Code Fellows

Class 24
Context API

seattle-javascript-401n14

Lab 23 Review

Code Challenge 22 Review

No Code Challenge 23

Focus on completing your **Resume** draft and **personal pitch** draft for your Career Coaching Workshop #1 on Wednesday

(Also I might increment code challenge numbers to better match with class numbers, so next CC will be #25)

Career Coaching Workshop #1

Snowmageddon 2020

Vocab Review!

What is a lifecycle method?

What is a state variable?

What is a hook?

What is a reducer?

When does useEffect run?

What can a custom hook return?

Quick Review

- **Hooks** let us make our functional components as powerful as class components
- `useState` - allows us to create a state variable with a simple **getter** and **setter**
- `useEffect` - allows us to do some action on **component mount**, **update** and **unmount**
  - Bonus: we can limit the “update” action only when *certain variables* change
- `useReducer` - allows us to create a state object with a **reducer function** that determines how to change the state
  - **Dispatch** gives us parameters / info on how to change the state
  - Reducer function has the logic that goes from current state >> new state based on what the dispatch sent
- `use___` - We can modularize any hook logic (usually a combo of useState and useEffect) into a **custom hook** that starts with the word “`use`”

useState

```
const [name, setName] =
  useState('Sarah');
```

getter = name
setter = setName(...)
initial value = 'Sarah'

To set a new name:

```
setName('Billy');
```

```
console.log(name);
  >> Billy
```

useReducer

```
const [state, dispatch] = useReducer(foo, {})
```

getter = state
setter = dispatch + foo
initial value = {} empty object

To set a new state:

```
dispatch({type: 'addName', name: 'Sarah'})
foo(state, dispatch)
{  newState = {...state}
   if (dispatch.type === 'addName')
      newState.name = dispatch.name
   return newState
}
```

```
console.log(state.name) >> Sarah
```

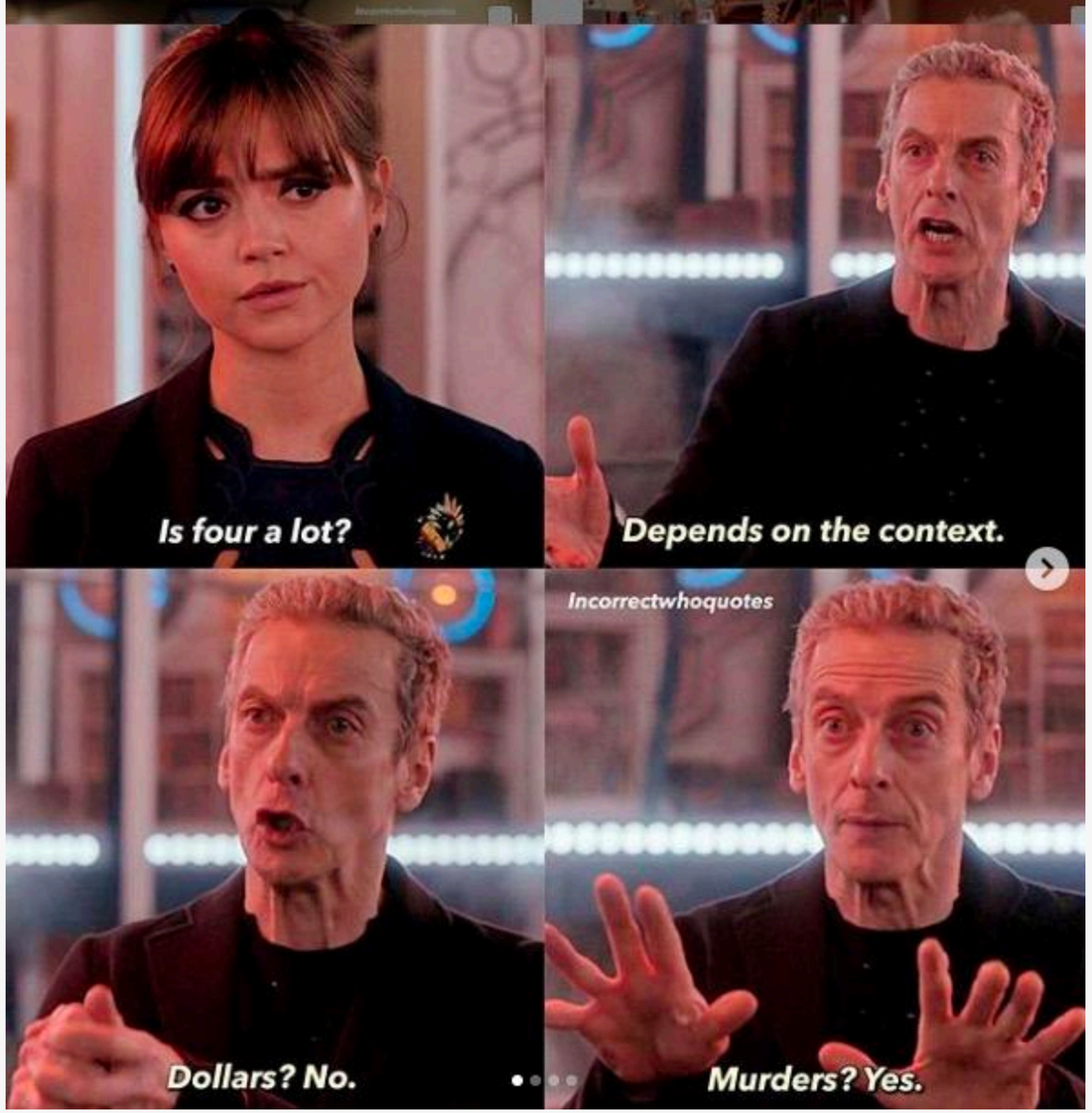
UseContext

```
const value = useContext(myContext);
```

