React JSX

As we have already seen that, all of the React components have a **render** function. The render function 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 **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.

Example

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

JSX File

```
1. <div>Hello JavaTpoint</div>
```

Corresponding Output

```
1. 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.

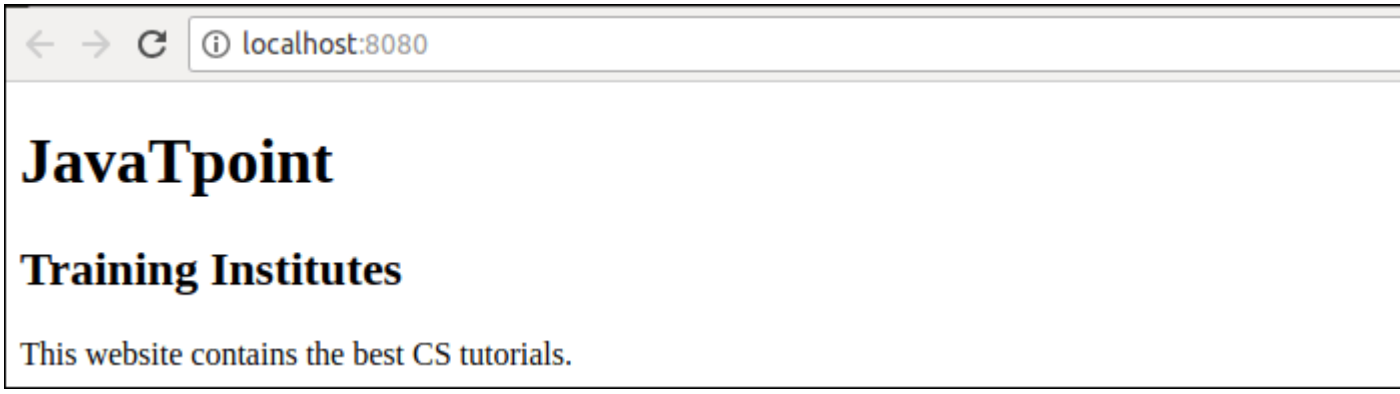
Nested Elements in JSX

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:

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

Example

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
        <h1 className = "hello" >JavaTpoint</h1>
        <p data-demoAttribute = "demo">This website contains the best CS tutorials.</p>
      </div>
    );
  }
}
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 {}:

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

Example

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
        <h1 className = "hello" >{25+20}</h1>
      </div>
    );
  }
}
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

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
        <h1 className = "hello" >Hello JavaTpoint</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.

Example

```
import React, { Component } from 'react';
class App extends Component{
   render(){
      var myStyle = {
         fontSize: 80,
         fontFamily: 'Courier',
         color: '#003300'
      }
      return (
         <div>
            <h1 style = {myStyle}>www.javatpoint.com</h1>
         </div>
      );
   }
}
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:

