COMPONENTS AND ADVANCED JSX

Use Multiline JSX in a Component

In this lesson, you will learn some common ways that JSX and React components work together. You'll get more comfortable with both JSX and components, while picking up some new tricks.

Take a look at this HTML:

```
<blockquote>
  <p>
    The world is full of objects, more or less interesting;
I do not wish to add any more.
  </p>
  <cite>
    <a target="_blank"
      href="https://en.wikipedia.org/wiki/Douglas_Huebler">
      Douglas Huebler
    </a>
  </cite>
</blockquote>
```

How might you make a React component that renders this HTML?

Select **QuoteMaker.js** to see one way of doing it.

The key thing to notice in `QuoteMaker` is the parentheses in the `return` statement, on lines 6 and 18. Until now, your render function `return` statements have looked like this, without any parentheses:

```
return <h1>Hello world</h1>;
```

However, a multi-line JSX expression should always be wrapped in parentheses! That is why `QuoteMaker`'s return statement has parentheses around it.

Instructions

1.

Here's another quote, formatted in the same way:

```
<blockquote>
  <p>
    What is important now is to recover our senses.
  </p>
```

```
  <cite>
    <a                                          target="_blank"
      href="https://en.wikipedia.org/wiki/Susan_Sontag">
      Susan                                              Sontag
    </a>
  </cite>
</blockquote>
```

In **app.js**, write a React component that renders this HTML. Render your component using `ReactDOM.render()`.

Use **QuoteMaker.js** as a guide! Remember to import `React` and `ReactDOM` at the top of the file, and remember to `ReactDOM.render()` your component at the bottom of the file.

If you don't like this quote, feel free to use a different one.

Checkpoint 2 Passed

Hint

You can copy **QuoteMaker.js** and make changes as needed.

QuoteMaker.js

```
import React from 'react';
import ReactDOM from 'react-dom';

class QuoteMaker extends React.Component {
  render() {
    return (
      <blockquote>
        <p>
          The world is full of objects, more or less interes
ting; I do not wish to add any more.
        </p>
        <cite>
          <a target="_blank"
            href="https://en.wikipedia.org/wiki/Douglas_Hueb
ler">
            Douglas Huebler
          </a>
        </cite>
      </blockquote>
```

```
    );
  }
};

ReactDOM.render(
  <QuoteMaker />,
  document.getElementById('app')
);
```

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

class NewQuoteMaker extends React.Component {
  render() {
    return (
      <blockquote>
        <p>
          What is important now is to recover our senses.
        </p>
        <cite>
          <a target="_blank"
            href="https://en.wikipedia.org/wiki/Susan_Sontag
">
            Susan Sontag
          </a>
        </cite>
      </blockquote>
    );
  };
};

ReactDOM.render(<NewQuoteMaker />, document.getElementById('
app'));
```

Use a Variable Attribute in a Component

Take a look at this JavaScript object named `redPanda`:

```
const redPanda = {
  src:
'https://upload.wikimedia.org/wikipedia/commons/b/b2/Endangered_Red_Panda.jpg',
  alt: 'Red Panda',
  width: '200px'
};
```

How could you render a React component, and get a picture with `redPanda`'s properties?

Select **RedPanda.js** to see one way to do it.

Note all of the curly-brace JavaScript injections inside of the render function! Lines 16, 17, and 18 all use JavaScript injections.

You can, and often will, inject JavaScript into JSX inside of a render function.

Instructions

1.
Select **app.js**.

On lines 1 and 2, import `React` and `ReactDOM`.

Checkpoint 2 Passed

Hint

To import React, you'll write a line that looks like this:

```
import React from 'react';
```

You'll have a line that does this and another line that imports `ReactDOM` from `'react-dom'`.

2.
Starting on line 10, declare a new component class named `Owl`. `Owl` should have a *render function* that returns an empty `<div></div>`.

Feel free to use **RedPanda.js** for guidance.

Checkpoint 3 Passed

Hint

Here's a skeleton for what your component will look like:

```
class Owl extends React.Component {
  render() {
  }
}
```

Use **RedPanda.js** for guidance.

3.

Nest an `<h1></h1>` inside of your `<div></div>`.

Inside of the `<h1></h1>`, put `owl.title`. Remember, you will have to use curly braces to access `owl.title`, since it is a JavaScript property.

Don't forget to wrap the whole multiline JSX expression in parentheses!

Checkpoint 4 Passed

Hint

Your `return` statement will look something like this:

