

# JAVASCRIPT DEVELOPMENT

*Sasha Vodnik, Instructor*

# HELLO!

1. Pull changes from the `svodnik/JS-SF-14-resources` repo to your computer
2. Open the `18-react` folder in your editor

**JAVASCRIPT DEVELOPMENT**

---

# **INTRODUCTION TO REACT**

# **LEARNING OBJECTIVES**

At the end of this class, you will be able to

- › Understand the roles of model, view, and controller
- › Describe the difference between frameworks and libraries
- › Recognize the primary uses of React
- › Build a React component function
- › Create a React component class
- › Implement composition in a React app
- › Install & use popular React developer tools

# **AGENDA**

- Model View Controller (MVC)
- Frameworks and libraries
- React overview
- Creating React components

---

## INTRODUCTION TO REACT

---

# WEEKLY OVERVIEW

**WEEK 10**

Final project lab / React

**WEEK 11**

Final Project Presentations

# **Final Project Presentation**

**What methods & properties have  
we used to change the DOM?**



---

**JAVASCRIPT DEVELOPMENT**

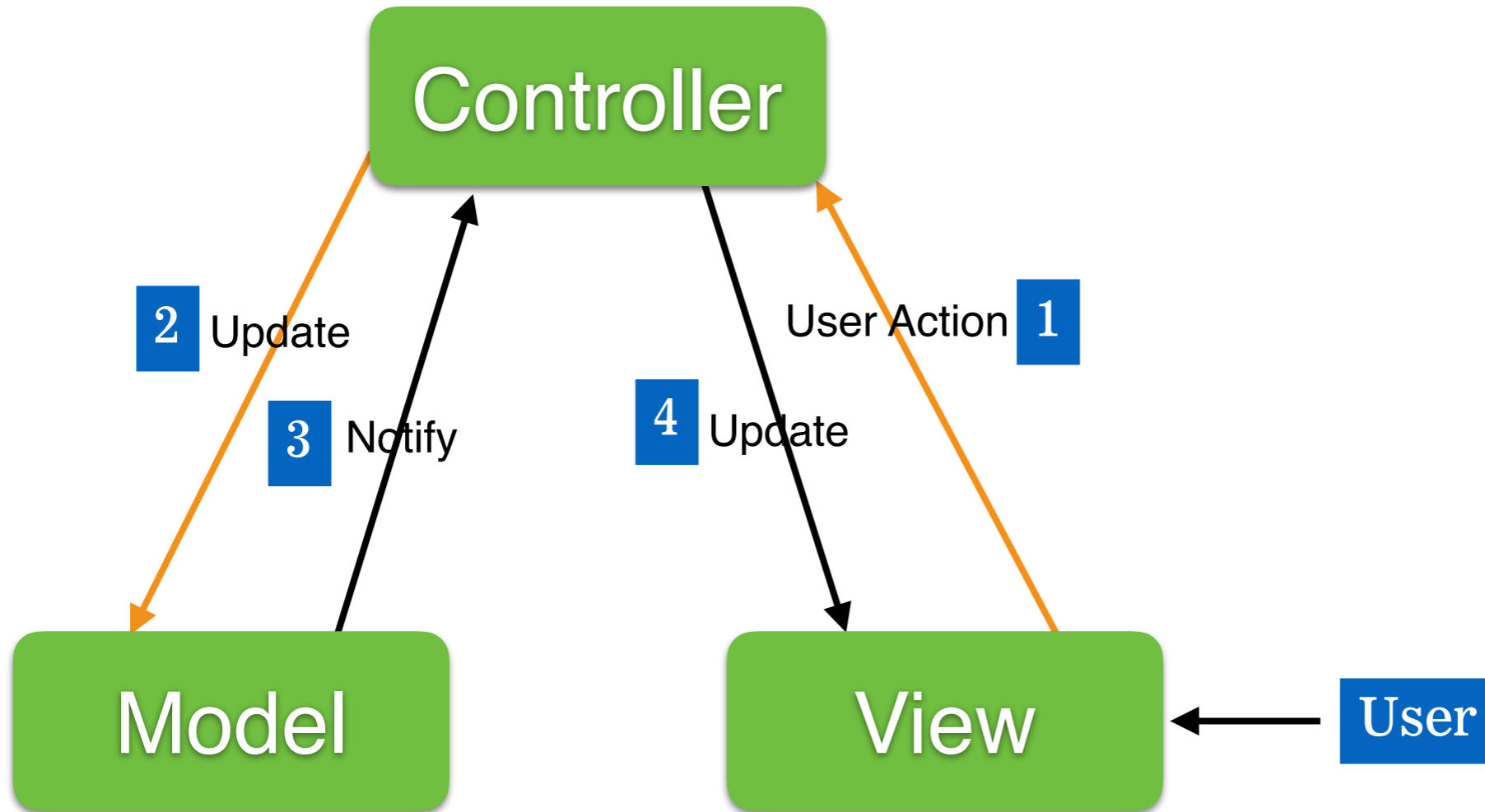
---

# **REACT BASICS**

# **MODEL-VIEW-CONTROLLER (MVC)**

- **Model:** data
- **View:** user interface
- **Controller:** coordinates between model and view