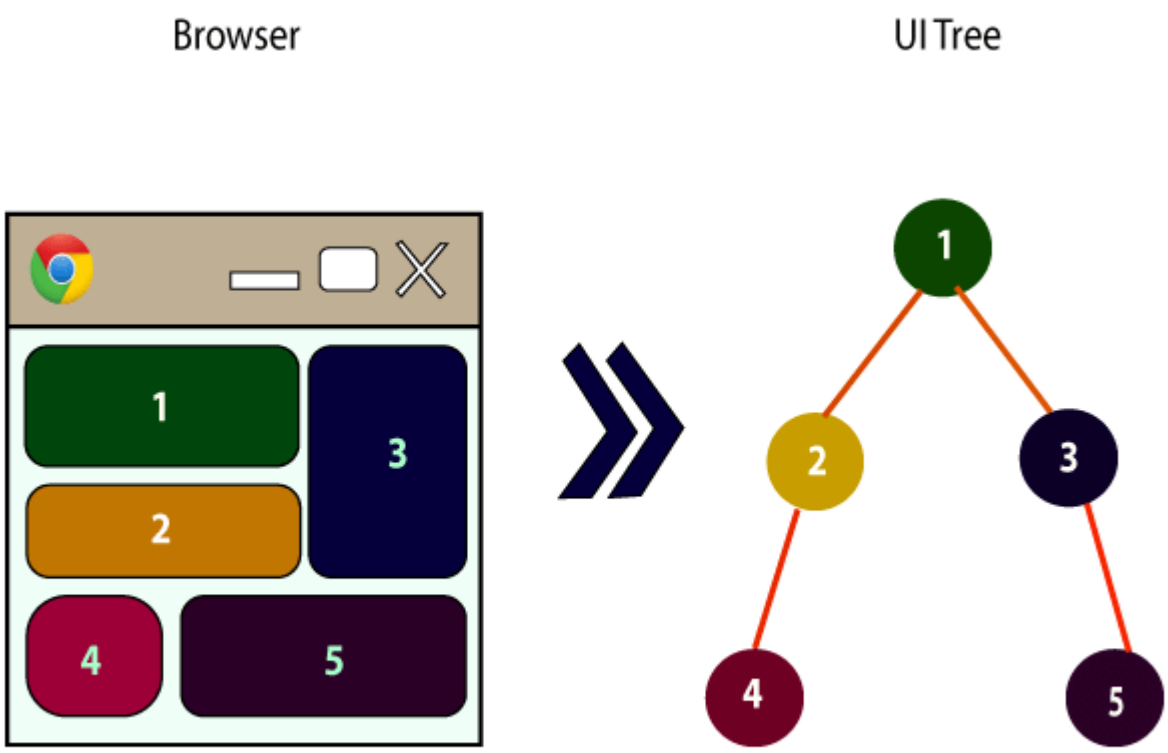
```
False!
```

React Components

Earlier, the developers write more than thousands of lines of code for developing a single page application. These applications follow the traditional DOM structure, and making changes in them was a very challenging task. If any mistake found, it manually searches the entire application and update accordingly. The component-based approach was introduced to overcome an issue. In this approach, the entire application is divided into a small logical group of code, which is known as components.

A Component is considered as the core building blocks of a React application. It makes the task of building UIs much easier. Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.

Every React component have their own structure, methods as well as APIs. They can be reusable as per your need. For better understanding, consider the entire UI as a tree. Here, the root is the starting component, and each of the other pieces becomes branches, which are further divided into sub-branches.



In ReactJS, we have mainly two types of components. They are

1. Functional Components
2. Class Components

Functional Components

In React, function components are a way to write components that only contain a render method and don't have their own state. They are simply JavaScript functions that may or may not receive data as parameters. We can create a function that takes props(properties) as input and returns what should be rendered. A valid functional component can be shown in the below example.

```
function WelcomeMessage(props) {
  return <h1>Welcome to the , {props.name}</h1>;
}
```

The functional component is also known as a stateless component because they do not hold or manage state. It can be explained in the below example.

Example

```
import React, { Component } from 'react';
class App extends React.Component {
   render() {
      return (
         <div>
            <First/>
            <Second/>
         </div>
      );
   }
}
class First extends React.Component {
   render() {
      return (
         <div>
            <h1>JavaTpoint</h1>
         </div>
      );
   }
```

```
20. }
21. class Second extends React.Component {
22.   render() {
23.     return (
24.       <div>
25.         <h2>www.javatpoint.com</h2>
26.         <p>This websites contains the great CS tutorial.</p>
27.       </div>
28.     );
29.   }
30. }
31. export default App;
```

Output:

Class Components

Class components are more complex than functional components. It requires you to extend from React. Component and create a render function which returns a React element. You can pass data from one class to other class components. You can create a class by defining a class that extends Component and has a render function. Valid class component is shown in the below example.

```
1. class MyComponent extends React.Component {
2.   render() {
3.     return (
4.       <div>This is main component.</div>
5.     );
6.   }
7. }
```

The class component is also known as a stateful component because they can hold or manage local state. It can be explained in the below example.

