# JSX and Components

## Understanding JSX

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

1. **JSX Syntax:**
   - JSX allows you to write HTML-like code within JavaScript.
   - JSX code is transformed into JavaScript code by tools like Babel.
   - Example:

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

2. **Embedding Expressions:**

- You can embed any JavaScript expression in JSX by wrapping it in curly braces {}.
- Example:

```jsx
const name = 'Saurabh';
const element = <h1>Hello, {name}!</h1>;
```

3. **JSX Attributes:**

- JSX attributes are similar to HTML attributes but follow camelCase naming conventions.
- Example:

```jsx
const element = <img src={user.avatarUrl} alt={user.name} />;
```

## Creating Functional Components

**Functional Components** are basic JavaScript functions that return JSX.

1. **Defining Functional Components:**
   - Functional components are defined as functions and can accept props as arguments.
   - Example:

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

2. **Rendering Functional Components:**

- Functional components can be used like HTML elements.
- Example

```jsx
const element = <Welcome name="Sara" />;
```

3. **Composing Components:**

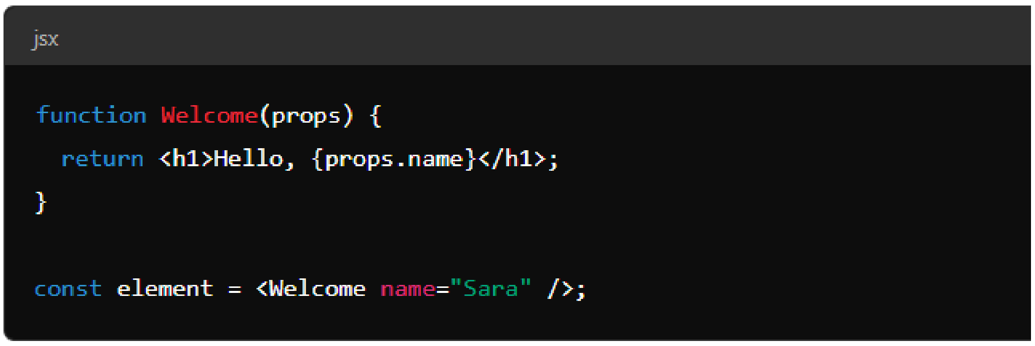
- Functional components can be composed to build complex UIs.
- Example:

```jsx
function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

## Props and State

**Props (Properties):**

- Props are inputs to components and are passed down from parent components to child components.
- Props are read-only; they should not be modified by the component receiving them.
- Example:



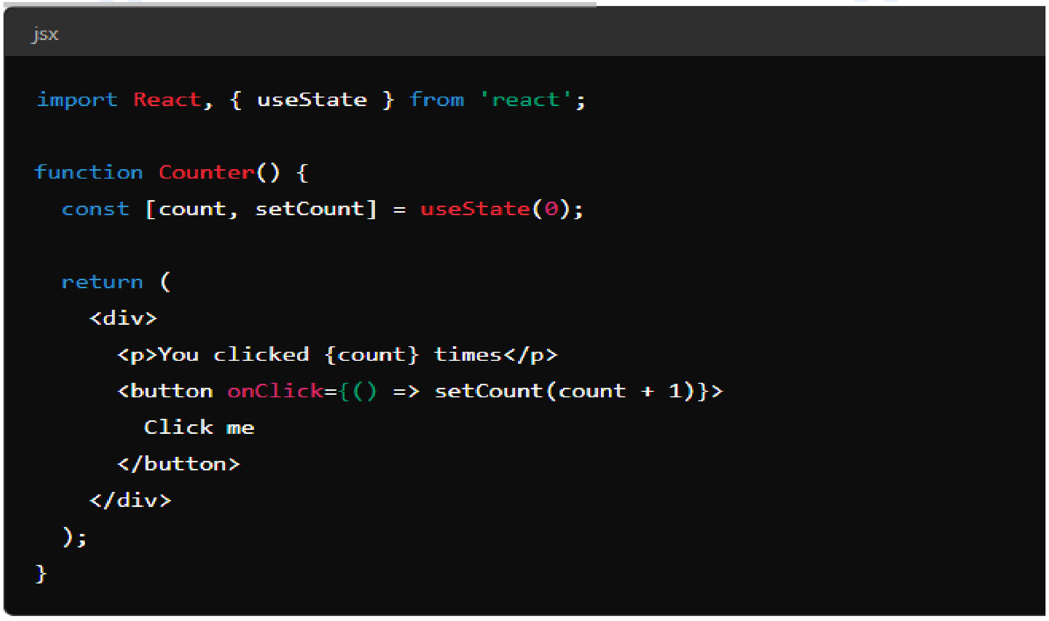
jsx

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
```

**State:**

- State is a built-in object that holds data that may change over the lifetime of the component.
- Unlike props, state is managed within the component itself and can be updated with the setState function.
- Example using Hooks (functional components):



jsx

```jsx
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

**Difference between Props and State:**

- **Props:**
  - Passed from parent to child components.
  - Immutable within the child component.
  - Used to pass data and event handlers down to child components.
- **State:**
  - Managed within the component.
  - Mutable, can be updated using setState.
  - Used to handle data that changes over time or in response to user actions.