**Tuesday May 7th, 2019**

0:08 — While aren’t we the early birds getting started real early today ;) Let’s continue the course now that we’ve gotten comfortable with mapping and props!

0:14 — Now we’re going to start learning about class-based components.

0:17 — Every class-based component needs at least one method… The render method ( ) { }

0:20 — Classed-based components will always start with the class keyword.

0:23 — Functional components and class-based components interact with props a little bit differently. For example…

Functional components would use: {props.item}

Class-based components would use: {this.props.item}

0:26 — Bob says that it is common for developers to forget the this keyword when converting one’s components from functional to class-based, but that this is a bug we are likely to quickly become adept at debugging.

0:28 — Bob recommends we have a solid understanding of ES6 classes in order to fully under class-based components. For that reason, I’m going to quickly watch a refresher video or two on classes in JavaScript.

0:30 — MPJ from FunFunFunction says that the class keyword in JavaScript does not represent true classes as in other programming languages, but that the keyword class in JavaScript is merely syntactic sugar on top of the prototypal inheritance model.

(source: <https://www.youtube.com/watch?v=Tllw4EPhLiQ>)

0:40 — MPJ says that while we should become familiar with classes, we should also be sure that if we are to use classes we do not do so simply because we are trying to avoid having to exert the effort needed to understand the prototypal inheritance model in JavaScript.

9:41 — I got some sleep and went through my morning routine. Now I’m going to review the class-based component video from last night before trying to work through my first exercise problem.

9:47 — Here’s an interesting note from Bob Ziroll… I missed it yesterday, but it’s a very interesting thought:

Note: *many* tutorials online will completely skip over functional components because they're technically not necessary. Some will even go so far as to say you *shouldn't* use functional components because of the mental overhead required of switching between the two. I figured it's best for you to get the whole picture and see what a common trend currently is (use functional components whenever you can and use class-based components only when you need to). It's up to you to decide which way you prefer to write your own React code.

9:49 — Bob says that he tends to teach functional components first in his courses because he thinks that they are very easy to understand. In truth, however, there are some things that class-based components are able to do that functional components simply can’t. For example, implementing state and life cycle methods.

9:52 — Bob recommended that we go check out a Scrimba ES6 course if we need to get more comfortable with class syntax in JavaScript. I searched Scrimba ES6 course and the top result is from none other than Dylan Israel! O\_O

9:54 — Dylan says that within the context of a class, a constructor uses the following basic syntax:

constructor(){

}

9:55 — What is a constructor’s purpose? My interpretation thus far of a constructor, is that it is used to set basic values for any instance of the class that it is referring to.

10:01 — Man this video from Dylan is dense lol. He explains things well, but I’m having to rewind every few seconds to get every little nugget and nuance in this video.

(source video: <https://scrimba.com/p/p4Mrt9/cQnMDHD>)

10:02 — Another keyword we may encounter when using classes is the ‘static’ keyword.

10:03 — Using the static keyword allows us to create a function within our class, and utilize said function without having to actually create an instance of our class.

10:10 — Now let’s talk about get methods. I remember learning about these in a codecademy course I completed a few months ago, but let’s review and try to get a deeper understanding now.

10:12 — A get method is basically a property we’re getting to retrieve some value (in most cases I would imagine this means retrieving a value from a particular instance of our class).

10:21 — Let’s look at the following code snippet taken straight from Dylan’s video:

class Cat extends Animal {

}

10:22 — What is going on here? For a little while I found the above syntax quite intimidating. A couple days at least. It turns out, however, this concept is actually pretty simple. What we’re doing in the above scenario is the following…

#1 — We’re creating a new class called Cat

#2 — Our Cat class is inheriting the properties and methods from the Animal class.

*While by golly that wasn’t so hard was it?*

10:26 — Something interesting to note is that we can actually have a conflict of sorts using this inheritance system. For example, if the default attitude in our Animal class is ‘friendly’ and our Cat class sets attitude to ‘lazy’, which class takes precedence? In this example, the Cat class value of ‘lazy’ would overwrite the value initially set by the Animal class.

10:33 — Now let’s talk about the ‘super’ keyword.

10:34 — When we use a constructor function that is being extended out from a parent, we need to use the super keyword to tell our child class which properties it should inherit from the parent.