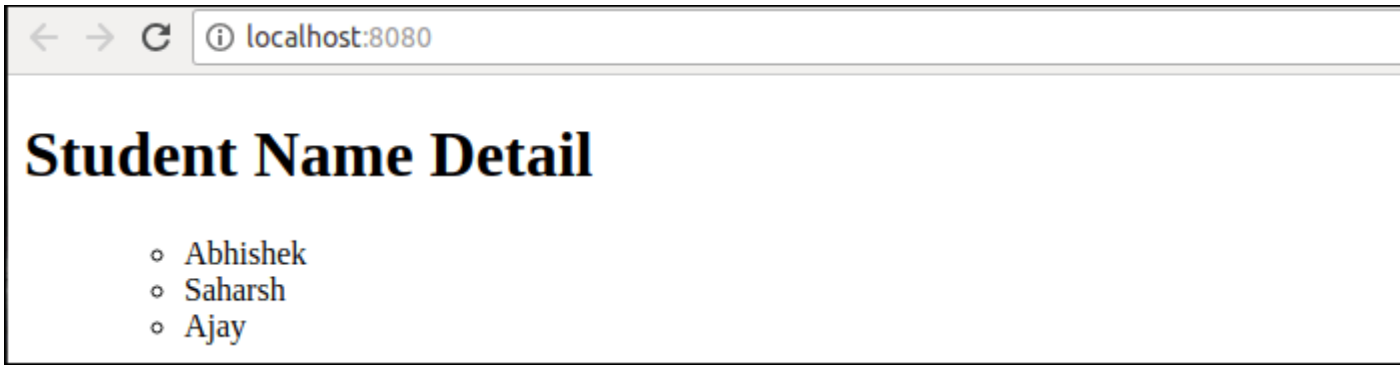
Example

In this example, we are creating the list of unordered elements, where we will dynamically insert StudentName for every object from the data array. Here, we are using ES6 arrow syntax (=>) which looks much cleaner than the old JavaScript syntax. It helps us to create our elements with fewer lines of code. It is especially useful when we need to create a list with a lot of items.

```
1.  import React, { Component } from 'react';
2.  class App extends React.Component {
3.   constructor() {
4.      super();
5.      this.state = {
6.        data:
7.        [
8.          {
9.            "name":"Abhishek"
10.         },
11.         {
12.           "name":"Saharsh"
13.         },
14.         {
15.           "name":"Ajay"
16.         }
17.       ]
18.     }
19.   }
20.   render() {
21.     return (
22.       <div>
23.         <StudentName/>
24.         <ul>
25.           {this.state.data.map((item) => <List data = {item} />)}
26.         </ul>
27.       </div>
28.     );
```

```
29.   }
30. }
31. class StudentName extends React.Component {
32.   render() {
33.     return (
34.       <div>
35.         <h1>Student Name Detail</h1>
36.       </div>
37.     );
38.   }
39. }
40. class List extends React.Component {
41.   render() {
42.     return (
43.       <ul>
44.         <li>{this.props.data.name}</li>
45.       </ul>
46.     );
47.   }
48. }
49. export default App;
```

Output:



React State

The state is an updatable structure that is used to contain data or information about the component. The state in a component can change over time. The change in state over time can happen as a response to user action or system event. A component with the state is known as stateful components. It is the heart of the react component which determines the behavior of the component and how it will render. They are also responsible for making a component dynamic and interactive.

A state must be kept as simple as possible. It can be set by using the **setState()** method and calling setState() method triggers UI updates. A state represents the component's local state or information. It can only be accessed or modified inside the component or by the component directly. To set an initial state before any interaction occurs, we need to use the **getInitialState()** method.

For example, if we have five components that need data or information from the state, then we need to create one container component that will keep the state for all of them.

Defining State

To define a state, you have to first declare a default set of values for defining the component's initial state. To do this, add a class constructor which assigns an initial state using this.state. The '**this.state**' property can be rendered inside **render()** method.

