React: JSX and Props

Springboard

Goals

- Better understand how to layout React applications
- Use conditionals and loops in JSX

Properties

aka. Props

A useful component is a reusable one.

This often means making it configurable or customizable.

```
const Order = (props) => {
  return (
    <div>
      <p>Your Order: </p>
      <p>I'd like coffee from Starbucks</p>
    </div>
  );
}
```

It would be better if we could configure our order.

Our order will be *I'd like* _____ from _____.

Let's make two "properties":

item
What we would like
restaurant
Where we are getting it from

Demo: Delivery-app

Let's add some props to this element.

props is an object that is defined for each component

demo/delivery-app/Order.js

```
const Order = (props) => {
  return (
    <div>
      <p>Your Order: </p>
      <p>I'd like {props.item} from {props.restaurant}</p>
    </div>
  );
}
```

We don't know what these values are yet, so let's see how to pass in props!

Using the component

demo/delivery-app/App.js

```
const App = () => {
  return (
    <div>
      <Order item="pizza" restaurant="Dominos" />
      <Order item="bread" restaurant="Panera" />
    </div>
  );
};

ReactDOM.render(<App/>,
  document.getElementById("root"));
```

Notice here, that we're passing in values for what the props will be.

The keys are separated using a = followed by the value.

You can either pass in props when you render the component or as default values which we will see soon!

Note: Rendering Multiple Top-Level Elements

Prior to React 16, every component had to render a single top-level element. In newer versions of React, it's possible to render siblings at the top level, but the syntax isn't quite as clean. You're welcome to look into this if you're curious, but all of our Component files will render a single element at the top of their hierarchy.

Properties Requirements

- Properties are for configuring your component
- Properties are immutable
- Properties can be strings:

```
<Employee name="Marcia" title="CTO" />
```

- For other types, embed JS expression using the curly braces:

```
<Employee name="Amanda" salary={ 170000 }
    hobbies={ ["running", "swimming", "gardening"] } />
```

Using Properties

- Get to properties inside class with ***propertyName***
- Properties are immutable -- cannot change!

Conditionals in JSX

Your functional components can render:

- a **single valid** DOM object (`return <div>...</div>`)
- an array of DOM objects (but don't do this yet!)
- **null** (**undefined** is not ok!)

You can put whatever logic you want in your function for this:

```
const Lottery = (props) => {
  if (props.winner) {
    return <b>You win!</b>;
  } else {
    return <b>You lose!</b>;
  }
}
```

Ternary

It's very common as well to use ternary operators when you have short conditional statements:

```
const Lottery = (props) => {
    return (
      <b>You {props.winner ? "win" : "lose"}!</b>
    )
}
```

Demo: Slots!

demo/slots/Machine.js

```
const Machine = (props) => {
  const winner = props.s1 === props.s2 && props.s2 === props.s3;

  return (
    <div className="Machine">
      <b>{props.s1}</b>
      <b>{props.s2}</b>
      <b>{props.s3}</b>
      <b>You {winner ? "win!" : "lose!"}</b>
    </div>
  );
};
```

demo/slots/App.js

```
ReactDOM.render(
  <Machine s1="🍒" s2="🍒" s3="🍒" />,
  document.getElementById("root")
);
```

Looping in JSX

It's very common to work with arrays of data and loop over them to render JSX

- A shopping list (display all the shopping items)
- A list of GitHub repositories (display each one)
- A list of songs from an album (display each title)

Let's see a demo: Lists!

demo/list/List.js

```
const List = (props) => {
  return (
    <div>
      <h1>{props.name}</h1>
      <ul>
        {props.items.map(item => <li>{item}</li>)}
      </ul>
    </div>
  );
}
```

demo/list/App.js

```
const App = () => {
  return (
    <div>
      <List name="Shopping List" items={["Salsa", "Avocado", "Beans"]} />
      <List name="Todo List" items={["Learn React", "Feed cats"]} />
    </div>
  );
};

ReactDOM.render(
  <App/>,
  document.getElementById("root")
);
```

We used map here, but what else can we do?

