

1. JSX, or JavaScript XML, is a syntax extension for JavaScript commonly used with the React library. It allows us to write HTML-like structures within JavaScript code. This approach enhances code readability and enables a declarative way to define user interfaces. JSX is not a requirement for React, but it is widely adopted because of its simplicity and expressiveness. The JSX code is transpiled to JavaScript, specifically to React.createElement() calls, during the build process.
2. ECMAScript, often abbreviated as ES, is the standardized specification that governs the core functionalities of JavaScript. Developed and maintained by ECMA International, this specification defines the syntax, types, statements, keywords, reserved words, and more. The most widely used versions include ES5 and ES6 , with subsequent versions introducing modern programming features such as arrow functions, classes, destructuring, modules, let and const, promises, and template literals. In React development, we extensively utilize these modern ECMAScript features to write efficient and maintainable code.
3. The React.createElement() function is a fundamental method provided by the React library. It is used to create React elements, which are the building blocks of React’s virtual DOM. The function takes three parameters: the element type, a properties object (commonly referred to as props), and any number of child elements or text nodes. This method enables the creation of complex user interface hierarchies without relying on JSX.

Syntax:

React.createElement(type, props, ...children)

The output of this function is a plain JavaScript object representing a DOM element in the virtual DOM.

1. In React, we can create React nodes using JSX by writing HTML-like syntax directly in our JavaScript code. These JSX expressions are compiled to React.createElement() calls. A React node, in this context, refers to any element or component that can be rendered, such as <div>, <h1>, or custom components. JSX supports nesting, allowing us to define complex layouts with clarity and structure.

Example:

const element = <div><h1>Welcome</h1><p>This is a paragraph.</p></div>;

1. import { createRoot } from 'react-dom/client';

const root = createRoot(document.getElementById('root'));

root.render(<App />);

This process mounts the specified JSX element or component into a DOM node.

1. JSX permits the embedding of JavaScript expressions within the markup using curly braces {}. Expressions such as variables, arithmetic operations, function calls, and ternary operators can be used to dynamically generate content within JSX.

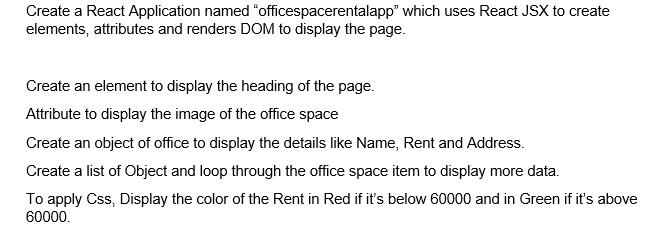
Example:

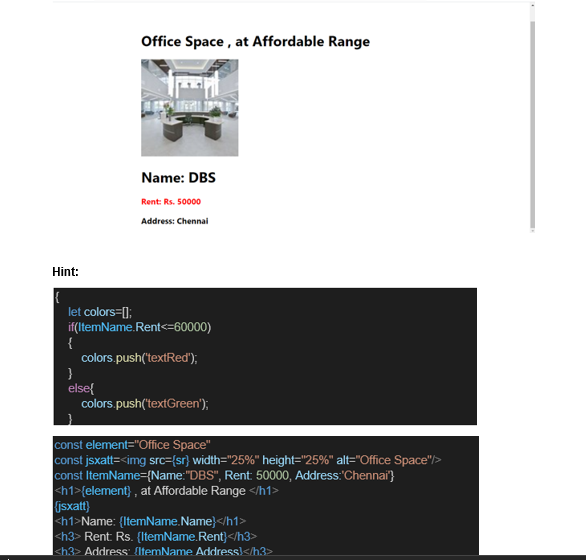
const user = "Arnav";  
const element = <h1>Welcome, {user}</h1>;

1. To apply inline CSS in JSX, we assign a JavaScript object to the style attribute of an element. The object must use camelCase property names instead of traditional CSS syntax. Values should be specified as strings or variables where appropriate.

Example:

const styleObject = {  
 backgroundColor: 'lightblue',  
 color: 'navy',  
 fontSize: '18px'  
};  
  
const element = <h2 style={styleObject}>Styled Component</h2>;





**Solution**

**App.js**

import logo from './logo.svg';

import React from 'react';

import './App.css';

function App() {

const element = "Office Space";

const sr = process.env.PUBLIC\_URL + '/office.jpg';

const jsxatt = <img src={sr} width="25%" height="25%" alt="Office Space" />;

const officeList = [

{ Name: "DBS", Rent: 50000, Address: "Chennai" },

{ Name: "WeWork", Rent: 70000, Address: "Bangalore" },

{ Name: "SmartDesk", Rent: 55000, Address: "Hyderabad" },

];

return (

<div className="App">

<h1>{element} , at Affordable Range</h1>

{jsxatt}

{officeList.map((office, index) => {

let colors = [];

if (office.Rent <= 60000) {

colors.push('textRed');

} else {

colors.push('textGreen');

}

return (

<div key={index}>

<h1>Name: {office.Name}</h1>

<h3 className={colors.join(' ')}>Rent: Rs. {office.Rent}</h3>

<h3>Address: {office.Address}</h3>

</div>

);

})}

</div>

);

}

export default App;

**App.css**

.App {

text-align: center;

font-family: Arial, sans-serif;

margin: 20px;

}

.textRed {

color: red;

font-weight: bold;

}

.textGreen {

color: green;

font-weight: bold;

}

**Output**

