Introduction to JSX

Topics covered:

- What is JSX?
- What is react.createElement?
- How to Return JSX?
- How to add comments in JSX?
- How to add Javascript in JSX?
- How to add conditions in JSX?
- How to add classes in JSX?
- How to add CSS in JSX?
- Mini project

1. JSX:

- JSX is an acronym for Javascript XML (eXtensible Markup Language).
- JSX (JavaScript Extension Syntax) makes it easy to combine JavaScript and HTML in React.
- If we use html code in javascript it will look like this:

```
const JSX = <h1>Hey, Testbook Users</h1>
```

- This is basic React JSX code. This JSX, however, is not understood by the browser because the JavaScript code is invalid. This is due to the fact that we are assigning an HTML tag to a variable that is simply HTML code and not a string.
- Therefore, we need a tool like Babel, a JavaScript compiler/transpiler, to translate it into browser-friendly JavaScript code.
- You can also try create-react-app, which internally utilizes Babel to convert JSX to JavaScript.
- This is how our React code may use the JSX from above example:

```
export default class demo extends Component {
  render() {
    return <h1>Hey, TestBook UsersX</h1>;
    }
}
```

2. React Without JSX:

- Using *React.createElement()*

```
export class demo1 extends Component {
  render() {
      return React.createElement("h1", null, "Hey, Testbook Users");
  }
}
```

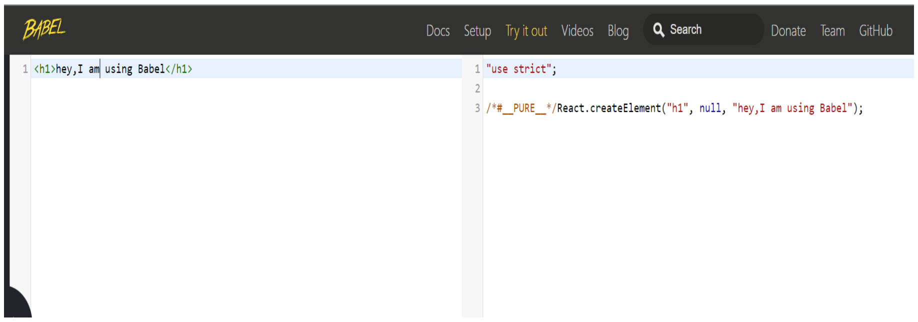
- This was the old approach of creating code in React - however writing the React is tedious. Every time, even when adding a basic div, use createElement.
- As a result, React released the JSX programming, which makes code easier to write and understand.

Let's talk about React.createElement() function:

- Each JSX file is translated to React. A browser-friendly createElement function call.
- The syntax for React.createElement is:

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

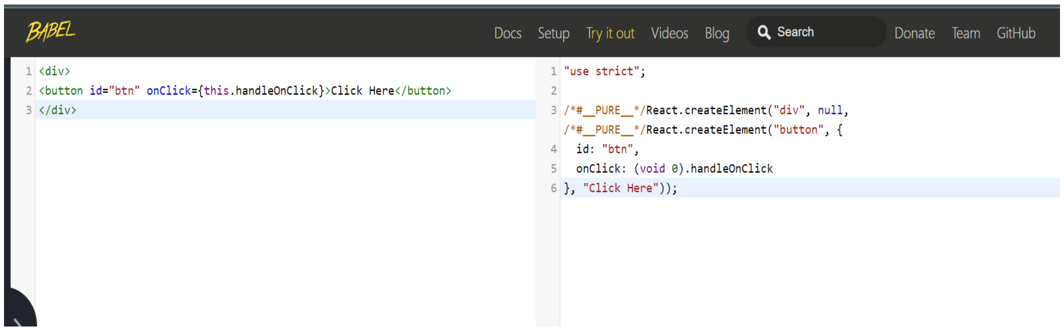
- Examine the parameters of the createElement function.
  - ➢ **Type** can be an HTML tag such as h1, div, or a React component.
  - ➢ **Props** are the characteristics that you want the element to have.
  - ➢ **Children** might be other HTML tags or a component.
- Let's see how babel convert JSX in createElement():



```
<h1>hey,I am using Babel</h1>
```

```
"use strict";

/*#__PURE__*/React.createElement("h1", null, "hey,I am using Babel");
```

- Now, let's make the JSX more complex to see how it's transformed to the React.createElement method.

