React 101

Codecademy

JSX

Advanced JSX

class vs className

Grammar in JSX is mostly the same as in HTML, but there are subtle differences to watch out for. Probably the most frequent of these involves the word class.

In HTML, it is common to use class as an attribute name:

<h1 class=”big”>Hey<h1>

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

<h1 className=”big”>Hey</h1>

This is because JSX gets translated into JavaScript, and class is a reserved word in JS.

When JSX is rendered, JSX className attributes are automatically rendered as class attribute.

Self-Closing Tags

Another JSX ‘gotcha’ involves self-closing tags.

Most HTML elements use two tags: an opening tag (<div>), and a closing tag (</div>). However, some HTML elements such as <img> and <input> use only one tag. The tag that belongs to a single-tag element isn’t an opening tag nor a closing tag; it’s a self-closing tag.

When we write a self-closing tag in HTML, it is optional to include a forward-slash immediately before the final angle-bracket:

Fine in HTML: <br /> Also fine in HTML: <br>

But!

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:

Fine in JSX: <br /> Not fine at all in JSX: <br>

JavaScript in JSX in JavaScript

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

Trying something like above, in JSX we would expect 5 as output on browser, but no what we will get is 2+3, i.e., JSX treats it as a string because it is inside the <h1> tags.

Any code in between the tags of a JSX element will be read as JSX, not regular JS!

So to deal with this problem we need some way so that even if code is present inside JSX tags, it is treated like ordinary JS and not like JSX.

This can be achieved by using curly braces!

Curly Braces in JSX

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

Now the output on the browser would be 5 as everything inside the curly braces will be treated as regular JS.

20 digits of PI in JSX

This means now we can inject regular JS into JSX expressions!.

import React from 'react';

import ReactDOM from 'react-dom';

const pi = (

<div>

<h1>

PI, YALL!

</h1>

<p>

{Math.PI.toFixed(20)}

</p>

</div>

);

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

* This code when written in JS file would by default be treated as regular JS.
* All the code between <div></div> tags would be treated as JSX.
* Math.PI.toFixed(20) is treated as JS again as it is inside curly braces.
* The curly braces themselves would not be treated as JSX nor JS. They are markers that signal the beginning and end of a JS injection into JSX, similar to the quotation marks that signal the boundaries of a string.

Variables in JSX

When we inject JavaScript into JSX, that JS is part of the same environment as the rest of the JS in your file.

That means that we can access variables while inside of a JSX expression, even if those variables were declared on the outside.

Variable Attributes in JSX

When writing JSX, it’s common to use variables to set attributes.

Here’s an example of how that might work:

const sideLength = "200px";

const panda = (

<img

src="images/panda.jpg"

alt="panda"

height={sideLength}

width={sideLength} />

);

Object properties are also often used to set attributes:

const pics = {

panda: "http://bit.ly/1Tqltv5",

owl: "http://bit.ly/1XGtkM3",

owlCat: "http://bit.ly/1Upbczi"

};

const panda = (

<img

src={pics.panda}

alt="Lazy Panda" />

);

const owl = (

<img

src={pics.owl}

alt="Unimpressed Owl" />

);

const owlCat = (

<img

src={pics.owlCat}

alt="Ghastly Abomination" />

);

Event Listeners in JSX