Introduction to JSX

36th **Day** {36%}

WHAT IS jsx?

JSX stands for JavaScript syntax extension which allows us to write HTML in React. JSX makes it easier to write and add HTML in React. In other words, JSX is an HTML-like syntax used by React that extends ECMAScript so that HTML-like syntax can co-exist with JavaScript/React code. The syntax is used by preprocessors (i.e., transpilers like babel) to transform HTML-like syntax into standard JavaScript objects that a JavaScript engine will parse.

JSX provides you to write HTML/XML-like structures (e.g., DOM-like tree structures) in the same file where you write JavaScript code, then preprocessor will transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children. JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.

JSX converts HTML tags into react elements.

Note that you are not required to use JSX, but JSX makes it easier to write React applications.

Example:

```
const myelement = <h1>First Dome  JSX  element </h1>;
ReactDOM.render(myelement, document.getElementById('root'));
```

The same element can be written with out JSX and it will appear as show below:

```
const myelement = React.createElement('h1', {}, 'I do not use JSX!');
ReactDOM.render(myelement, document.getElementById('root'));
```

As you can see JSX allows us to write HTML directly within the JavaScript code. JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.

Why use JSX ?

- It is faster than regular JavaScript because it performs optimization while translating the code to JavaScript.
- Instead of separating technologies by putting markup and logic in separate files, React uses components that contain both. We will learn components in a further section.
- It is type-safe, and most of the errors can be found at compilation time.
- It makes easier to create templates.

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

example:

```
const myelement = <h1>React is {5 + 5} times better with JSX</h1>;
```

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

Example:

```
const myelement = (
  <ul>

    <li>Apples</li>

    <li>Bananas</li>

    <li>Cherries</li>

  </ul>

);
```

The HTML code must be wrapped in ONE top level element. So if you like to write two headers, you must put them inside a parent element, like a div element

```
const myelement = (


  <div>

    <h1>I am a Header.</h1>

    <h1>I am a Header too.</h1>

  </div>

);
```

JSX Attributes

JSX use attributes with the HTML elements same as regular HTML. JSX uses camelcase naming convention for attributes rather than standard naming convention of HTML such as a class in HTML becomes className in JSX because the class is the reserved keyword in JavaScript. We can also use our own custom attributes in JSX. For custom attributes, we need to use data- prefix. In the below example, we have used a custom attribute data-demoAttribute as an attribute for the <p> tag.

Example:

```
import React, { Component } from 'react';

class App extends Component{
    render(){
       return(
          <div>
              <h1> #100 Days of Code with Lux</h1>
            <h2>Training Institutes</h2>
              <p data-demoAttribute = "demo">
                  The best Frontend course in 2020.
             </p>
          </div>
       );
    }
 }
 export default App;
```

In JSX, we can specify attribute values in two ways:

a). As String Literals : We can specify the values of attributes in double quotes:

Demo: https://github.com/HarunHM/React-Tutorial/blob/master/string-literals.jsx

```
var element = <h2 className = "firstAttribute">Hello JS learners</h2>;
```

b). As Expressions: We can specify the values of attributes as expressions using curly braces {}

Demo: https://github.com/HarunHM/React-Tutorial/blob/master/expressions.jsx

```
Var element = <h2 className = {varName}>Hello JS learners</h2>;
```

JSX Comments.

JSX allows us to use comments that begin with /* and ends with */ and wrapping them in curly braces {} just like in the case of JSX expressions.

Below example shows how to use comments in JSX.

```
import React, { Component } from 'react';

 class App extends Component{
    render(){
       return(
          <div>
             <h1 className = "hello" > #100 Days of Code with Lux </h1>
       {/* This is a comment in JSX */}
          </div>
      );
    }
 }
 export default App;
```

JSX Styling.

React always recommends to use **inline** styles. To set inline styles, you need to use **camelCase** syntax. React automatically allows appending **px** after the number value on specific elements.

The following example shows how to use styling in the element.

```
import React, { Component } from 'react';

 class App extends Component{
    render(){
      var myStyle = {
          fontSize: 80,
          fontFamily: 'Courier',
          color: '#003300'
       }
```

```
        return (
            <div>
                <h1 style = {myStyle}> #100 Days of Code with Lux </h1>
            </div>
        );
    }
}
 export default App;
```

It is important to note that you cannot use if-else statements in JSX.

Instead of it, you can use conditional (ternary) expressions.

Example:

```
import React, { Component } from 'react';

class App extends Component{
   render(){
      var i = 5;
      return (
         <div>
            <h1>{i == 1 ? 'True!' : 'False!'}</h1>
         </div>
      );
   }
}
export default App;
```

More resources: – https://reactjs.org/docs/introducing-jsx.html

– https://reactjs.org/docs/jsx-in-depth.html

Best wishes Lux Tech Academy