Before we begin...

- Open up these slides:
  - https://bit.ly/2IE9zTD

React

Learning Objectives

- Effectively **use** props in React
- Effectively **use** state in React
- **Use** event listeners in React
- **Identify** common lifecycle methods and their purpose
- **Use** lifecycle methods
- **Use** APIs with React context
- **Use** DotEnv effectively

Agenda

- DotEnv
- React
  - Props
  - Events
  - State
  - Lifecycle Methods
  - APIs

A quick review

- React
  - Props
  - Events
  - State
  - Lifecycle Methods

Project Time!

React

Events

Events

```
const React = require("react");
const ReactDOM = require("react-dom");

class Hello extends React.Component {
  render() {
    const { name } = this.props;
    return (
        <h1 onClick={() => alert(`${name} was clicked!`)}>
            Hello {name}!
        </h1>
    )
  }
}

ReactDOM.render(<Hello name="Bill" />, document.getElementById("roo
```

Events

```
const React = require("react");
const ReactDOM = require("react-dom");

class Hello extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    alert(`${this.props.name} was clicked!`);
  }
  render() {
    return <h1 onClick={this.handleClick}>Hello {this.props.name}!<
  }
```

More Events

```
const React = require("react");
const ReactDOM = require("react-dom");

class Hello extends React.Component {
  constructor(props) {
    super(props);
    this.handleMouseEnter = this.handleMouseEnter.bind(this);
    this.handleRightClick = this.handleRightClick.bind(this);
  }
  handleMouseEnter() {
    alert(`The mouse moved over ${this.props.name}`);
  }
  handleRightClick(event) {
    event.preventDefault();
    alert(`${this.props.name} was right-clicked!`);
  }
  render() {
    const { name } = this.props;
    return (
      <h1
        onMouseEnter={this.handleMouseEnter}
        onContextMenu={this.handleRightClick}
      >
        Hello {name}!
      </h1>
    );
  }
}
```

State

What is state?

- State holds information about the current component (trapped within an instance of the component, can't be accessed from anywhere else)
- It is mutable and is where you add anything that changes (e.g. data and AJAX requests, user inputs)
- It is an object. We can add whatever data we want
- It is initialised in the *constructor*
- It is changed with the *setState* method
  - As soon as you call *setState*, it will re-run the *render method*. It handles the hard work for you!

Click Counter

```
const React = require("react");
const ReactDOM = require("react-dom");

class ClickCounter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { clicks: 0 };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState({ clicks: this.state.clicks + 1 });
  }
  render() {
    return (
      <div>
        <button onClick={this.handleClick}>Click Me!</button>
        <p>Number of clicks: {this.state.clicks}</p>
      </div>
    );
  }
}

ReactDOM.render(<ClickCounter />, document.getElementById("roo
```

Date Printer

```
const React = require("react");
const ReactDOM = require("react-dom");

class DatePrinter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
    setInterval(() => {
      this.setState({
        date: new Date()
      });
    }, 100);
  }
  render() {
    const { date } = this.state;
    return (
      <p>
        {date.toLocaleDateString()} {date.toLocaleTimeString()
      </p>
    );
  }
}

ReactDOM.render(<DatePrinter />, document.getElementById("root
```

Name Printer

```
const React = require("react");
const ReactDOM = require("react-dom");

class InputPrinter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: "" };
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(event) {
    this.setState({ value: event.target.value });
  }
  render() {
    return (
      <div>
        <input
          type="text"
          onChange={this.handleChange}
          value={this.state.value}
          placeholder="Type here!"
        />
        <p>
          You have typed: <strong>{this.state.value}</strong>
        </p>
      </div>
    );
  }
}

ReactDOM.render(<InputPrinter />, document.getElementById("root
```

Sign Up Page

```
const React = require("react");
const ReactDOM = require("react-dom");

class SignUpPage extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "",
      email: "",
      imageURL: ""
    };
    this.createAccount = this.createAccount.bind(this);
    this.updateName = this.updateName.bind(this);
    this.updateEmail = this.updateEmail.bind(this);
    this.updateImage = this.updateImage.bind(this);
  }
  createAccount(e) {
    e.preventDefault();
    console.log(this.state);
  }
```