```
return (
  <div>
    <h1></h1>
  </div>
);
```

You'll need to put something inside of the `<h1>`.

4.

Still inside of the `<div></div>`, make a new line after the `<h1></h1>`.

On your new line, add an `<img />` element.

Your `<img />` should have two attributes:

- an `src` of `owl.src`
- an `alt` of `owl.title`

Checkpoint 5 Passed

Hint

After the `<h1></h1>`, put an `<img />` element.

Give it two attributes: `src={owl.src}` and `alt={owl.title}`.

5.

At the bottom of the file, use `ReactDOM.render()` to render an instance of `Owl`.

`ReactDOM.render()`'s second argument should be `document.getElementById('app')`.

Checkpoint 6 Passed

Hint

Call `ReactDOM.render()` with two arguments: `<Owl />` and `document.getElementById('app')`.

RedPanda.js

```
import React from 'react';
import ReactDOM from 'react-dom';

const redPanda = {
  src: 'https://upload.wikimedia.org/wikipedia/commons/b/b2/Endangered_Red_Panda.jpg',
  alt: 'Red Panda',
  width:  '200px'
};

class RedPanda extends React.Component {
  render() {
    return (
      <div>
        <h1>Cute Red Panda</h1>
        <img
          src={redPanda.src}
          alt={redPanda.alt}
          width={redPanda.width} />
      </div>
    );
  }
}

ReactDOM.render(
  <RedPanda />,
  document.getElementById('app')
);
```

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';


const owl = {
  title: 'Excellent Owl',
  src: 'https://content.codecademy.com/courses/React/react_photo-owl.jpg'
};

// Component class starts here:
class Owl extends React.Component {
  render() {
    return (
      <div>
        <h1>
          {owl.title}
        </h1>
        <img
          src={owl.src}
          alt={owl.title} />
      </div>
    );
  };
};

ReactDOM.render(<Owl />, document.getElementById('app'));
```

Put Logic in a Render Function

A `render()` function must have a `return` statement. However, that isn't *all* that it can have.

A `render()` function can also be a fine place to put simple calculations that need to happen right before a component renders. Here's an example of some calculations inside of a `render` function:

```
class Random extends React.Component {
  render() {
```

```
    //  First,  some  logic  that  must  happen
    //              before              rendering:
    const  n  = Math.floor(Math.random()  * 10  + 1);
    //          Next,          a return          statement
    //            using            that            logic:
    return  <h1>The  number  is  {n}!</h1>;
  }
}
```

Watch out for this common mistake:

```
class  Random  extends  React.Component  {
  //  This  should  be  in  the  render  function:
  const  n  = Math.floor(Math.random()  * 10  + 1);

  render()                                          {
    return  <h1>The  number  is  {n}!</h1>;
  }
};
```

In the above example, the line with the `const n` declaration will cause a syntax error, as is it should not be part of the class declaration itself, but should occur in a method like `render()`.

Instructions

1.

Let's make a `render()` function with some logic in it.

On lines 1 and 2, import `React` and `ReactDOM`.

Checkpoint 2 Passed

Hint

To import React, you'll write a line that looks like this:

```
import React from 'react';
```

You'll also need to import `ReactDOM` from `'react-dom'`.

2.

Starting on line 20, create a new *component class* named `Friend`. Remember, the component class declaration syntax is `class YourClassName extends React.Component {}`

Give your component class the following render function:

```
render()                                                    {

  return                                                    (
    <div>
    </div>
  );
}
```

Checkpoint 3 Passed

Hint

Declare a component class with `class Friend extends React.Component {}`. Inside of the curly braces, copy the `render()` method from the instructions.

3.

Inside the body of the *render* function, before the `return` statement, declare a new variable named `friend`.

Set `friend` equal to either `friends[0]`, `friends[1]`, or `friends[2]`, depending on which friend sounds most appealing to you.

Checkpoint 4 Passed

Hint

Assign `friend` to `friends[0]`, `friends[1]`, or `friends[2]`. Make sure to put that assignment inside of the `render()` function, after the `{` but before `return`.

4.

Inside of the `return` statement, and inside of the `<div></div>`, write a nested `<h1></h1>`.

Inside of the `<h1></h1>`, inject `friend.title`.

Checkpoint 5 Passed

Hint

Find the empty `<div>` element and put `<h1>{friend.title}</h1>` inside.

