Agenda

- What is JSX?
- How to write complex JSX?
- Using Javascript in JSX and Adding CSS Classes in JSX
- What is a component in react?
- Functional and Class components
- Conditional Rendering In React

What is JSX?

Before studying about components in react we must first understand what is JSX. JSX stands for JavaScript XML.JSX is a JavaScript Extension Synta used in React to easily write HTML and JavaScript together. Take a look at the below code:

```
const jsx = <h1>This is JSX</h1>
```

This is simple JSX code in React. But the browser does not understand this JSX because it's not valid JavaScript code. This is because we're assignir an HTML tag to a variable that is not a string but just HTML code. So to convert it to browser understandable JavaScript code, react uses a to like Babel which is a JavaScript compiler/transpiler.

Babel is responsible to transform JavaScript code from one format to another based on specified configuration. Now, if your browser needs to display th above information, it needs to know to get hold of the HTML view and the regular old JavaScript. This looks like a perfect job for 'Babel' which accep the JSX input and transforms it into a valid JavaScript that can be understood by the browser.

We can use the above JSX in our React code like this:

```
class JSXDemo extends React.Component {
    render() {
        return <h1>This is JSX</h1>;
    }
}

ReactDOM.render(<JSXDemo />, document.getElementById('root'));
```

Here, we're returning the JSX from the `JSXDemo` component and rendering that on the screen using the `ReactDOM.render` method.

When the Babel executes the above JSX, it converts it to the following code:

```
class JSXDemo extends React.Component {
    render() {
        return React.createElement("h1", null, "This is JSX");
    }
}
```

As you can see in the above Code Sandbox, the code still correctly prints the contents to the screen using `React.createElement` .

This was the old way of writing code in React – but it's tedious to write the `React.createElement` every time, even for adding a simple div.

So React introduced the JSX way of writing code which makes code easy to write and understand.

> Knowing how to convert JSX to `React.createElement` is very important as a React developer (it's also a popular interview question).

What is the React.createElement Function?

Every JSX is converted to the `React.createElement` function call that the browser understands.

The `React.createElement` has the following syntax:

```
React.createElement(type, [props], [...children])
```

Let's look at the parameters of the `createElement` function.

- **type** can be an HTML tag like h1, div or it can be a React component
- **props** are the attributes you want the element to have

- **children** contain other HTML tags or can be a component

The `React.createElement` call will also be converted to the object representation like this:

```
{
 type: 'h1',
 props: {
   children: 'This is JSX'
 }
}
```

You can see this object representation if you assign the JSX to some local variable and log it as shown below:

```
class JSXDemo extends React.Component {
    render() {
        const jsx = <h1>This is JSX</h1>;
        console.log(jsx);
        return jsx;
    }
}

ReactDOM.render(<JSXDemo />, document.getElementById('root'));
```

You will see the log printed as shown below:

Now, take a look at the below code:

```
class JSXDemo extends React.Component {
  render() {
    const jsx = <h1 id="jsx">This is JSX</h1>;
    console.log(jsx);
    return jsx;
  }
}

ReactDOM.render(<JSXDemo />, document.getElementById("root"));
```

Here, we've used the JSX like this:

```
<h1 id="jsx">This is JSX</h1>
```

So React, will convert this JSX to the below code:

```
React.createElement("h1", { id: "jsx" }, "This is JSX");
```

If there are any attributes added to the HTML tag as in our case, they will be passed as the second parameter for the `React.createElement` call. Th object representation will look like this:

```
{
  type: 'h1',
  props: {
   id: 'jsx',
   children: 'This is JSX'
```

```
  }
}
```

You will see the log printed as shown below:

So from all the above examples, it's clear that JSX is converted to a `React.createElement` call and it's then converted to its object representation.

How to Return Complex JSX

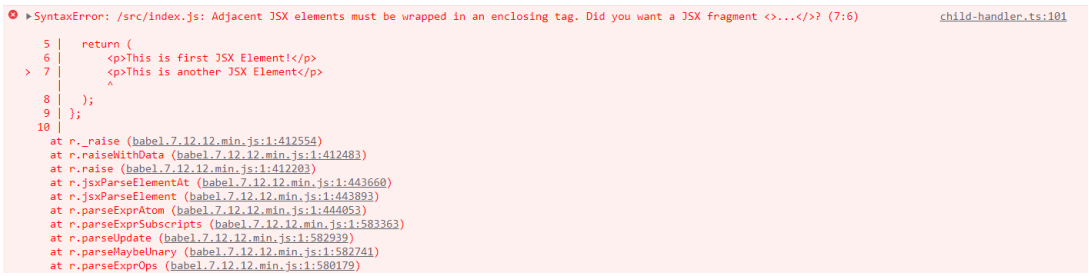
Take a look at the below code:

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  return (
      <p>This is first JSX Element!</p>
      <p>This is another JSX Element</p>
  );
};

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

Here, we're returning two paragraphs from the App component. But if you run the code, you will get this error:



We're getting an error because React requires adjacent elements to be wrapped in a parent tag.

