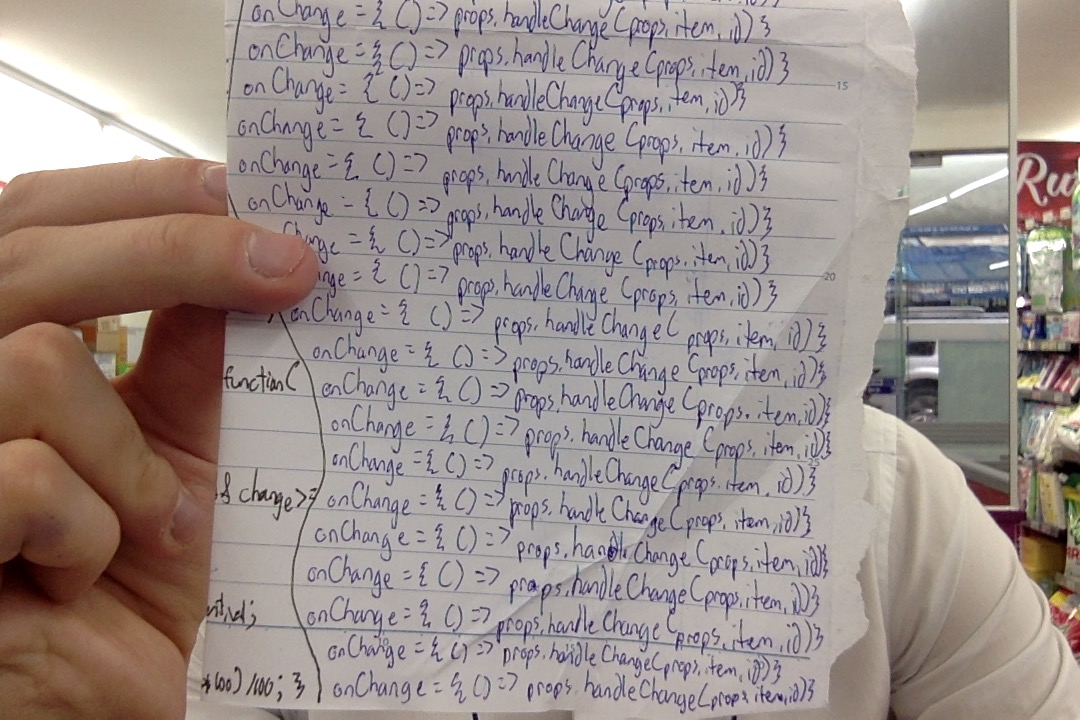
17:25 — I always knew we could log multiple things inside of a console.log method using the plus operator but seeing the use of a comma was something new to me.

17:37 — I’m kind of just meditating on the following line of code right now:

onChange = {( ) => props.handleChange(props.item.id)}

17:45 — When learning something unfamiliar, something I’ve found that really helps me to understand new concepts is to take code samples and write them by hand a few dozen times.

(UPDATE: For reference, I spend the next 25 or so minutes looking at code from the following video: <https://scrimba.com/p/p7P5Hd/cgDqBHP>)

17:46 — Here’s an example based on the code I was just looking at:

17:47 — Apart from possibly forming some muscle memory on the correct syntax, I find this type of exercise useful because it forces the mind to focus on a small bit of code for extended periods of time. Usually the process I go through goes something like this:

#1 Complete confusion

#2 Write the code by hand several times

#3 Begin to think about why the code works the way it does

#4 Continue writing the code by hand several more times

#5 Begin to have a surface level understanding of the code

#6 Write the code by hand several more times

#7 Begin being able to replicate some of the code in my application

#8 Begin to understand the concept at a deeper level after being forced to use it in several real life applications throughout my projects

17:51 — Looking at the code above, I’m probably at about step #5 thus far. Based on my current understanding, I’d theorize that the code is doing something like this:

*onChange* — This is the easiest part to understand. This is the event handler that’s “listening” for some event.

