Title: What is JSX?

Answer:

JSX is a technology that was introduced by React.

Although React can work completely fine without using JSX, it's an ideal technology to work with components, so React benefits a lot from JSX.

At first, you might think that using JSX is like mixing HTML and JavaScript (and as you'll see CSS).

But this is not true, because what you are really doing when using JSX syntax is writing a declarative syntax of what a component UI should be.

And you're describing that UI not using strings, but instead using JavaScript, which allows you to do many nice things.

A JSX PRIMER

Here is how you define a h1 tag containing a string:

```
const element = <h1>Hello, world!</h1>
```

It looks like a strange mix of JavaScript and HTML, but in reality it's all JavaScript.

What looks like HTML, is actually syntactic sugar for defining components and their positioning inside the markup.

Inside a JSX expression, attributes can be inserted very easily:

```
const myId = 'test'
const element = <h1 id={myId}>Hello, world!</h1>
```

You just need to pay attention when an attribute has a dash (`-`) which is converted to camelCase syntax instead, and these 2 special cases:

- `class` becomes `className`
- `for` becomes `htmlFor`

because they are reserved words in JavaScript.

Here's a JSX snippet that wraps two components into a `div` tag:

```
<div>
  <BlogPostsList />
  <Sidebar />
</div>
```

A tag always needs to be closed, because this is more XML than HTML (if you remember the XHTML days, this will be familiar, but since then the HTML5 loose syntax won). In this case a self-closing tag is used.

Notice how I wrapped the 2 components into a `div` . Why? Because **the render() function can only return a single node**, so in case you want to return 2 siblings, just add a parent. It can be any tag, not just `div` .

TRANSPILING JSX

A browser cannot execute JavaScript files containing JSX code. They must be first transformed to regular JS.

How? By doing a process called **transpiling**.

We already said that JSX is optional, because to every JSX line, a corresponding plain JavaScript alternative is available, and that's what JSX is transpiled to.

For example the following two constructs are equivalent:

Plain JS

```
ReactDOM.render(
  React.DOM.div(
    { id: 'test' },
    React.DOM.h1(null, 'A title'),
    React.DOM.p(null, 'A paragraph')
  ),
  document.getElementById('myapp')
)
```

JSX

```
ReactDOM.render(
  <div id="test">
    <h1>A title</h1>
    <p>A paragraph</p>
  </div>,
  document.getElementById('myapp')
)
```

This very basic example is just the starting point, but you can already see how more complicated the plain JS syntax is compared to using JSX.

At the time of writing the most popular way to perform the **transpilation** is to use **Babel**, which is the default option when running `create-react-app`, so if you use it you don't have to worry, everything happens under the hoods for you.

If you don't use `create-react-app` you need to setup Babel yourself.

JS IN JSX

JSX accepts any kind of JavaScript mixed into it.

Whenever you need to add some JS, just put it inside curly braces `{}`. For example here's how to use a constant value defined elsewhere:

```
const paragraph = 'A paragraph'
ReactDOM.render(
  <div id="test">
    <h1>A title</h1>
    <p>{paragraph}</p>
  </div>,
  document.getElementById('myapp')
)
```

This is a basic example. Curly braces accept *any* JS code:

```
const paragraph = 'A paragraph'
ReactDOM.render(
  <table>
    {rows.map((row, i) => {
      return <tr>{row.text}</tr>
    })}
  </div>,
  document.getElementById('myapp')
)
```

As you can see *we nested JavaScript inside a JSX defined inside a JavaScript nested in a JSX*. You can go as deep as you need.

HTML IN JSX

JSX resembles a lot HTML, but it's actually a XML syntax.

In the end you render HTML, so you need to know a few differences between how you would define some things in HTML, and how you define them in JSX.

YOU NEED TO CLOSE ALL TAGS

Just like in XHTML, if you have ever used it, you need to close all tags: no more `<br>` but instead use the self-closing tag: `<br />` (the same goes for other tags)

CAMELCASE IS THE NEW STANDARD

In HTML you'll find attributes without any case (e.g. `onchange`). In JSX, they are renamed to their camelCase equivalent:

- `onchange => onChange`
- `onclick => onClick`
- `onsubmit => onSubmit`

`class` BECOMES `className`

Due to the fact that JSX is JavaScript, and `class` is a reserved word, you can't write

```
<p class="description">
```

but you need to use

```
<p className="description">
```

The same applies to `for` which is translated to `htmlFor` .

THE STYLE ATTRIBUTE CHANGES ITS SEMANTICS

The `style` attribute in HTML allows to specify inline style. In JSX it no longer accepts a string, and in CSS in React you'll see why it's a very convenient change.