- What is this all about ??? (Let’s find out!)
- Quick summary: It lets you load in objects from another component, and lets you use it as if that object was defined locally

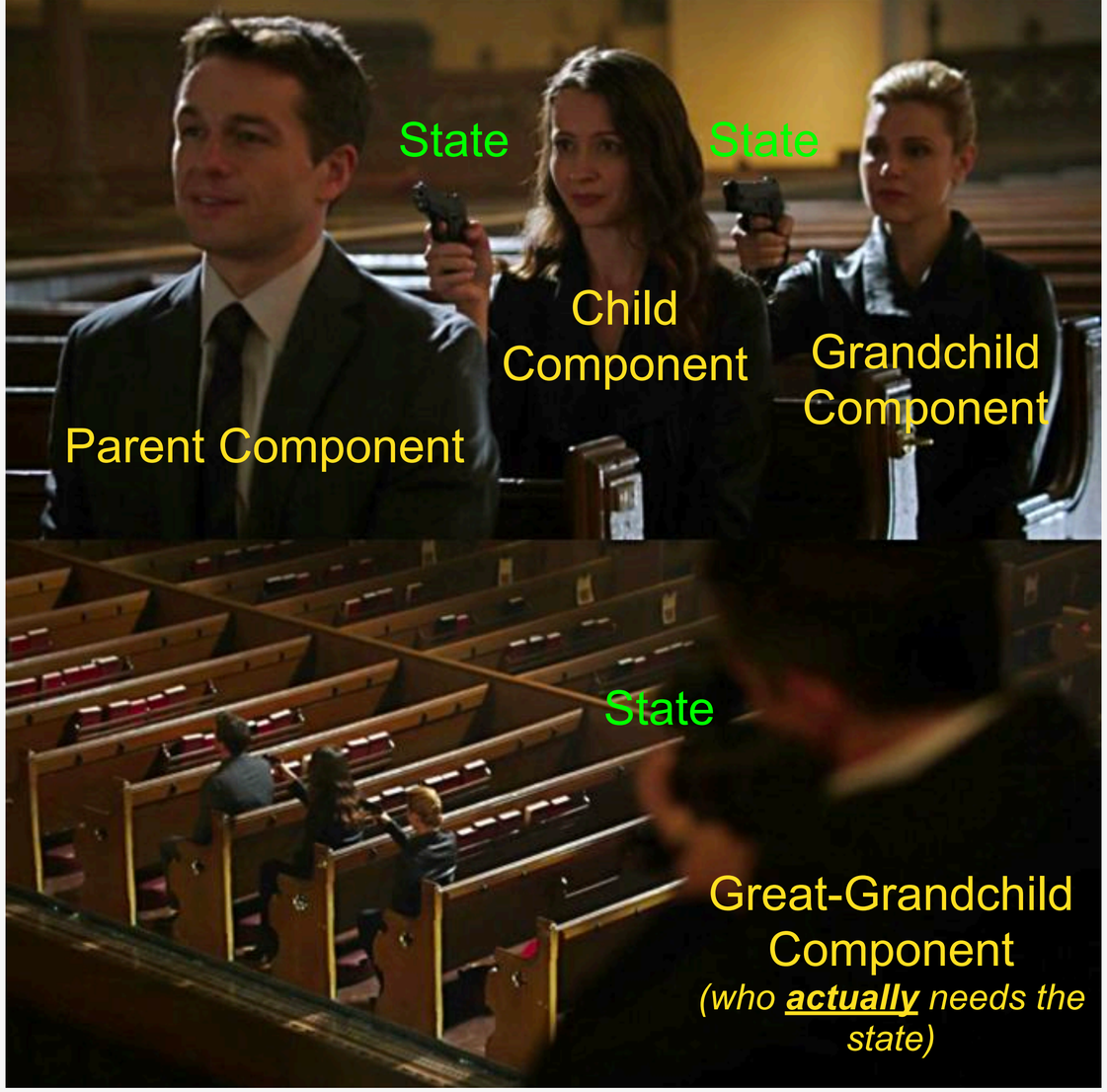
What is Context?

