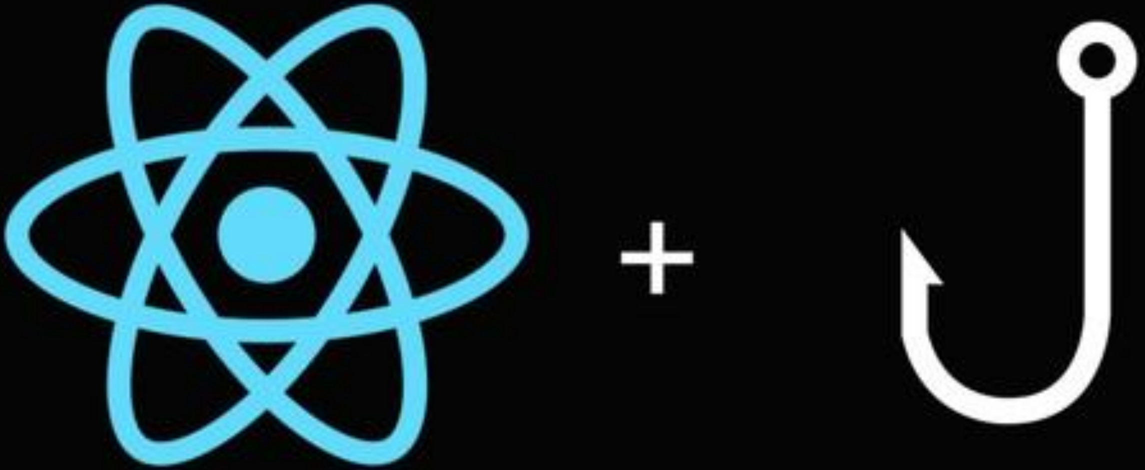


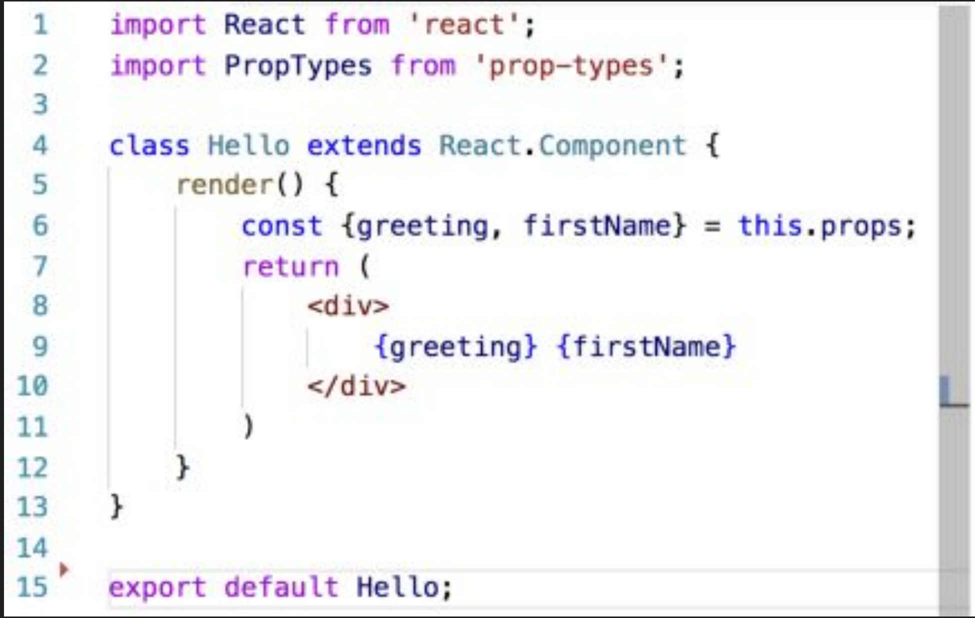
## What?

