Code Fellows

# Class 22
# Hooks API

seattle-javascript-401n14

# Welcome Back!

# Lab 21 Review

# Code Challenge 21 Review

# Vocab Review!

# What is a component?

# What is a prop?

# What is a state variable?

# What does parent and child component mean?

# What is `props.children` used for?

# What is Sass?

# What We've Learned in React

- How to build **components** that contain JavaScript and HTML code bundled together
- How to write UI tests using **Enzyme** and **shallow mount**, **full mount**, **render** or **snapshot testing**
- How to deploy React apps using **Netlify**
- How to use **prop** variables for parent to child data
- How to use **state** variables for local component data, triggering re-**render**
- How to break up our UI into modular components, utilizing `props.children` for nested custom components
- How to add routing using `react-router` and `<BrowserRouter />`
- How to style using a CSS preprocessor, **Sass**
- How to think about our style in a more code-like, modular format

# What is the difference between class components and functional components?

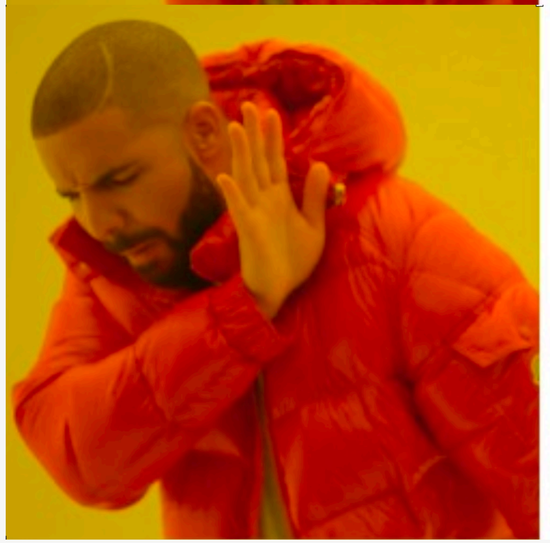
# Benefits of Classes

- Lets you manage a **state** to store data or record user input
- Gives you **lifecycle methods**, such as `componentDidMount`
- Self contained module that you can plug-and-play in many places

# Not-Benefits of Classes

- Classes can't **share their state** with one another, and instead need to rely on passing props around
- We need to manage **context** (make sure the "`this`" reference works)
- Classes can easily get very large and confusing

Stateless functional components



Stateful class components

# The Hooks API makes our functions better

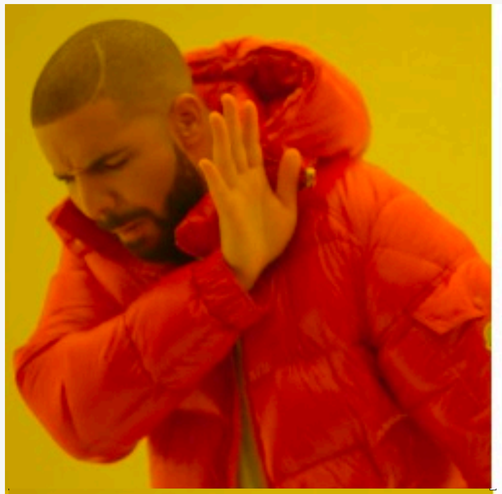
The Hooks API allows us to improve our **functional components,** letting them “hook into” the **state** and **lifecycle** features of a class. This lets you get all the benefits of a class within a simple function!

# Demo

Let’s play around with hooks!

# Benefits of Hooks

- Gives you the **best of both worlds**; the power of a class in the simplicity of a function
- Easily **pass states** between functions
- Cut down on code size and automate / standardize manual work



