React JSX:-

What is JSX?

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX makes it easier to write and add HTML in React.

Coding JSX:-

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.

You are not required to use JSX, but JSX makes it easier to write React applications.

Here are two examples. The first uses JSX and the second does not:

Example 1 with jsx :-

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = <h1>I Love JSX!</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Example 2 **Without JSX:-**

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = React.createElement('h1', {}, 'I do not use JSX!');

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

As you can see in the first example, 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.

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

Execute the expression `5 + 5`:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

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

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

output :-

React is 10 times better with JSX

Inserting a Large Block of HTML

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

Example

Create a list with three list items:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

output :-

- Apples
- Bananas
- Cherries

One Top Level Element

The HTML code must be wrapped in *ONE* top level element.

So if you like to write *two* paragraphs, you must put them inside a parent element, like a `div` element.

Example

Wrap two paragraphs inside one DIV element:

```
import React from 'react';
import ReactDOM from 'react-do/client';

const myElement = (
  <div>
    <h1>I am a Header.</h1>
    <h1>I am a Header too.</h1>
  </div>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

output:-

I am a Header.

I am a Header too.

JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element.

Alternatively, you can use a "fragment" to wrap multiple lines. This will prevent unnecessarily adding extra nodes to the DOM.

A fragment looks like an empty HTML tag: `<></>`.

Example

Wrap two paragraphs inside a fragment:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = (
    <>
      <p>I am a paragraph.</p>
      <p>I am a paragraph too.</p>
    </>
  );

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

output :-

```
I am a paragraph.

I am a paragraph too.
```

Elements Must be Closed

JSX follows XML rules, and therefore HTML elements must be properly closed.

Example

Close empty elements with `/>`

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = <input type="text" />;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

JSX will throw an error if the HTML is not properly closed.

Attribute class = className

The `class` attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the `class` keyword is a reserved word in JavaScript, you are not allowed to use it in JSX.

Use attribute `className` instead.

JSX solved this by using `className` instead. When JSX is rendered, it translates `className` attributes into `class` attributes.

Example:-

Use attribute `className` instead of `class` in JSX:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = <h1 className="myclass">Hello World</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Conditions - if statements

React supports `if` statements, but not *inside* JSX.

To be able to use conditional statements in JSX, you should put the `if` statements outside of the JSX, or you could use a ternary expression instead:

Option 1:

Write `if` statements outside of the JSX code:

Example

Write "Hello" if `x` is less than 10, otherwise "Goodbye":

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const x = 5;
let text = "Goodbye";
if (x < 10) {
  text = "Hello";
}

const myElement = <h1>{text}</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

To run in directly in html page code examples:-

```
<!DOCTYPE html>

<html>

  <head>

    <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>

    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
crossorigin></script>

    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

  </head>

  <body>




<div id="root"></div>



          <script type="text/babel">
```

```
const x = 5;
let text = "Goodbye";
if (x < 10) {
  text = "Hello";
}

const myElement = <h1>{text}</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

```
        </script>

 </body>

</html>
```

Output:-

Hello