Hooks are functions that let you share stateful logic between components

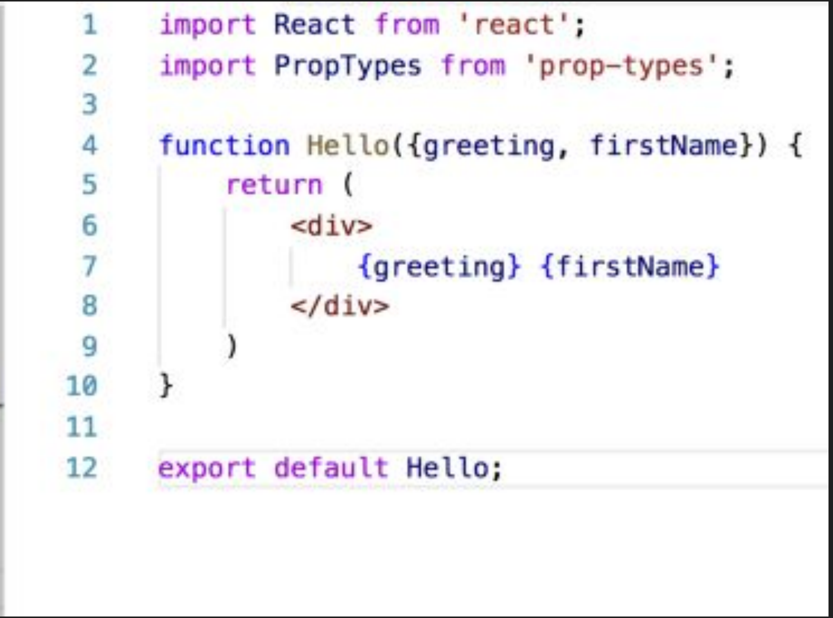
## Why?

## Class Component

## Function Component



```
import React from 'react';
import PropTypes from 'prop-types';

class Hello extends React.Component {
    render() {
        const {greeting, firstName} = this.props;
        return (
            <div>