*{ }* — Because we’re using JavaScript inside of JSX, we need to use these curly braces

*( ) =>* — My understanding of this part is a bit shaky. It seems to me, however, that our event handlers look for some kind of “event” to occur when they fire. For that reason we need to create an empty function that’s returned as the “event” and that function can then return props.handleChange.

*props.handleChange* — Taking a look at the wider context of code:

function TodoItem(props) {

return (

<div className="todo-item">

<input

type="checkbox"

checked={props.item.completed}

onChange={() => props.handleChange(props.item.id)}

/>

<p>{props.item.text}</p>

</div>

)

}

handleChange is actually being passed into TodoItem as one of its props. That is why it is necessary for our referencing of this method to include props and as such be written as props.handleChange.

*props.item.id* — What’s going on here? Well, in our handleChange method, remember that handleChange takes in an id as an argument. Where does that come from? It comes from *props.item.id*. Why does *props.item.id* have the relevant values that we are looking for? Because using the map method, we took data from a JSON file (or for simplicity sake in this course, another .js file), and passed item for our argument as we mapped each item in our array into a new component.

Basically, what we’re saying is take this item’s id and pass it into our handleChange method. Then, take this item’s if and pass it into our handleChange method, and so on until we’ve gone through all of the items.

18:09 — While I’m sure I didn’t explain all of the above in a perfectly coherent manner, pushing oneself to explain concepts also forces oneself to look at every isolated part of a component (no pun intended) to understand the greater whole. I feel as if I’ve done just that with the code above.

18:11 — Anyway, I’d call this a successful pomodoro session. I’m going to go to the bathroom and go for a quick walk. Then I’ll try to squeeze one more pomodoro session in before dinner.

23:06 — So, we’re a little bit out of order now. My doll arrived at the convenience store about 45 minutes early so I didn’t have time to get that pomodoro session in before dinner. No problem, we’ll get it in now.

23:11 — It looks like I retained all the information I practiced earlier today. Now, let’s see if I can push any further…

23:16 — I know I need to implement the .map method and an if statement to change the state of my task if its id is equal to the id of the todo-item that is clicked. But, how can I go about doing that?

23:26 — I’m doing some basic debugging stuff to try to see if I’m on the right track. Seemingly as soon as I fix one error, however, I get another one:

handleChange(id) {

this.setState(this.todoItems.map( (item) => {

if (item.id = 1) {

console.log("take out the trash confirmed")

}

}))

The above code for example gave me the following error:

TypeError: “Cannot read property map of undefined”

23:29 — What is going wrong? As usual, I don’t have much of an idea to be honest. Mostly I just finagle things until something works and in that process I can usually internalize the concept for how things work.

23:32 — Look at the following code:

handleChange(id) {

console.log(this.todoItems)

}

render() {

const todoItems = this.state.todos.map(item => <TodoItem key={item.id} item={item} handleChange={this.handleChange} />)

I’m not sure why this logs null instead of console.logging the entire todoItems component/array.

23:35 — I’m going to watch the video a couple times to try to see where I was going wrong and then head to bed.

23:39 — It looks like one problem with what I was trying appears to be that I may have been attempting to modify state directly.

23:40 — On the other hand, I don’t see exactly how that is true given that I was trying to use the .map method to create a new array and modify the state of those array items.

23:41 — In any case, the key point I appear to have missed has to do with state. See the following code:

this.state = {

todos: todosData

}

I did not realize that all of the information for the todos was coming from the todoData.js file. I should have paid better attention to all of the things closely related to state giving that was something I was focusing on in this video.

23:45 — I didn’t realize that my todoItem component was actually created by mapping from this.state.todos.map. Now it seems obvious, but earlier it wasn’t something I paid much attention to.

23:49 — Man I’m beat. I’m proud of myself for putting in this extra session after a long night out, but at this point pushing forward any further is unlikely to be productive. Fortunately, I think tomorrow may just be the day that I’m finally able to get through all of this exercise in one sitting.

**Total time spent coding today: 2 hours 8 minutes**

**Total time spent coding thus far in May 2019: 23 hours 24 minutes**

**Total lifetime hours of coding: 519 hours 18 minutes**