Stateless functional components

Stateful class components

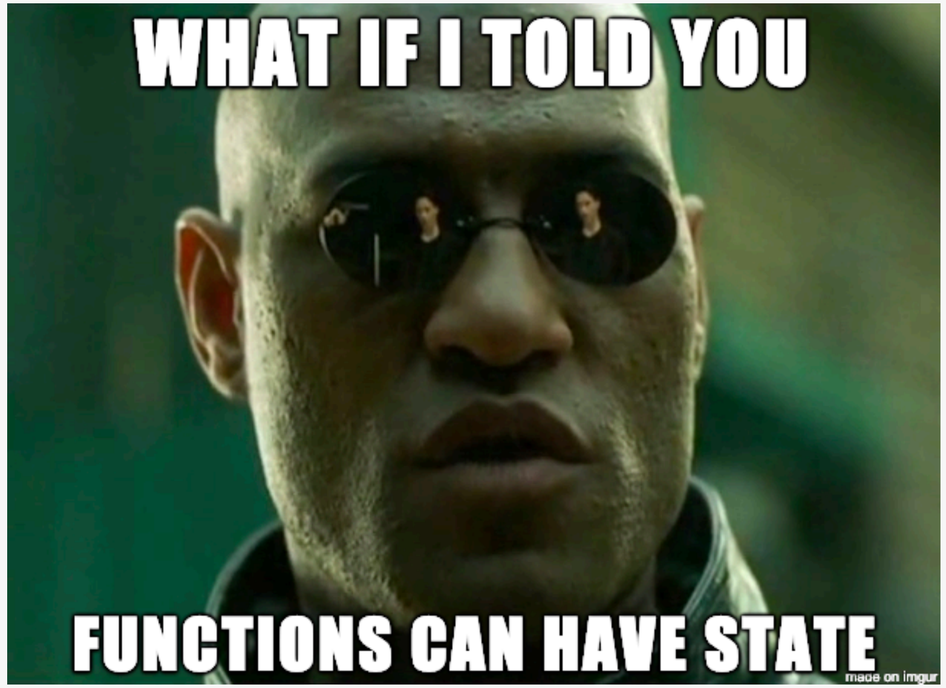
Stateful functional components with Hooks

# UseState

```
const [getter, setter] =
  useState(initValue);
```

- Creates a **getter** and **setter**, and causes a re-render on value change (just like a state variable!)

```
1  import React, { useState } from "react";
2
3  export default function MyFunction(props) {
4      const [firstName, setFirstName] = useState("John");
5      const [lastName, setLastName] = useState("Doe");
6
7      function changeFirstName(e) {
8          setFirstName(e.target.value);
9      }
10
11     function changeLastName(e) {
12         setLastName(e.target.value);
13     }
```



# UseEffect

```
useEffect( () => {

  // function contents

  return () => { // do on unMount }

}, [ optional trigger var(s) ] );
```

- Executes code during component lifecycle
- Runs on every render by default, but you can customize to run when a trigger variable changes

```
7    useEffect(() => {
8        document.title = firstName + " " + lastName;
9    }, [firstName, lastName]);
10
11   useEffect(() => {
12       window.addEventListener("resize", handleResize);
13
14       return () => {
15           window.removeEventListener("resize", handleResize);
16       };
17   });
```

# Use__

```
function useHookName() {

    //function contents

    return value;

}

const hookedVar = useHookName();
```

