**Define JSX**

**JSX (JavaScript XML)** is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. It was developed by Facebook for use with React.

**Key Characteristics:**

* **HTML-like syntax**: Write markup that looks like HTML but is actually JavaScript
* **JavaScript integration**: Embed JavaScript expressions directly in markup
* **Component-based**: Create reusable UI components
* **Transpilation required**: JSX must be converted to regular JavaScript to run in browsers

**Basic JSX Example:**

const element = <h1>Hello, World!</h1>;  
const greeting = <div>Welcome to React!</div>;

**JSX vs HTML differences:**

* Use className instead of class
* Use htmlFor instead of for
* All tags must be closed (self-closing tags need /)
* Camel case for attributes (e.g., onClick instead of onclick)

**Explain About ECMA Script**

**ECMAScript (ES)** is the standardized scripting language specification that JavaScript is based on. It defines the syntax, types, statements, keywords, and objects that JavaScript should support.

**Key Versions:**

* **ES5 (2009)**: Added strict mode, JSON support, array methods
* **ES6/ES2015 (2015)**: Major update with classes, modules, arrow functions, let/const
* **ES2016-ES2023**: Annual releases with incremental improvements

**ES6 Features Used in React:**

* **Arrow functions**: const func = () => {}
* **Classes**: class Component extends React.Component
* **Modules**: import React from 'react'
* **Template literals**: Hello ${name}
* **Destructuring**: const {name, age} = props
* **Spread operator**: {...props}

**Example in React:**

// ES6 features in React component  
import React from 'react'; // Modules  
  
const MyComponent = ({name, age}) => { // Arrow function, destructuring  
 return (  
 <div>  
 <h1>{`Hello ${name}`}</h1> {/\* Template literals \*/}  
 <p>Age: {age}</p>  
 </div>  
 );  
};

**Explain React.createElement()**

**React.createElement()** is the underlying JavaScript function that JSX gets transpiled into. It creates React elements (virtual DOM nodes).

**Syntax:**

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

**Parameters:**

* **type**: HTML tag name (string) or React component
* **props**: Object containing element properties/attributes
* **children**: Child elements or text content

**JSX to createElement Transformation:**

// JSX version  
const element = <h1 className="greeting">Hello, World!</h1>;  
  
// Gets transpiled to:  
const element = React.createElement(  
 'h1',  
 { className: 'greeting' },  
 'Hello, World!'  
);

**Complex Example:**

// JSX  
const element = (  
 <div className="container">  
 <h1>Welcome</h1>  
 <p>This is a paragraph</p>  
 </div>  
);  
  
// createElement equivalent  
const element = React.createElement(  
 'div',  
 { className: 'container' },  
 React.createElement('h1', null, 'Welcome'),  
 React.createElement('p', null, 'This is a paragraph')  
);

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

React nodes are the building blocks of React applications created using JSX syntax.

**Types of React Nodes:**

**1. Element Nodes:**

// HTML elements  
const heading = <h1>Main Title</h1>;  
const paragraph = <p>This is content</p>;  
const image = <img src="image.jpg" alt="Description" />;

**2. Component Nodes:**

// Custom components  
const Welcome = () => <h1>Welcome!</h1>;  
const App = () => <Welcome />;

**3. Fragment Nodes:**

// React Fragments to group elements without extra DOM nodes  
const element = (  
 <React.Fragment>  
 <h1>Title</h1>  
 <p>Content</p>  
 </React.Fragment>  
);  
  
// Short syntax  
const element = (  
 <>  
 <h1>Title</h1>  
 <p>Content</p>  
 </>  
);

**4. Nodes with Props:**

const Button = (props) => (  
 <button   
 className={props.className}   
 onClick={props.onClick}  
 >  
 {props.children}  
 </button>  
);  
  
// Usage  
const app = <Button className="primary" onClick={handleClick}>Click Me</Button>;

**Define How to Render JSX to DOM**

**ReactDOM.render()** (React 17 and below) or **createRoot()** (React 18+) is used to render JSX elements to the actual DOM.