As we have seen, `<p>This is first JSX Element!</p>` will be converted to `React.createElement("p", null, "This is first JSX Element!")` and `<p>This is another JSX Element</p>` will be converted to `React.createElement("p", null, "This is another JSX Element")` .

The converted code will look like this now:

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  return (
            React.createElement("p", null, "This is first JSX Element!");
                        React.createElement("p", null, "This is another JSX Element");
        );
};

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

Here we are returning two things from the `App` component which will not work because there is no parent element to wrap both of them.

To make it work, the obvious solution is to wrap both of them in some parent element, most probably a `div` like this:

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  return (
    <div>
      <p>This is first JSX Element!</p>
      <p>This is another JSX Element</p>
    </div>
  );
};

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

The other way to fix it is by using the `React.Fragment` component:

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  return (
    <React.Fragment>
      <p>This is first JSX Element!</p>
      <p>This is another JSX Element</p>
    </React.Fragment>
  );
};

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

`React.Fragment` was added in React version 16.2 because we always have to wrap multiple adjacent elements in some tag (like div) inside eve JSX returned by a component. But that adds unnecessary div tags.

This is fine most of the time but there are certain cases where it's not fine.

For example, if we're using Flexbox, then there is a special parent-child relationship in Flexbox's structure. And adding divs in the middle makes it hard keep the desired layout.

So using `React.Fragment` fixes this issue.

Apart from the regular *React.Fragment* syntax, there's a very simple and easy JSX syntax available to use fragments, which is expressed by an emp JSX tag like below

```
<> </>
```

This is called the **Empty tag** in React. This is a shorter syntax for React Fragments.

So we can also write the previous example of fragment by using the empty tags syntax like below

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  return (
    <>
      <p>This is first JSX Element!</p>
      <p>This is another JSX Element</p>
    </>
  );
};
```

```
const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

How to Add Comments to JSX Code

If you have a line of code like this:

```
<p>This is some text</p>
```

and you want to add a comment for that code, then you have to wrap that code in JSX expresssion syntax inside the `/*` and `*/` comment symbo like this:

```
{/* <p>This is some text</p> */}
```

> *Tip:* Instead of manually typing the comment, you can use `Cmd + /` (Mac) or `Ctrl + /` shortcut keys to add or remove the comment.

How to Add JavaScript Code in JSX

Up to this point, we have only used HTML tags as a part of JSX. But JSX becomes more useful when we actually add JavaScript code inside it.

To add JavaScript code inside JSX, we need to write it in curly brackets like this:

```
const App = () => {
 const number = 10;
 return (
  <div>
   <p>Number: {number}</p>
  </div>
 );
};
```

> Inside the curly brackets we can only write an expression that evaluates to some value.

So, often this syntax of using curly brackets is known as JSX Expression Syntax.

Following are the valid things you can have in a JSX Expression:

- A string like "hello"
- A number like 10
- An array like [1, 2, 4, 5]
- An object property that will evaluate to some value
- A function call that returns some value which may be JSX
- A map method that always returns a new array
- JSX itself

Following are the invalid things and cannot be used in a JSX Expression:

- A for loop or while loop or any other loop
- A variable declaration
- A function declaration
- An if condition
- An object

We can write arrays in JSX Expressions because `<p>{[1, 2, 3, 4]}</p>` is finally converted to `<p>{1}{2}{3}{4}</p>` when rendering (whic can be rendered without any issue).

In the case of an object, it's not clear how the object should be displayed. For example, should it be comma-separated key-value pairs or should it b displayed as JSON? So you will get an error if you try to display the object in a JSX expression. But we can use object properties instead.

> Also note that undefined, null, and boolean are not displayed on the UI when used inside JSX.

So if you have a boolean value and you want to display it on the UI you need to wrap it in ES6 template literal syntax like this:

```
const App = () => {
  const isAdmin = true;
  return (
    <div>
      <p>isAdmin is {`${isAdmin}`} </p>
    </div>
  );
};
```

> isAdmin is true
>
> isAdmin is

How to Add a Class in JSX

We can add attributes to the JSX elements, for example `id` and `class`, the same as in HTML.