                {greeting} {firstName}
            </div>
        )
    }
}

export default Hello;
```



```
import React from 'react';
import PropTypes from 'prop-types';

function Hello({greeting, firstName}) {
    return (
        <div>
            {greeting} {firstName}
        </div>
    )
}

export default Hello;
```

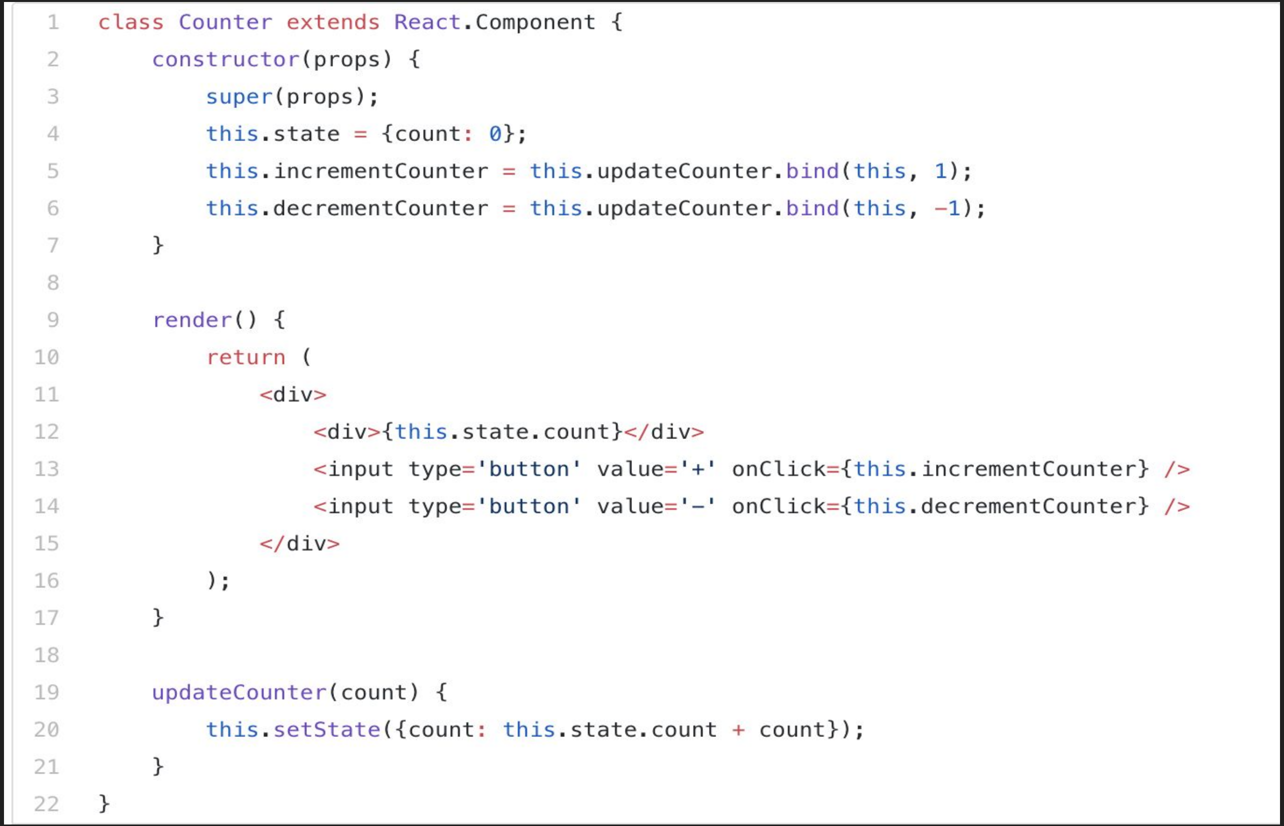
## Functional components vs Class components