**React 18+ Method (Current):**

import React from 'react';  
import { createRoot } from 'react-dom/client';  
  
const element = <h1>Hello, World!</h1>;  
  
// Get DOM container  
const container = document.getElementById('root');  
  
// Create root and render  
const root = createRoot(container);  
root.render(element);

**React 17 and Below Method:**

import React from 'react';  
import ReactDOM from 'react-dom';  
  
const element = <h1>Hello, World!</h1>;  
  
ReactDOM.render(  
 element,  
 document.getElementById('root')  
);

**Complete Example:**

// App.js  
import React from 'react';  
import { createRoot } from 'react-dom/client';  
  
const App = () => {  
 return (  
 <div>  
 <h1>My React App</h1>  
 <p>This is rendered to the DOM</p>  
 </div>  
 );  
};  
  
const container = document.getElementById('root');  
const root = createRoot(container);  
root.render(<App />);

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

JavaScript expressions can be embedded in JSX using **curly braces {}**.

**Types of Expressions:**

**1. Variables:**

const name = "John";  
const age = 25;  
const element = <h1>Hello, {name}! You are {age} years old.</h1>;

**2. Functions:**

const formatName = (firstName, lastName) => `${firstName} ${lastName}`;  
const element = <h1>Welcome, {formatName('John', 'Doe')}</h1>;

**3. Mathematical Operations:**

const a = 5;  
const b = 10;  
const element = <p>Sum: {a + b}, Product: {a \* b}</p>;

**4. Conditional Expressions:**

const isLoggedIn = true;  
const element = (  
 <div>  
 {isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign in</h1>}  
 </div>  
);

**5. Array Methods:**

const items = ['apple', 'banana', 'orange'];  
const element = (  
 <ul>  
 {items.map((item, index) => <li key={index}>{item}</li>)}  
 </ul>  
);

**6. Object Properties:**

const user = { name: 'Alice', email: 'alice@example.com' };  
const element = (  
 <div>  
 <h2>{user.name}</h2>  
 <p>{user.email}</p>  
 </div>  
);

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

Inline CSS in JSX is applied using the style attribute with a JavaScript object, not a string like in HTML.

**Basic Syntax:**

const element = <div style={{property: 'value'}}>Content</div>;

**Key Rules:**

* **Double curly braces**: Outer braces for JavaScript, inner braces for object
* **Camel case**: CSS properties must be in camelCase (e.g., backgroundColor not background-color)
* **String values**: Most values should be strings
* **Number values**: For pixel values, numbers can be used (automatically adds 'px')

**Examples:**

**1. Basic Styling:**

const element = (  
 <div style={{  
 color: 'blue',  
 fontSize: '20px',  
 backgroundColor: 'lightgray',  
 padding: '10px',  
 margin: '5px'  
 }}>  
 Styled content  
 </div>  
);

**2. Dynamic Styling:**

const isActive = true;  
const buttonStyle = {  
 backgroundColor: isActive ? 'green' : 'red',  
 color: 'white',  
 padding: '10px 20px',  
 border: 'none',  
 borderRadius: '5px'  
};  
  
const element = <button style={buttonStyle}>Click Me</button>;

**3. Conditional Styling:**

const Component = ({ isError }) => {  
 return (  
 <p style={{  
 color: isError ? 'red' : 'black',  
 fontWeight: isError ? 'bold' : 'normal'  
 }}>  
 {isError ? 'Error message' : 'Normal text'}  
 </p>  
 );  
};

**4. Using Variables:**

const primaryColor = '#007bff';  
const fontSize = 16;  
  
const element = (  
 <h1 style={{  
 color: primaryColor,  
 fontSize: fontSize + 4, // Will be 20px  
 textAlign: 'center'  
 }}>  
 Title  
 </h1>  
);

**5. Complete Example:**

const Card = ({ title, content, isHighlighted }) => {  
 const cardStyle = {  
 border: '1px solid #ccc',  
 borderRadius: '8px',  
 padding: '16px',  
 margin: '8px',  
 backgroundColor: isHighlighted ? '#fff3cd' : 'white',  
 boxShadow: '0 2px 4px rgba(0,0,0,0.1)'  
 };  
  
 const titleStyle = {  
 color: '#333',  
 fontSize: '1.5em',  
 marginBottom: '8px'  
 };  
  
 return (  
 <div style={cardStyle}>  
 <h2 style={titleStyle}>{title}</h2>  
 <p style={{ color: '#666', lineHeight: '1.4' }}>{content}</p>  
 </div>  
 );  
};