Conditional Rendering

Login / Logout

```
const React = require("react");
const ReactDOM = require("react-dom");

class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.state = { loggedIn: true };
    this.logIn = this.logIn.bind(this);
    this.logOut = this.logOut.bind(this);
  }
  logOut() {
    this.setState({ loggedIn: false });
  }
  logIn() {
    this.setState({ loggedIn: true });
  }
  render() {
    if (this.state.loggedIn) {
      return <button onClick={this.logOut}>Log Out!</button>;
    }
    return <button onClick={this.logIn}>Log In!</button>;
  }
}

ReactDOM.render(<LoginControl />, document.getElementById("roo
```

Ron Swanson Jokes

```
const React = require("react");
const ReactDOM = require("react-dom");

class RonSwansonJokes extends React.Component {
  constructor(props) {
    super(props);
    this.state = { data: null, error: null };
    fetch("//ron-swanson-quotes.herokuapp.com/v2/quotes")
      .then(r => r.json())
      .then(data => {
        this.setState({ data: data[0] });
      });
  }
  render() {
    const text = this.state.data || "Loading...";
    return (
      <div>
        <h3>{text}</h3>
      </div>
    );
  }
}

ReactDOM.render(<RonSwansonJokes />, document.getElementById("root
```

Ron Swanson Jokes (More)

```
const React = require("react");
const ReactDOM = require("react-dom");

class RonSwansonJokes extends React.Component {
  constructor(props) {
    super(props);
    this.state = { data: null };
    this.getJoke = this.getJoke.bind(this);
    this.handleClick = this.handleClick.bind(this);
    this.getJoke();
  }
  handleClick() {
    this.setState({ data: null });
    this.getJoke();
  }
  getJoke() {
    fetch("//ron-swanson-quotes.herokuapp.com/v2/quotes")
      .then(r => r.json())
      .then(data => {
        this.setState({ data: data[0] });
      });
  }
  render() {
    const text = this.state.data || "Loading...";
    return (
      <div>
        <h3>{text}</h3>
        <button onClick={this.handleClick}>Get another!</button>
      </div>
    );
  }
}

ReactDOM.render(<RonSwansonJokes />, document.getElementById("root
```

DotEnv

Structuring React Code

Breaking into Components

- **F**ocussed
- **I**ndependent
- **R**eusable
- **S**mall
- **T**estable

Container Components

- Manage all state for a part of your application
- Pass state down as props to other components
- This means we need to update state from child components!
- Container components focus on how the app works (what data, using APIs etc.)
- Presentational components use that data but only focus on presentation
  - Receive data exclusively through props!

Lifting State Up

Todo App

```
const React = require("react");
const ReactDOM = require("react-dom");

class TodoView extends React.Component {
  render() {
    return <li>{this.props.todo}</li>;
  }
}

class TodoInputView extends React.Component {
  constructor(props) {
    super(props);
    this.state = { text: "" };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
```

Edit Profile Page

```
import React, { Component } from "react";
import { render } from "react-dom";

class EditProfileForm extends Component {
  render() {
    const {
      name,
      email,
      imageURL,
      updateName,
      updateEmail,
      updateImage
    } = this.props;
    return (
      <form>
        <input
```

