**JSX**

JSX is a syntax extension for JavaScript that is used with React. Browsers cannot read JSX, so code written with JSX must be compiled before it is used on the web. The example below is a single JSX element.

const header = <h1>Hello World</h1>

JSX elements:

* are treated as JavaScript expressions.
* can be stored individually in variables.
* can be stored in objects.
* can have attributes like regular HTML elements.
* use className as an attribute rather than class.
* must include the closing slash (/>) if they are self-closing elements like <img /> or <input />.
* can be nested inside of other JSX elements (see example of multi-line element below).
* nested JSX elements must have a single outermost element; if they don't, you might be able to fix the problem by wrapping the whole thing inside of a single <div>.

const nestedElements = (

<div class="container">

<h1>Header Text!</h1>

<img src="#" alt="an image"/>

</div>

);

RENDERING JSX

The JavaScript file must contain

* import React from 'react' and import ReactDOM from 'react-dom' at the top.
* ReactDOM.render(<JSX element or a variable>, <element where to append first argument>) at the bottom.

INJECTING JAVASCRIPT

let myName = 'Ian';

<p>Hello, {myName}</p>

* Variables can be used to set attributes.
* It is common to save variables in objects.

const styles = {

height: 100px;

width: 100px;

}

const catImage = (

<img

src="#"

alt="a cat"

height: {styles.height}

width: {styles.width} />

)

* if statements cannot be injected into JSX expressions.
* JSX expressions can be injected into conditionals.

let elements;

if (condition) {

elements = (

// Elements to render

);

} else {

elements = (

// Different elements to render

);

}

* Ternary operators can be used in JSX.

const headline = (

<h1>

{ age >= votingAge ? 'Vote' : 'Do not vote' }

</h1>

);

**ITERATOR METHODS IN JSX**

Iterator methods can be used to render information from arrays that will receive the same styling/presentation.

const strings = ['Home', 'Shop', 'About Me'];

const listItems = strings.map(string => <li>{string}</li>);

<ul>{listItems}</ul>

LISTS IN JSX

A JSX list only needs keys if

* The list items have memory from one render to the next, such as if elements are checked off.
* A list's order might be shuffled, such as if they are search results.

<ul>

<li key="li-01">Element One</li>

<li key="li-02">Element Two</li>

<li key="li-03">Element Three</li>

</ul>

**REACT WITHOUT JSX**

Elements can be created without JSX:

const h1 = React.createElement(

"element type",

null,

"element contents"

);

**COMPONENTS**

A component is a small, reusable chunk of code that is responsible for one job, generally to render some HTML.

class ClassName extends React.Component {

render() {

return (

// JSX expression

)

}

}

ReactDOM.render(

<ClassName />,

document.getElemetByID('app');

);

You can also inject JavaScript expressions into JSX.

class TodaysPlan extends React.Component {

render() {

let task;

if(!apocalypse) {

task = 'learn React.js'

} else {

task = 'run around'

}

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

}

}

ReactDOM.render(

<TodaysPlan />,

document.getElememtById('app');

);

**USING THIS**

class IceCreamGuy extends React.Component {

get food(){

return 'ice cream';

}

render() {

return <h1>I like {this.food}.</h1>;

}

}

EVENT LISTENERS IN RENDER()

render() {

return (

<div onHover={myFunction}>

</div>

);

}

In the example above, onHover is an event listener. Whenever the user hovers over this <div>, myFunction() will be called. myFunction() is an event handler.