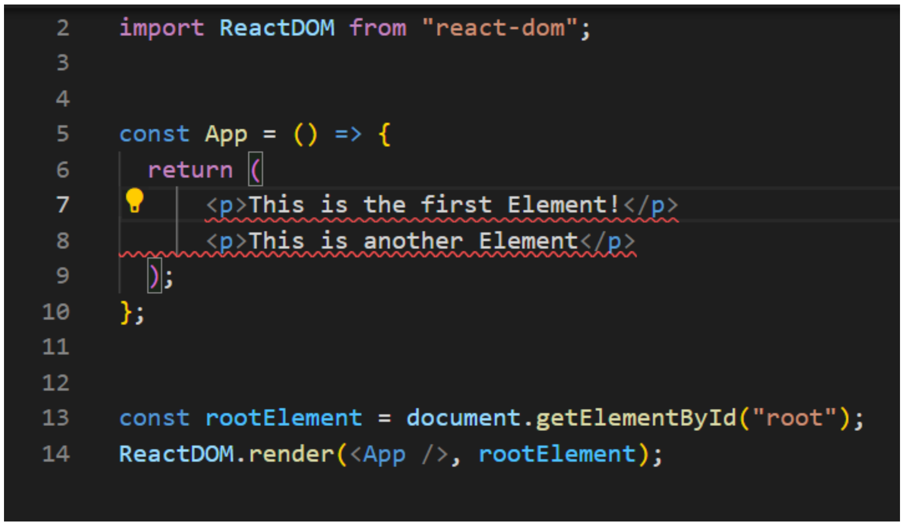


```
<div>
<button id="btn" onClick={this.handleOnClick}>Click Here</button>
</div>
```

```
"use strict";

/*#__PURE__*/React.createElement("div", null,
/*#__PURE__*/React.createElement("button", {
  id: "btn",
  onClick: (void 0).handleOnClick
}, "Click Here"));
```

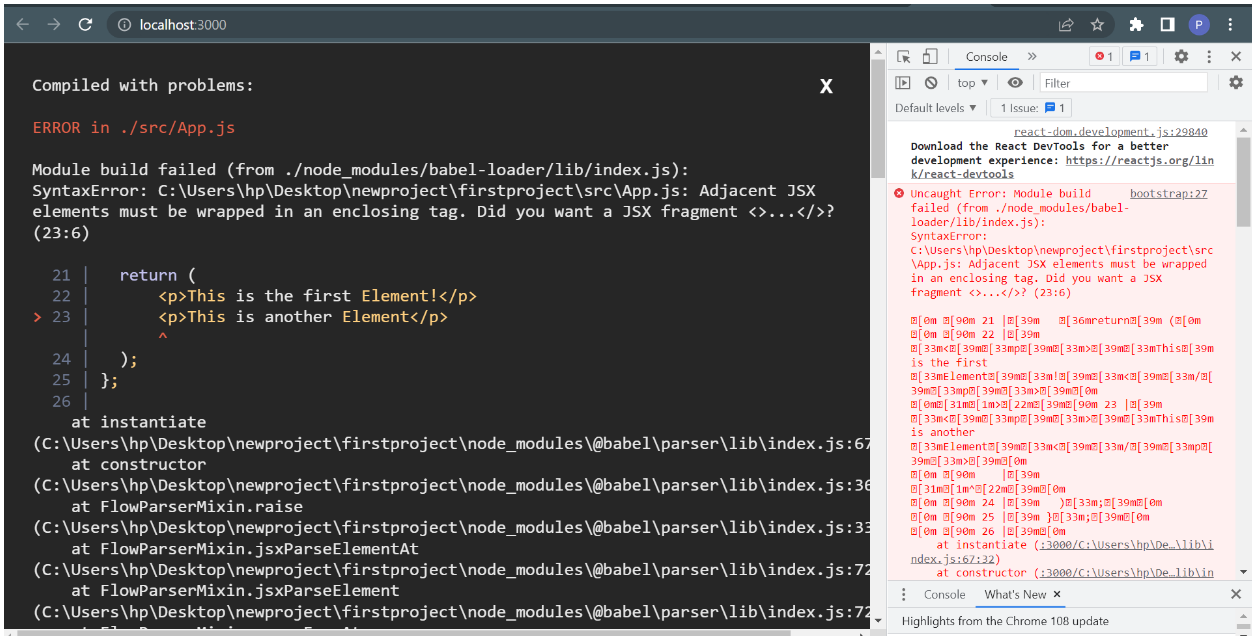
3. How to Return Complex JSX?