# MODEL-VIEW-CONTROLLER (MVC)



# LIBRARIES VS FRAMEWORKS

**Libraries**



# LIBRARIES VS FRAMEWORKS

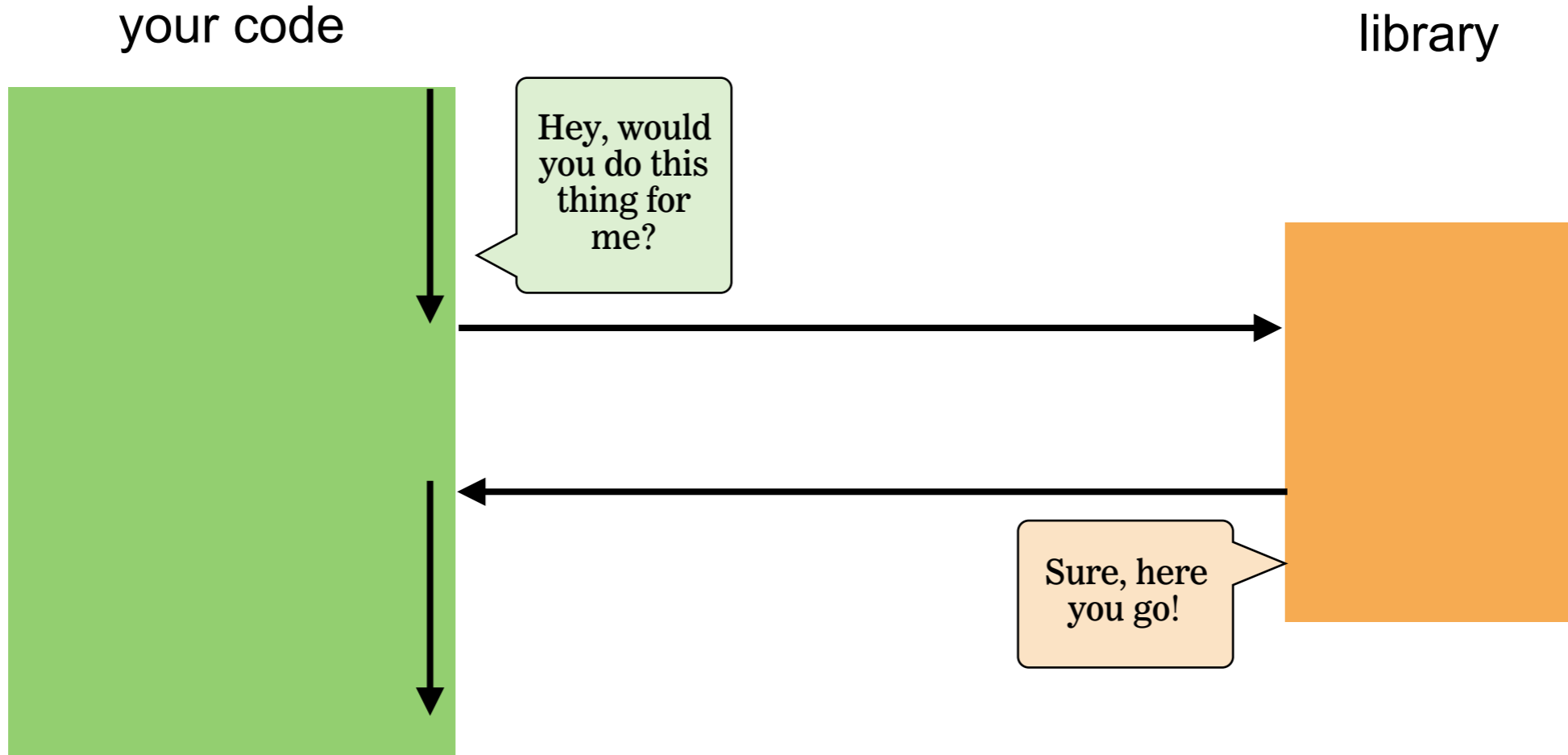
## Libraries



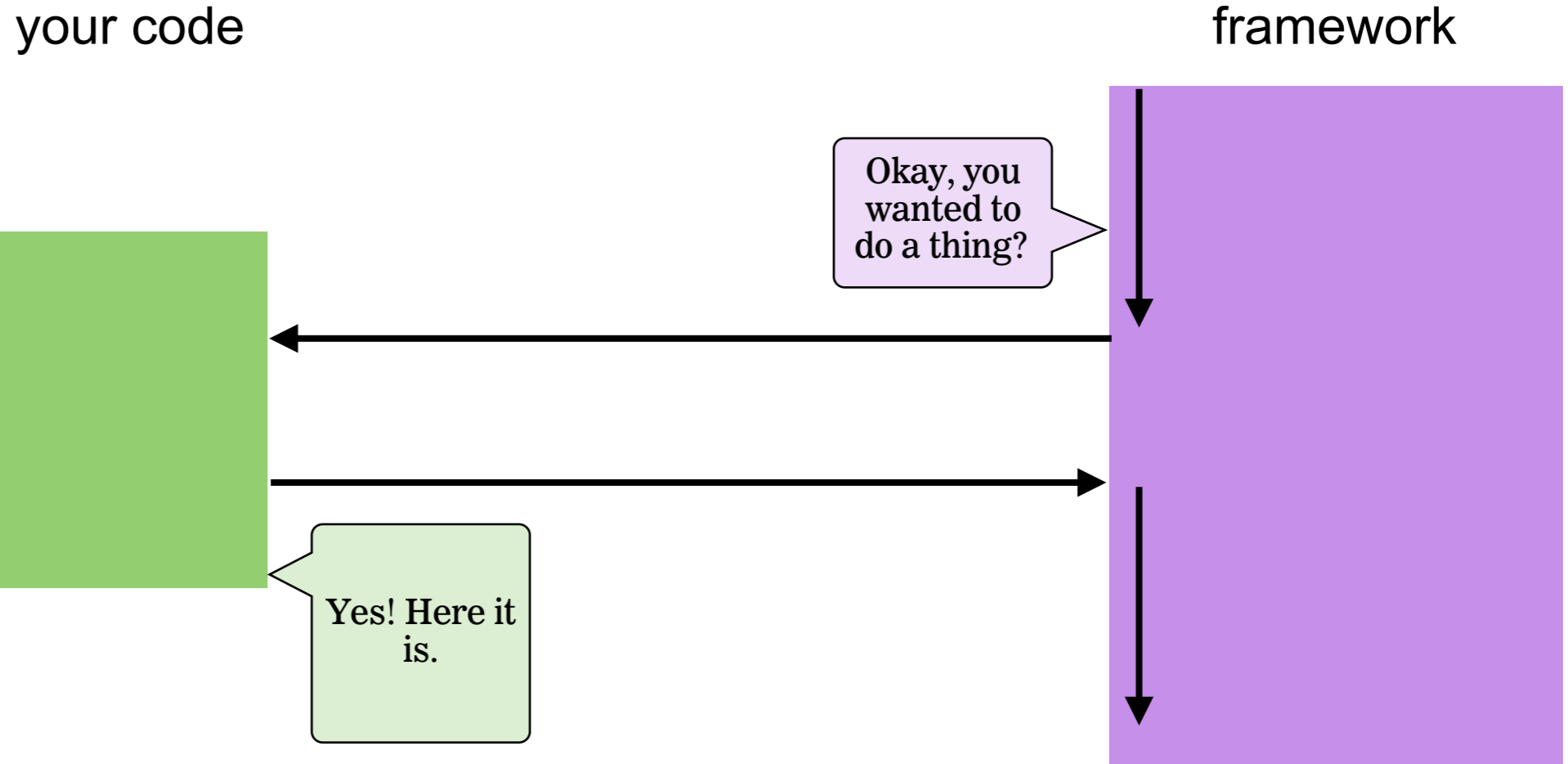
## Frameworks



# YOUR CODE CALLS A LIBRARY



# A FRAMEWORK CALLS YOUR CODE



# LIBRARIES VS FRAMEWORKS

## Libraries

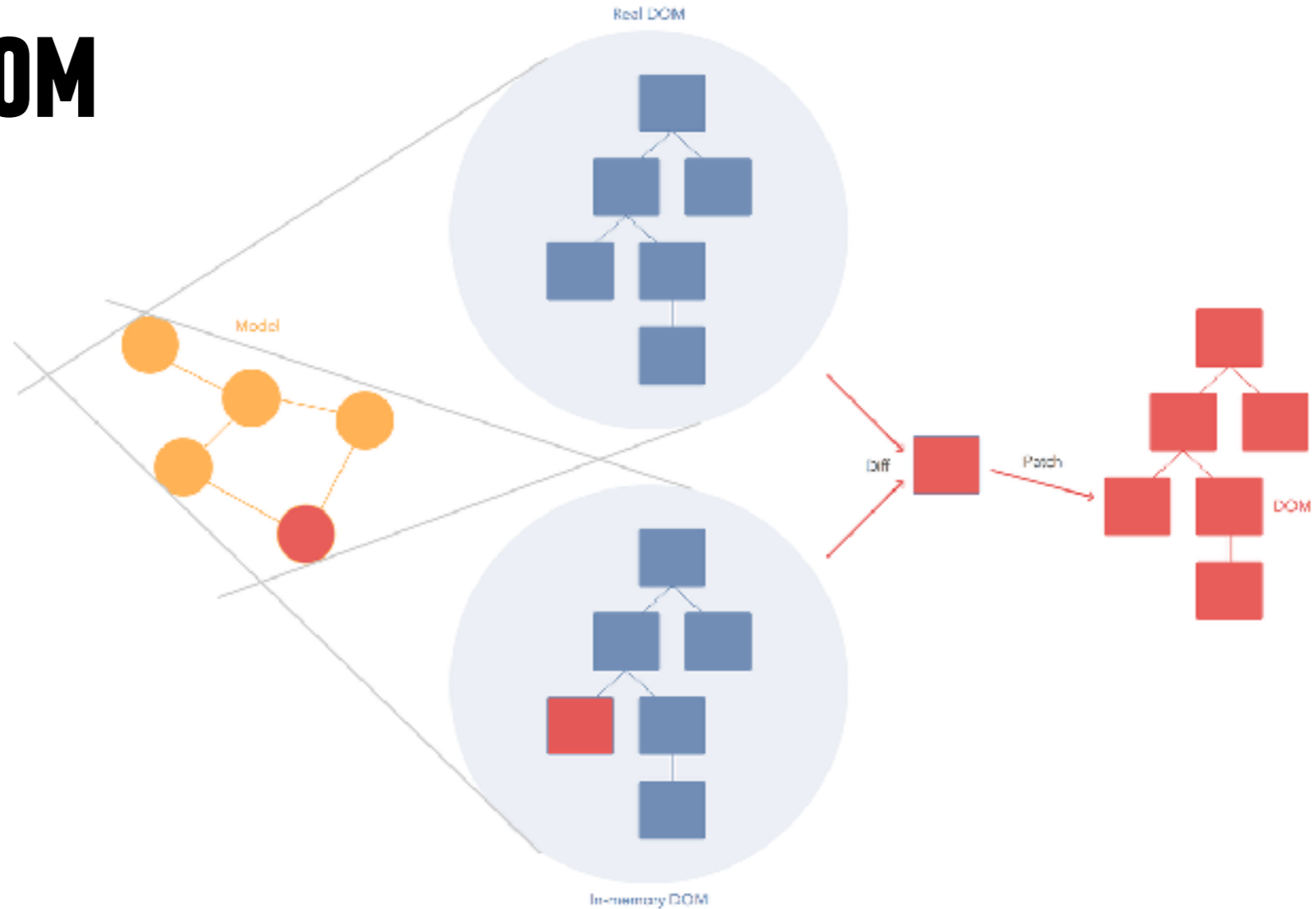


## Frameworks





# VIRTUAL DOM



# COMPONENTS

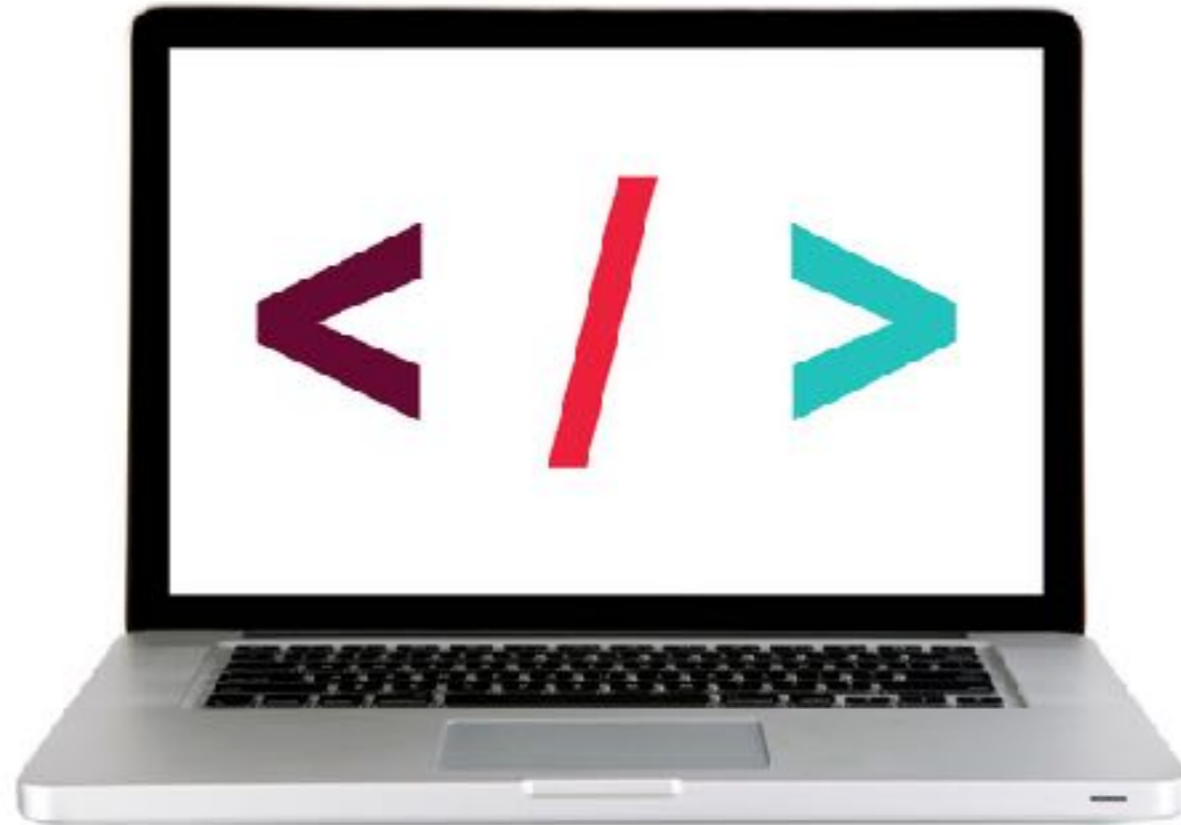
The image shows a screenshot of a Facebook page for 'Facebook for Developers'. Red boxes highlight several key components of the page's layout:

- Page Header:** A navigation bar containing 'Like', 'Share', 'Suggest Edits', and a menu icon.
- Profile Card:** The top section of the page showing the profile picture, name 'Facebook for Developers', and handle '@FacebookforDevelopers'.
- Post Content:** The main body of the post, including the text description, a video player for the 'F8 2018 San Jose, California' event, and engagement metrics like '56K Views', 'Like', and 'Comment'.
- Right Sidebar:** A vertical column of information including 'Sign Up', 'Send Message', 'Community' statistics, 'About' information, 'People' who liked the post, and 'People Also Like' recommendations.
- Left Sidebar:** A navigation menu with options like 'Home', 'Posts', 'Videos', 'About', 'Photos', 'Events', 'Notes', 'Community', and a 'Create a Page' button.

