State Management

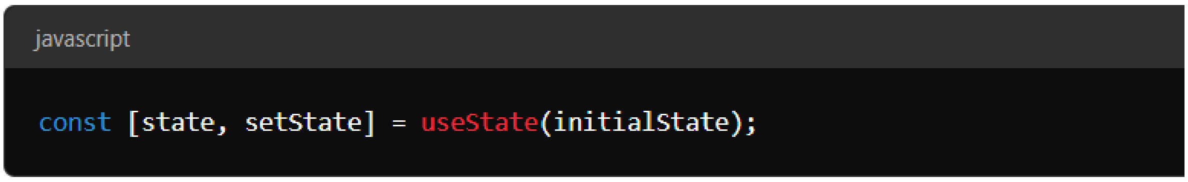
State Management

- **State** in React is an object that determines how a component renders and behaves.
- Each component can have its own state, which is managed within that component.
- State allows React components to change their output over time in response to user actions, network responses, or any other event.

Introduction to useState Hook

- The useState hook is a fundamental hook in React for managing state in functional components.
- It allows you to add state to a functional component.

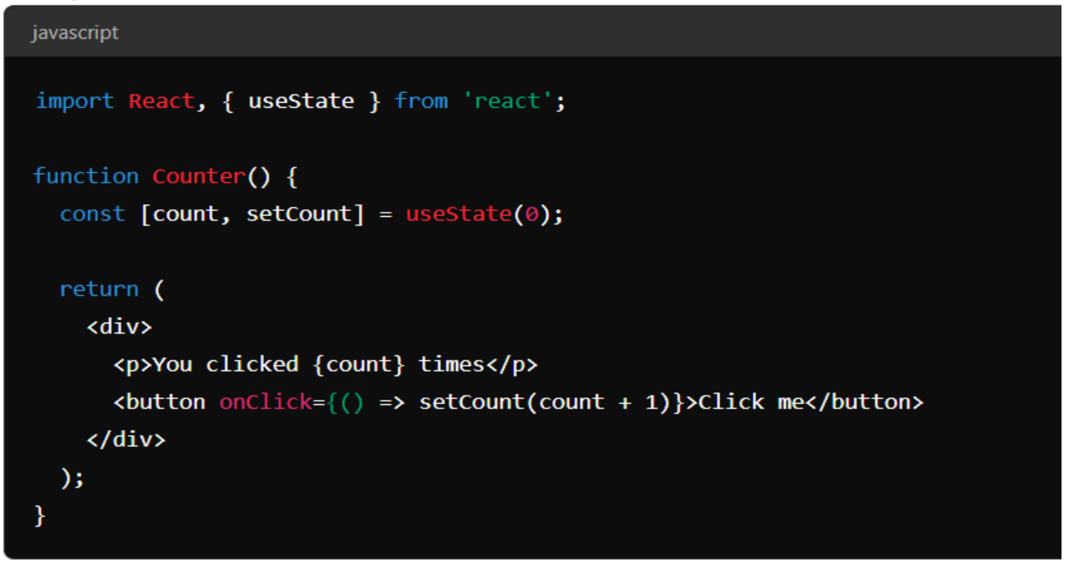
Syntax



```javascript
const [state, setState] = useState(initialState);
```

- state: The current state.
- setState: A function that updates the state.
- initialState: The initial value of the state.