Examine the following code:



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


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


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

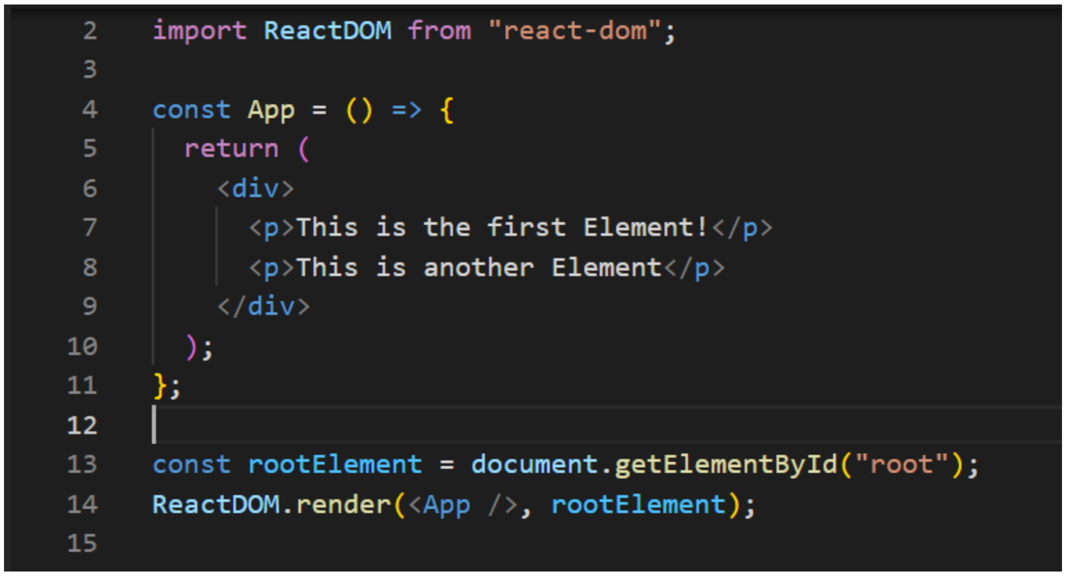
We're returning two paragraphs from the App component here. However, when you run the code, you will see the following error:



```
Compiled with problems:                                                      X

ERROR in ./src/App.js

Module build failed (from ./node_modules/babel-loader/lib/index.js):
SyntaxError: C:\Users\hp\Desktop\newproject\firstproject\src\App.js: Adjacent JSX
elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>?
(23:6)

  21 |   return (
  22 |       <p>This is the first Element!</p>
> 23 |       <p>This is another Element</p>
     |       ^
  24 |   );
  25 | };
  26 |
    at instantiate
(C:\Users\hp\Desktop\newproject\firstproject\node_modules\@babel\parser\lib\index.js:67
    at constructor
(C:\Users\hp\Desktop\newproject\firstproject\node_modules\@babel\parser\lib\index.js:36
    at FlowParserMixin.raise
(C:\Users\hp\Desktop\newproject\firstproject\node_modules\@babel\parser\lib\index.js:33
    at FlowParserMixin.jsxParseElementAt
(C:\Users\hp\Desktop\newproject\firstproject\node_modules\@babel\parser\lib\index.js:72
    at FlowParserMixin.jsxParseElement
(C:\Users\hp\Desktop\newproject\firstproject\node_modules\@babel\parser\lib\index.js:72
```

