**Define JSX**  
JSX (JavaScript XML) is a syntax extension for JavaScript used in React to describe what the UI should look like. It allows developers to write HTML-like code within JavaScript, making it easier to create and visualize component structures. JSX is not valid JavaScript by itself; it is transpiled by tools like Babel into standard JavaScript calls, such as React.createElement(), before being executed by the browser.

**Explain about ECMA Script**  
ECMA Script (often abbreviated as ES) is the standardized scripting language specification upon which JavaScript is based. It is governed by ECMA International and updated through a series of editions. Each edition introduces new features to the JavaScript language. For example, ES5 introduced stricter syntax and JSON support, while ES6 (ECMAScript 2015) introduced major enhancements such as let, const, arrow functions, classes, modules, promises, and new data structures.

**Explain React.createElement()**  
React.createElement() is a core function in React used to create a React element (or node). It takes three arguments: the type of the element (e.g., 'div' or a React component), optional props, and the element’s children. JSX is syntactic sugar for this function, so when JSX is used, it gets converted into React.createElement() calls during compilation. This function is responsible for creating the virtual DOM representation of the UI.

**Explain how to create React nodes with JSX**  
React nodes can be created using JSX by writing HTML-like tags directly in the component’s return statement. For example, writing <h1>Hello World</h1> inside a return block of a function or class component creates a React node. These JSX elements are transformed into JavaScript objects using React.createElement() behind the scenes and then rendered as UI components.

**Define how to render JSX to DOM**  
To render JSX to the DOM, the ReactDOM.render() method is used. This method takes a JSX element and a DOM container as arguments. For example, ReactDOM.render(<App />, document.getElementById('root')) renders the App component (written in JSX) into the HTML element with the id root. The JSX is first compiled to JavaScript and then translated into DOM elements that the browser can display.

**Explain how to use JavaScript expressions in JSX**  
JavaScript expressions can be used inside JSX by enclosing them within curly braces {}. This allows dynamic content, such as variables, function calls, or arithmetic expressions, to be embedded within the JSX. For example, <h1>{user.name}</h1> will display the value of user.name in the rendered output. Only expressions (not statements like if or for) can be used inside the braces.

**Explain how to use inline CSS in JSX**  
Inline CSS in JSX is applied using the style attribute, which accepts a JavaScript object instead of a string. The keys in the object use camelCase notation instead of kebab-case. For example, <div style={{ backgroundColor: 'blue', fontSize: '14px' }}>Text</div> applies background color and font size using inline styles. This approach allows dynamic styling by passing variables and expressions as style values.