---

**LET'S TAKE A CLOSER LOOK**

---



# REACT DEVELOPER TOOLS



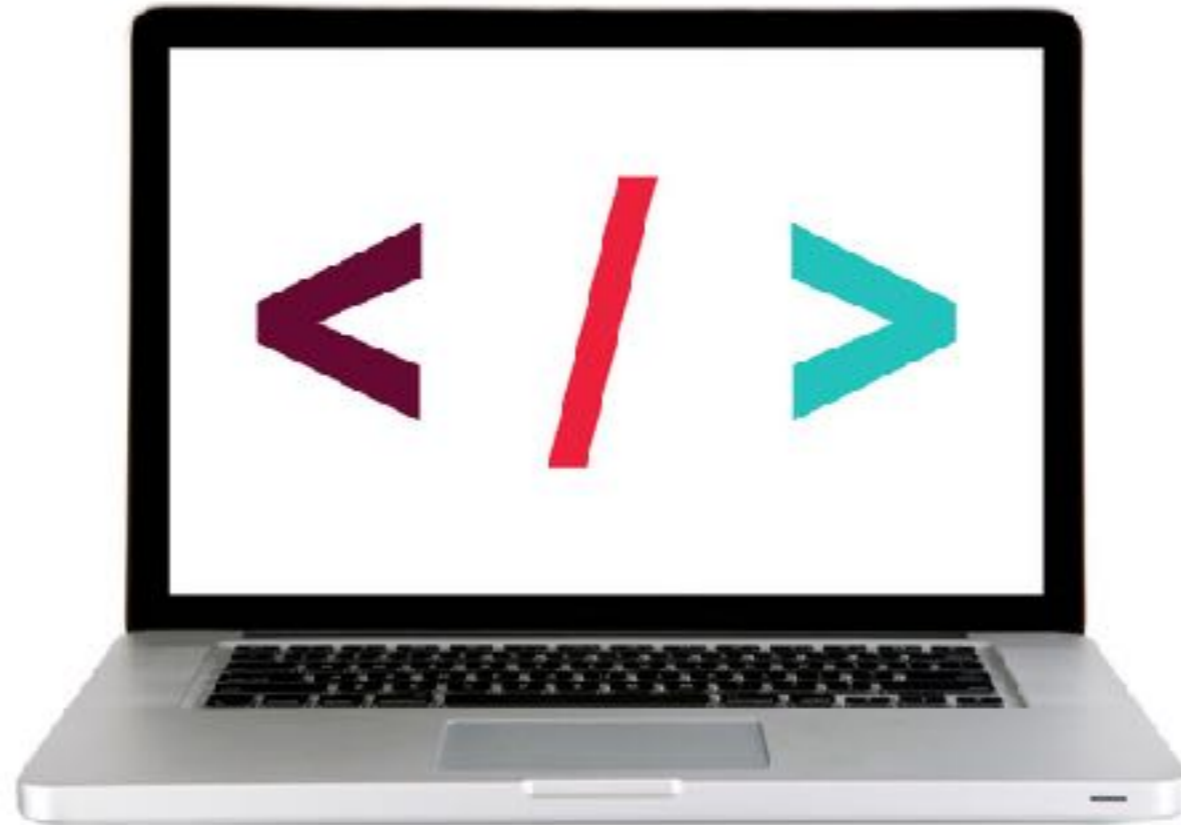
## React Developer Tools

Offered by: Facebook

---

**LET'S TAKE A CLOSER LOOK**

---



**INTRODUCTION TO REACT**

---

# **CREATING REACT COMPONENTS**

---

**INTRODUCTION TO REACT**

---

# **FUNCTIONAL COMPONENTS**

# FUNCTIONAL COMPONENTS

function name has an initial cap

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```



# FUNCTIONAL COMPONENTS

standard parameter name is props

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

# FUNCTIONAL COMPONENTS

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

function always includes a return statement

# FUNCTIONAL COMPONENTS

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

content of the return statement is JSX

# FUNCTIONAL COMPONENTS

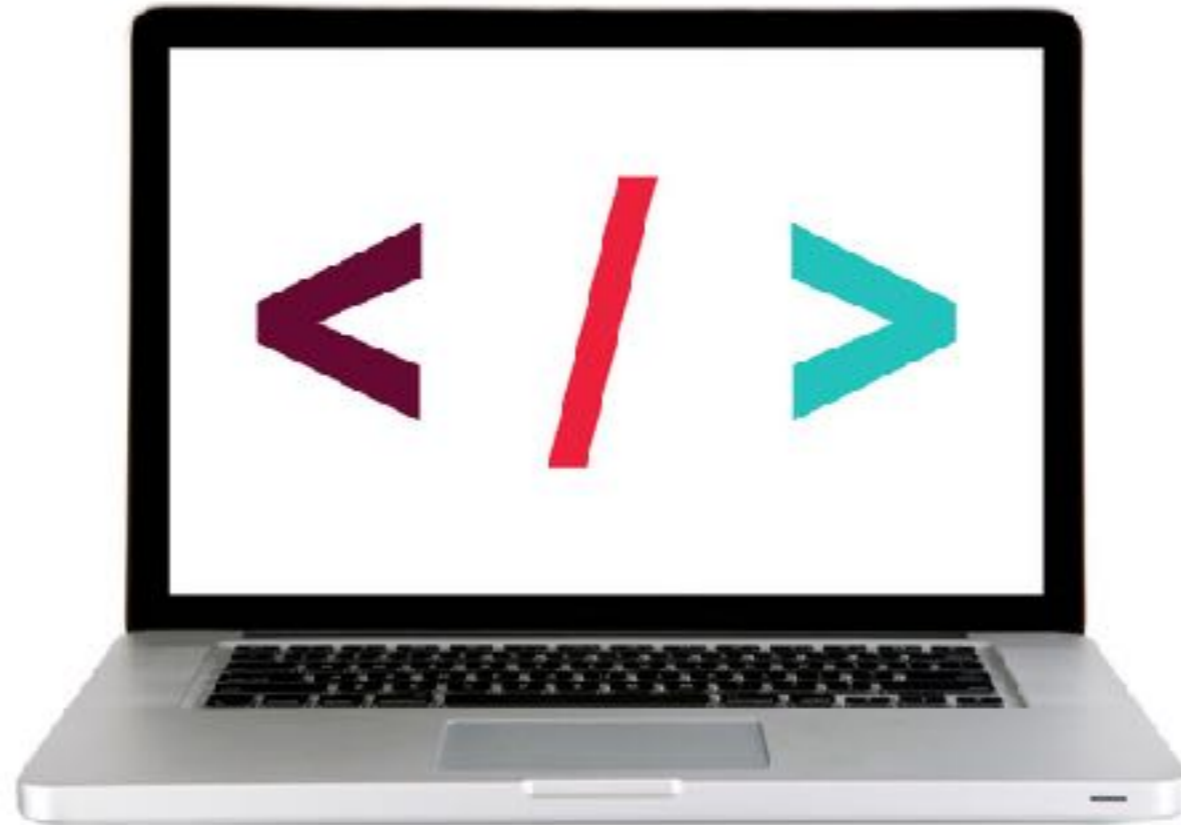
```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

JSX can include JavaScript expressions wrapped in {}

---

**LET'S TAKE A CLOSER LOOK**

---



# JSX

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

could be rendered by React using

```
`<h1>Hello, ${props.name}</h1>`;
```

# JSX

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

could also be rendered by React using

```
createElement('h1').textContent =  
  'Hello, ' + props.name;
```

# LOOPING IN REACT COMPONENTS

```
const listItems = numbers.map((number) =>
  <li>{number}</li>
);

return (
  <ul>{listItems}</ul>
);
```



# LOOPING IN REACT COMPONENTS

```
[  
  'carrot',  
  'tomato',  
  'cucumber'  
]  
  maps to  
[  
  '<li>carrot</li>',  
  '<li>tomato</li>',  
  '<li>cucumber</li>'  
]
```

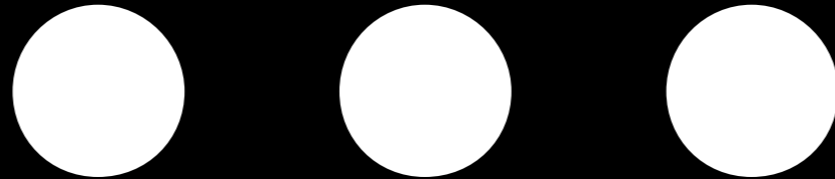
# LOOPING IN REACT COMPONENTS

```
[  
  '<li>carrot</li>',  
  '<li>tomato</li>',  
  '<li>cucumber</li>'  
]
```

rendered as

```
<li>carrot</li>  
<li>tomato</li>  
<li>cucumber</li>
```

# ES6 SPREAD OPERATOR



# ES6 SPREAD OPERATOR

```
{  
  firstName: 'Ben',  
  lastName: 'Hector'  
}  
  
return <Greeting {...props} />;
```

is parsed as

```
return <Greeting firstName="Ben" lastName="Hector" />;
```

---

# EXERCISE — CREATE FUNCTIONAL COMPONENTS

---



## **KEY OBJECTIVE**

---

- ▶ Build a React functional component

## **TYPE OF EXERCISE**

---

- ▶ Solo or in pairs

## **LOCATION**

---

- ▶ `starter-code > 1-functional-component-exercise`

## **TIMING**

---

*10 min*

1. The start file contains the components we've already been working with, along with additional data in the state variable.
2. Create variables storing references to the two new elements in the DOM.
3. Create components to render the contents of the new state properties.
4. Call the render method for each of your two new components.

---

**INTRODUCTION TO REACT**

---

# **CLASS COMPONENTS**

# CLASS COMPONENTS

class name has an initial cap