- What “`this`” refers to
  - The class
  - The function
  - The object
- In the case of React, we usually need “`this`” for referring to “`this.state`” in classes
- React has an actual Context object we can create, which lets us share state in a new way!



The Problem

- The current only way to share state is to pass `Parent -> Child` props
- This can be tedious if there is a long chain of descendants that need that state:
  `A` (provides state) `->` `B` (sends state) `->` `C` (sends state) `->` `D` (consumes state)
- Wouldn't it be nice if you could just have the state sent to all descendants automatically?
  `A` (provides state) `->` `B` `->` `C` `->` `D` (consumes state)



React Context *(a solution)*

- Allows us to share state variables WITHOUT having to pass the state through props
- Has an idea of a **Provider** and **Consumer**
  - The Provider exposes the context (typically the state)
  - The Consumer subscribes to the context and is able to use anything the Provider “provides”
  - The consumer can be any descendant!

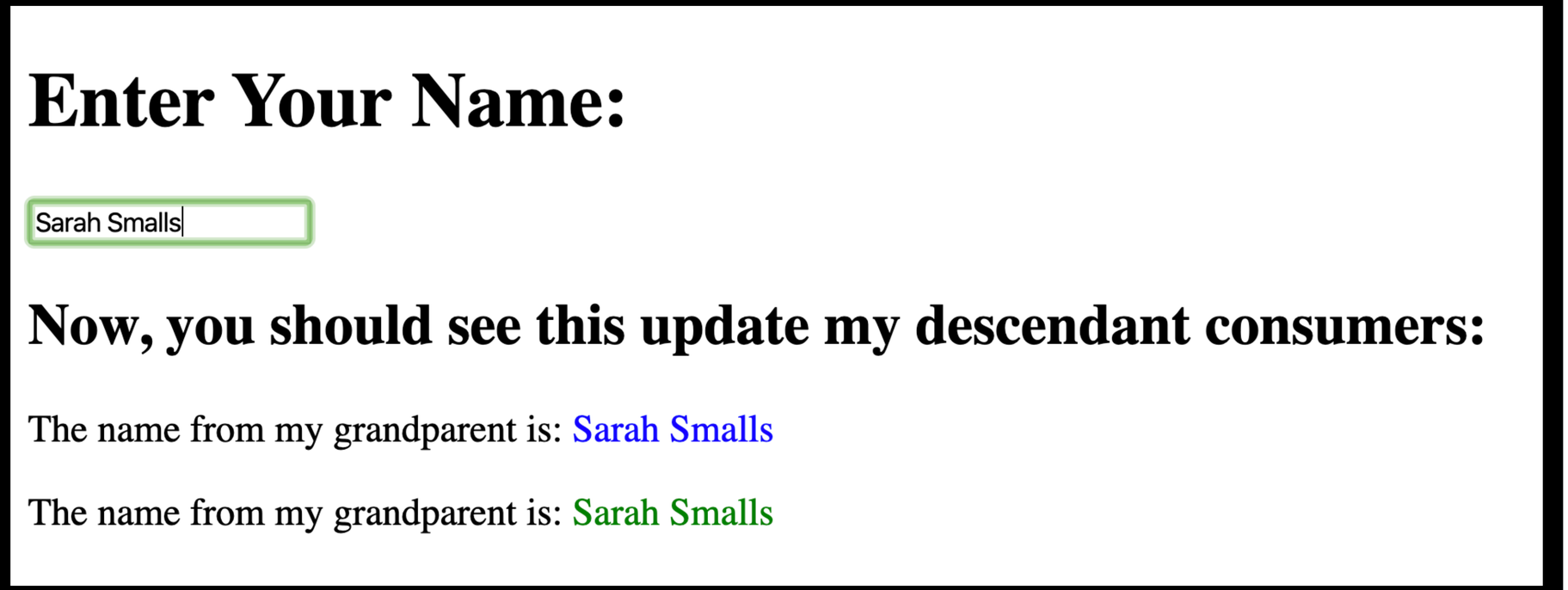
```
MyContext = React.createContext();

class Provider
  this.state = {name: 'Sarah'};

  render:
    <MyContext.Provider value={this.state}>
      {this.props.children}
    </MyContext.Provider>
```