- for loops
- while loops

Using for loops

Let's see another example!

demo/jokes-loops/Jokes-imperative.js

```
const JokesLoop = () => {
  const jokes = [
    {
      id: 1,
      text: "How do you comfort a JavaScript bug? You console it!"
    },
    {
      id: 2,
      text: "Why did the developer quit? Because he didn't get arrays"
    }
  ];
  const jokeList = [];

  for (let joke of jokes) {
    jokeList.push(<li>{joke.text}</li>);
  }

  return <ul>{jokeList}</ul>;
};
```

Here's what it looks like with map

It's common to use **array.map(fn)** to output loops in JSX:

demo/jokes-loops/Jokes-declarative.js

```
const JokesMap = () => {
  const jokes = [
    {
      id: 1,
      text: "How do you comfort a JavaScript bug? You console it!"
    },
    {
      id: 2,
      text: "Why did the developer quit? Because he didn't get arrays"
    }
  ];
  return (
    <ul>
      {jokes.map(j => <li>{j.text}</li>) }
    </ul>
  );
}
```

Which one should I use?

We highly recommend map - it's what you will see everywhere. Here's just a few reasons why

- you know that your code is going to run on each element of the array in the right order.
- your original array will be unaffected as map returns a new array each time it is called.
- it's more "declarative".

Warnings about key props

If you look in the console, when you are rendering multiple adjacent elements with JSX you'll see that React is mad at you for not adding something called a "**key**" prop when you map over an array and render components.

At a high level, keys help React identify which items have changed, are added, or are removed. They should always be unique and not change (also called stable)

You don't need to worry about this for now, We'll talk more about what's happening here shortly. Let's get comfortable with the fundamentals first!

Fixing our previous key prop issue

If we wanted to fix our previous issue, here's how we could do it:

```
const JokesMap = () => {
  const jokes = [
    {
      id: 1,
      text: "How do you comfort a JavaScript bug? You console it!"
    },
    {
      id: 2,
      text: "Why did the developer quit? Because he didn't get arrays"
    }
  ];
  return (
    <ul>
      {jokes.map(j => <li key={j.id}> {j.text}</li>) }
    </ul>
  );
}
```

Default Props

Components can specify default values for missing props

Demo: message-app

demo/message-app/Message.js

```
const Message = ({ from = "Marissa", messageText }) => {
  return (
    <p>
      {from} says {messageText}
    </p>
  );
};
```

Set properties on element; get using ***propName***.

demo/message-app/App.js

```
const App = () => {
  return (
    <div>
      <Message messageText="👋" from="Lana" />
      <Message messageText="💙💙" />
    </div>
  );
};

ReactDOM.render(<App />,
  document.getElementById("root"));
```

Let's take another look at this:

```
const Message = ({ from = "Marissa", messageText }) => {
  return (
    <p>
      {from} says {messageText}
    </p>
  );
};
```

We're destructuring the ***from*** and ***messageText*** keys from ***props*** and setting a default value for ***from*** to "Marissa".

It's extremely common to see destructuring from props and other functions in React.

If you're not sure about what this code is doing, check out the Modern JS course to learn about destructuring!

Props.children

- Some components don't know their children (containing data) ahead of time.
- React provides a special children prop to pass children elements directly to a parent component

An example

demo/props-children/CustomGreeting.js

```
const CustomGreeting = ({message, children}) => (
  <div>
    <h2>{message}</h2>
    {children}
  </div>
)
```

demo/props-children/App.js

```
const App = () => (
  <div>
    <CustomGreeting message={"Hello!"}>
      <button>Confirm Greeting</button>
    </CustomGreeting>

    <CustomGreeting message={"Nothing else here!"} />

    <CustomGreeting message={"Lets say nice things!"}>
      Inside this greeting we will give you a compliment
      <ul>
        <li>You are kind!</li>
      </ul>
    </CustomGreeting>
  </div>
);

ReactDOM.render(<App />,
  document.getElementById("root"));
```

Where do you see this?

- UI libraries like Reactstrap and Material UI
- When you don't know what the children will be when defining a component