What is JSX

React uses JSX

- That's the weird not-quite HTML stuff
- Notice it is NOT a JS string

Why JSX?

Best practice for both server-side and client-side:

- functions take state data and return HTML

Functions that modify the DOM can be broken down:

- Convert state data to HTML
- Update page with HTML

JSX

- is this kind of function
- is written like HTML
- is still actually a function (transpiled)

JSX

Basic Rules of JSX Syntax

- All JSX tags can self-close (and must close)
- JSX tags require `className` instead of `class`
- JSX tags can take an object (not string) for `style`
    - If you use `style` - we won't
- Anything inside `{}` replaced with evaluated results
    - Notice how this is NOT `${}`!
- Whitespace trims as much as possible
    - Not just to 1 space!

JSX example

JSX (`coolCat` is a JS variable that holds 'Maru')

```
<div className="demo">
  <span>{ 1 + 1 }</span>
  { coolCat }
</div>
```

Actual output:

```
<div class="demo"><span>2</span>Maru</div>
```

JSX vs Component

Component is the function/object that returns output

JSX is a syntax for the output

- Components return JSX
- Components are NOT JSX

Composition using many component files

```
import Cat from './Cat';
import Box from './Box';

const Room = () => {
  return (
    <div className="room">
      <Box><Cat/></Box>
      <Box><Cat/></Box>
      <Box/>
    </div>
  );
};
export default Room;
```

Components: Classes vs function

React Components can be defined as classes:

```
class MyComponent extends React.Component {
  render() {
    return ( <div>
      // ...
```

or as functions:

```
function MyComponent()
  return ( <div>
    // ...
```

Feb 2019, "hooks" released

- classes are no longer required

Will we do class-based components?

This course will only do function-based components

- Class-based components still exist
- But basically no new work done with them
- hooks have definite advantages

React Docs still use class-based components

- Huge bummer and confusion
- use their new docs! **http://beta.reactjs.org**

Props

Like HTML, React Components can be passed attributes, called "props"

The component gets them as arguments:

```
<MyComp name="Bao"/>
```

```
function MyComp(props) {
  return (<div>{props.name}</div>);
}
```

```
<div>Bao</div>
```

You can destructure like any object/function call:

```
function MyComp({ name }) {
  return (<div>{ name }</div>);
}
```

About Props

In HTML

- attributes must be strings
- properties have no value

In JSX, props can be ANY DATA (if in {})

```
<MyComp info={ [ 1, 2, 3 ] }/>
```

In JSX, properties should be set as boolean

```
<MyComp disabled={true}/>
```

JSX is often passed callback functions as props!

```
<MyComp onLogout={logoutCallback}/>
```

Children (tag contents)

To access JSX contents, use special prop "children":

```
<MyComp>This is some content</MyComp>
```

```
const MyComp = ({ children }) => {
  return (
    <div>I heard: <b>{children}</b></div>
  );
};
```

```
<div>I heard: <b>This is some content</b></div>
```

```
import Box from './Box';
import Cat from './Cat';
const Room = () => {
  return (
    <div className="room">
      <Box><Cat name="Maru"/></Box>
      <Box><Cat name="Grumpy"/></Box>
      <Box/>
    </div>
  );
};
export default Room;
```

```
const Box = ({ children }) => {
  const contents = children ? children : <div>Nothing</div>;
  return ( <div> A box contains: {contents} </div>);
};
export default Box;
```

```
const Cat = ({ name }) => (<div>{name}</div>);
export default Cat;
```

When to use props

Props are essential to using components

Think of them as arguments to a function call

- Because they are!
- Makes components more flexible and reusable
- Keeps components "dumb" and unaware of app state

We make components reusable and decoupled

- Just like we do functions, like service calls