```
class Welcome extends React.Component {  
  render() {  
    return (  
      <p>Hello, {this.props.name}</p>  
    );  
  }  
}
```

# CLASS COMPONENTS

component class is always based on `React.Component`

```
class Welcome extends React.Component {  
  render() {  
    return (  
      <p>Hello, {this.props.name}</p>  
    );  
  }  
}
```



# CLASS COMPONENTS

class definition always calls the render() function

```
class Welcome extends React.Component {  
  render() {  
    return (  
      <p>Hello, {this.props.name}</p>  
    );  
  }  
}
```

# CLASS COMPONENTS

render function call always includes a return statement

```
class Welcome extends React.Component {  
  render() {  
    return (  
      <p>Hello, {this.props.name}</p>  
    );  
  }  
}
```

# CLASS COMPONENTS

content of the return statement is JSX

```
class Welcome extends React.Component {  
  render() {  
    return (  
      <p>Hello, {this.props.name}</p>  
    );  
  }  
}
```

# CLASS COMPONENTS

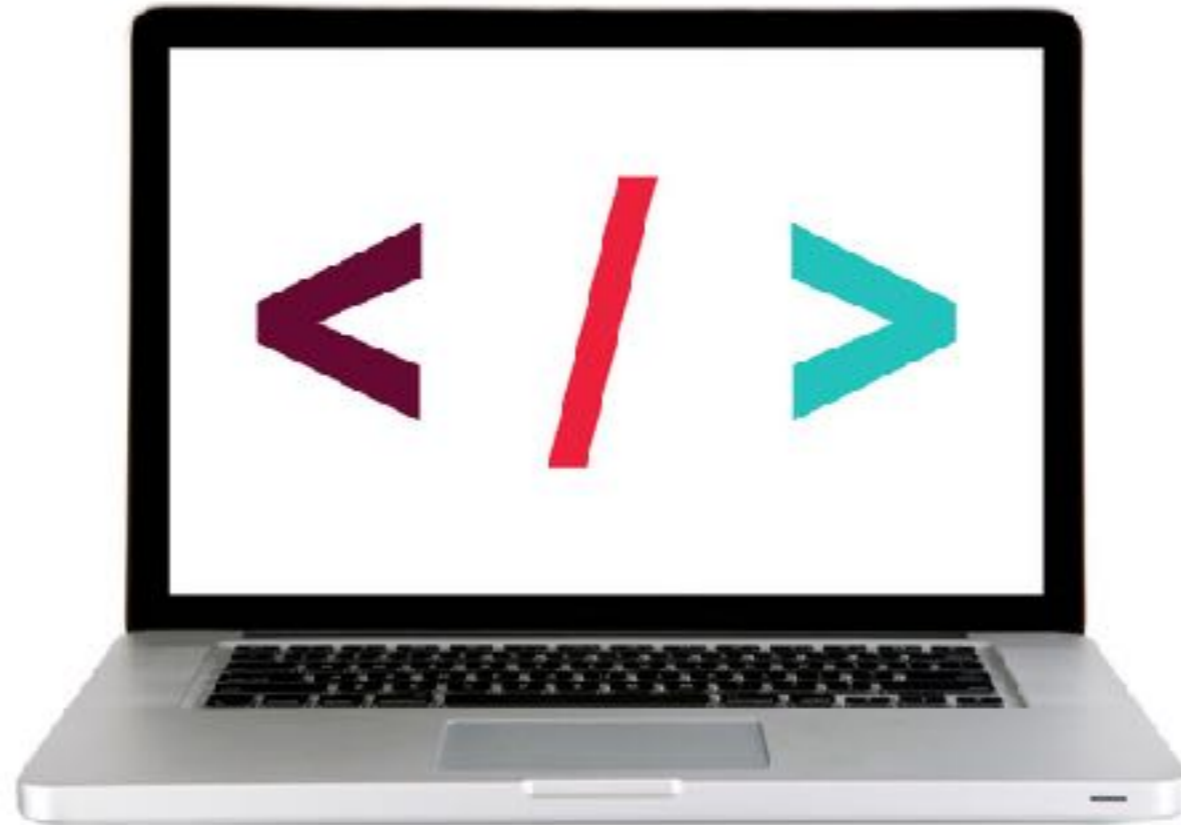
JSX can include JavaScript expressions wrapped in {}

```
class Welcome extends React.Component {  
  render() {  
    return (  
      <p>Hello, {this.props.name}</p>  
    );  
  }  
}
```

---

**LET'S TAKE A CLOSER LOOK**

---



---

# EXERCISE — CREATE CLASS COMPONENTS

---



EXERCISE

## **KEY OBJECTIVE**

---

- ▶ Build a React class component

## **TYPE OF EXERCISE**

---

- ▶ Solo or in pairs

## **LOCATION**

---

- ▶ `starter-code > 3-class-component-exercise`

## **TIMING**

---

*10 min*

1. The start file contains the components we've already been working with, along with additional data in the state variable.
2. Create variables storing references to the two new elements in the DOM.
3. Create components to render the contents of the new state properties.
4. Call the render method for each of your two new components.

---

**INTRODUCTION TO REACT**

---

# **COMPOSITION**

## **COMPOSITION**

- In parent class, call each child with JSX using element syntax
- Pass necessary props as attributes, referencing `this.props`
- For child classes, move data manipulation outside of `render()` method, and reference the result instead
- Call `ReactDOM.render()` only on parent class



# COMPOSITION

```
class Menu extends React.Component {  
  render() {  
    return (  
      <nav>  
        <ul>  
          {this.props.menu.map(function(item, index) {  
            return <li key={index}>{item}</li>;  
          })}  
        </ul>  
      </nav>  
    );  
  }  
}
```

# COMPOSITION

```
class Menu extends React.Component {
  items = this.props.menu.map(function(item, index) {
    return <li key={index}>{item}</li>;
  });
  render() {
    return (
      <nav>
        <ul>
          {this.items}
        </ul>
      </nav>
    );
  }
}
```

# COMPOSITION