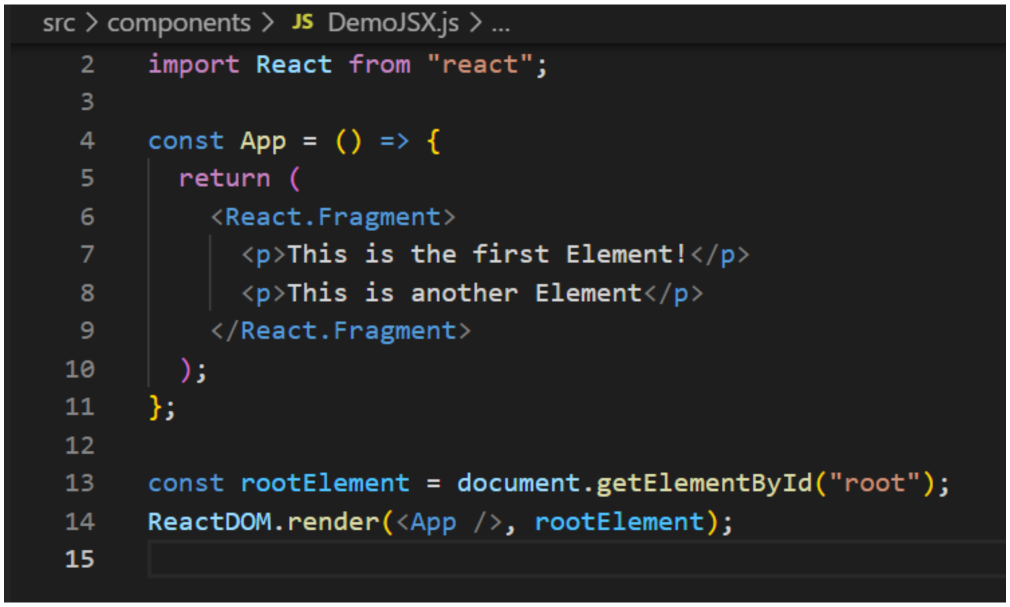
```
react-dom.development.js:29840
Download the React DevTools for a better
development experience: https://reactjs.org/lin
k/react-devtools

Uncaught Error: Module build      bootstrap:27
failed (from ./node_modules/babel-
loader/lib/index.js):
SyntaxError:
C:\Users\hp\Desktop\newproject\firstproject\src
\App.js: Adjacent JSX elements must be wrapped
in an enclosing tag. Did you want a JSX
fragment <>...</>? (23:6)

[0m [90m 21 |[39m   [36mreturn[39m ([0m
[0m [90m 22 |[39m
[33m<[39m[33mp[39m[33m>[39m[33mThis[39m
is the first
[33mElement[39m[33m![39m[33m<[39m[33m/[
39m[33mp[39m[33m>[39m[0m
[0m[31m[1m>[22m[39m[90m 23 |[39m
[33m<[39m[33mp[39m[33m>[39m[33mThis[39m
is another
[33mElement[39m[33m<[39m[33m/[39m[33mp[
39m[33m>[39m[0m
[0m [90m    |[39m
[31m[1m^[22m[39m[0m
[0m [90m 24 |[39m   )[33m;[39m[0m
[0m [90m 25 |[39m }[33m;[39m[0m
[0m [90m 26 |[39m[0m
    at instantiate (:3000/C:\Users\hp\De…\lib\i
ndex.js:67:32)
    at constructor (:3000/C:\Users\hp\De…lib\in
```

We're getting an error because React needs neighboring elements to be wrapped in a parent tag. We're getting an error since react only returns one element. As a result, a parent tag is required to enclose these nearby items.