1. State management
2. Life Cycle
3. Sharing non-visual logic with other components

## Functional components vs Class components

1. **State management**
2. Life Cycle
3. Sharing non-visual logic with other components

# Class Component State Management



```
class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = {count: 0};
        this.incrementCounter = this.updateCounter.bind(this, 1);
        this.decrementCounter = this.updateCounter.bind(this, -1);
    }

    render() {
        return (
            <div>
                <div>{this.state.count}</div>
                <input type='button' value='+' onClick={this.incrementCounter} />
                <input type='button' value='-' onClick={this.decrementCounter} />
            </div>
        );
    }

    updateCounter(count) {
        this.setState({count: this.state.count + count});
    }
}
```

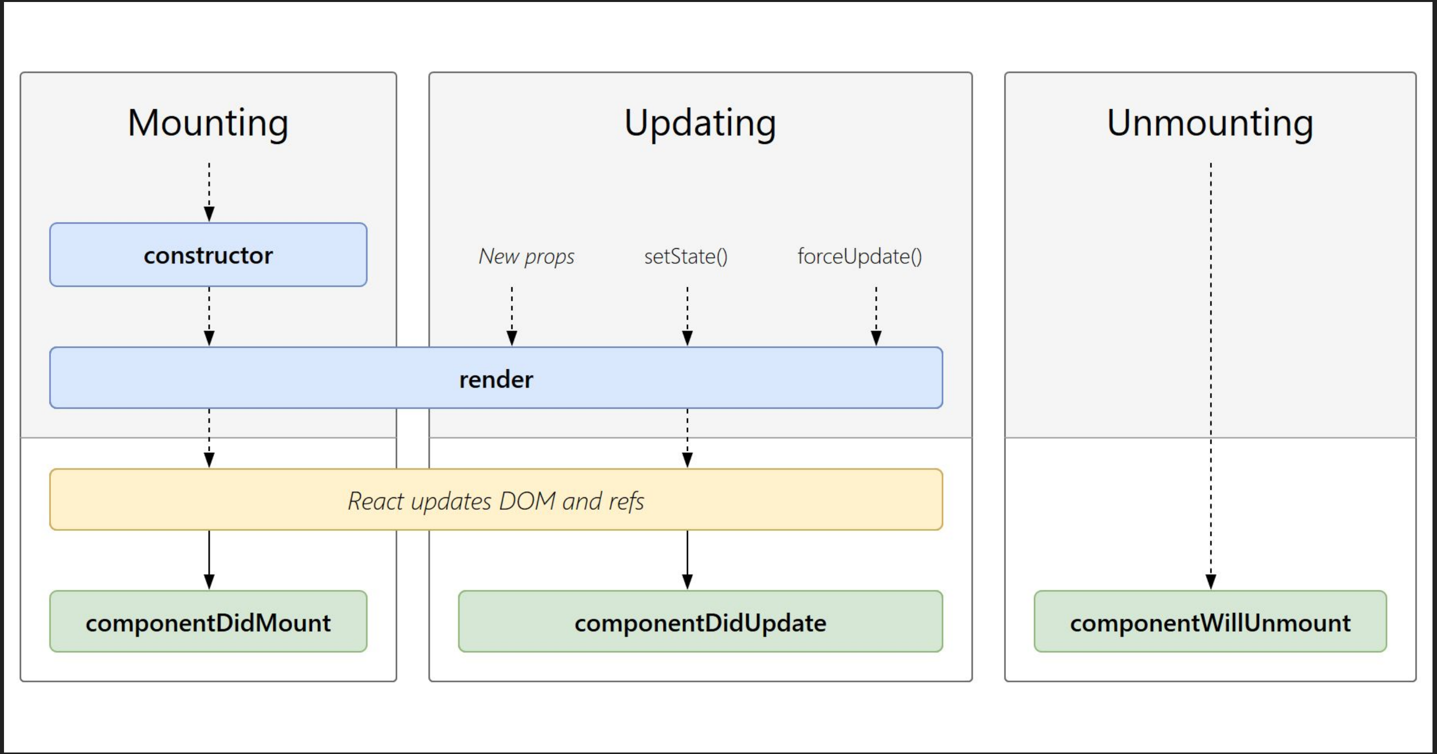