All descendants can became a consumer of the value property (context)

```
class Consumer
  render:
    <MyContext.Consumer>
      {context => {return(<div>{context.name}</div>)}}
    </MyContext.Consumer>
```



Demo

Let’s try to create some context and share that around different components

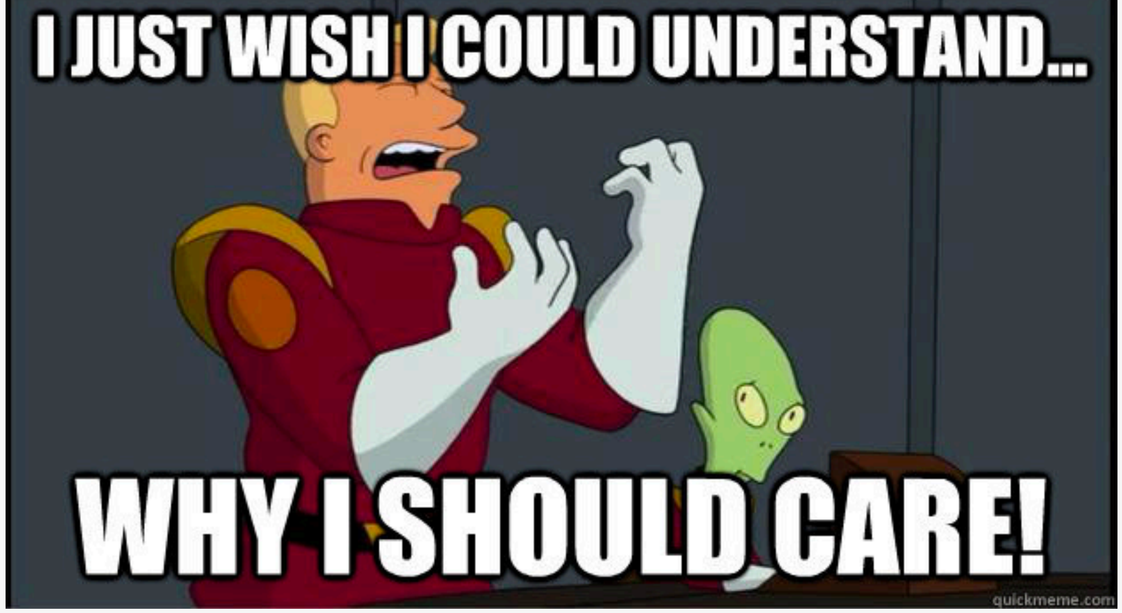
UseContext

```
const value = useContext(myContext);
```

- Here, `value` is going to be whatever you set the Provider's `value` prop to be (usually `this.state`)
- `myContext` is the `React.CreateContext()` initially created (usually in the same file as the Provider

Why Is This Useful?