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

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

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

Additionally, you can use the React.Fragment component to solve the problem:


src > components > JS DemoJSX.js > ...

```
import React from "react";

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

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

- ➢ In React version 16.2, React.Fragment was introduced because we always have to wrap multiple adjacent elements in some tag (like div) inside every JSX returned by a component. As a result, unnecessary div tags are added.
- ➢ Most of the time, this is fine, but there are some exceptions.
- ➢ Flexbox, for example, has a special parent-child relationship in its structure. And adding divs in the middle makes it difficult to achieve the desired layout.
- ➢ Using ReactFragment solves this problem:
    - ❖ Fragments allow you to group a list of children without introducing new nodes to the DOM.

4. Comments in JSX:

If you have a line of code that looks like this:

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

If you want to add a comment to that code, you must wrap it in JSX expression syntax inside the /* and */ comment symbols, as seen below:

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

5. Use JavaScript in JSX:
We have just used HTML tags as part of JSX up to this point. However, JSX becomes more useful when we include JavaScript code within it.

To include JavaScript code within JSX, use curly brackets like this:

```
const App = () => {
  const name = "Rohit";
  return (
    <div>
      <p>Name: {name}</p>
    </div>
  );
};
```

We can only write an expression inside curly brackets that evaluates to some value**.** Therefore, JSX Expression Syntax is a common name for this use of curly brackets in syntax.

The following elements are valid in a JSX expression:

- A string similar to "hello"
- A number such as ten
- An array of the form [1, 2, 4, 5]
- An object property that gives a value.
- A function call that returns a value that could be anything. JSX
- A map method that returns a distinct array every time.
- Also JSX

  Arrays can be written in JSX Expressions because <p>[{1, 2, 3, 4}]</p> is finally converted to <p>{1}{2}{3}{4}</p> when rendered (which can be rendered without any issue).

The following are invalid and cannot be used in a JSX Expression:

- The following are invalid and cannot be used in a JSX Expression:
- A for loop, a while loop, or any other type of loop
- A declaration of variables
- A declaration of a function
- An if condition
- An object

Also, when used within JSX, undefined, null, and boolean are not displayed on the UI.

So, if you have a boolean value that you wish to display on the UI, you must wrap it in ES6 template literal syntax, as shown below:

```
const App = () => {

  const isAdmin = true;

  return (

    <div>

      <p>isAdmin is {`${isAdmin}`} </p>

    </div>

  );

};
```

6. Conditional Operators in JSX Expressions:

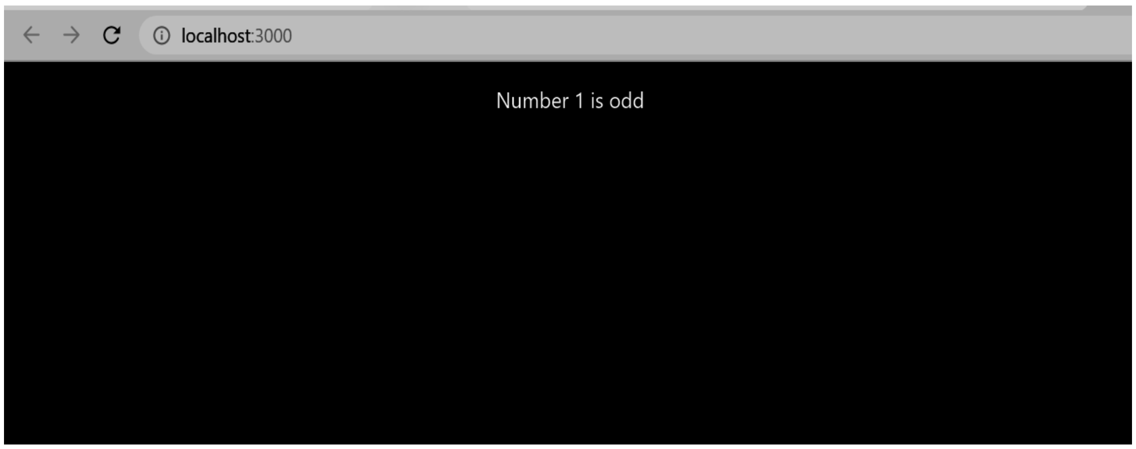
We can't write if conditions in JSX expressions, which may seem like a limitation. However, React allows us to build conditional operators, such as ternary operators, as well as the logical short circuit && operator, as shown below:

