

February 24th, 2018 –

REACT_JS [CODECADEMY]

JAVASCRIPT LIBRARY DEVELOPED AT FACEBOOK

[OPEN SOURCE PROJECTS](#)

REACT IS FAST – COMPLEX UPDATES QUICKLY

REACT IS MODULAR – MANY SMALLER, REUSABLE FILES

REACT IS SCALABLE – BEST USED DISPLAYING CHANGING DATA

REACT IS FLEXIBLE – POTENTIAL STILL UNKNOWN

REACT IS POPULAR – HELPS TO BECOME EMPLOYABLE

1. WHAT IS JSX

A. A syntax extension for JavaScript. Written to be used with React (looks a bit like HTML)

1. *This means JSX is not valid JavaScript and must be compiled and translated to JavaScript before reaching a web browser*

B. Basic unit of JSX is called a JSX element

1. *Example: `<h1>Hello World</h1>` looks like HTML, but in a .js file*
2. *JSX element treated like JavaScript expression in that it can be:*
 - a. *Saved in a variable*
 - b. *Passed to a function*
 - c. *Stored in an object or array*
 - i. `const navBar = <nav>thing goes here</nav>;`
 - ii. `const myTeam = { center: Tim, pointGuard: Jim, ... };`
 - d. *Etc.*

C. JSX elements can have attributes

1. *Looks like HTML element (can have one or multiple)*
 - a. `const navBar = <nav id="nav-bar">thing goes here</nav>;`

D. Nested JSX

1. *To make it readable use HTML-style line breaks and indentation*
2. *If expression takes up more than one line, then you must wrap the multi-line JSX expression in parenthesis*
3. *Can be saved as variables, passed to functions, etc.*

a. `const nestedExample = (

 <h1> Click link </h1>

);`

4. *JSX Outer Elements*

- a. *A JSX expression must have exactly one outermost element*
 - i. i.e. the first and closing tag of a JSX expression must be the same
 - ii. You can always just wrap it in a `<div>` if this is an issue

E. Rendering JSX - Make it appear on the screen

1. *ReactDOM*

- a. *Name of the JavaScript library that deal with the [DOM](#)*

2. *ReactDOM.render()*

- a. *Most common way to render JSX*
 - i. Only updates DOM elements that have changed (called “diffing”)
 - 1. React is so successful because of this significant ability
 - 2. Accomplishes this because of [the virtual DOM](#)
 - a. *Entire Virtual DOM gets updated*
 - b. *Virtual DOM is compared to snapshot of DOM right before the update*
 - c. *React figures out which objects have changed and change only those objects in the real DOM*
 - d. *Changes on the real DOM cause the screen*
- b. *Takes the JSX expression, creates corresponding tree DOM nodes, and adds that tree to the DOM*
- c. *The first argument (HTML looking thing) being passed should evaluate to a JSX expression, and it will be rendered on the screen*
 - i. It doesn’t have to literally be a JSX expression
 - ii. It could be a variable as long as it evaluates to a JSX expression
- d. *The second argument tells where to put the first argument on the screen*
 - i. Example: `document.getElementById(‘app’)`
 - ii. Note: The first argument is appended to whatever element is selected by the second argument

2. ADVANCED JSX

- A. Grammar in JSX is mostly the same as HTML with subtle differences

1. *class vs className*

- a. *class in HTML is className in JSX because class is a reserved word in JS which JSX get translated you can't use class*
 - i. JSX className attribute automatically render as class attributes
- 2. *Self-Closing Tags*
 - a. *Must include the / in self closing tags with JSX (optional in HTML)*
 - i. `
` is JSX is ok but `
` is not (even tho both ok in HTML)
- B. JavaScript in JSX (which is in JavaScript file)
 - 1. *Wrap in { } for JSX code to be read as JavaScript*
 - a. *Example: `<h1>{2 + 3}</h1>` will show 5 but without the { } it will literally show 2 + 3*
 - 2. *Injected JavaScript is part of same environment as rest of file so you can access variables inside of JSX expressions even if variable declared outside*
 - 3. *Object properties are often used to set attributes (organize code)*
 - 4. *Event Listeners ([valid event names](#))*
 - a. *Attribute value should be a valid/defined function*
 - b. *Written in camelCase for JSX not all lowercase like HTML*
 - 5. *Conditionals: If statements that don't work (can't use an 'if' in JSX)*
 - a. *Explained [here](#)*
 - b. *Common to keep the if else outside of JSX tags, not injected between*
 - c. *Ternary Operator – more compact way to write conditionals*
 - i. [Explanation](#): `x ? y : z` (if x truth return y, if x false return z)
 - d. *&& operator*
 - i. Works best in conditionals that will sometimes do an action but other times do nothing at all
 - e. *.map()*
 - i. Is best bet for creating lists in JSX for example:


```
1. const arrays = ['thing1', 'thing2', 'thing3'];
   const listArray = arrays.map( arrayItem =>
       <li>{arrayItem} </li>);
   ReactDOM.render(<ul>{listArray}</ul>, document.get ... );
```
 - f. *Keys – JSX attribute and the value should be unique (like and id)*
 - i. React uses them internally (don't see it) to track lists
 - ii. React might scramble lists if you don't use keys correctly
 - iii. Needs keys if either of the following is true:
 - 1. The list-items have 'memory' from one render to the next
 - a. *i.e. was something checked off a list?*
 - 2. A list's order might be shuffled

