1. Why React?

React.js is a JavaScript library. It was developed by engineers at Facebook.

What is JSX?

JSX is a syntax extension for JavaScript. It was written to be used with React. JSX code looks a lot like HTML.

it means that JSX is not valid JavaScript. Web browsers can’t read it!

```
const h1 = <h1>Hello world</h1>;
```

Can also have ids in JSX

```
Const p1=<p1 id =”large”>foo</p1>
```

What is Nested JSX?

```
var myDiv = <div>
     <h1>Hello World</h1>
   </div>;
```

JSX must have only one outer element.

```
const blog = (
<div>  <img src="pics/192940u73.jpg" />
  <h1>
    Welcome to Dan's Blog!
  </h1>
  <article>
    Wow I had the tastiest sandwich today.  I <strong>literally</strong> almost freaked out.
  </article>
  </div>
);
```

Rendering JSX:

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

// Copy code here:
ReactDOM.render(<h1>Hello world</h1>, document.getElementById('app'));
```

What is ReactDOM?

ReactDOM is the name of a JavaScript library. This library contains several React-specific methods, all of which deal with the DOM in some way or another.

ReactDOM.render() is the most common way to render JSX. It takes a JSX expression, creates a corresponding tree of DOM nodes, and adds that tree to the DOM. That is the way to make a JSX expression appear onscreen.

Passing a Variable to ReactDOM.render()?

```
var myList = (
  <ul>
    <li>Apple</li>
    <li>Orange</li>
  </ul>);
ReactDOM.render(myList, document.getElementById('app'));
```

The Virtual DOM

One special thing about ReactDOM.render()is that it only updates DOM elements that have changed.
That means that if you render the exact same thing twice in a row, the second render will do nothing:

```
const hello = <h1>Hello world</h1>;

// This will add "Hello world" to the screen:

ReactDOM.render(hello, document.getElementById('app'));

// This won't do anything at all:

ReactDOM.render(hello, document.getElementById('app'));
```

In JSX, you can’t use the word class! You have to use className instead:

```
<h1 class="big">Hey</h1>   → wrong
<h1 className="big">Hey</h1> → correct

var myDiv= <div className="big">I AM A BIG DIV</div>;   → correct
ReactDOM.render(myDiv, document.getElementById('app'));
```

In JSX, you have to include the slash. If you write a self-closing tag in JSX and forget the slash, you will raise an error:

```
const profile = (
 <div>
```

```
    <h1>I AM JENKINS</h1>
    <img src="images/jenkins.png" />
    <article>
      I LIKE TO SIT
      <br/>
      JENKINS IS MY NAME
      THANKS HA LOT
    </article>
  </div>
);
```

JavaScript In Your JSX In Your JavaScript:

```
ReactDOM.render(<h1>2 + 3 = 2 + 3</h1>, document.getElementById('app'));

//2 + 3
```

Any code in between the tags of a JSX element will be read as JSX, not as regular JavaScript! JSX doesn't add numbers - it reads them as text, just like HTML.

```
ReactDOM.render(<h1>2 + 3 = {2 + 3}</h1>, document.getElementById('app'));
//5
```

---

20 Digits of Pi in JSX

```
const pi = (
  <div>
    <h1>
      PI, YALL!
    </h1>
    <p>
      {Math.PI.toFixed(20)}
    </p>
  </div>
);

ReactDOM.render(
        pi,
        document.getElementById('app')
);
//
```

- The code is written in a JavaScript file. By default, it will all be treated as regular JavaScript.
- Find on line 5. From there up through , the code will be treated as JSX.

- Find Math. From there up through (20), the code will be treated as regular JavaScript again.
- The curly braces themselves won't be treated as JSX nor as JavaScript. They are markers that signal the beginning and end of a JavaScript injection into JSX, similar to the quotation marks that signal the boundaries of a string.

Variables in JSX

When you inject JavaScript into JSX, that JavaScript is part of the same environment as the rest of the JavaScript in your file.
That means that you can access variables while inside of a JSX expression, even if those variables were declared on the outside.

```
const theBestString ='tralalalala i am da best';

ReactDOM.render(<h1>{theBestString}</h1>, document.getElementById('app'));
```

Variable Attributes in JSX

```
const goose =
'https://s3.amazonaws.com/codecademy-content/courses/React/react_photo-goose.jpg';

// Declare new variable here:
var gooseImg=(
<img src={goose}/>
);
ReactDOM.render(gooseImg, document.getElementById('app'));
```

Event Listeners in JSX

JSX elements can have event listeners, just like HTML elements can. Programming in React means constantly working with event listeners.
You create an event listener by giving a JSX element a special attribute. Here's an example:

```
<img onClick={myFunc} />
function myFunc() {
  alert('Make myFunc the pFunc... omg that was horrible i am so sorry');
}
```

Note that in HTML, event listener names are written in all lowercase, such as onclick or onmouseover. In JSX, event listener names are written in camelCase, such as onClick or onMouseOver.