10:36 — Haha, was that difficult to understand? Let me try to explain things more clearly with a syntactic example. Recall that we were previously discussing our Cat class extending our Animal class. In this case, we’d be working with the following syntax:

class Cat extends Animal {

constructor (type, attitude, fur)

// Let’s assume the Animal class had type and attitude properties.

super(type, attitude)

this.fur = fur

}

10:42 — The above code would utilize super to determine which properties we wanted our Cat class to inherit from our Animal class. Of course, because fur was not included as an argument using our super keyword, we would have to define fur when we instantiate a new instance of our Cat class.

10:46 — I’ve finished the video. It’s been a good session to start the day. I’m going to commit this to GitHub and then maybe go exercise or eat breakfast. See you later!

14:42 — I ate breakfast, walked a couple miles, took care of some other obligations but now I’m back for a quick 15 minute pomodoro session.

14:46 — I briefly reviewed class-based components and now I’m going to practice converting a functional component into a class-based component.

14:56 — I’ve completed most of the conversion successfully, but I’m having a small error related to perhaps an unexpected use of const or incorrect syntax with new Date().

19:06 — I’m back. Let’s see if we can figure out this error.

19:17 — Because it had been several hours since I looked at the code, Scrimba refreshed everything and I had to do the exercise over again.

19:29 — This is so bizarre. It doesn’t matter what I do. I can’t seem to get my method to display the hours. I can’t even get it to log to the console. Here’s my current code:

const hours = new Date().getHours()

console.log(hours)

I’ve changed this up a bit. I’ve tried to use let or var, to no use. I’ve also tried using the this keyword but I’m a bit stumped now to be honest.

19:47 — It took forever, but I finally got everything working. Here was my final solution:

class Greeting extends React.Component {

render() {

const hours = new Date().getHours();

let timeOfDay;

if (hours < 12) {

timeOfDay = "morning"

} else if (hours >= 12 && hours < 17) {

timeOfDay = "afternoon"

} else {

timeOfDay = "night"

}

return (

<h1>Good {timeOfDay} to you, sir or madam!</h1>

)

}

}

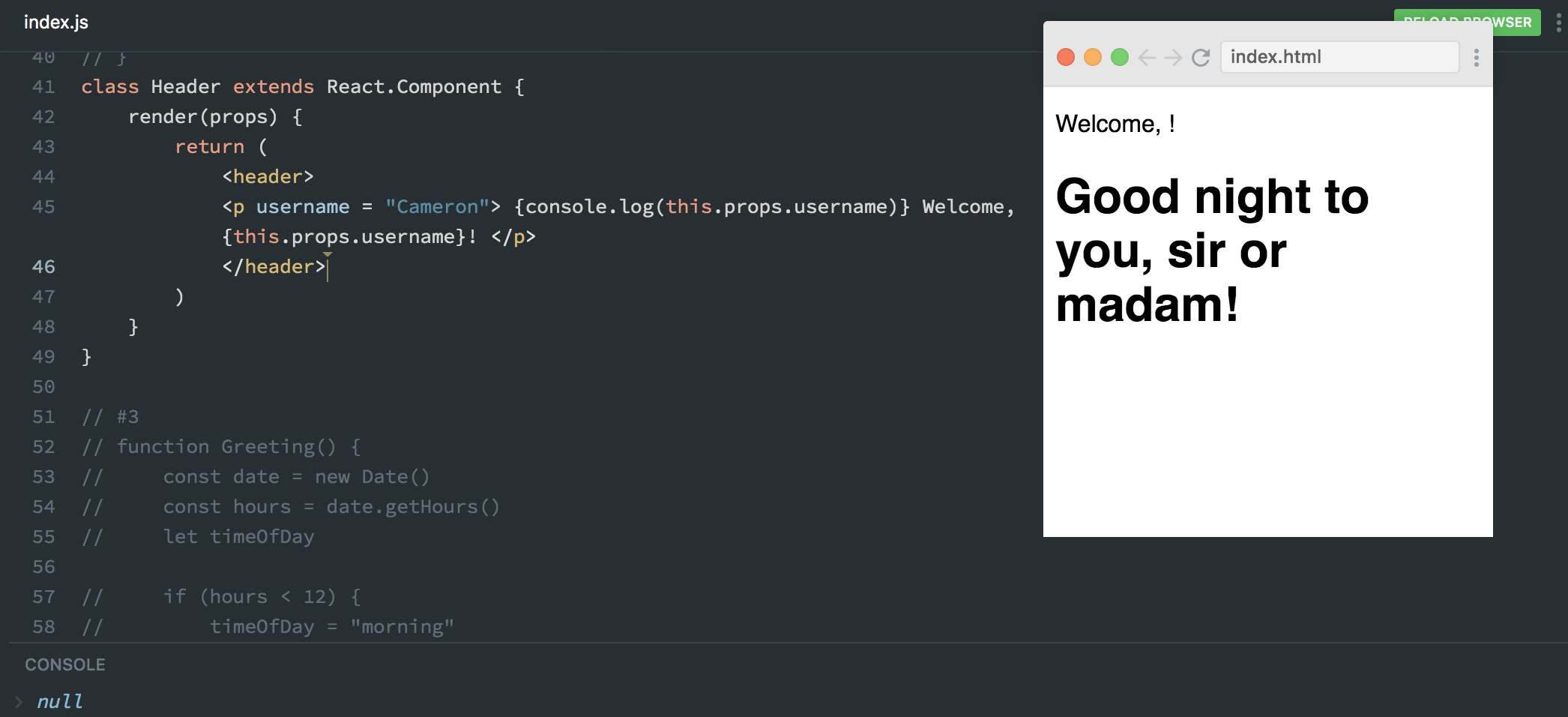
19:48 — It seems the problem (perhaps one of many haha) was that earlier I was defining a method outside of render *()*