# A More Complicated Counter Example

```
class Child extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            counter: 100
        }
    }
    render = () => {
        return (
            <div>
                <h2>{this.state.counter}</h2>
                <button onClick={this.props.clickHandler.bind(this)}>Click</button>
            </div>
        )
    }
}

export default class Parent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            counter: 0
        }
    }
    addCounter = () => {
        this.setState({
            counter: ++this.state.counter
        })
    }
    render = () => {
        return (
            <div>
                <h1>{this.state.counter}</h1>
                <Child clickHandler={this.addCounter} />
            </div>
        )
    }
```

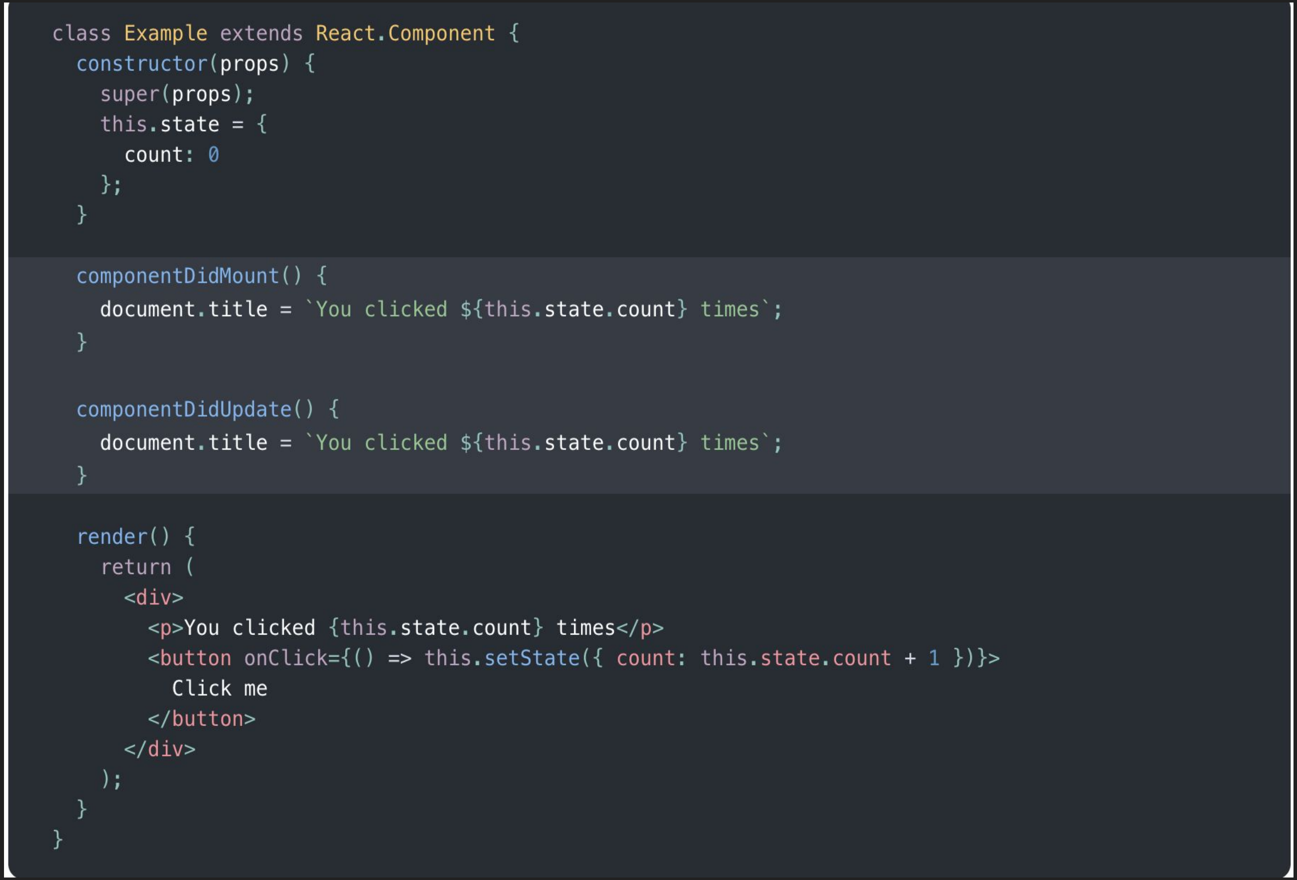
## Functional components vs Class components

