What is JSX?

What is JSX?

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

Think of it as a convenient and intuitive way to write user interface structures (like buttons, paragraphs, images) using a syntax that's very familiar to anyone who's worked with HTML, but with the full power of JavaScript embedded within it.

Key points about JSX:

1. **Not HTML, Not a String:** Despite looking like HTML, JSX is **not** raw HTML and it's not a string literal that React parses. It's a syntactic sugar over `React.createElement()` calls. When you write JSX, it gets **transpiled** (converted) by tools like Babel into regular JavaScript function calls that React can understand.

JSX:
JavaScript
const element = <h1>Hello, world!</h1>;

-

What it transpiles to (roughly):
JavaScript
const element = React.createElement('h1', null, 'Hello, world!');

-

2. This transformation happens before your code runs in the browser.
3. **Declarative UI:** JSX promotes a **declarative** way of building UIs. Instead of telling the browser "create a div, add a class, then put some text inside it," you simply *describe* what you want: `<div>Hello</div>`. React then takes care of the actual DOM manipulation.

Embed JavaScript Expressions: One of JSX's most powerful features is its ability to **embed any valid JavaScript expression within curly braces `{}`**. This allows you to dynamically render content, apply styles, handle events, and include variables.
JavaScript
const name = 'Alice';
const greeting = <h2>Hello, {name}!</h2>; // Embedding a variable

```
function formatUser(user) {
  return user.firstName + ' ' + user.lastName;
}
const user = { firstName: 'Bob', lastName: 'Smith' };
const userDisplay = <p>User: {formatUser(user)}</p>; // Embedding a function call

const isLoggedIn = true;
const authButton = (
  <button>
    {isLoggedIn ? 'Logout' : 'Login'} // Conditional rendering
  </button>
);
```

4.

Attributes in CamelCase: HTML attributes are written in `camelCase` in JSX. For example, `class` becomes `className`, and `tabindex` becomes `tabIndex`. This is because `class` is a reserved keyword in JavaScript.

JavaScript

```
<div className="my-class" tabIndex="0">
  Click me
</div>
```

5.

Self-Closing Tags: If an element has no children, it can be self-closed, similar to XML.

JavaScript

```
<img src="logo.png" alt="Logo" />
<input type="text" />
```

6.

Requires a Single Root Element: When returning multiple JSX elements from a component, they must be wrapped in a single parent element. This can be a `<div>`, a `<section>`, or a React Fragment (`<React.Fragment>` or the shorthand `<>...</>`).

JavaScript

```
// Valid JSX: Wrapped in a div
function MyComponent() {
  return (
    <div>
      <h1>Title</h1>
      <p>Paragraph</p>
    </div>
```

```
  );
}

// Also valid: Using a React Fragment
function MyOtherComponent() {
  return (
    <>
      <h1>Title</h1>
      <p>Paragraph</p>
    </>
  );
}

// Invalid JSX: Multiple top-level elements
/*
function InvalidComponent() {
  return (
    <h1>Title</h1>
    <p>Paragraph</p>
  );
}
*/
```

7.

Why is JSX Helpful?

- **Clarity and Readability:** It makes the UI structure much clearer and more intuitive to read and write compared to deeply nested `React.createElement()` calls. It looks very similar to the final HTML output.
- **Component-Based:** It allows you to define your UI directly alongside the JavaScript logic that controls it, reinforcing React's component-based paradigm. This colocation improves maintainability.
- **Familiarity:** For developers already familiar with HTML, JSX's syntax feels very natural, reducing the learning curve for building UIs with React.
- **Type-Safety (with TypeScript):** When combined with TypeScript, JSX can provide compile-time type checking for your UI components and their props, catching errors early.
- **Performance (indirectly):** While JSX itself doesn't directly improve runtime performance, its readability and declarative nature allow developers to build more complex UIs efficiently, and the underlying `React.createElement` calls are highly optimized by React's reconciliation algorithm.

In summary, JSX is an elegant and powerful syntax extension that bridges the gap between JavaScript and HTML, making it the preferred way to write React components and define their user interfaces

Student Portfolio Component:

```
import React from 'react';

function StudentPortfolio({name=”teja”}) {

propd.name=”teja”
props.kobbies
  const student = {
    name: 'John Doe',
    imageUrl: 'https://via.placeholder.com/150',
    description: 'A passionate student who loves web development and open-source projects.'
  };

  return (
    <div style={styles.container}>
      <h2>{student.name}</h2>
      <img src={student.imageUrl} alt="Student" style={styles.image} />
      <p>{student.description}</p>
    </div>
  );
}

const styles = {
  container: {
    border: '1px solid #ccc',
    padding: '16px',
    borderRadius: '8px',
    width: '300px',
    fontFamily: 'Arial'
  },
  image: {
    width: '150px',
    height: '150px',
    borderRadius: '8px'
  }
};
```

```
export default StudentPortfolio;

import React from 'react';
import StudentPortfolio from './StudentPortfolio';

function App() {
  return (
    <div>
      <h1>Student Portfolio</h1>
      <StudentPortfolio name=”mohsn” age={20} hobbies={[“singing”,,,,,]} />
    </div>
  );
}

export default App;
```

Props:-

In React, **Props (short for "properties")** are a fundamental mechanism for **passing data from a parent component to a child component.**

Think of them like **arguments you pass to a function**, but for React components.