5.

Still inside of the `<div></div>`, make a new line after the `<h1></h1>`.

On the new line, write an `<img />`.

Give the `<img />` an attribute of `src={friend.src}`.

Checkpoint 6 Passed

Hint

After the `<h1>`, put the `<img />`. It will have a single attribute, `src={friend.src}`.

6.

At the bottom of the file, use `ReactDOM.render()` to render an instance of `Friend`. Use the example code as a guide.

Checkpoint 7 Passed

Hint

Call `ReactDOM.render()` with two arguments: `<Friend />` and `document.getElementById('app')`.

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';


const friends = [
  {
    title: "Yummmmmmm",
    src: "https://content.codecademy.com/courses/React/react_photo-monkeyweirdo.jpg"
  },
  {
    title: "Hey Guys!  Wait Up!",
    src: "https://content.codecademy.com/courses/React/react_photo-earnestfrog.jpg"
  },
  {
    title: "Yikes",
    src: "https://content.codecademy.com/courses/React/react_photo-alpaca.jpg"
  }
];

// New component class starts here:
class Friend extends React.Component {
```

```
  render() {
    const friend = friends[1];
    return (
      <div>
        <h1>
          {friend.title}
        </h1>
        <img
          src={friend.src}
        />
      </div>
    );
  };
};

ReactDOM.render(<Friend />, document.getElementById('app'))
```

Use a Conditional in a Render Function

How might you use a *conditional* statement inside of a `render()` function?

Select **TodaysPlan.js** to see one way of doing it.

Notice that the `if` statement is located *inside* of the render function, but *before* the `return` statement. This is pretty much the only way that you will ever see an `if` statement used in a render function.

Instructions

1.

Select **app.js**. You can see a variable named `fiftyFifty`.

`fiftyFifty` will equal `true` half the time and `false` half the time.

Starting on line 7, write a new component class named `TonightsPlan`.

If `fiftyFifty` is true, then `TonightsPlan` should render this element:

```
<h1>Tonight I'm going out WOOO</h1>
```

If `fiftyFifty` is *false*, then `TonightsPlan` should render this element:

```
<h1>Tonight I'm going to bed WOOO</h1>
```

Use `TodaysPlan` as a guide, but you don't have to stick to it exactly. There are many valid ways to solve this problem using a conditional.

Checkpoint 2 Passed

Hint

Refer to the example in **TodaysPlan.js**. Notice the conditional assignment that starts on line 6 with `let task` and ends on line 11.

Your code in **app.js** will look similar but will use `fiftyFifty` instead of `apocalypse`.

2.

Render an instance of `TonightsPlan` and see what fate has in store.

Checkpoint 3 Passed

Hint

Call `ReactDOM.render()` with two arguments: `<TonightsPlan />` and `document.getElementById('app')`.

TodayPlan.js

```
import React from 'react';
import ReactDOM from 'react-dom';

class TodaysPlan extends React.Component {
  render() {
    let task;
    if (!apocalypse) {
      task = 'learn React.js'
    } else {
      task = 'run around'
    }
```

```
    return <h1>Today I am going to {task}!</h1>;
  }
}

ReactDOM.render(
  <TodaysPlan />,
  document.getElementById('app')
);
```

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

const fiftyFifty = Math.random() < 0.5;

// New component class starts here:
class TonightsPlan extends React.Component {
  render() {
    if (fiftyFifty) {
      return (
        <h1>Tonight I'm going out WOOO</h1>
      )
    }
    else {
      <h1>Tonight I'm going to bed WOOO</h1>
    }
  }
}

ReactDOM.render(<TonightsPlan />, document.getElementById('a
pp'))
```

Use this in a Component

The word `this` gets used in React a lot!

You are especially likely to see `this` inside of the body of a component class declaration. Here's an example:

```
class IceCreamGuy extends React.Component {
  get food() {
    return 'ice cream';
  }

  render() {
    return <h1>I like {this.food}.</h1>;
  }
}
```

In the code, what does `this` mean?

Once you have a guess, scroll down to see the answer.

…

…

…

…

…

…

The simple answer is that `this` refers to an instance of `IceCreamGuy`. The less simple answer is that `this` refers to the object on which `this`'s enclosing method, in this case `.render()`, is called. It is almost inevitable that this object will be an instance of `IceCreamGuy`, but technically it could be something else.

