# 

# 

# Section 4: State and Props

### **Section 4: State and Props**

* **State:**
  + **Managing component data**
  + **Updating state**
* **Props:**
  + **Passing data between components**
* **Example:**
  + **Create a counter component using state.**

## 

### **State: Managing Component Data**

State is a local data storage mechanism that's specific to a component. It's used to manage data that can change over time within a component.

**Updating state:** State can be updated using the setState method (for class components) or the useState hook (for functional components). It's important to understand that updating state is asynchronous.

### **Props: Passing Data Between Components**

Props are read-only properties that allow you to pass data from a parent component to a child component. They are similar to function arguments.

### **Example: Create a counter component using state**

**Software to download:**

* **Node.js:** Provides a runtime environment for JavaScript.
* **npm (or yarn):** Package manager for Node.js.

**import React, { useState } from 'react';**

**function Counter() {**

**const [count, setCount] = useState(0);**

**const increment = () => {**

**setCount(count + 1);**

**};**

**return (**

**<div>**

**<p>Count: {count}</p>**

**<button onClick={increment}>Increment</button>**

**</div>**

**);**

**}**

**export default Counter;**

**Explanation:**

1. Import useState from React.
2. Create a functional component Counter.
3. Use useState to initialize the count state with a value of 0.
4. Define an increment function to update the count state by adding 1.
5. Render the current count value and a button that triggers the increment function when clicked.

By understanding state and props, you'll be able to build interactive and dynamic React components.