```
class Menu extends React.Component {  
  items = this.props.menu.map(function(item, index) {  
    return <li key={index}>{item}</li>;  
  });  
  render() {  
    return (  
      <nav>  
        <ul>  
          {this.items}  
        </ul>  
      </nav>  
    );  
  }  
}
```

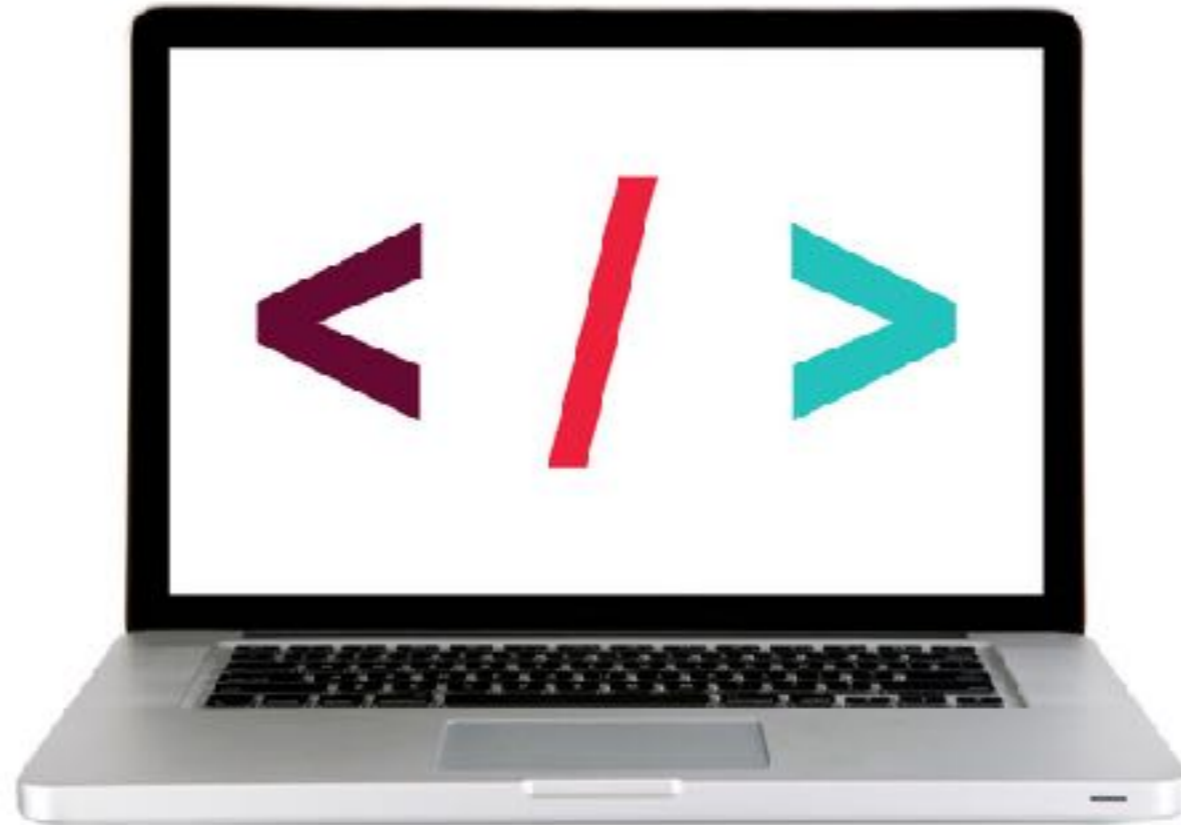
# COMPOSITION

```
class App extends React.Component {
  render() {
    return (
      <div className="container">
        <Menu menu={this.props.menu} />
        <Heading title={this.props.title} />
        <Articles articles={this.props.articles} />
        <Footer footer={this.props.footer} />
      </div>
    );
  }
}
```

---

**LET'S TAKE A CLOSER LOOK**

---



---

# EXERCISE — REUSE COMPONENTS WITH COMPOSITION

---



## **KEY OBJECTIVE**

---

- Implement composition in a React app

## **TYPE OF EXERCISE**

---

- Solo or in pairs

## **LOCATION**

---

- `starter-code > 5-composition-exercise`

## **TIMING**

---

- 20 min*
1. Open `CawCaw comp.png` and examine the view you'll be creating.
  2. Follow the instructions in `script.js` to build the `User`, `Content`, `Date`, and `App` components.

---

**INTRODUCTION TO REACT**

---

**REUSE**

# REUSE

```
class Favorites extends React.Component {
  items = this.props.favorites.map(function(item, index) {
    return <img key={index} src={item.src} />;
  });
  render() {
    return (
      <nav>
        {this.items}
      </nav>
    );
  }
}
```



# REUSE

```
class Suggestions extends React.Component {
  items = this.props.suggestions.map(function(item, index) {
    return <img key={index} src={item.src} />;
  });
  render() {
    return (
      <nav>
        {this.items}
      </nav>
    );
  }
}
```

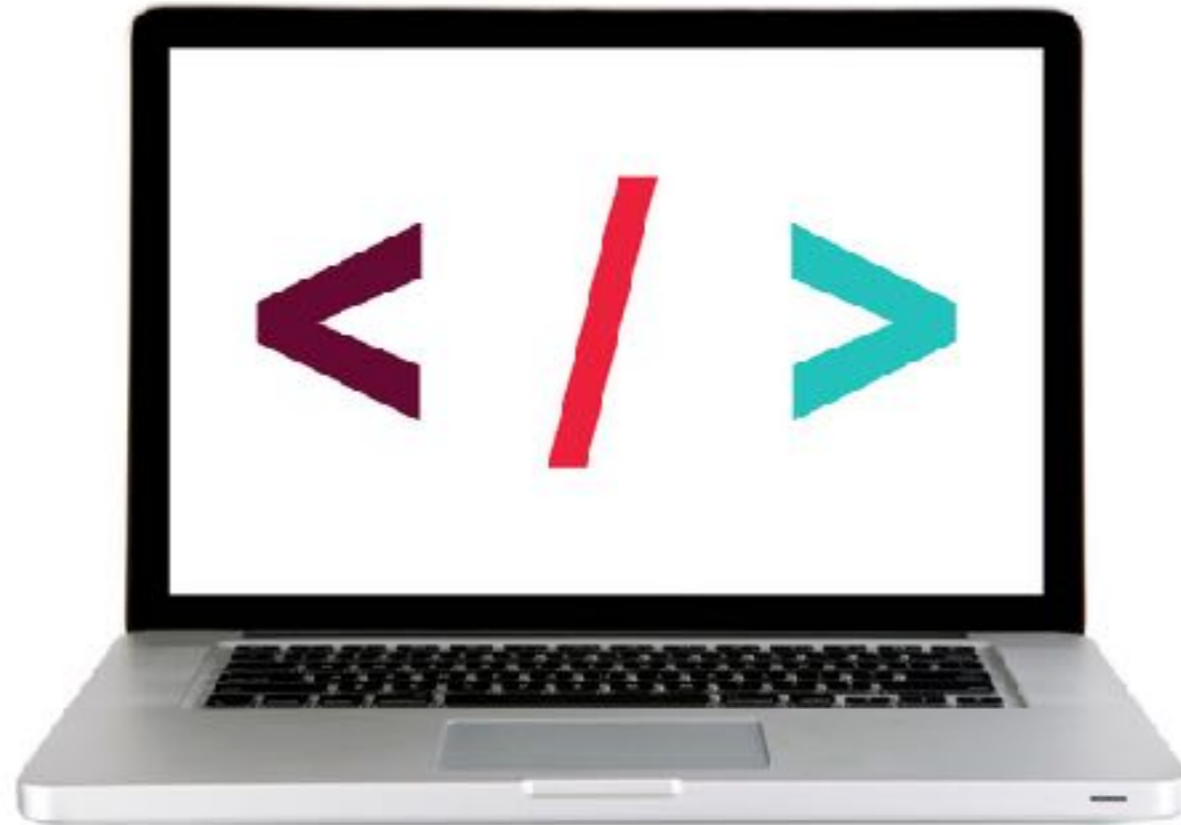
# REUSE