a. *i.e. maybe a lists search results*

3. Otherwise you don't have to use keys (but doesn't hurt if you do)

C. [React.createElement](#)

1. *You can write React code without using JSX (majority of programmers do use JSX, but don't have to)*

a. *Example in JSX*

i. `const title = <h1>Hello World</h1>`

b. *Example of React without JSX*

i. `Const title = React.createElement(
 "h1",
 null,
 "Hello World"
);`

c. *When a JSX element is compiled the compiler transforms the JSX into the method above*

3. REACT COMPONENTS

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

B. `import React from 'react';`

- // create a variable named React: import React from 'react';
// evaluate this variable and get a particular, imported JavaScript object: React
// { imported object properties here... }*
- This imported object contains methods that you need in order to use React. The object is called the React library.*

C. `import ReactDOM from 'react-dom';`

- The methods imported from 'react-dom' are meant for interacting with the [DOM](#)*
- The methods imported from 'react' don't deal with the DOM at all. They don't engage directly with anything that isn't part of React.*
- To clarify: the DOM is used in React applications, but it isn't part of React. After all, the DOM is also used in countless non-React applications. Methods imported from 'react' are only for pure React purposes, such as creating components or writing JSX elements.*

D. Component Class

- Every component must come from a component class (component class is not a component)*
- If you have a component class, you can create as many components as you want*

3. To make a component class you use a base class from React library (`React.Component`)
4. Links to more info on classes: [1](#) [2](#) [3](#) [4](#)
5. Component class variable names must begin with capital letters
6. This adheres to JavaScript's class syntax (and [broader programming convention](#))

E. Review Components

Let's review what you've learned so far! Find each of these points in `app.js`:

- On line 1, `import React from 'react'` creates a JavaScript object. This object contains properties that are needed to make React work, such as `React.createElement()` and `React.Component`.
- On line 2, `import ReactDOM from 'react-dom'` creates another JavaScript object. This object contains methods that help React interact with the DOM, such as `ReactDOM.render()`.
- On line 4, by subclassing `React.Component`, you create a new *component class*. This is not a component! A component class is more like a factory that produces components. When you start making components, each one will come from a component class.
- Whenever you create a component class, you need to give that component class a name. That name should be written in UpperCamelCase. In this case, your chosen name is `MyComponentClass`.

Something that we *haven't* talked about yet is the *body* of your component class: the pair of curly braces after `React.Component`, and all of the code between those curly braces.

Like all JavaScript classes, this one needs a body. The body will act as a set of instructions, explaining to your component class how it should build a React component.

Here's what your class body would look like on its own, without the rest of the class declaration syntax. Find it in `app.js`:

```
{
  render() {
    return <h1>Hello world</h1>;
  }
}
```

That doesn't look like a set of instructions explaining how to build a React component! Yet that's exactly what it is.

F. Render Function

1. This property must be included, name is `render` and value is a function

G. Component Instance

- A. JSX elements can be either HTML-like, or component instances.
- B. JSX uses capitalization to distinguish
 1. That is why component class names begin with capital letters – says “I’m a component instance, not an HTML tag”

4. COMPONENTS AND ADVANCED JSX

- A. `Render()` must have a return, but can also contain more
 1. Example: `Math.floor(Math.random() * 10 + 1);`
- B. If statement is located *inside* the render, but *before* the return statement
- C. Using `this` in a component
 1. This refers to an object on which this's enclosing method (often `.render()`) is called