Example

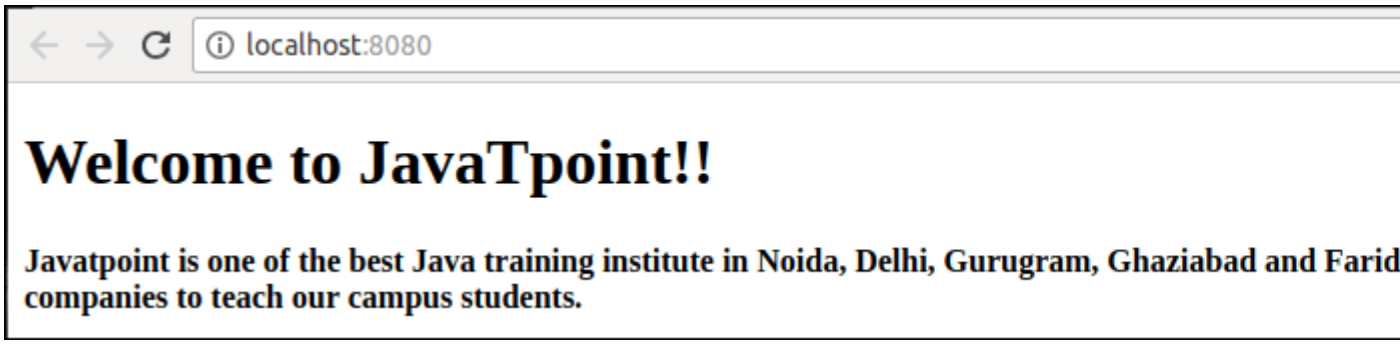
The below sample code shows how we can create a stateful component using ES6 syntax.

```
1. import React, { Component } from 'react';
2. class App extends React.Component {
3.  constructor() {
4.      super();
5.      this.state = { displayBio: true };
6.      }
7.      render() {
8.         const bio = this.state.displayBio ? (
9.             <div>
```

```
10.                 <p><h3>Javatpoint is one of the best Java training institute in Noida, Delhi, Guru
    gram, Ghaziabad and Faridabad. We have a team of experienced Java developers and trainer
    s from multinational companies to teach our campus students.</h3></p>
11.             </div>
12.              ) : null;
13.              return (
14.                 <div>
15.                     <h1> Welcome to JavaTpoint!! </h1>
16.                     { bio }
17.                 </div>
18.             );
19.     }
20. }
21. export default App;
```

To set the state, it is required to call the super() method in the constructor. It is because this.state is uninitialized before the super() method has been called.

Output



Changing the State

We can change the component state by using the setState() method and passing a new state object as the argument. Now, create a new method toggleDisplayBio() in the above example and bind this keyword to the toggleDisplayBio() method otherwise we can't access this inside toggleDisplayBio() method.

```
1. this.toggleDisplayBio = this.toggleDisplayBio.bind(this);
```

Example

In this example, we are going to add a **button** to the **render**() method. Clicking on this button triggers the toggleDisplayBio() method which displays the desired output.

```
1. import React, { Component } from 'react';
2. class App extends React.Component {
3.  constructor() {
4.      super();
5.      this.state = { displayBio: false };
6.      console.log('Component this', this);
7.      this.toggleDisplayBio = this.toggleDisplayBio.bind(this);
8.      }
9.      toggleDisplayBio(){
10.        this.setState({displayBio: !this.state.displayBio});
11.        }
12.     render() {
13.        return (
14.           <div>
15.              <h1>Welcome to JavaTpoint!!</h1>
16.              {
17.                 this.state.displayBio ? (
18.                    <div>
19.                       <p><h4>Javatpoint is one of the best Java training institute in Noida, Del
    hi, Gurugram, Ghaziabad and Faridabad. We have a team of experienced Java developers an
    d trainers from multinational companies to teach our campus students.</h4></p>
20.                       <button onClick={this.toggleDisplayBio}> Show Less </button>
21.                    </div>
22.                    ) : (
23.                       <div>
24.                          <button onClick={this.toggleDisplayBio}> Read More </button>
25.                       </div>
26.                    )
27.              }
28.           </div>
29.       )
30.    }
31. }
32. export default App;
```

Output:

When you click the **Read More** button, you will get the below output, and when you click the **Show Less** button, you will get the output as shown in the above image.