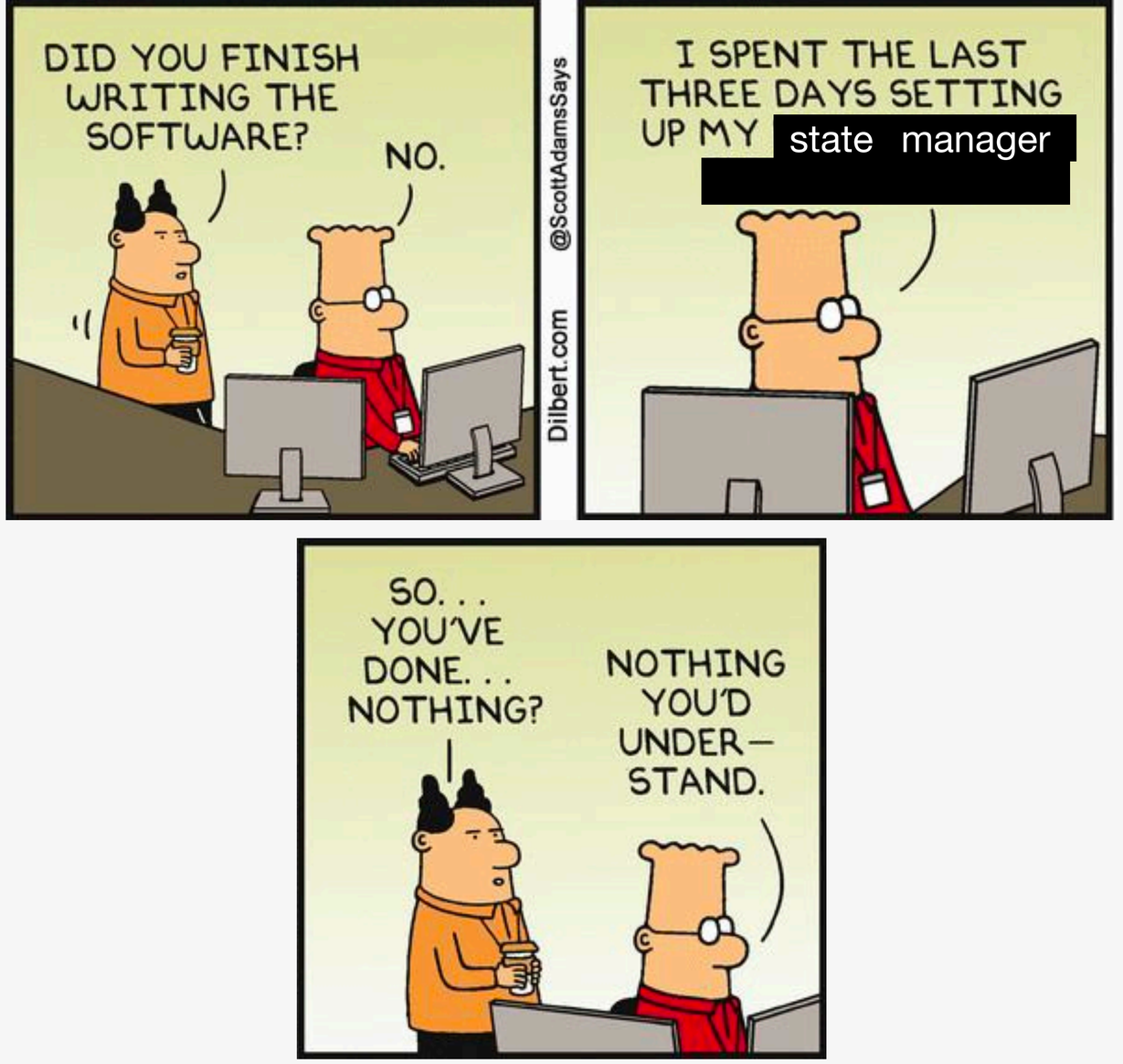
- Define your own hooks by using a function with “`use`” in the beginning of its name
- Great for separating out API logic, animation logic, form handling, etc

# UseReducer

```
const [stateObject, dispatch] =

useReducer(reduceFunction,
initState);
```

- Reducers are more complex state managers which we dispatch a payload to, and the reducer function figures out how to change the state
- Useful when we want to do something more complicated than just updating a value



# UseContext

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

- We didn’t get to this! Just know that it exists - we’ll learn more about it soon!
- Quick summary: It lets you load in objects from another component, and lets you use it as if that object was defined locally

# Rules for Hooks

- Don’t put hooks in any `if` statements, `for` or `while` loops, or any other conditionals

  **“We love hooks unconditionally”**

- Custom hook functions should always start with “`use`”

- Don’t use hooks in classes

# Lab 22 Preview

# What's Next:

- Due by Midnight Tonight:
    - **Learning Journal 22**
    - **Partner Power Hour Report**
- Due by Midnight Sunday:
    - **Feedback Week 14**
    - **Career Coaching Workshop #2 Prep**
- Due by Tuesday January 7th, 6:30pm:
    - **Lab 22**
    - **Reading Class 23**
- Next Class on Saturday: **Class 23 - Custom Hooks: Sockets and Fetch**


# Questions?