```
class Gallery extends React.Component {
  items = this.props.image.map(function(item, index) {
    return <img key={index} src={item.src} />;
  });
  render() {
    return (
      <nav>
        {this.items}
      </nav>
    );
  }
}
```

---

**LET'S TAKE A CLOSER LOOK**

---



---

**INTRODUCTION TO REACT**

---

# **THINKING IN REACT**

## THINKING IN REACT

Data returned from a JSON API

```
[  
  {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},  
  {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},  
  {category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketball"},  
  {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},  
  {category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"},  
  {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}  
];
```

Mock from designer

Only show products in stock  

Name	Price
<strong>Sporting Goods</strong>	
Football	\$49.99
Baseball	\$9.99
<b>Basketball</b>	\$29.99
<strong>Electronics</strong>	
iPod Touch	\$99.99
<b>iPhone 5</b>	\$399.99
Nexus 7	\$199.99

# THINKING IN REACT

## DRAW SOME BOXES

The diagram illustrates a user interface for a product list. It features a search bar, a checkbox for filtering products, and a table of products. The components are highlighted with colored boxes: a blue box for the search and filter area, a green box for the table, and red boxes for individual table rows. A cyan box highlights the category headers within the table.

Name	Price
<b>Sporting Goods</b>	
Football	\$49.99
Baseball	\$9.99
<b>Basketball</b>	\$29.99
<b>Electronics</b>	
iPod Touch	\$99.99
<b>iPhone 5</b>	\$399.99
Nexus 7	\$199.99

# THINKING IN REACT

## NAME THE BOXES (SEMANTICALLY!)

- FilterableProductTable
- SearchBar
- ProductTable
- ProductCategoryRow
- ProductRow

Name	Price
<b>Sporting Goods</b>	
Football	\$49.99
Baseball	\$9.99
<b>Basketball</b>	\$29.99
<b>Electronics</b>	
iPod Touch	\$99.99
<b>iPhone 5</b>	\$399.99
Nexus 7	\$199.99

# THINKING IN REACT

## MAKE A HIERARCHY

components!

- FilterableProductTable
  - ▶ SearchBar
  - ▶ ProductTable
    - » ProductCategoryRow
    - » ProductRow

Name	Price
<b>Sporting Goods</b>	
Football	\$49.99
Baseball	\$9.99
<b>Basketball</b>	\$29.99
<b>Electronics</b>	
iPod Touch	\$99.99
<b>iPhone 5</b>	\$399.99
Nexus 7	\$199.99



---

# EXERCISE

---



## **KEY OBJECTIVE**

---

- ▶ Create a component hierarchy

## **TYPE OF EXERCISE**

---

- ▶ Individual/pair

## **TIMING**

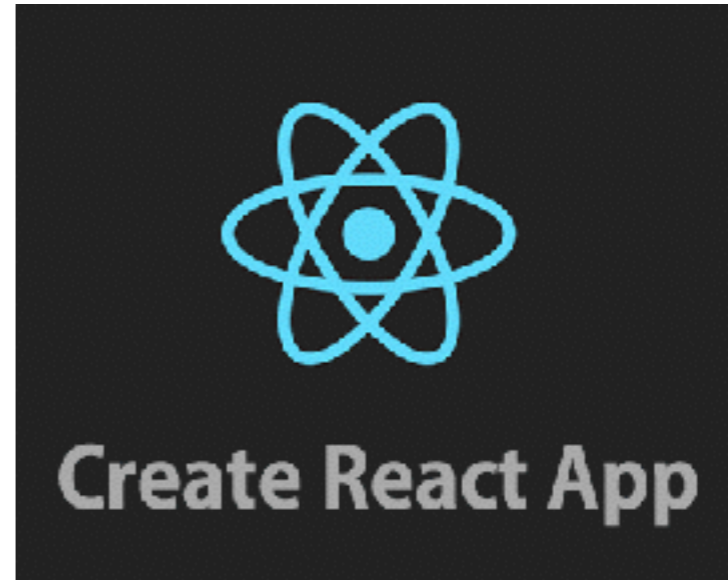
---

*10 min*

1. Choose a section of your favorite website
2. Write down the component hierarchy (remember the steps: 1. Mock, 2. Boxes, 3. Name, 4. Hierarchy)
3. Don't forget to use semantic names!

# CREATE-REACT-APP

- › npm package
- › generates files & folder structure



# **Exit Tickets!**

**(Class #18)**

## **LEARNING OBJECTIVES – REVIEW**

- Understand the roles of model, view, and controller
- Describe the difference between frameworks and libraries
- Recognize the primary uses of React
- Build a React component function
- Create a React component class
- Implement composition in a React app

# **Q&A**