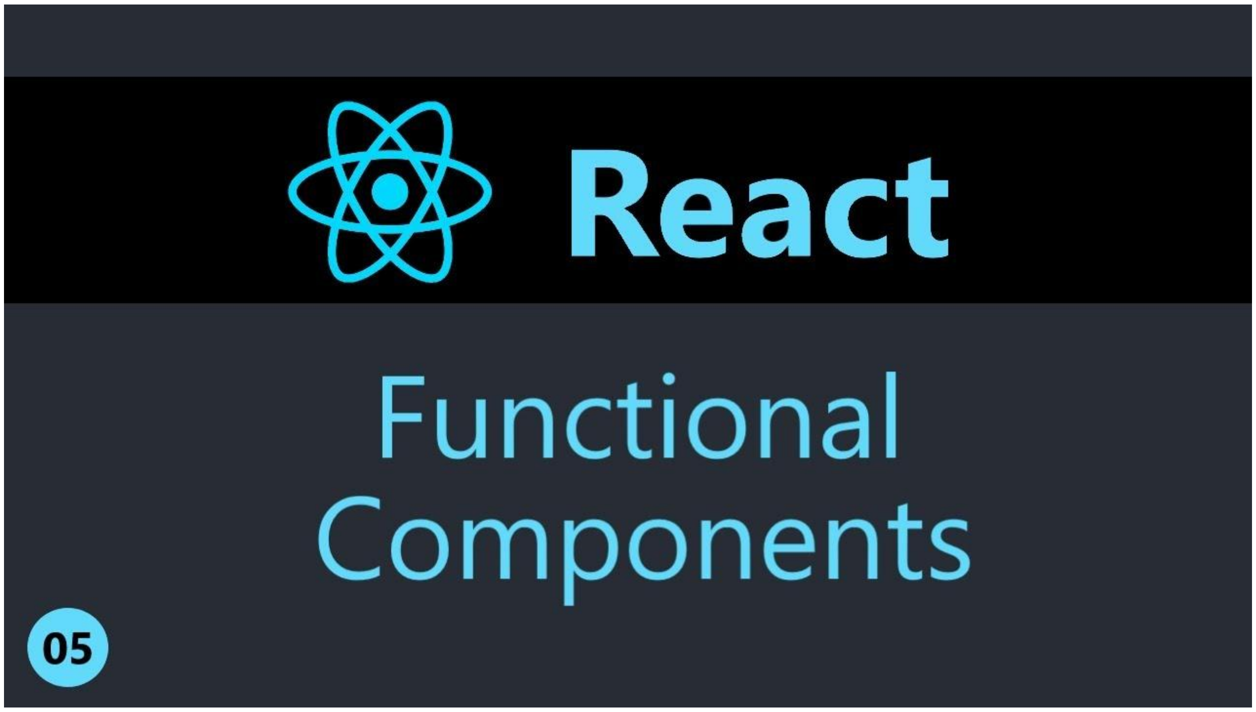
React Functional Components

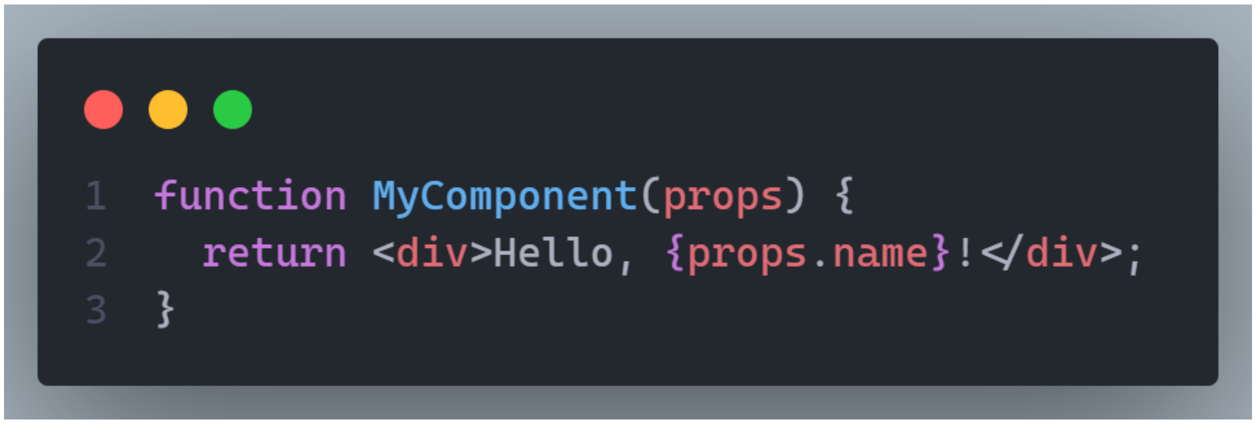


What is a functional component in React?

In React, functional components are one of two main types of components that can be used to create UI elements. Functional components are simpler and more lightweight than class components, and developers often prefer them because they are easier to read and understand.