1. State management
2. **<u>Life Cycle</u>**
3. Sharing non-visual logic with other components

# Life Cycle Methods



# Life Cycle Methods Example



```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }

  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

## Functional components vs Class components

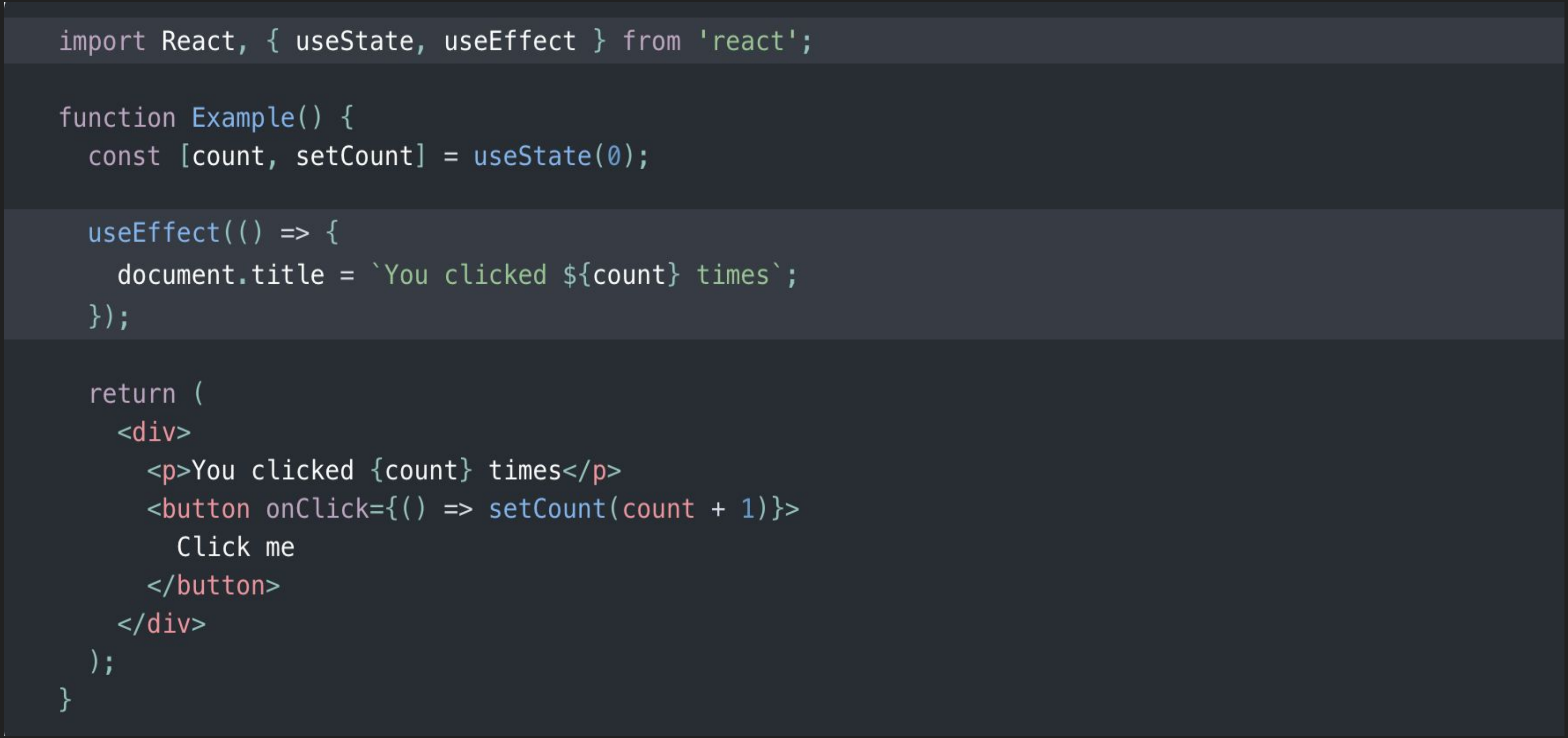
1. State management
2. Life Cycle
3. **Sharing non-visual logic with other components**

## Problems With Class Component

1. Calling Super
2. No one knows how “this” works in javascript
3. Bind is annoying
4. Often requires duplicate logic in life cycle methods
5. Life cycle method is verbose and unintuitive

## React Hooks

# React Hooks Example

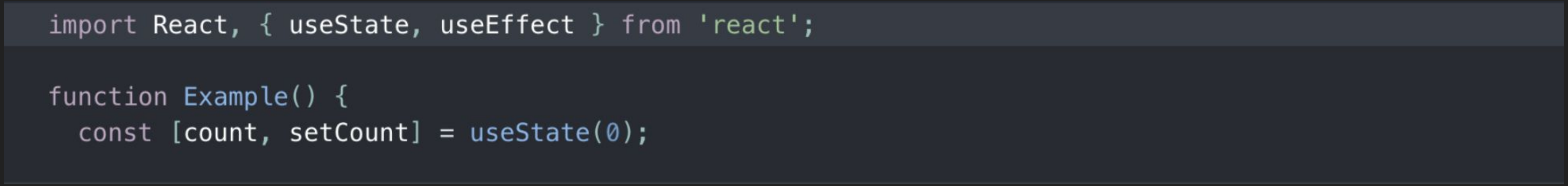


```
import React, { useState, useEffect } from 'react';