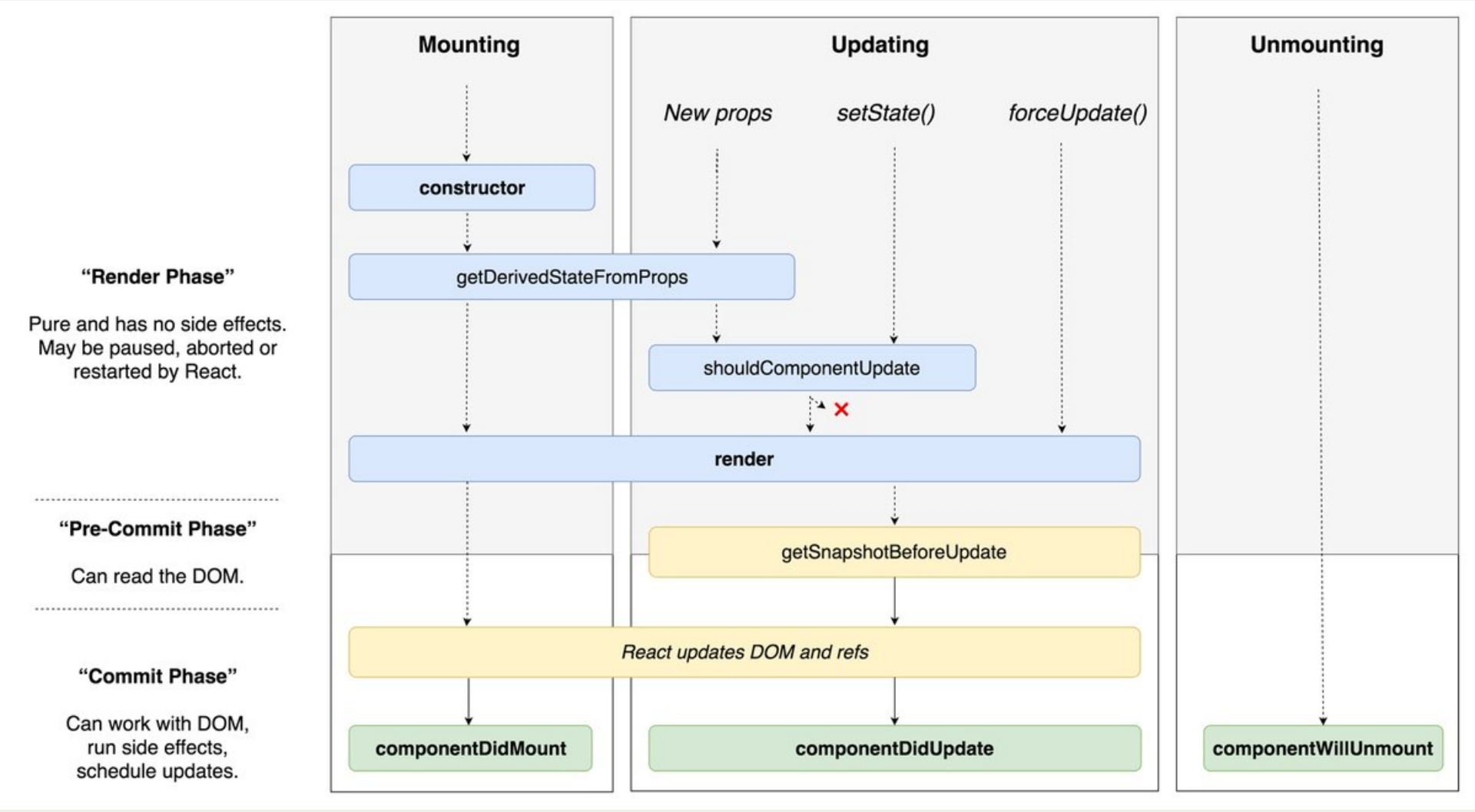
Lifecycle Methods

What are lifecycle methods?

- Lifecycle methods represent the life of a React component
  - From the birth (instantiation)
  - To the death (unmounting)
- They signify important moments
  - When it gets added to the page, when it gets updated, when state changes etc.

What are lifecycle methods?

- They are broken down into four main phases:
  - Initialisation
  - Mounting
  - Updating
  - Unmounting


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.

Resources

- ReactJS Website
- Egghead: Beginner's Guide to React
- React for Beginners
- Codecademy
- Cabin: Learn React
- React Armory: Learn React
- The Road to Learn React
- SurviveJS: React

Homework

- It's Project Time!
- Do some React Tutorials
- Add Babel and Webpack to previous homework!
- Read up on ES2015
- - Translate some of your previous code into it!
- Finish all exercises from class
- Upload your homework to GitHub
- Prepare for next lesson

Homework (Extra)

- Go through some tasks in Exercism
- Get into JavaScript30
- Go through The Modern JavaScript Tutorial
- Read Exploring ES6
- Read Eloquent JavaScript
- Read Speaking JavaScript

What's next?

- Deploying your apps
- Your choice!

Questions?

Feedback time!

Lesson 15: ***React***

https://ga.co/js05syd

Thanks!