*{*

*}*

but I was forgetting to call that method from within render. I made the mistake of calling my created method from outside render which just doesn’t work.

19:54 — I’m so disappointed. I thought I was done with this challenge and I felt proud and ready to take a picture. Turns out, I still have another bug to fix lol. I need to get the app to properly display my username when it says Hello, \_\_\_\_\_\_\_\_. As of now, the username I’m passing is not displaying.

19:59 — I’m hanging by a thread. Attention beginning to wane hahaha.

20:00 — I don’t know why but it says that the value of username is null. It does not seem to matter whether I attempt to initialize username within the header tags or p tags.

20:06 — Oh man I’m stuck. I’ve rewatched the previous video in the course several times and it’s gotten me close but he doesn’t seem to cover exactly what I need to do in order to use props correctly with class-based functions.

20:07 — I’m going to go back and watch the original props video from several days or maybe even a week ago to see if there are any gems that can help me here.

20:09 — I’m going to take a break and finish watching that video after dinner.

21:53 — I just finished dinner and a 30 minute walk around the city. Now let’s get some more work done.

21:58 — OMG. Everything refreshed and I have to do the exercise a third time O\_O.

22:01 — Halfway done with the basic parts. I’m so tired of this exercise #halfLol

22:07 — I’m going to watch the video and see what I’ve been missing. I can’t seem to figure out, and feel like I’m just wandering in circles at this point.

22:09 — Say whaaaat? Apparently when we use the render( ) method inside of a class-based component we don’t need to say render(props). We just say render( ) and can directly add props to the component when it is being rendered in the App.js file.

22:11 — In other WTF news, my username didn’t work because I allowed myself to be confused by how he structured the files. Instead of assigning a username when I had a specific instance of my Header component, I wasn’t careful enough and allowed myself to become confused — accidentally assigning/hard-coding a username into the Header component itself rather than a specific instance of the Header component.

22:13 — Uh-oh, I think things are about to get real now. It’s time to learn about state in React.

22:14 — Props are immutable/unchangable.

22:15 — First important note to make. If we want to use state, we can do so using a class-based component. Functional components and state don’t play together ;)

22:16 — The parent class is also sometimes referred to as the super class.

22:19 — Bob describes constructor functions as a place we’re going to be initializing values.

22:21 — Bob says that the first thing we should always do inside of a constructor is make a call to a global function called super( ).

22:22 — Bob says that super( ) is necessary because it goes to the parent or super class and brings down some “goodies” so that our child class can use said “goodies”.

22:23 — Adding state to a component is *SUPER* easy. Haha! Get it, super? ;)

22:24 — All we have to do to add state to a component is add the property state to “this” and set state equal to an object.

22:27 — The course author mentions that we can pass state down to a child component using props.

22:28 — I’m going to call it a night here. Tomorrow morning I’ll review state and then do the state practice exercise.

**Total time spent coding today: 3 hours 40 minutes**

**Total time spent coding thus far in May 2019: 15 hours 47 minutes**

**Total lifetime hours of coding: 511 hours 41 minutes**