**REACT ASSIGNMENT**

**Define JSX**

JSX (JavaScript XML) is a syntax extension for JavaScript used with React that allows developers to write HTML-like code within JavaScript. It makes code more readable and expressive by enabling you to describe the UI structure in a way that resembles HTML. Behind the scenes, JSX is transpiled into React.createElement() calls using tools like Babel, and helps in building components more intuitively with familiar markup syntax embedded in JavaScript logic.

**Explain About ECMA Script**

ECMAScript (or ES) is the standardized scripting language specification that JavaScript is based on. Maintained by ECMA International, the ECMAScript standards (like ES5, ES6, etc.) define the core features and functionality of the language. ES6 (ECMAScript 2015) was a major release introducing modern features such as let, const, arrow functions, classes, promises, and modules, significantly improving JavaScript's power and readability.

**Explain React.createElement()**

React.createElement() is a core method provided by React to create virtual DOM elements. It takes three arguments: the type of element (e.g., 'div'), a configuration object (props), and children (which can be strings, numbers, or other React elements). Although developers often use JSX, it’s ultimately transformed into React.createElement() calls during compilation, enabling React to efficiently construct and update the virtual DOM.

**Explain How to Create React Nodes with JSX**

In React, nodes (elements) can be created using JSX by writing HTML-like syntax directly inside JavaScript. For example, you can write <h1>Hello, World!</h1> inside a component's return statement to create an h1 node. These JSX elements are converted into virtual DOM nodes behind the scenes, enabling React to render and update them efficiently based on application state or props.

**Define How to Render JSX to DOM**

To render JSX to the DOM, React uses the ReactDOM.render() method. This method takes a JSX element and a DOM container as arguments and mounts the component into the specified HTML element. For example, ReactDOM.render(<App />, document.getElementById('root')) renders the App component into the <div id="root"> in your HTML file, effectively connecting React’s virtual DOM to the real browser DOM.

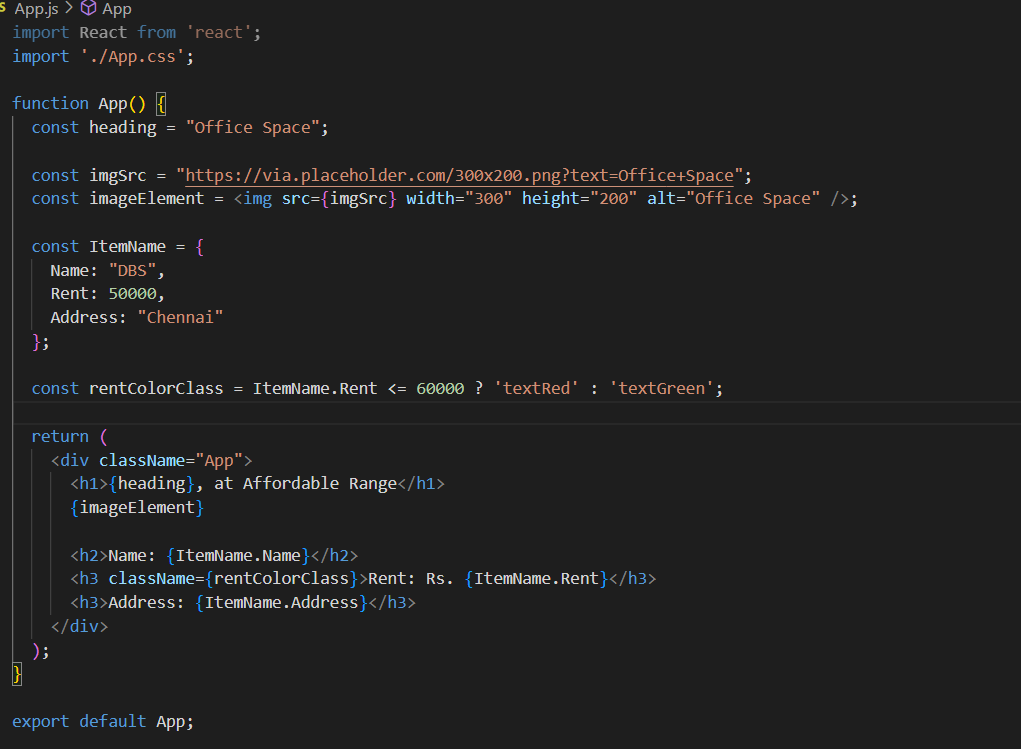
**Explain How to Use JavaScript Expressions in JSX**

JavaScript expressions can be embedded in JSX using curly braces {}. This allows you to dynamically render variables, function calls, or calculations within JSX. For example, you can write <h1>{user.name}</h1> or <p>{2 + 2}</p> to inject values into the component output. However, only expressions—not statements like if or for—can be used directly inside JSX.

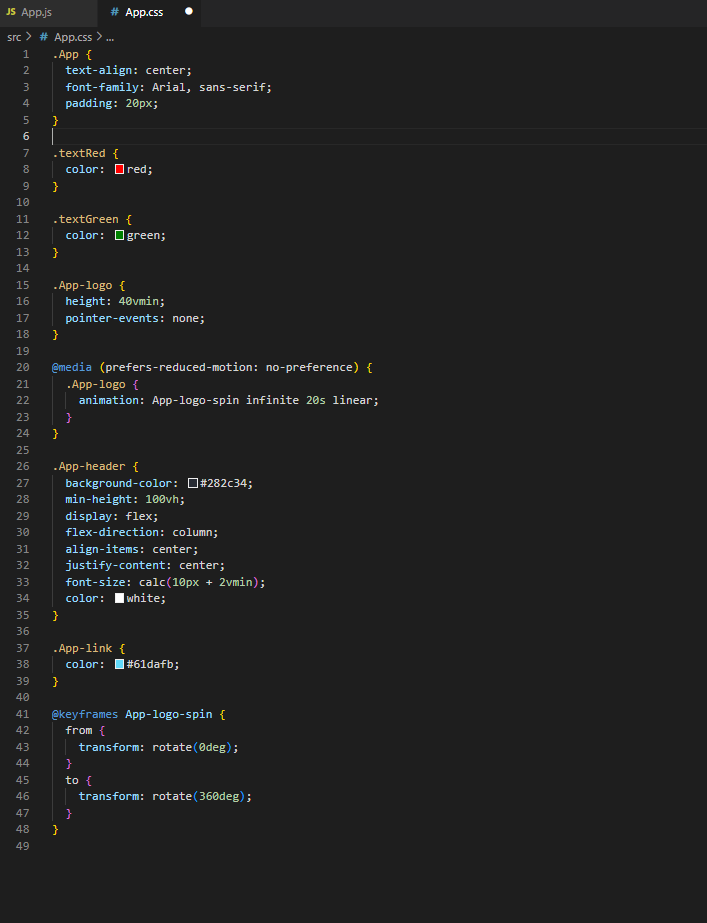
**Explain How to Use Inline CSS in JSX**

In JSX, inline styles are applied using a style attribute that takes a JavaScript object with camelCase property names instead of traditional CSS syntax. For instance, instead of font-size, you’d use fontSize, and string values are provided for most styles. Example: <div style={{ color: 'blue', fontSize: '20px' }}>Hello</div>. This approach allows dynamic styling using variables and logic within the component.

**App.js:**

****

**App.css:**

****

**OUTPUT:**

****