A functional component is a JavaScript function that returns JSX (a syntax extension for JavaScript that allows you to write HTML-like code).

The syntax for functional Components -



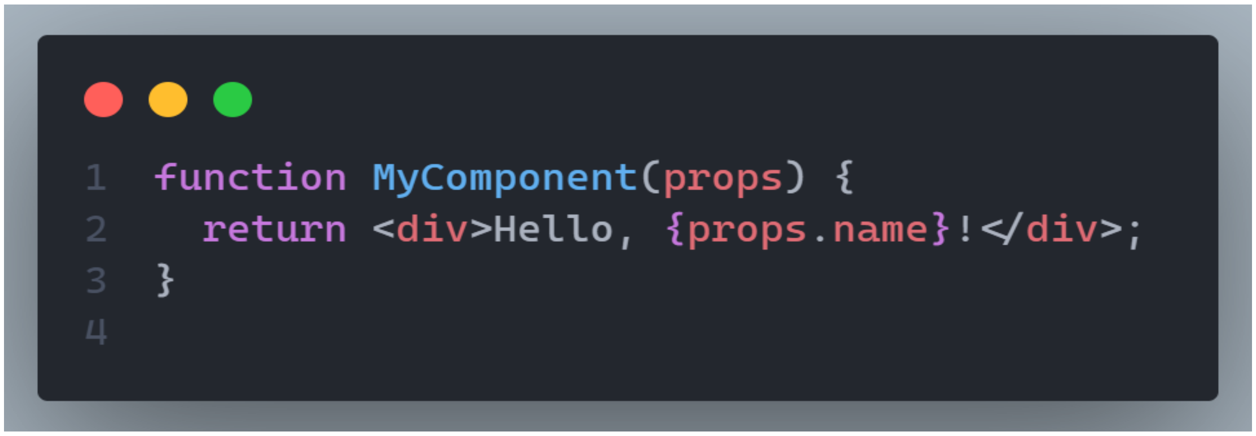
```
function MyComponent(props) {
  return <div>Hello, {props.name}!</div>;
}
```

In this example, MyComponent is a functional component that takes in a props object and returns a JSX element that displays a greeting message.

Props and state in functional components

Props and state are two important concepts in React. Props are used to pass data from a parent component to a child component, while state is used to manage data within a component.

In functional components, props are passed in as an argument to the function and can be accessed using dot notation (e.g. props.name). Here is an example:

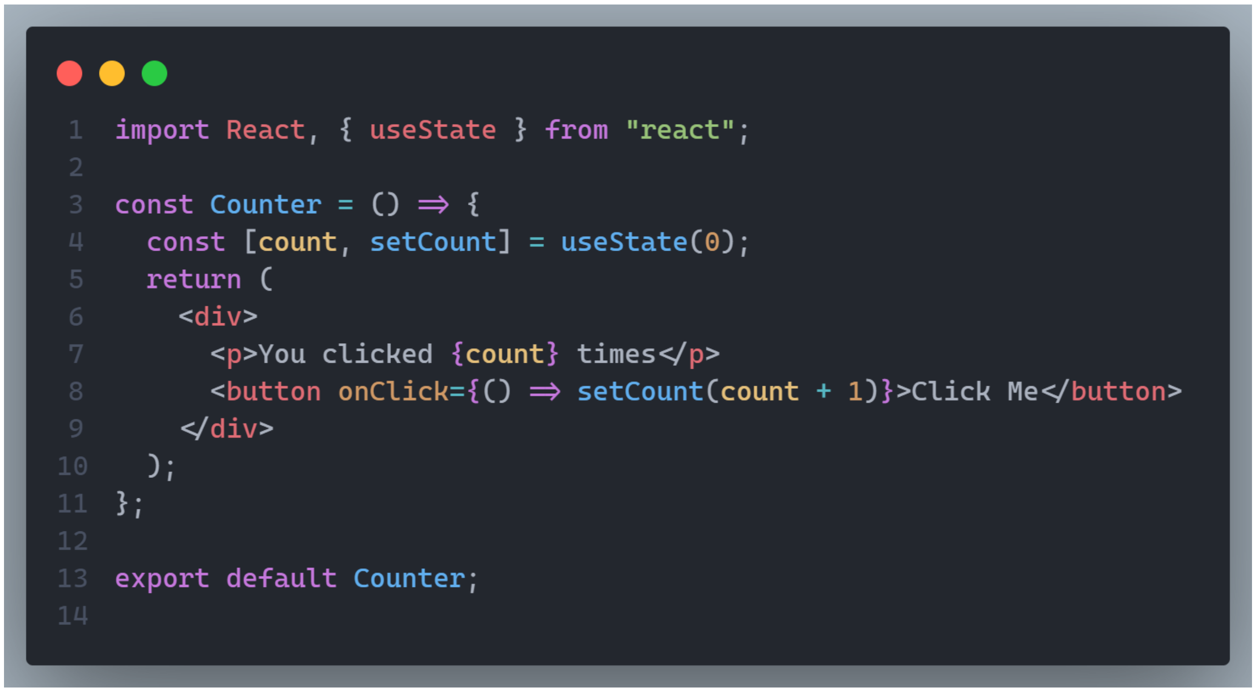


```
function MyComponent(props) {
  return <div>Hello, {props.name}!</div>;
}

```

In this example, **props.name** is used to display the name passed in through props.