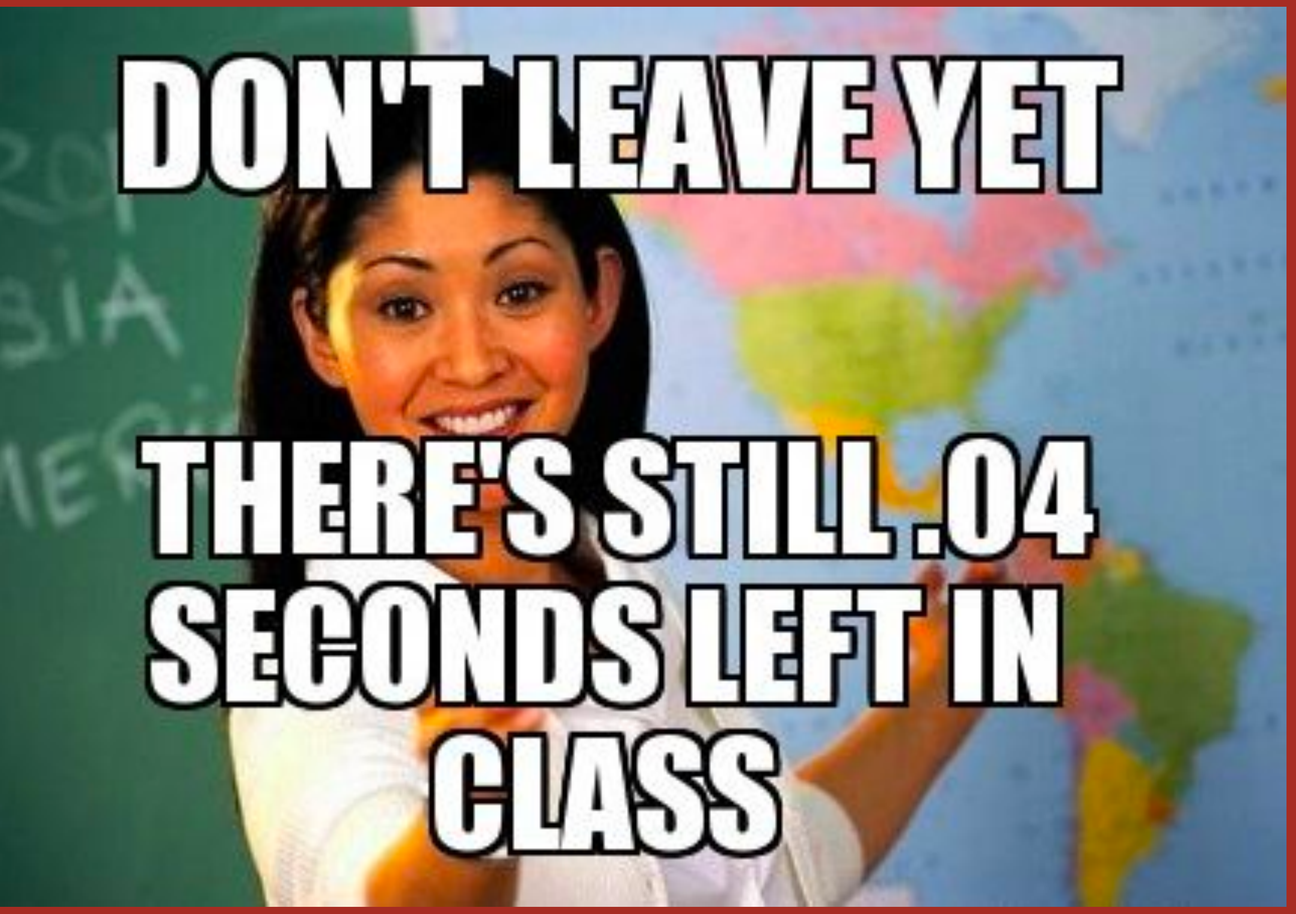
- It allows you to create global settings / theme variables
- Any component can access things like local language, theme color, etc without you having to pass down so many props
- Allows for cleaner sharing of stateful data
- You can have multiple contexts! One for theme colors, one for language, etc



Lab 24 Preview

What's Next:

- Due by Midnight Tonight:
    - **Learning Journal 24**
    - **Partner Power Hour Report #6**
- Due by Midnight Sunday:
    - **Feedback Week 16**
- Due by Tuesday January 14th, 6:30pm:
    - **Lab 24**
    - **Reading Class 25**
- Next Class on Tuesday: **Class 25 - Login and Auth**



Questions?