Day 4

# What Is JSX?

Learn About JSX

kodemaven-portfolio.vercel.app

# What Is JSX?

JSX **(JavaScript XML)** is a syntax extension for JavaScript that looks similar to HTML.

It allows you to write HTML-like code directly within your JavaScript files, making it easier to create and visualize the structure of your React components.

React then transforms this JSX into regular JavaScript that browsers can understand.

Although it looks like HTML, **JSX is not HTML;** it's a syntax sugar that React uses to create "React elements" (the building blocks of React components).

# Why Use JSX In React?

**1. Readability:** Writing HTML-like code inside JavaScript makes it easier to understand the structure of a component.

**2. Simplicity:** JSX helps you write UI logic and markup in a single file, making development faster and more streamlined.

**3. Fewer Errors:** JSX includes helpful error messages that guide you when something goes wrong.

# JSX vs. Regular Javascript

In **regular JavaScript**, you would create elements like this:

```
1 const element = React.createElement('h1', {}, 'Hello,
  world!');
```

With **JSX**, you can write the same thing in a much clearer way:

```
1 const element = <h1>Hello, world!</h1>;
```

✅ Cleaner Syntax
✅ Easy to Understand
✅ Quicker to Write

JSX helps you write more concise and readable code!

# How Does JSX Work?

When you write JSX, it is compiled by tools like Babel into regular JavaScript code that React can understand. For example:

```jsx
const element = <h1>Hello, world!</h1>;
```

is compiled into:

```jsx
const element = React.createElement('h1', {}, 'Hello, world!');
```

This means JSX is not required to write React code, but it makes the code easier to write and understand.

# Basic Rules of JSX In React

**1. Return a Single Parent Element:** JSX expressions must have one root element. This means you can't return multiple sibling elements directly; instead, wrap them in a single parent, like a  or a React Fragment (<>...</>).

```jsx
// Correct
return (
  <> // a React fragment wrapping all elements
    <h1>Hello!</h1>
    <p>Welcome to React.</p>
  </>
);

// Incorrect (multiple root elements)
return (
  <h1>Hello!</h1>
  <p>Welcome to React.</p>
);
```

**2. Use Curly Braces for JavaScript Expressions:**
You can insert any JavaScript expression inside curly braces { } in your JSX.

```jsx
const MyComponent = () => {
    const name = 'Alice';

    return (
      <>
        <h1>Hello, {name}!</h1>
      </>
    )
// Outputs: Hello, Alice!
 }

export default MyComponent
```

The variable name was accessed in JSX by wrapping a curly braces around the variable.

**3. Self-Closing Tags:** Tags without children (like <img> or <input>) must be self-closed.

```jsx
const MyComponent = () => {

  return (
    <>
      <img src="logo.png" alt="Logo" title="Logo" />
    </>
  )
}

export default MyComponent
```

**4. Class vs. className:** Since **class** is a reserved keyword in JavaScript, JSX uses **className** instead of class to define CSS classes.

```jsx
const MyComponent = () => {

  return (
    <>
      <h1 className="greetings">Hello, {name}!</h1>
    </>
  )
}

export default MyComponent
```

# Embedding Expressions In JSX

You can use any JavaScript expression within JSX by **wrapping it in curly braces { }.** This allows you to dynamically display content and use JavaScript functions or variables to compute the values that will be rendered.

For Example:

```jsx
const MyComponent {
  const numbers = [1, 2, 3, 4, 5];

  // Using the map() function to create a list of JSX elements
  const listItems = numbers.map((number) => <li key={number}>{number}</li>);

  return (
    <>
      <h2>Numbers List:</h2>
      <ul>{listItems}</ul> {/* Embedding the dynamic list in JSX */}
    </>
  );
}

export default MyComponent;

```

This simple React component displays a list of numbers dynamically by embedding the mapped list element in JSX as the variable “listItems”.

# Styling in JSX

JSX allows you to style components directly using inline styles or by applying CSS classes.

**1.** Inline Styles: Use an object to define inline styles. Remember that style properties in JSX are written in camelCase.

```jsx
const MyComponent = () => {

  const headingStyle = { color: 'blue', fontSize: '20px' };

  return (
    <>
      <h1 style={headingStyle}>Styled Heading</h1>
    </>
  )
}
// Output: Hello, John Doe!

export default MyComponent
```

Style applied inline

**2**. CSS Classes: Use the className attribute to apply CSS classes defined in a stylesheet.

```jsx
const MyComponent = () => {

  return (
    <>
      <h1 className="greetings">Hello, {name}!</h1>
    </>
  )
}

export default MyComponent
```

The **“greetings”** class will be used as a selector to write styles for the h! element in the *index.css* or **custom css** file.

Remember: It’s **className**, not class, because class is a reserved keyword in JavaScript.

# Conditional Rendering in JSX

Conditional rendering in React allows you to display different content based on certain conditions.

You can use **JavaScript logical operators** (like if, &&, or the ternary operator ? :) directly in JSX.

```jsx
const MyComponent {

 return (
    <div>
      {isLoggedIn ? ( // Using a ternary operator for conditional rendering
        <h1>Welcome back, User!</h1>
      ) : (
        <h1>Please sign in to continue.</h1>
      )}
    </div>
  );
}

export default MyComponent;

```

- If **isLoggedIn** is true, it renders <h1>Welcome back, User!</h1>.
- If **isLoggedIn** is false, it renders <h1>Please sign in to continue.</h1>.

I hope you found this material useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be helpful to someone 👌

kodemaven-portfolio.vercel.app

# Hi There!

## Thank you for reading through

Did you enjoy this knowledge?

💼 Follow my LinkedIn page for more work-life balancing and Coding tips.

🌐 LinkedIn: Oluwakemi Oluwadahunsi



kodemaven-portfolio.vercel.app