function Example() {
  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>
  );
}
```

## useState

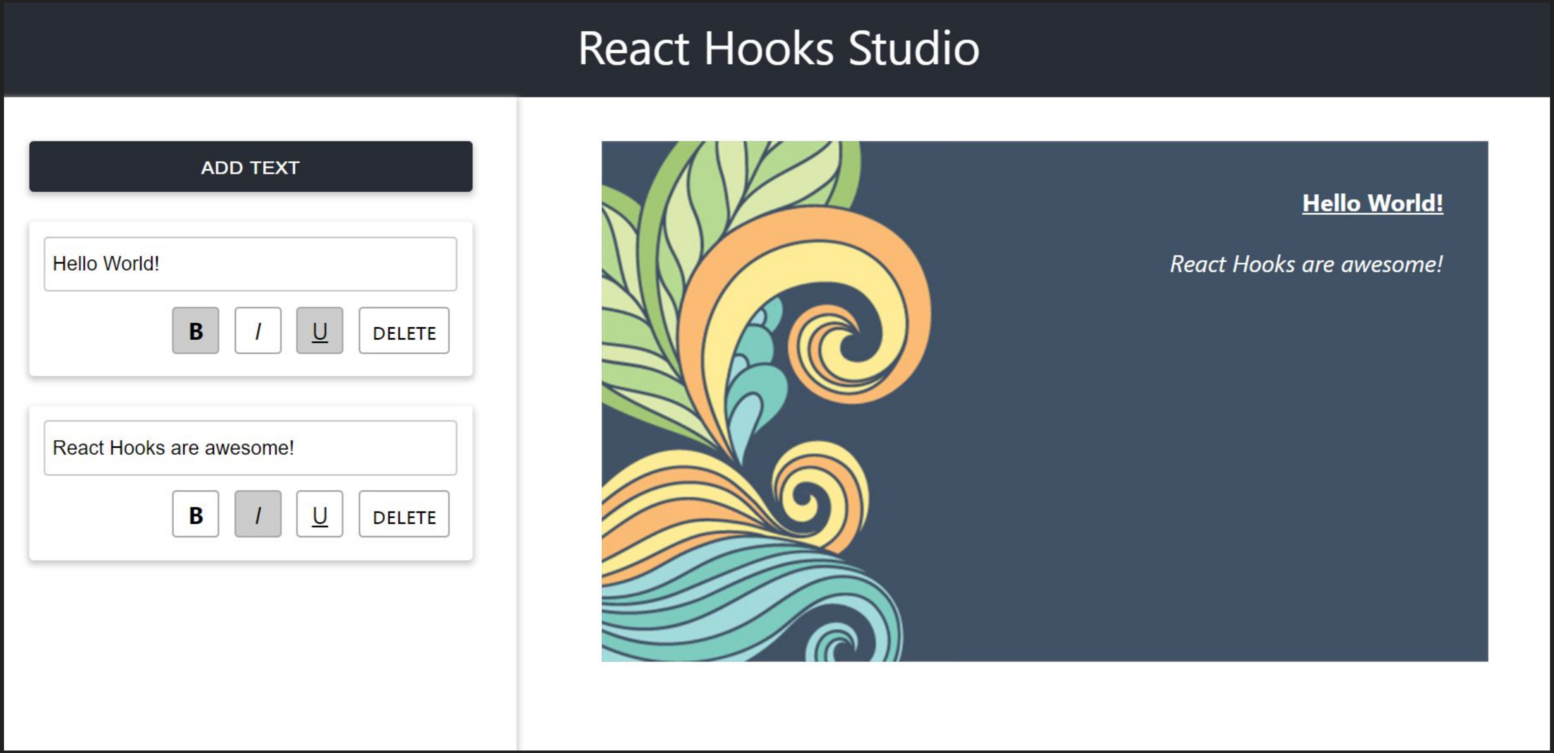
# Hook: useState

useState returns a pair: the current state value and a function that lets you update it



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

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

# Demo - Text Fields in Studio!



Hello World!

React Hooks are awesome!

**<u>Hello World!</u>**

*React Hooks are awesome!*

## useEffect

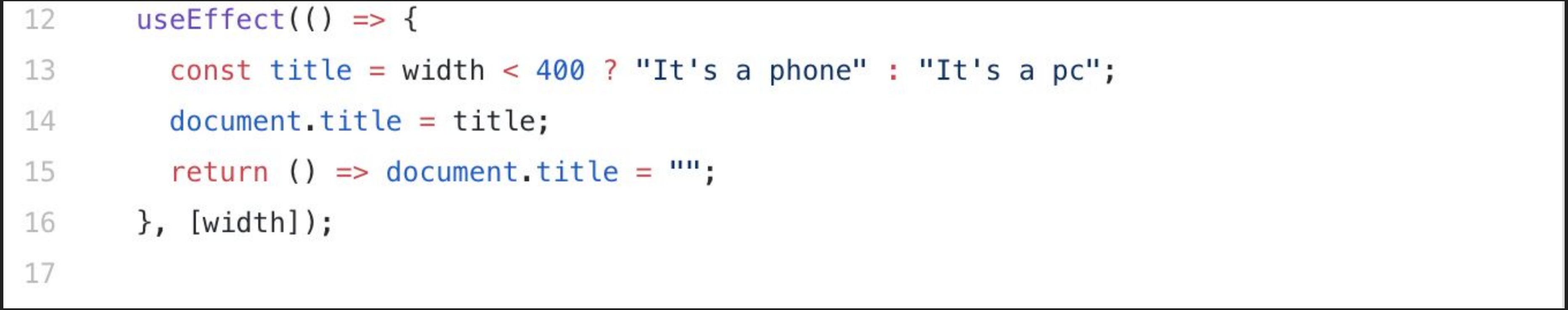