FORMS

Form fields definition and events are changed in JSX to provide more consistency and utility.

Forms in JSX goes into more details on forms.

CSS IN REACT

JSX provides a cool way to define CSS.

If you have a little experience with HTML inline styles, at first glance you'll find yourself pushed back 10 or 15 years, to a world where inline CSS was completely normal (nowadays it's demonized and usually just a "quick fix" go-to solution).

JSX style is not the same thing: first of all, instead of accepting a string containing CSS properties, the JSX `style` attribute only accepts an object. This means you define properties in an object:

```
var divStyle = {
  color: 'white'
}

ReactDOM.render(<div style={divStyle}>Hello World!</div>, mountNode)
```

or

```
ReactDOM.render(<div style={{ color: 'white' }}>Hello World!</div>, mountNod
e)
```

The CSS values you write in JSX is slightly different than plain CSS:

- the keys property names are camelCased
- values are just strings
- you separate each tuple with a comma

WHY IS THIS PREFERRED OVER PLAIN CSS / SASS / LESS?

CSS is an **unsolved problem**. Since its inception, dozens of tools around it rose and then fell. The main problem with JS is that there is no scoping and it's easy to write CSS that is not enforced in any way, thus a "quick fix" can impact elements that should not be touched.

JSX allows components (defined in React for example) to completely encapsulate their style.

IS THIS THE GO-TO SOLUTION?

Inline styles in JSX are good until you need to

1. write media queries
2. style animations
3. reference pseudo classes (e.g. `:hover`)
4. reference pseudo elements (e.g. `::first-letter`)

In short, they cover the basics, but it's not the final solution.

FORMS IN JSX

JSX adds some changes to how HTML forms work, with the goal of making things easier for the developer.

VALUE AND DEFAULTVALUE

The `value` attribute always holds the current value of the field.

The `defaultValue` attribute holds the default value that was set when the field was created.

This helps solve some weird behavior of regular DOM interaction when inspecting `input.value` and `input.getAttribute('value')` returning one the current value and one the original default value.

This also applies to the `textarea` field, e.g.

```
<textarea>Some text</textarea>
```

but instead

```
<textarea defaultValue={'Some text'} />
```

For `select` fields, instead of using

```
<select>
  <option value="x" selected>
    ...
  </option>
</select>
```

use

```
<select defaultValue="x">
  <option value="x">...</option>
</select>
```

A MORE CONSISTENT ONCHANGE

Passing a function to the `onChange` attribute you can subscribe to events on form fields.

It works consistently across fields, even `radio`, `select` and `checkbox` input fields fire a `onChange` event.

`onChange` also fires when typing a character into a `input` or `textarea` field.

JSX AUTO ESCAPES

To mitigate the ever present risk of XSS exploits, JSX forces automatic escaping in expressions.

This means that you might run into issues when using an HTML entity in a string expression.

You expect the following to print `© 2017`:

```
<p>{'&copy; 2017'}</p>
```

But it's not, it's printing `&copy; 2017` because the string is escaped.

To fix this you can either move the entities outside the expression:

```
<p>&copy; 2017</p>
```

or by using a constant that prints the Unicode representation corresponding to the HTML entity you need to print:

```
<p>{'\u00A9 2017'}</p>
```

WHITE SPACE IN JSX

To add white space in JSX there are 2 rules:

HORIZONTAL WHITE SPACE IS TRIMMED TO 1

If you have white space between elements in the same line, it's all trimmed to 1 white space.

```
<p>Something       becomes              this</p>
```

becomes

```
<p>Something becomes this</p>
```

VERTICAL WHITE SPACE IS ELIMINATED

```
<p>
  Something
  becomes
  this
</p>
```

becomes

```
<p>Somethingbecomesthis</p>
```

To fix this problem you need to explicitly add white space, by adding a space expression like this:

```
<p>
  Something
  {' '}becomes
  {' '}this
</p>
```

or by embedding the string in a space expression:

```
<p>
  Something
  {' becomes '}
  this
</p>
```

ADDING COMMENTS IN JSX

You can add comments to JSX by using the normal JavaScript comments inside an expression:

```
<p>
  {/* a comment */}
  {
    //another comment
  }
</p>
```

SPREAD ATTRIBUTES

In JSX a common operation is assigning values to attributes.

Instead of doing it manually, e.g.

```
<div>
  <BlogPost title={data.title} date={data.date} />
</div>
```

you can pass

```
<div>
  <BlogPost {...data} />
</div>
```

and the properties of the `data` object will be used as attributes automatically, thanks to the *ES6 spread operator*

Tags: react