React Js Online Training

sahosoft

Online Learning Platform

07

By: Ajeet Kumar

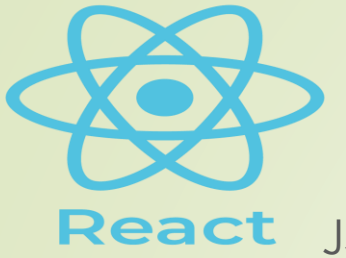
sahosoft

Online Learning Platform

JSX In Depth (Complete JSX)

07

sahosoft
Online Learning Platform



JSX In Depth

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() methods.

Note: *You are not required to use JSX, but JSX makes it easier to write React applications.*

Using JSX: Single Element

If we want to return single element then we can return single element easily. JSX looks like a regular HTML in most cases. Even though it's similar to HTML, there are a couple of things we need to keep in mind when working with JSX.

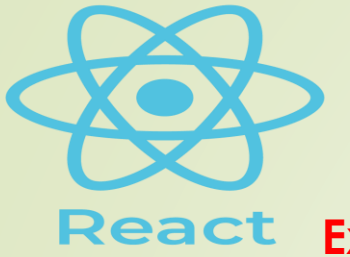
Using JSX: Nested Elements

If we want to return more elements, we need to wrap it with one container element. Means ***Inserting a Large Block of HTML: with a wrapper ()***

By wrapping our entire JSX return with parentheses, we can make a multi-line JSX return.

To write HTML on multiple lines, put the HTML inside parentheses

Note: You can return a JSX element directly or you can have it wrapped with () for larger blocks of code.



JSX In Depth

Expressions in JSX

With JSX you can write expressions inside curly braces { }.

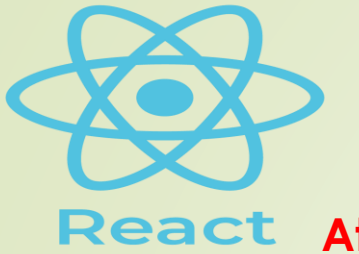
The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result.

Elements Must be Closed

JSX follows XML rules, and therefore HTML elements must be properly closed. Means you need to close all tags.

You need to close all tags: no more **
** but instead use the self-closing tag: **
** (the same goes for other tags)

Note: JSX will throw an error if the HTML is not properly closed.



JSX In Depth

Attributes

JSX allows us to use attributes with the HTML elements just like we do with normal HTML. But instead of the normal naming convention of HTML, JSX uses **camelcase** convention for attributes.

For example, **class** in HTML becomes **className** in JSX. The main reason behind this is that some of the attribute names in HTML like **'class'** are reserved keywords in JavaScripts. So, in order to avoid this problem, JSX uses the camel case naming convention for attributes.

- We can also use custom attributes in JSX. For custom attributes, the names of such attributes should be prefixed by **data-**.

Specifying attribute values: JSX allows us to specify attribute values in two ways:

1.As string literals: We can specify the values of attributes as hard-coded strings using quotes:

```
var ele = <h1 className = "firstAttribute">Hello Sahosoft</h1>;
```

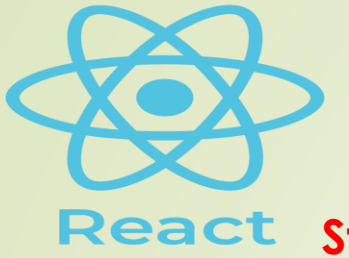
2.As expressions: We can specify attributes as expressions using curly braces {}:

```
var ele = <h1 className = {varName}>Hello Sahosoft</h1>;
```

The style attribute changes its semantics

The style attribute in HTML allows to specify inline style. In JSX it no longer accepts a string.

JSX style attribute only accepts an object. This means you define properties in an object.



JSX In Depth

Styling

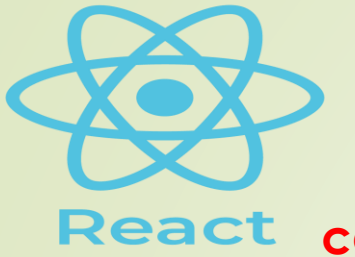
JSX provides a cool way to define CSS.

When we want to set inline styles, we need to use camelCase syntax. React will also automatically append px after the number value on specific elements.

JSX style attribute only accepts an object. This means you define properties in an object.

The CSS values you write in JSX are slightly different from plain CSS:

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



JSX In Depth

camelCase is the new standard

In HTML you'll see 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

tabindex becomes tabIndex.

The same applies to for which is translated to htmlFor.