```
<p>{a > b ? "Greater" : "Smaller"}</p>
<p>{shouldShow && "Shown"}</p>
```

Example of conditional rendering:

```
const EvenOdd = () => {
  const number = 1;
  return (
    <div>
      {number % 2 == 0 ? (
        <p>Number {number} is even</p>
      ) : (
        <p>Number {number} is odd</p>
      )}
    </div>
  );
};
export default EvenOdd;
```

Output:

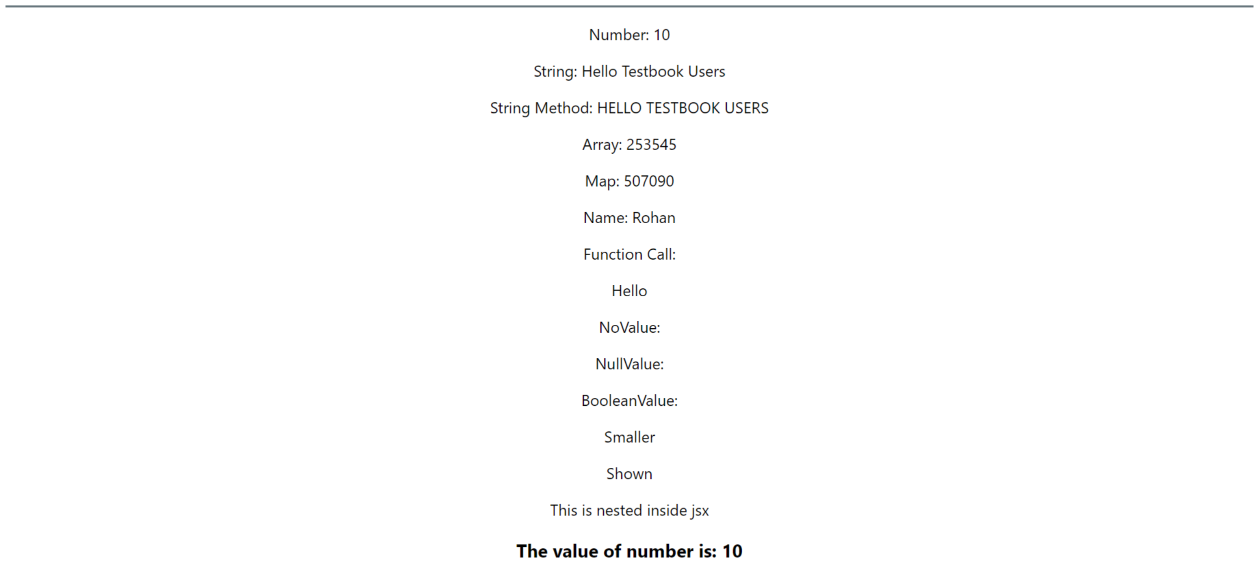


