**Learn React: Advanced JSX and Everything Else You Need To Know –**

[Aphinya Dechalert](https://medium.com/@PurpleGreenLemon?source=post_page-----ac867fb79c5c--------------------------------) Nov 4, 2021

Part 2 of Learn React from Scratch Series

JSX is easy once you’ve got the syntax down. If you’ve got experience or knowledge in JavaScript, JSX is pretty easy to pick up quickly. Here are more JSX syntaxes that you should know to be an effective React developer.

**class and className**

In HTML, class is used as an attribute that lets the browser know which CSS class to apply. However, class is also a JavaScript reserved word.

JSX compiles down into JavaScript, which means that conflicts can occur. To fix this, we use the attribute className instead of class. Don't worry, it will compile into the HTML class attribute during the build process. The differences between className and class is simply to help JSX determine what kind of *class*we're dealing with.

Here is an example:

const greeting = "hello";   
<h1 className="header">{greeting}</h1>

This will render the following when compiled:

<h1 class="header">hello</h1>

**Self-closing tags only in JSX**

Browsers are pretty chill when it comes to rendering HTML. However, JSX strictly requires all tags to be closed. This means that certain elements such as <img> and <input> needs to be closed off with /.

If a tag is not self-closed, then JSX will throw an error.

Here is an example:

<div>  
 <br / >  
 <p>some text here</p>  
 <img src=”{imgLink} />  
</div>

**Adding JavaScript to JSX**

JSX is JavaScript. However, because it mixes in HTML, JSX needs a way to distinguish between statically rendering contents vs. reading it as JavaScript syntax.

To do this, you need to use {} around your JavaScript code. Here is an example of how to add a variable to JSX:

const name = “Bob”;  
const number = 1 + 2 + 3;  
const greeting = (<h1>Hi {name}{number}</h1>);// greeting will print:  
// <h1>Hi Bob6</h1>

You can also set attributes and event listeners in JSX with JavaScript using the curly braces {}. Here is an example:

const altDescription = “This is a picture of a cat”;function doSomethingI{}const meow = (  
 <img   
 src=”images/cat.jpg”  
 alt={altDescription}  
 onClick={doSomething}  
 />  
);

**Writing conditionals in JSX**

Following the above logic, we’re under the impression that all you have to do is put your JavaScript between {} curly braces and it’ll just work. However, this won’t work for conditionals. This is because JSX wants you to return entire blocks rather than start adding conditional logic within the block to return parts of it.

The idea is to keep things as simple as possible and once you start breaking up HTML blocks, it can get messy over time.

So how do you write conditionals in JSX? You need to start with the condition and then return whatever block of HTML you want. Here is a syntax example:

if(random() === ‘cat’){  
 text = <p className=’cat’>random returned cat</p>;  
}else{  
 text = <p className=’other’>random returned something else</p>;  
}

So that’s the syntax for if statements in JSX. What about ternary operators?

A ternary operator looks something like this:

x ? y : z

But what exactly is the above example saying? In short, x refers to the condition, y is what gets executed if x is truthy, else z comes into effect.

Here is an example:

random() === ‘cat’ ? ‘print cat’ : ‘print dog’

Here is how you’d read the above ternary example: if random === ‘cat’, execute print cat , else execute print dog. You can run ternary operators inside JSX HTML tags.

For example:

const printText (  
 <p>  
 {random() === ‘cat’ ? ‘print cat’ : ‘print dog’}  
 </p>  
);

Which leaves us with the && operator.

The && operator is similar to the ? ternary, but without the second fallback condition. It’s basically, if the condition is truthy, do something. If the condition does not apply, then don’t do anything.

Here is the above example, but rewritten with a && operator.

Const printText (  
 <p>  
 {random() === ‘cat’ && ‘print cat’ }  
 </p>  
);

**Using .map in JSX**

The array method .map() is not something that is exclusive to JSX but already exists in JavaScript.