Let's assume that `this` refers to an instance of your component class, as will be the case in all examples in this course. `IceCreamGuy` has two methods: `.food` and `.render()`. Since `this` will evaluate to an instance of `IceCreamGuy`, `this.food` will evaluate to a call of `IceCreamGuy`'s `.food` method. This method will, in turn, evaluate to the string "ice cream."

Why don't you need parentheses after `this.food`? Shouldn't it be `this.food()`?

You don't need those parentheses because `.food` is a *getter* method. You can tell this from the `get` in the above class declaration body.

There's nothing React-specific about getter methods, nor about `this` behaving in this way! However, in React you will see `this` used in this way almost constantly.

`this` in JavaScript can be a difficult concept! Here is a good resource for understanding `this` in JavaScript.

Instructions

1.

On line 6, add a getter method to your class body. Your getter method should have a *name* of `name`, and a *return value* of a string:

```
get name() {
  return 'whatever-your-name-is-goes-here';
}
```

Checkpoint 2 Passed

Hint

This getter will be a "sibling" of the `render()` method and will be defined right above it inside of the `MyName` class.

If your name is Esmeralda, your getter will look like this:

```
get name() {
  return 'Esmeralda';
}
```

2.

Inside of the render function, in between the `<h1></h1>` tags, add the text `My name is _.`.

In place of `_`, get `name` from `this` with `this.name`. Feel free to use the example code as a guide.

Checkpoint 3 Passed

Hint

The `<h1></h1>` is currently empty, but not for long! You'll put `My name is _.` inside, where `_` should be replaced with `{this.name}`.

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyName extends React.Component {
  // name property goes here:
  get name() {
    return 'ANDRES R. BUCHELI'
  }

  render() {
    return <h1>My name is {this.name}.</h1>;
  }
}

ReactDOM.render(<MyName />, document.getElementById('app'));
```

Use an Event Listener in a Component

Render functions often contain event listeners. Here's an example of an event listener in a render function:

```
render()                                                    {
  return                                                    (
    <div                                              onHover={myFunc}>
    </div>
  );
}
```

Recall that an event *handler* is a function that gets called in response to an event. In the above example, the event handler is `myFunc()`.

In React, you define event handlers as *methods* on a component class. Like this:

```
class MyClass extends React.Component {
  myFunc() {
    alert('Stop it. Stop hovering.');
  }

  render() {
    return (
      <div onHover={this.myFunc}>
      </div>
    );
  }
}
```

Notice that the component class has two methods: `.myFunc()` and `.render()`. `.myFunc()` is being used as an *event handler*. `.myFunc()` will be called any time that a user hovers over the rendered `<div></div>`.

Instructions

1.

In **app.js**, find the `<button></button>` inside of the render function.

Give this `<button></button>` an `onClick` attribute. The attribute's *value* should be the `.scream()` method.

Feel free to use the example code as a guide.

Checkpoint 2 Passed

Hint

If you wanted to give the `<button></button>` an `id` attribute of `this.foo`, you would do something like this:

```
<button id={this.foo}>
```

Your code will look similar but will use `onClick` instead of `id` and `this.scream` instead of `this.foo`.

2.

At the bottom of the file, render a `<Button />` using `ReactDOM.render()`. For `ReactDOM.render()`'s second argument, pass in `document.getElementById('app')`.

Once your component renders, click on the button in the browser. Bone-chilling!

Checkpoint 3 Passed

Hint

Call `ReactDOM.render()` with two arguments: `<Button />` and `document.getElementById('app')`.

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

class Button extends React.Component {
  scream() {
    alert('AAAAAAAAHHH!!!!!');
  }

  render() {
    return <button onClick={this.scream}>AAAAAH!</button>;
  }
}

ReactDOM.render(<Button />, document.getElementById('app'));
```

Components Recap

Congratulations! You have finished the unit on React components.

React components are complicated. Their syntax is complicated, and the reasoning behind their syntax is especially complicated.

You have learned a lot about both their syntax and their reasoning. You have learned about component classes and component instances. You have learned about `React.Component`, and about the instructions that you must provide to a component class. You have learned how to `import`, and how to render a component instance.

You have been introduced to some common ways of using JSX in React components. You have rendered components using multiline JSX expressions, logic inside of the render function, a conditional statement, `this`, and an event listener.

You have spent a lot of time studying React components in isolation! Now, it’s time to start learning how components fit into with the world around them.