Example

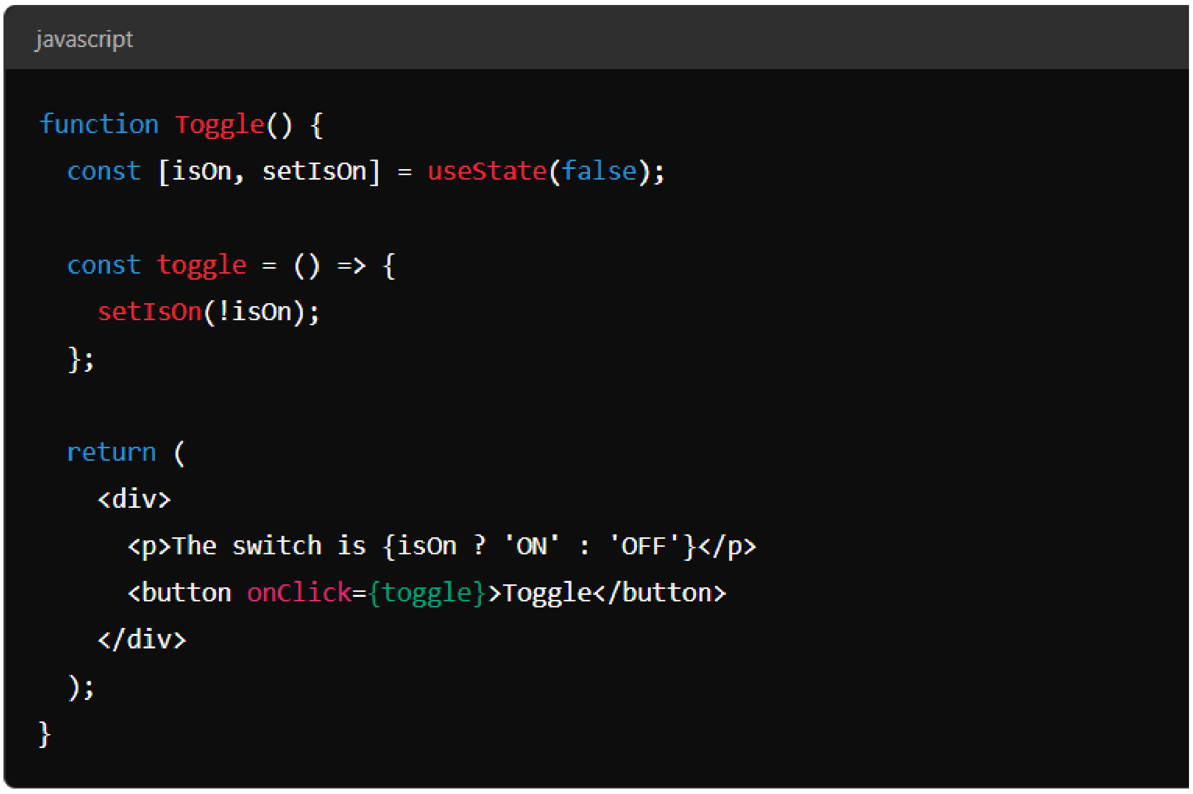


```javascript
import React, { useState } from 'react';

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

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

Managing State in Functional Components

- **Initialization**: State is initialized using the useState hook.
- **Reading State**: State can be accessed directly from the variable returned by useState.
- **Updating State**: State is updated using the setter function returned by useState.

Example



javascript

```javascript
function Toggle() {
  const [isOn, setIsOn] = useState(false);

  const toggle = () => {
    setIsOn(!isOn);
  };