```
Number 1 is odd
```

Example of trying every expression:

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

const greet = () => {
  return <p>Hello</p>;
};

const Expression = () => {
  const number = 10;
  const string = "Hello Testbook Users";
  const array = [25, 35, 45];
  const object = { name: "Rohan" };
  const noValue = undefined;
  const nullValue = null;
  const booleanValue = false;
  const a = 20;
  const b = 30;
  const shouldShow = true;
  const isFalse = false;
  return (
    <div>
      <p>Number: {number}</p>
      <p>String: {string} </p>
      <p>String Method: {string.toUpperCase()} </p>
      <p>Array: {array}</p>
      <p>Map: {array.map((value) => value * 2)}</p>
      <p>Name: {object.name} </p>
      <p>Function Call: {greet()}</p>
      <p>NoValue: {noValue} </p> {/* This will not be displayed */}
      <p>NullValue: {nullValue} </p>
      <p>BooleanValue: {booleanValue}</p>
      <p>{a > b ? "Greater" : "Smaller"}</p>
      <p>{shouldShow && "Shown"}</p>
      <p>{isFalse && "Won't Displayed"}</p>
      <p>{true && <p>This is nested inside JSX</p>}</p>
      <p>{<h3>The value of number is: {number}</h3>}</p>
    </div>
  );
};

export default Expression;
```

Output:



```
Number: 10
String: Hello Testbook Users
String Method: HELLO TESTBOOK USERS
Array: 253545
Map: 507090
Name: Rohan
Function Call:
Hello
NoValue:
NullValue:
BooleanValue:
Smaller
Shown
This is nested inside jsx
The value of number is: 10
```

7. How to add class in JSX:

As in HTML, we can add properties to JSX elements, such as id and class.
It is important to note that with React, we must use className instead of class.
Example:

```
import React from "react";

export const Classes = () => {
  const id = "id-1";
  return (
    <div>
      <h1 id={id}>This is heading 1</h1>
      <h2 className="active">This is heading 2</h2>
    </div>
  );
};
```

8. Styling in React with CSS:
There are various ways to style React using CSS, mainly 3 ways with CSS:

- Inline styling
- CSS stylesheets
- CSS Modules

1. **Inline Styling:**

   The value of the inline style attribute must be a JavaScript object in order to style an element:

Example:

```
const Header = () => {
  return (
    <>
      <h1 style={{ color: "red" }}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
};
```

In JSX, JavaScript expressions are enclosed by curly braces, and because JavaScript objects are also enclosed by curly braces, the style in the above example is enclosed by two sets of curly braces.

Due to the fact that the inline CSS is written in a JavaScript object, values with hyphen separators, such as background-color, must be written in camel case:

```
const Header = () => {
  return (
    <>
```

```
      <h1 style={{ backgroundColor: "lightblue" }}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
};
```

JavaScript Object (Internal Css):

You can also build a styled object and refer to it in the style attribute:

Example:

```
const Header = () => {
  const myStyle = {
    color: "white",
    backgroundColor: "DodgerBlue",
    padding: "10px",
    fontFamily: "Sans-Serif",
  };
  return (
    <>
      <h1 style={myStyle}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
};
```

2. **Css Stylesheet:**

You can write your CSS styling in a separate file and import it into your application by saving it with the .css file extension.

Example: App.css
```css
body {
  background-color: #282c34;
  color: white;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}
```

Import the stylesheet in App.js

```jsx
import React from "react";
import ReactDOM from "react-dom/client";
import "./App.css";

const App = () => {
  return (
    <>
      <h1>Hello Style!</h1>
      <p>Add a little style!.</p>
    </>
  );
};
```

3. **CSS Modules:**
    - CSS Modules are another option to add styles to your application.
    - CSS Modules are useful for components that are placed in different files.
    - The CSS inside a module is only available to the component that imported it, and there are no name conflicts.
    - Make a CSS module using the.module.css extension, such as my-style.module.css.

Create a CSS module with the .module.css extension, such as my-style.module.css.

Example: my-style.module.css
```css
.bigblue {
  color: DodgerBlue;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}
```

Import the stylesheet in your component:

```
import styles from './my-style.module.css';

const Desktop = () => {
  return <h1 className={styles.bigblue}>Hello Car!</h1>;
}
export default Desktop;
```

Import the component in index.js:

```
import ReactDOM from "react-dom/client";
import Desktop from "./Desktop.js";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<Desktop />);
```