D. Event Listeners

1. *Render () {
 Return (
 <div onMouse= {myFunc}> </div>
);
}*

E. Review Component

5. COMPONENTS RENDER OTHER COMPONENTS

A. Component Instances: when you render a component in another component

B. By default every JavaScript file is invisible to other JavaScript files

1. *Use import statement to use variables between files*
 - a. *Also will need an export statement (exporting variable you hope to grab)*
 - b. *Rarely will you see import without export and visa versa*
2. *If string at end of import is a / or . then import treats the string as a file path*
 - a. *.js is assumed so is not necessary at the end of file name*
3. *This [Module system](#) is not specific to React.*
 - a. *React's import/export specific [module system](#) comes from ES6*
 - i. *More in depth info [here](#)*

6. STORE DYNAMIC INFORMATION IN REACT

A. Dynamic Information – information that can change

1. *React needs dynamic info to render*
2. *Two ways a component can get dynamic information*
 - a. *Props*
 - i. *Passed in from the outside*
 - b. *State*
 - i. *Component decides its own state*
3. *Every other component besides these two should always stay the same*
4. *React apps is really just components setting state and passing in props to one another*
5. *this.props*
 - a. *A component can pass information to another component*
 - i. *This information is known as “props”*

1. A prop is an object
- ii. You can pass information to a React component by adding an attribute
 1. Set name attribute equal to info you want to pass, use { } if passing something that is not a string
- iii. Most common way to use props is to pass info from one component to another
- b. *How to make a component display the info it is passed in*
 - i. Find component class that will receive the info
 - ii. Include this.props.name-of-info in that component class's render method return statement
- c. *Props clarification*
 - i. Props references the object that stores all the info
 - ii. Props also is the plural of prop, which are the individual pieces of the props object
- d. *Props to make decisions*
 - i. Props are not always shown on the screen, but often used to make decisions on what should be shown on the screen based on the attribute
- e. *Functions as props*
 - i. Especially common for event handlers
 1. Must define event handlers in class before passing them anywhere
 - a. *Define event handler as a method on component class (just like render)*

```
import React from 'react';

class Example extends React.Component {
  handleClick() {
    alert('I am an event handler.
    If you see this message,
    then I have been called.');
```

example: }

- f. *Name like onClick only create event listeners if they're used on HTML-like JSX elements. Otherwise, they're just ordinary prop names*

<pre>// Button.js // The attribute name onClick // creates an event listener: <button onClick= {this.props.onClick}> Click me! </button></pre>	<pre>// Talker.js // The attribute name onClick // is just a normal attribute name: <Button onClick= {this.handleClick} /></pre>
---	---

example:

- g. *this.props.children*
 - i. Will return everything between a component's opening and closing JSX tags
 - ii. If a component has more than one child they will be returned in an array, but if there is only one child it will just be returned (no array)
- h. *defaultProps*
 - i. Set default props so that if there is no prop it isn't left blank
- 6. *this.props Recap (skills learned)*
 - a. *Passing a prop by giving an attribute to a component instance*
 - b. *Accessing a passed in prop via this.props.prop-name*
 - c. *Displaying a prop*
 - d. *Using a prop to make decisions about what to display*
 - e. *Defining an event handler in a component class*
 - f. *Passing an event handler as a prop*
 - g. *Receiving a prop event handler and attaching it to an event listener*
 - h. *Naming event handlers and event handler attributes according to convention*
 - i. *this.props.children*
 - j. *getDefaultProps*
- 7. *this.state*
 - a. *Should be equal to an object*
 - i. That object is the initial state
 - b. *Constructor and super are [features of ES6](#), not unique to React*
 - i. React components always have to call super in their constructors to be set up properly
 - c. *Note: methods should never be separated by a comma if inside of a class body.*
 - i. This is to emphasize the fact that classes and object literals are different
 - d. *To read a component's state use: this.state.name-of-property*
 - i. Just like this.props, this.state can be used from any property defined inside of a component class's body
 - e. *Update State using this.setState()*
 - i. this.setState() takes two arguments
 - 1. An object that will update the component's state
 - 2. A callback → you basically never need the callback

example:

```

React.Component {
  constructor(props) {
    super(props);
    this.state = { mood:
    'decent' };
  }

  render() {
    return <div></div>;
  }
}
<Example />

```


- ii. Bound the correct this [explanation](#)
 - 1. Simplified for now, in React when using an event handler that uses this, you need to add `this.methodName = this.methodName.bind(this)` to constructor function
- iii. As soon as `this.setState()` is called, it essentially then calls `.render()`
 - 1. Thus, `this.setState()` cannot be called in `.render()` or it'd be an infinite loop