  return (
    <div>
      <p>The switch is {isOn ? 'ON' : 'OFF'}</p>
      <button onClick={toggle}>Toggle</button>
    </div>
  );
}
```

Lifting State Up

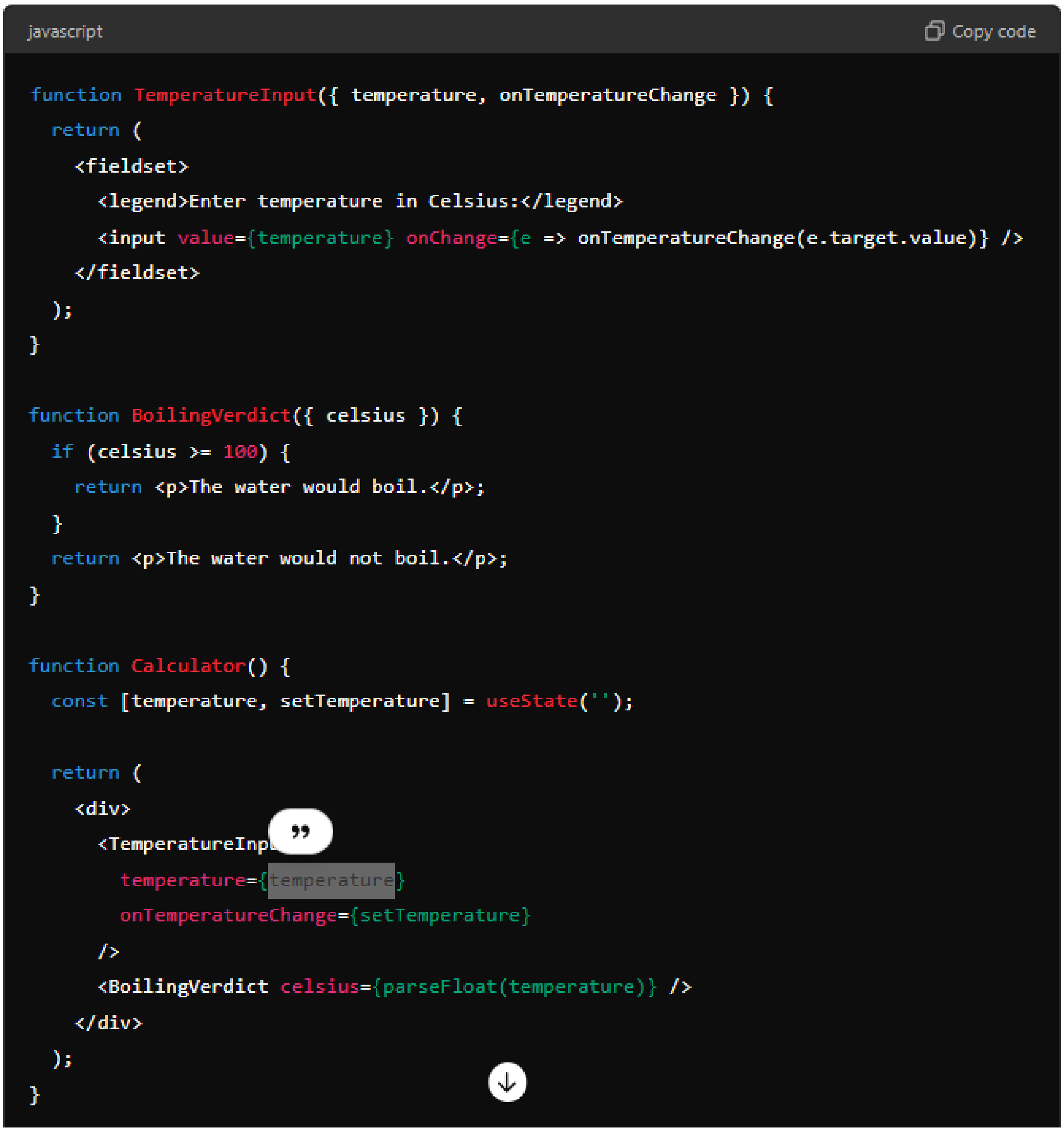
- **Lifting state up** is a technique for managing state shared by multiple components.
- The shared state is lifted up to the closest common ancestor component and passed down as props to child components.

Steps for Lifting State Up

1. Identify the common ancestor component.
2. Move the state to the common ancestor.
3. Pass the state and the state updater function as props to child components.

Example

Consider two components, TemperatureInput and BoilingVerdict, which need to share the same temperature state.



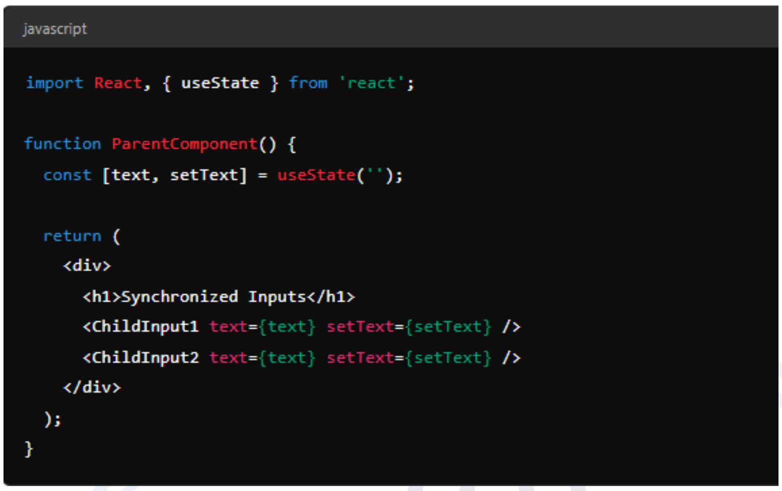
```javascript
function TemperatureInput({ temperature, onTemperatureChange }) {
  return (
    <fieldset>
      <legend>Enter temperature in Celsius:</legend>
      <input value={temperature} onChange={e => onTemperatureChange(e.target.value)} />
    </fieldset>
  );
}