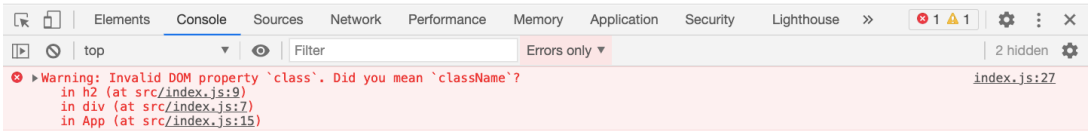
```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  const id = "some-id";
  return (
    <div>
      <h1 id={id}>This is a heading</h1>
      <h2 className="active">This is another heading</h2>
    </div>
  );
};

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

Note that in React, we need to use `className` instead of `class`.

This is because if you use `class` instead of `className`, you will get a warning in the console as shown below:



To understand why the warning is being shown, print the object representation of it and you will see the following:

As you can see, the props object has the `class` property with a value `active`. But in JavaScript, `class` is a reserved keyword s accessing `props.class` will result in an error.

This is why React decided to use `className` instead of `class`.

This use of `className` instead of `class` is a frequently asked question in React interviews.

> Note that in React, all the attribute names are written in camelCase.

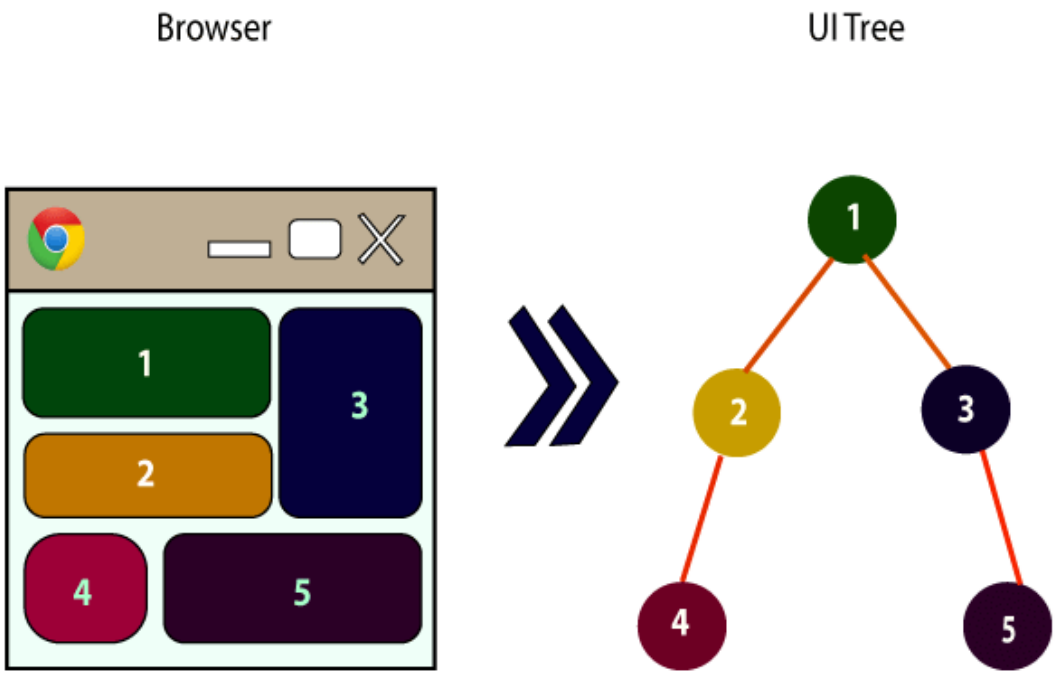
You can find all the changed and unchanged attributes list here.

What Is A Component In React?

Previously, developers had to write thousands of lines of code to create a single-page application. Making updates to these programs was a difficu undertaking because they follow the standard DOM structure. If a mistake is discovered, the entire application is manually searched and updated. Th component-based methodology was developed to address a problem. This method breaks down the entire application into a handful of manageab logical groups of code, or 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 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, consid 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 su branches.



For example, if we were building the UI of Twitter with React:

Rather than building the whole UI under one single file, we can and we should divide all the sections (marked with red) into smaller independent piece In other words, these are components

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

1. Functional Components
2. Class Components

Functional Components

A functional component is basically a JavaScript/ES6 function that returns a React element (JSX). We will learn more about JSX in the upcoming sectio 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 simp JavaScript functions that may or may not receive data as parameters. We can create a function that takes props(properties) as input and returns wh should be rendered. A valid functional component can be shown in the below example.

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

Alternatively, you can also create a functional component with the arrow function definition:

```
const WelcomeMessage = (props) => {
  return <h1>Welcome to the React</h1>;
}
```

To be able to use a component later, you need to first export it so you can import it somewhere else:

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

export default WelcomeMessage;
```

After importing it, you can call the component like in this example:

```
import WelcomeMessage from './WelcomeMessage';

function App() {
  return (
    <div className="App">
      <WelcomeMessage />
    </div>
  );
}
```

So a Functional Component in React:

- is a JavaScript/ES6 function
- must return a React element (JSX)
- always starts with a capital letter (naming convention)
- takes props as a parameter if necessary

Class Components

The second type of component is the class component. Class components are ES6 classes that return JSX. Below, you see our same WelcomeMessag function, this time as a class component:

```
class WelcomeMessage extends React.Component {
  render() {
    return <h1>Welcome to the React</h1>;
  }
}
```

Different from functional components, class components must have an additional render( ) method for returning JSX.

A Class Component:

- is an ES6 class, will be a component once it ‘extends’ a React component.
- takes Props (in the constructor) if needed
- must have a render( ) ****method for returning JSX

Apart from these **two main components** we also have **Pure Components** and **High Order Components** which we will study in future.

Conditional Rendering

In React, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state your application.

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like `if` or the conditional operator create elements representing the current state, and let React update the UI to match them.

Consider these two components:

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

We'll create a `Greeting` component that displays either of these components depending on whether a user is logged in:

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```

Conclusion

So till now we learnt basics of React components, JSX and Conditional rendering in react in react

In this module we have learned about:

- What is JSX
- What is a component in React
- What is a Functional Component
- What is a Class Component
- Conditional Rendering In React.

Thank You !