Map() lets you create a list based on the array that’s passed into the function. Here is an example:

const cats = [‘Tibbles’, ‘Tibbers’, ‘Bimber’];const printCats = cats.map(name => <li>{name}</li>);  
   
const listCats = <ul>{printCats}</ul>

The above will return something like this:

<ul>  
 <li>Tibbles</li>  
 <li>Tibbers</li>  
 <li>Bimber</li>  
</ul>

**Using keys in JSX with map()**

When there are lists involved, you need to have a unique key for each of your list items. key is a JSX attribute and is used by React internally to keep track of items. It's fine to work without it but can lead to your list order rendering in the wrong order.

We usually use key when the list items need to have a memory of the previous render. map() has a second parameter that lets you track the array item position.

Here is an example of how to use it:

const cats = ['Tibbles', 'Tibbers', 'Bimber'];const printCats = cats.map((name, i) => {  
 <li key={'cat\_'+ i}>{name}</li>);  
}  
   
const listCats = <ul>{printCats}</ul>

i is used as the placeholder for the second parameter in map().

You can use i by itself as the key. However, if you have multiple lists, then it might cause some issues through duplication.

**React FAQ:**

**In addition to**class**, is there any other JavaScript reserved words that is also an HTML attribute?**

Yes. In addition to class, which is represented as className in JSX, there is also for, which is written as htmlFor.

**Is JSX HTML or JavaScript?**

JSX is used to render HTML and JavaScript logic can sit inside the HTML to render logic and present data in a seamless way.

**Can we use JSX with anything else? (eg Java, PHP, C++)**

No. If you’re planning to mix and mingle JSX with other languages, it won’t work. JSX is an extension of JavaScript, not Java, PHP, C++ or anything else. It can sit inside these projects and configured to work, but you cannot put things in between {} and hope that it will run the same way as JavaScript does with JSX.

**Should I be using set variables or object properties to set attributes?**

It really depends on the situation. If you only need a single attribute, then a set variables is a good way to go. If you need unique instances for each one, then object properties may be the way to go.

Here is an example based on the scenario:

//single variable example   
const cats = ['Tibbles', 'Tibbers', 'Bimber'];  
const catClass = 'highlight';  
const printCats = cats.map((name, i) => {  
 <li   
 key={'cat\_'+i}   
 className={catClass}  
 >  
 {name}  
 </li>);  
}//object porperties example   
const navClasses = {  
 highlight: 'highlight',  
 normal: 'normal',  
 fade: 'fade'  
};const nav =(  
 <nav>  
 <span className={navClasses.highlight}>Home</span>  
 <span className={navClasses.normal}>About</span>   
 <span className={navClasses.normal}>Shop</span>  
 <span className={navClasses.fade}>Cart</span>  
 </nav>  
);

**Why don’t we set event listener attributes as function calls?**

This is how we set event listeners functions in JSX:

const text = (  
 <p onClick={switchText}>Click Me!</p>   
)

switchText is called as {switchText} instead of {switchText()} because if we turned it into a function, the function will get called automatically on page load. Without the () part, our event listener only triggers the function when the event occurs.

**Why can we use a ternary operator but not if statements in JSX?**

With if statements, there are multiple ways to write it and can get messy over time. With ternary ? and &&, the expected outcome is finite and absolute. Ternaries also evaluate to a value while if statements can involve mother expressions.

**Can we use**.forEach()**loop like we do with**.map()**?**

No. forEach() operates differently from .map(), so you cannot use it to render the same way. map() involves creating a new array. In contrast, forEach() loops through everything and doesn't keep any memory of the value.

However, if you want to use forEach(), you can still do so, but it is a bit longer to write because you'll need an array to hold what forEach() evaluates.

Here is an example of forEach() in action:

const cats = ['Tibbles', 'Tibbers', 'Bimber'];  
const catNames = [];cats.forEach((cat, i) => {  
 catNames.push(<li>{cat}</li>);  
});  
   
const listCats = <ul>{catNames}</ul>

In contrast, map() has one less step:

const cats = ['Tibbles', 'Tibbers', 'Bimber'];  
const printCats = cats.map(name => <li>{name}</li>);   
const listCats = <ul>{printCats}</ul>