To add stateful behaviour to a functional component, React provides a useState hook. Here is an example:



```
import React, { useState } from "react";

const Counter = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click Me</button>
    </div>
  );
};

export default Counter;

```

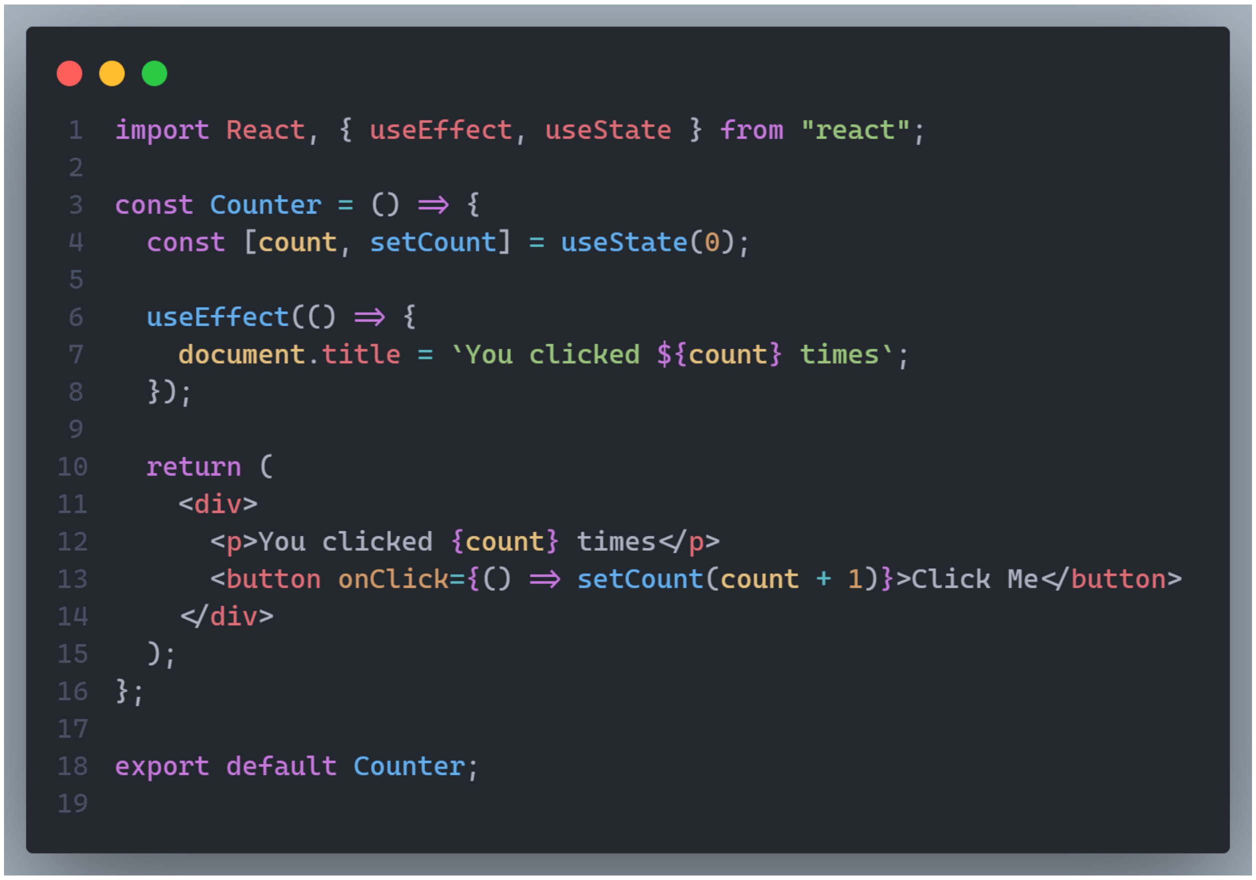
In this example, the useState hook is used to create a count state variable and a setCount function that updates the state. The count variable is used to display the number of times the button has been clicked.

Lifecycle methods and hooks in functional components

Lifecycle methods are methods that are called at specific points in a component's lifecycle, such as when it is mounted, updated, or unmounted. Class components have access to a number of lifecycle methods, but functional components do not.

Instead, React provides a number of hooks that can be used to add lifecycle-like behaviour to functional components. Some common hooks include useEffect, useMemo, and useCallback.

Here is an example of the useEffect hook:



```
import React, { useEffect, useState } from "react";

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

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

export default Counter;
```

In this example, the useEffect hook is used to update the document title whenever the count state variable changes. The useEffect hook