|  |  |
| --- | --- |
| **ch8.ReactJS JSX** | **Date: 22-02-2022** |

**Topics**

React JSX, Why use JSX, Nested Elements in JSX, JSX Attributes, JSX Comments, JSX Styling

**React JSX**

* In React class-based components have a **render** function which consist **return** statements and Functional based component just have **return** statement.
* The render function or return statement specifies the HTML output of a React component.
* JSX (JavaScript Extension), is a React extension which allows writing JavaScript code that looks like HTML.
* 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 **pre-processors** (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 pre-processor will transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

**Example**

Here, we will write JSX syntax in JSX file and see the corresponding JavaScript code which transforms by preprocessor(babel).

|  |  |
| --- | --- |
| **JSX syntax**  <div>Hello JavaTpoint</div> | **corresponding JavaScript code**  *React*.createElement("div", null, "Hello JavaTpoint"); |

|  |
| --- |
| The above line creates a **react element** and passing **three arguments** inside where the first is the name of the element which is div, second is the **attributes** passed in the div tag, and last is the **content** you pass which is the "Hello JavaTpoint." |

## **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.

## **Nested Elements in JSX**

To use more than one element, you need to wrap it with one container element. Here, we use **div** as a container element which has **three** nested elements inside it.

**App.JSX**

import React, { Component } from 'react';

class App extends Component{

   render(){

      return(

         <div>

            <h1>JavaTpoint</h1>

          <h2>Training Institutes</h2>

            <p>This website contains the best CS tutorials.</p>

         </div>

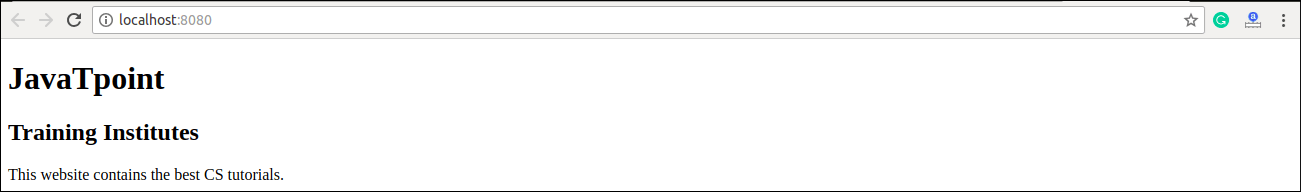
      );

   }

}

export default App;

**Output:**



## **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**

1. **import** React, { Component } from 'react';
2. **class** App **extends** Component{
3. render(){
4. **return**(
5. <div>
6. <h1>JavaTpoint</h1>
7. <h2>Training Institutes</h2>
8. <p data-demoAttribute = "demo">This website contains the best CS tutorials.</p>
9. </div>
10. );
11. }
12. }
13. export **default** App;

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

**1. As String Literals:** We can specify the values of attributes in double quotes:

1. var element = <h2 className = "firstAttribute">Hello JavaTpoint</h2>;

**Example**

1. **import** React, { Component } from 'react';
2. **class** App **extends** Component{
3. render(){
4. **return**(
5. <div>
6. <h1 className = "hello" >JavaTpoint</h1>
7. <p data-demoAttribute = "demo">This website contains the best CS tutorials.</p>
8. </div>
9. );
10. }
11. }
12. export **default** App;

**Output:**

JavaTpoint

This website contains the best CS tutorials.

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

1. var element = <h2 className = {varName}>Hello JavaTpoint</h2>;

**Example**

1. **import** React, { Component } from 'react';
2. **class** App **extends** Component{
3. render(){
4. **return**(
5. <div>
6. <h1 className = "hello" >{25+20}</h1>
7. </div>
8. );
9. }
10. }
11. export **default** App;

**Output:**

45

## **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.

### **Example**

1. **import** React, { Component } from 'react';
2. **class** App **extends** Component{
3. render(){
4. **return**(
5. <div>
6. <h1 className = "hello" >Hello JavaTpoint</h1>
7. {/\* This is a comment in JSX \*/}
8. </div>
9. );
10. }
11. }
12. 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.

### **Example**

1. **import** React, { Component } from 'react';
2. **class** App **extends** Component{
3. render(){
4. var myStyle = {
5. fontSize: 80,
6. fontFamily: 'Courier',
7. color: '#003300'
8. }
9. **return** (
10. <div>
11. <h1 style = {myStyle}>www.javatpoint.com</h1>
12. </div>
13. );
14. }
15. }
16. export **default** App;

**Output:**



#### **NOTE: JSX cannot allow to use if-else statements. Instead of it, you can use conditional (ternary) expressions. It can be seen in the following example.**

### **Example**

1. **import** React, { Component } from 'react';
2. **class** App **extends** Component{
3. render(){
4. var i = 5;
5. **return** (
6. <div>
7. <h1>{i == 1 ? 'True!' : 'False!'}</h1>
8. </div>
9. );
10. }
11. }
12. export **default** App;

**Output:**

False!