**COMPONENTS RENDER OTHER COMPONENTS**

class OMG extends React.Component {

render() {

return <h1>Hello World!</h1>;

}

}

class Crazy extends React.Component {

render() {

return <OMG />;

}

}

RENDERING COMPONENTS FROM OTHER FILES

If a component that you wish to render is written in another file, you can use the following syntax to retrieve it:

import React from 'react';

import ReactDOM from 'react-dom';

import { NavBar } from './NavBar.js'; // Be sure to export the variable in the other file

class ProfilePage extends React.Component {

render() {

return (

<div>

<NavBar />

</div>

);

}

}

**PROPS**

Information that is passed from one component to another is known as "props." Information can be passed a component by giving it an attribute with the information:

<SecretMessageRenderer message="Secret!" />

To render the props of an element:

class Greeting extends React.Component {

render() {

return <h1>Hi there, {this.props.firstName}!</h1>;

}

}

ReactDOM.render(

<Greeting firstName='Roberta' />,

document.getElementById('id')

);

To pass a prop from one component to another:

// exportComponent.js

import React from 'react';

export class Greeting extends React.Component {

render() {

return <h1>Hi there, {this.props.name}!</h1>;

}

}

// importComponent.js

import React from 'react';

import ReactDOM from 'react-dom';

import { Greeting } from './exportComponent';

class App extends React.Component {

render() {

return (

<div>

<h1>Welcome to my site!</h1>

<Greeting name = "Gilberto" />

</div>

);

}

}

ReactDOM.render(

<App />,

document.getElementById('app')

);

PROPS AS EVENT HANDLERS

class Example extends React.Component {

handleEvent() {

alert(`I am an event handler.

If you see this message,

then I have been called.`);

}

render() {

return (

<h1 onClick={this.handleEvent}>

Hello world

</h1>

);

}

}

CHILDREN

// App.js

class App extends React.Component {

render() {

return (

<div>

<List type='Living Musician'>

<li>Sachiko M</li>

<li>Harvey Sid Fisher</li>

</List>

<List type='Living Cat Musician'>

<li>Nora the Piano Cat</li>

</List>

</div>

);

}

}

ReactDOM.render(

<App />,

document.getElementById('app')

);

// List.js

export class List extends React.Component {

render() {

let titleText = `Favorite ${this.props.type}`;

if (this.props.children instanceof Array) {

titleText += 's';

}

return (

<div>

<h1>{titleText}</h1>

<ul>{this.props.children}</ul>

</div>

);

}

}

**STATE**

The state property of a component is declared in the constructor()function.

class App extends React.Component {

constructor(props) {

super(props);

this.state = { title: 'Best App' };

}

render() {

return (

<h1>{this.state.title}</h1>

);

}

}

.SETSTATE()

State can be changed in functions other than the constructor() function using .setState().

When passing a method that uses this down to a child, it is necessary to bind this in the constructor of the parent method.

class Mood extends React.Component {

constructor(props) {

super(props);

this.state = { mood: 'good' };

this.toggleMood = this.toggleMood.bind(this); // bind(this) to .toggleMood()

}

toggleMood() {

const newMood = this.state.mood == 'good' ? 'bad' : 'good';

this.setState({ mood: newMood });

}

render() {

return (

<div>

<h1>I'm feeling {this.state.mood}!</h1>

<button onClick={this.toggleMood}>

Click Me

</button>

</div>

);

}

}

CHILD COMPONENTS UPDATING PARENT'S STATE

There are several parts to creating a child component that can update its parent's state:

1. In the parent component, define a method that calls this.setState().
2. Bind the new method to the current instance of the component, like this:

this.handleClick = this.handleClick.bind(this)

1. Pass the method down to the child as a prop in the render()function:

<ChildClass onClick={this.handleClick} />

1. The child receives the passed down function and uses it as an event handler:

<button onClick={this.props.onClick}></button>

When the button is clicked, the state of the parent component will be updated.

Below is an example of a child component that updates its parent's state:

import React from 'react';

export class Child extends React.Component {

constructor(props) {

super(props);

this.handleChange = this.handleChange.bind(this);

}

handleChange(e) {

const name = e.target.value;

this.props.onChange(name);

}

render() {

return (

<div>

<h1>

Hey my name is {this.props.name}!

</h1>

<select id="great-names" onChange={this.handleChange}>

<option value="Frarthur">

Frarthur

</option>

</select>

</div>

);

}

}

React Documentation link: <https://reactjs.org/>

Deploying a created React app: <https://daveceddia.com/deploy-create-react-app-surge/>