ReactJS

Lecture 4

- React JSX

React JSX

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.

React JSX

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.

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.

JSX Attributes

JSX supports HTML like attributes. All HTML tags and its attributes are supported. Attributes has to be specified using camelCase convention (and it follows JavaScript DOM API) instead of normal HTML attribute name. For example, class attribute in HTML has to be defined as *className*. The following are few other examples −

- *htmlFor* instead of *for*
- *tabIndex* instead of *tabindex*
- *onClick* instead of *onclick*

JSX Attributes

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
        <h1>JavaTpoint</h1>
       <h2>Training Institutes</h2>
        <p data-demoAttribute = "demo">This website contains the best CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

JSX Attributes

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>;
```

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

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

```
  return(
      <div>
        <h1 className = "hello" >{25+20}</h1>
      </div>
    );
```

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" >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.

```
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;
```

Use of ternary operators

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.

```
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;
```