React JSX in Javascript

Using JSX Elements as Values and in Conditionals

Using JSX

- We recently learned JSX, a way of writing React that looks like HTML
- A lot of it is intuitive, but it comes with some new things to learn
  - What is a JSX value, and what functionality does it have?
  - How do I embed logic inside of JSX?
  - How do I change DOM elements made in JSX?
- There are answers to all of these, but they require you to think of writing code the "React way"

JSX as a Value

JSX as a Value

- If you look at a React component render function, you'll notice that they always return JSX
- In Javascript, anything that can be returned could also be assigned to a value
- JSX is no different, and could easily be assigned as a variable
- Likewise, this means it can be passed around, and even rendered like a variable
- This allows us to render things conditionally, or create functions that return JSX

JSX as a Value - Conditional Example

- This shows a logout or login button, depending on the `isLoggedIn` prop

```
// components/Navigation.js
render() {
    let userButton;

    if (this.props.isLoggedIn) {
        userButton = <a href="/logout">Log out</a>;
    }
    else {
        userButton = <a href="/login">Log in</a>;
    }

    return (
        <nav>
            <h1>My Cool Website</h1>
            {userButton}
        </nav>
    );
}
```

JSX as a Value - Function Example

```
function renderGreeting(name = "Stranger") {
    return <h1>Hello {name}!</h1>;
}
```

JSX as a Value - Notes

- You can also just not have anything render, rendering a variable that is `null` will simply render nothing
- JSX elements can also be placed in arrays, and rendered that way. It'll render each element in the array.
  - But each element will need a unique `key` prop (more on that later)
- JSX need only be wrapped in parentheses if it spans multiple lines
- In addition to functions and variables, JSX can be used as a property!
  - This is actually how the `children` prop works

JSX Logic & Conditionals

JSX Logic & Conditionals

- While we can resolve all of our logic and assign JSX to variables before we return the final JSX object, we can also embed some logic into JSX itself
- When you enter "JS mode" with the curly braces, you can call functions or use logic operators to render different things
- This is a handy technique, but try to keep it simple

JSX Logic & Conditionals - Conditional Render

- You can use && to check some logic before rendering something
- If anything is falsey (You can string multiple checks together) it won't render

```
// components/SignupForm.js
render() {
    const { error } = this.props;

    return (
        <form className="SignupForm">
            {error && <p className="SignupForm-error">{error}</p>}
            ...
        </form>
    );
}
```

JSX Logic & Conditionals - If / Else Render

- You can use a ternary operator (condition ? trueCase : falseCase) to do an "if / else"
- If it's true, the first thing gets rendered, otherwise the second one is rendered

```
// components/Button.js
render() {
    const { children, isLoading } = this.props;

    return (
        <button className="Button">
            {isLoading ? "Loading..." : children}
        </button>
    );
}
```

JSX Logic & Conditionals - If / Else Render (cont.)

- You can use parens to multi-line bigger conditionals
- This is the same as a previous example, but more succinct

```
// components/Navigation.js
render() {
    const { isLoggedIn } = this.props;

    return (
        <nav>
            <h1>My Cool Website</h1>
            {isLoggedIn ? (
                <a href="/logout">Log out</a>
            ) : (
                <a href="/login">Log in</a>
            )}
        </nav>
    );
}
```

Manipulating JSX

Manipulating JSX

- One might think that because we can assign JSX to a variable, we can alter it, right?
- This is not the case. Once some JSX has been rendered, it **cannot be altered**.
  - This is a core principle of React, reproducible renders
- Likewise, once JSX is on the document, **we cannot alter it there**
  - Technically this can be done, but will be reset next render
- When using react, you *never* use `document.getElement*` functions
- You must get into the mindset of making your changes so that you could re-render your component, and the changes would be reflected because props or state changed
- This goes for all changes, like adding a class, or changing a style

Manipulating JSX - Changing Props

```
// components/ColorText.js
render() {
    this.text = (
        <font color="blue">
            {this.props.children}
        </font>
    );

    return this.text;
}

someOtherFunction() {
    this.text.color = "red";
}
```

- JSX values have no attributes or functions that can manipulate how they're rendered

Manipulating JSX - Changing Props

```
// components/ColorText.js
constructor() {
    this.color = "blue";
}

render() {
    return (
        <font color={this.color}>
            {this.props.children}
        </font>
    );
}

someOtherFunction() {
    this.color = "red";
}
```

- Manipulating values that are used in render don't cause a re-render

Manipulating JSX - Changing Props

```
// components/ColorText.js
render() {
    return (
        <font id="my-font" color="blue">
            {this.props.children}
        </font>
    );
}

someOtherFunction() {
    const text = document.getElementById("my-font");
    text.color = "red";
}
```

- This may work initially, but re-rendering the component will return it to its original state

Manipulating JSX - Changing Props

```
// components/BlueText.js
state = {
    color: "blue",
};

render() {
    return (
        <font color={this.state.color}>
            {this.props.children}
        </font>
    );
}

someOtherFunction() {
    this.setState({ color: "red" });
}
```

Additional Reading

- React Docs - Introducing JSX
- React Docs - Conditional Rendering
- React Docs - JSX in Depth (Advanced reading)