function BoilingVerdict({ celsius }) {
  if (celsius >= 100) {
    return <p>The water would boil.</p>;
  }
  return <p>The water would not boil.</p>;
}

function Calculator() {
  const [temperature, setTemperature] = useState('');

  return (
    <div>
      <TemperatureInp
        temperature={temperature}
        onTemperatureChange={setTemperature}
      />
      <BoilingVerdict celsius={parseFloat(temperature)} />
    </div>
  );
}
```

In this example, the Calculator component manages the state and passes it down to TemperatureInput and BoilingVerdict components

Example 2: Synchronized Inputs

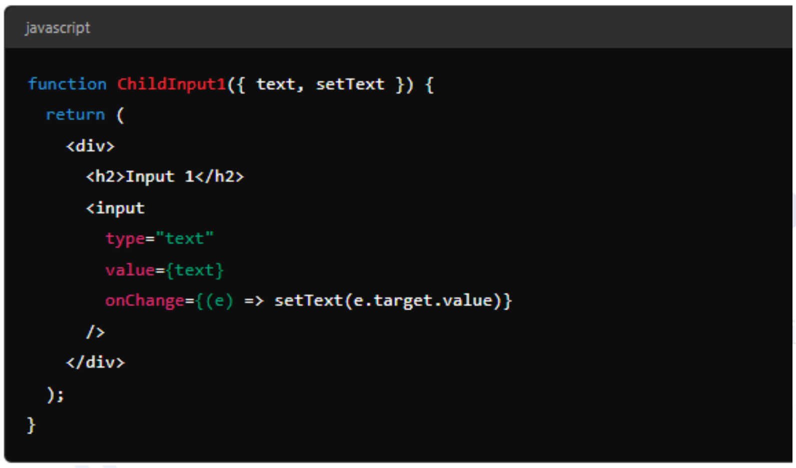
In this example, we have two input fields that need to be synchronized with each other.

Parent Component



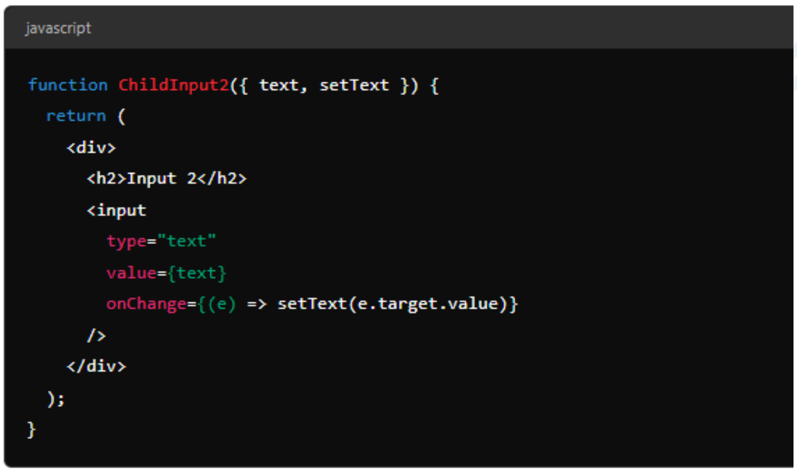
```javascript
import React, { useState } from 'react';

function ParentComponent() {
  const [text, setText] = useState('');

  return (
    <div>
      <h1>Synchronized Inputs</h1>
      <ChildInput1 text={text} setText={setText} />
      <ChildInput2 text={text} setText={setText} />
    </div>
  );
}
```

Child Input 1



```javascript
function ChildInput1({ text, setText }) {
  return (
    <div>
      <h2>Input 1</h2>
      <input
        type="text"
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
    </div>
  );
}
```

Child Input 2



```javascript
function ChildInput2({ text, setText }) {
  return (
    <div>
      <h2>Input 2</h2>
      <input
        type="text"
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
    </div>
  );
}
```

Here, the state text is lifted up to the ParentComponent, and both ChildInput1 and ChildInput2 share the same state. When you type in one input, the other input automatically updates.