```
function makeDoggy(e) {
  e.target.setAttribute('src',
'https://s3.amazonaws.com/codecademy-content/courses/React/react_photo-puppy.jpeg');
  e.target.setAttribute('alt', 'doggy');
}

const kitty = (
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/React/react_photo-kitty.jpg"
                    alt="kitty" onClick={makeDoggy}/>
);

ReactDOM.render(kitty, document.getElementById('app'));
```

JSX Conditionals: If Statements That Don't Work

Here’s a rule that you need to know: you can not inject an if statement into a JSX expression.
This code will break:

```
(
  <h1>
    {
      if (purchase.complete) {
        'Thank you for placing an order!'
      }
    }
  </h1>
)
```

Correct Example:

```
function coinToss() {
  // This function will randomly return either 'heads' or 'tails'.
  return Math.random() < 0.5 ? 'heads' : 'tails';
}

const pics = {
  kitty: 'https://s3.amazonaws.com/codecademy-content/courses/React/react_photo-kitty.jpg',
  doggy:
'https://s3.amazonaws.com/codecademy-content/courses/React/react_photo-puppy.jpeg'
};

let img;
```

```
if (coinToss() === "heads"){
  img =(
  <img src={pics.kitty}/>
    );
}else {
  img =(
  <img src={pics.doggy}/>
);
}
ReactDOM.render(img, document.getElementById('app'));
```

Ternery operator:

```
const headline = (
  <h1>
    { age >= drinkingAge ? 'Buy Drink' : 'Do Teen Stuff' }
  </h1>
);

Example;
function coinToss () {
  // Randomly return either 'heads' or 'tails'.
  return Math.random() < 0.5 ? 'heads' : 'tails';
}

const pics = {
  kitty: 'https://s3.amazonaws.com/codecademy-content/courses/React/react_photo-kitty.jpg',
  doggy:
'https://s3.amazonaws.com/codecademy-content/courses/React/react_photo-puppy.jpeg'
};

const img = <img src={pics[coinToss()=='heads' ? 'kitty' : 'doggy']} />;

ReactDOM.render(
          img,
          document.getElementById('app')
);
```

JSX Conditionals: &&

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

// judgmental will be true half the time.
const judgmental = Math.random() < 0.5;
```

```
const favoriteFoods = (
  <div>
    <h1>My Favorite Foods</h1>
    <ul>
      <li>Sushi Burrito</li>
      <li>Rhubarb Pie</li>
      { !judgmental && <li>Nacho Cheez Straight Out The Jar</li>}
      <li>Broiled Grapefruit</li>
    </ul>
  </div>
);

ReactDOM.render(
          favoriteFoods,
          document.getElementById('app')
);
```

.map()

```
const people = ['Rowe', 'Prevost', 'Gare'];

const peopleLis = people.map(person =>
  // expression goes here:
<li>{person}</li>;
);

// ReactDOM.render goes here:
ReactDOM.render(<ul>{peopleLis}</ul>, document.getElementById('app'));
```

Keys:
A key is a JSX attribute. The attribute's nameis key. The attribute's value should be something unique, similar to an id attribute.

keys don't do anything that you can see! React uses them internally to keep track of lists. If you don't use keys when you're supposed to, React might accidentally scramble your list-items into the wrong order.

```
const people = ['Rowe', 'Prevost', 'Gare'];

const peopleLis = people.map((person, i) =>
  // expression goes here:
<li key={'person_' + i}>{person}</li>;
);

// ReactDOM.render goes here:
```

ReactDOM.render(
{peopleLis}, document.getElementById('app'));

React.createElement

Const greatestDivEver = i am div ;

It can be rewritten like:

const greatestDivEver = React.createElement("div",null, "i am div");

What's a component?

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

Ex:

```
import React from 'react';                  // importing React library from react
import ReactDOM from 'react-dom';   // importing ReactDOM library from react-dom

class MyComponentClass extends React.Component {
  render() {
    return <h1>Hello world</h1>;
  }
};

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

What is Component class?

Every component must come from a component class.
A component class is like a factory that creates components. If you have a component class, then you can use that class to produce as many components as you want.

Ex;

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

```
class x extends React.Component{

};
```

Name a Component Class

```
class MyComponentClass extends React.Component{

};
```

Use Multiline JSX in a Component

```
class QuoteMaker extends React.Component {
  render() {
    return (
      <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>
    );
  }
};

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

Use a Variable Attribute in a Component

```
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')
);
```

Use a Conditional in a Render Function

```
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(){
    let message;
    if(fiftyFifty == 'true'){
      message ="I'm going out WOOO";
    } else {
     message = "I'm going to bed WOOO";
    }
    return <h1>Tonight {message}</h1>;
  }
};

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

Use this component:

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

```
class MyName extends React.Component {
          // name property goes here:
get name(){
  return 'Shobana';
}
  render() {
    return <h1>My name is {this.name}.</h1>;
  }
}

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

Use an Event Listener in a Component

```
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'));
```

Render methods can also return another kind of JSX: component instances.

```
class OMG extends React.Component {
  render() {
    return <h1>Whooaa!</h1>;
  }
}

class Crazy extends React.Component {
  render() {
    return <OMG />;
  }
}
```