Here's a breakdown of what props are and why they are crucial:

What are Props?

1. **Data Carriers:** Props are how components receive data from the outside world (specifically, from their parent components).
2. **Read-Only (Immutable):** A key rule of React is that props are **read-only** within the child component that receives them. A child component should *never* directly modify its own props. If a child needs to change something based on its props, it usually requests the parent to update its state, and the parent then passes down new props.
3. **Flow Downwards:** Data with props always flows in a **unidirectional (one-way) flow, from parent to child** down the component tree. A child component cannot directly pass props back up to its parent or to a sibling component.
4. **Passed as HTML Attributes (in JSX):** When you use a component in JSX, you pass props to it just like you would pass attributes to an HTML element.

Program:

```
import React from 'react';

function StudentProfile(props) {
  return (
    <div style={styles.container}>
      <h2>{props.name}</h2>
      <img src={props.image} alt="Student" style={styles.image} />
      <p>{props.description}</p>
    </div>
  );
}

const styles = {
  container: {
    border: '1px solid #ddd',
    padding: '16px',
    margin: '16px',
    borderRadius: '8px',
    width: '300px',
    fontFamily: 'Arial',
    backgroundColor: '#f9f9f9'
  },
  image: {
    width: '150px',
    height: '150px',
    borderRadius: '8px'
  }
};

export default StudentProfile;

—----------------------------------------
import React from 'react';
import StudentProfile from './StudentProfile';

function App() {
  return (
    <div>
      <h1>Student Profiles</h1>
```

```
    <StudentProfile
      name="Alice Johnson"
      image="https://via.placeholder.com/150"
      description="A dedicated student who loves design and frontend development."
    />

    <StudentProfile
      name="Bob Smith"
      image="https://via.placeholder.com/150"
      description="An aspiring software engineer interested in AI and machine learning."
    />
   </div>
  );
}

export default App;
```

Default PRops:-

```
import React from 'react';

function StudentCard(props) {
  return (
    <div style={styles.card}>
      <h2>{props.name}</h2>
      <img src={props.image} alt="Student" style={styles.image} />
      <p>{props.description}</p>
    </div>
  );
}

// ✅ Default Props
StudentCard.defaultProps = {
  name: "Unnamed Student",
  image: "https://via.placeholder.com/150",
  description: "No description provided."
};

const styles = {
  card: {
    border: '1px solid #ccc',
```

```
    borderRadius: '8px',
    padding: '16px',
    margin: '16px',
    width: '300px',
    fontFamily: 'Arial'
  },
  image: {
    width: '150px',
    height: '150px',
    borderRadius: '8px'
  }
};

export default StudentCard;
```

App

```
import React from 'react';
import StudentCard from './StudentCard';

function App() {
  return (
    <div>
      <h1>Student List</h1>

      {/* Props provided */}
      <StudentCard
        name="Jane Doe"
        image="https://via.placeholder.com/150/0000FF/FFFFFF?text=Jane"
        description="A curious learner with a passion for backend development."
      />

      {/* Missing all props - defaults will be used */}
      <StudentCard />

      {/* Only one prop provided - rest use defaults */}
      <StudentCard
        name="Mark Lee"
      />
    </div>
  );
}

export default App;
```

Type validation:

```
import React from 'react';
import PropTypes from 'prop-types';

function StudentInfo(props) {
  return (
    <div style={styles.card}>
      <h2>{props.name}</h2>
      <p>Age: {props.age}</p>
      <p>Status: {props.isEnrolled ? 'Enrolled' : 'Not Enrolled'}</p>
    </div>
  );
}

// ✅ PropTypes validation
StudentInfo.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number.isRequired,
  isEnrolled: PropTypes.bool
};

// Default prop if not provided
StudentInfo.defaultProps = {
  isEnrolled: false
};

const styles = {
  card: {
    border: '1px solid #ccc',
    borderRadius: '8px',
    padding: '16px',
    margin: '16px',
    width: '300px',
    fontFamily: 'Arial'
  }
};

export default StudentInfo;
```

```
import React from 'react';
import StudentInfo from './StudentInfo';

function App() {
  return (
    <div>
      <h1>Student Info</h1>

      {/* Valid props */}
      <StudentInfo name="Alice" age={20} isEnrolled={true} />

      {/* Missing isEnrolled (uses default) */}
      <StudentInfo name="Bob" age={22} />

      {/* Invalid type: age as string (will show warning in console) */}
      <StudentInfo name="Charlie" age="twenty-one" isEnrolled={true} />
    </div>
  );
}

export default App;
```