1)Define JSX?

**JSX (JavaScript XML)** is a syntax extension for JavaScript that allows you to write HTML-like code directly within your JavaScript files. It's primarily used with React to describe what the UI should look like.

* **Syntax Extension:** It's not plain JavaScript, nor is it plain HTML. It's a syntactic sugar that gets compiled down to regular JavaScript.
* **HTML-like Structure:** It looks very similar to HTML, using tags like <div>, <h1>, <p>, etc., with attributes.
* **JavaScript Power:** Inside JSX, you can embed JavaScript expressions by enclosing them in curly braces {}. This is how you display dynamic data, call functions, or apply conditional logic directly within your UI structure.
* **Declarative:** JSX allows you to declare what your UI *should* look like, rather than imperatively describing *how* to build it step-by-step. React then takes care of efficiently updating the actual browser's DOM to match your declared UI.
* **Compilation:** Browsers don't understand JSX directly. Tools like Babel (which is included in React's build setup) transpile (convert) JSX code into standard JavaScript calls to React.createElement().

2) Explain about ECMA Script?

**ECMAScript (ES)** is a **standardized scripting language specification** that forms the **foundation of JavaScript**.

Think of **ECMAScript** as the **rules and features** behind JavaScript.  
JavaScript is the language we use — ECMAScript defines how it behaves.

**Key Points :**

1. 🔹 **Developed by**: ECMA International (specifically, the TC39 committee).
2. 🔹 **JavaScript follows** ECMAScript rules (JS is the implementation).
3. 🔹 First version released in **1997**.
4. 🔹 **Modern JavaScript (ES6 and above)** is more powerful and readable.

**Major ECMAScript Versions**

| **Version** | **Year** | **Features Introduced (Examples)** |
| --- | --- | --- |
| ES5 | 2009 | strict mode, JSON, Array.forEach, Object.create |
| **ES6 (ES2015)** | **2015** | let, const, arrow functions, classes, promises, template literals, destructuring |
| ES7 (2016) | 2016 | Array.prototype.includes(), exponentiation operator (\*\*) |
| ES8 (2017) | 2017 | async/await, Object.entries(), Object.values() |
| ES9–ES13 | 2018–2022 | Optional chaining ?., nullish coalescing ??, top-level await, etc. |

3) Explain React.createElement() ?

React.createElement() is a **core React API** used to create React elements (virtual DOM nodes).

It is the function that runs **behind the scenes** when you use JSX like <div>Hello</div>.

**Syntax**

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

**Parameters:**

1. **type:** The type of element (e.g., 'div', 'h1', or a React component).
2. **props:** An object with attributes/properties (e.g., { className: "title" }).
3. **children:** The content or nested elements (can be multiple).

**Example :**

import React from "react";

import ReactDOM from "react-dom/client";

const element = React.createElement(

"h1", // type

{ className: "title" }, // props

"Hello World!" // child (text)

);

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(element);

**Output on UI:**

<h1 class="title">Hello World!</h1>

4)Explain how to create React nodes with JSX?

In React, a **node** is anything React can render:

* HTML elements (<div>)
* Components (<MyComponent />)
* Strings or numbers ("Hello", 123)
* Fragments
* Arrays of elements

**JSX** is the most common way to create these nodes because it is **simple and readable**.

**🔹 Creating React Nodes with JSX**

JSX (JavaScript XML) lets you write HTML-like syntax in JavaScript, which React converts into nodes internally using React.createElement().

**Example: Simple Element**

import React from "react";

import ReactDOM from "react-dom/client";

const element = <h1>Hello React</h1>; // JSX node

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(element);

**Output:**

<h1>Hello React</h1>

**How JSX Works Internally**

const element = <h1 className="title">Hello</h1>;

// React converts it internally to:

const element = React.createElement("h1", { className: "title" }, "Hello");

JSX is **just syntax sugar** for React.createElement().

5) Define how to render JSX to DOM?

endering JSX to the DOM means **displaying React elements or components in the browser’s actual DOM**.

**Steps to Render JSX to the DOM:**

1. **Create a root container in index.html:**

<div id="root"></div>

This is where React will insert the elements.

1. **Write JSX in JavaScript:**

const element = <h1>Hello React</h1>;

1. **Use ReactDOM.createRoot() and .render():**

import React from "react";

import ReactDOM from "react-dom/client";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(element);

**What Happens Internally?**

* **JSX** is first compiled to React.createElement() calls by Babel.
* React builds a **Virtual DOM tree** from JSX.
* React updates the **real DOM** inside <div id="root"> with the Virtual DOM changes.

6)Explain how to use JavaScript expressions in JSX

In React, **JavaScript expressions** can be embedded directly inside JSX using **curly braces {}**.  
This allows you to dynamically display values, run functions, or perform calculations inside your JSX.

**Syntax**

{ JavaScriptExpression }

**Key points:**

* You can use variables, functions, mathematical expressions, array methods, etc.
* You **cannot** use statements like if, for directly inside {}.

**Examples**

**Using Variables**

const name = "Alice";

const element = <h1>Hello, {name}!</h1>;

**Output:**

Hello, Alice!

7)Explain how to use inline CSS in JSX?

In React (JSX), **inline CSS** is applied differently than in plain HTML because JSX is JavaScript.

**Key Rules for Inline CSS in JSX**

1. Use the style attribute, but pass it a **JavaScript object** (not a string).
2. CSS property names must be written in **camelCase** instead of kebab-case:
   * HTML: background-color → JSX: backgroundColor
3. Property values (strings) must be inside quotes.

**Example 1: Simple Inline CSS**

function App() {

return (

<h1 style={{ color: "blue", backgroundColor: "yellow" }}>

Hello React!

</h1>

);

}

**Explanation:**

* style={{ ... }} uses **double curly braces**:
  1. Outer {} → tells JSX to evaluate JavaScript.
  2. Inner {} → the actual JavaScript object containing CSS.

**Example 2: Using a CSS Object Variable**

const headingStyle = {

color: "white",

backgroundColor: "green",

padding: "10px",

textAlign: "center"

};

function App() {

return <h2 style={headingStyle}>Welcome to React</h2>;

}