# Hook: useEffect

useEffect replaces:

componentDidMount(), componentWillUnmount(), componentDidUpdate()

Parameters:

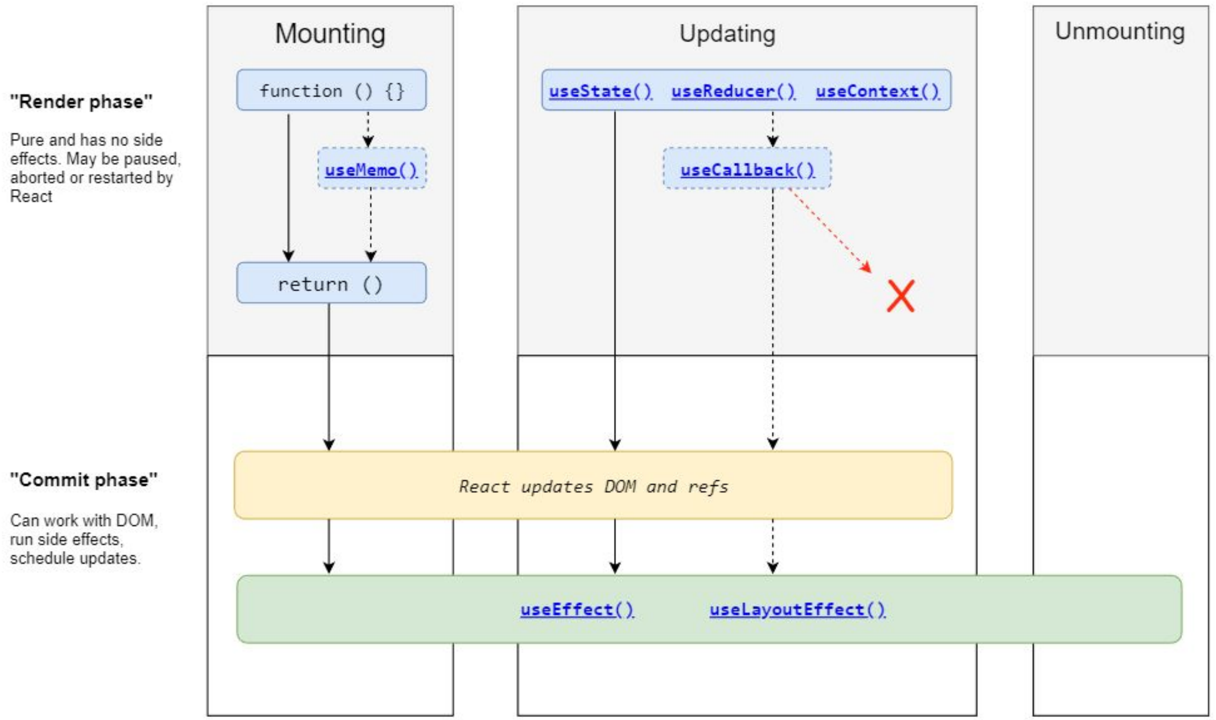
1. A call Function to be executed when component mounts
2. An array of variables to monitor. Any change to those variables will rerun useEffect()
3. The function returned inside the callback is executed after component unmount



```
  useEffect(() => {
    const title = width < 400 ? "It's a phone" : "It's a pc";
    document.title = title;
    return () => document.title = "";
  }, [width]);

```

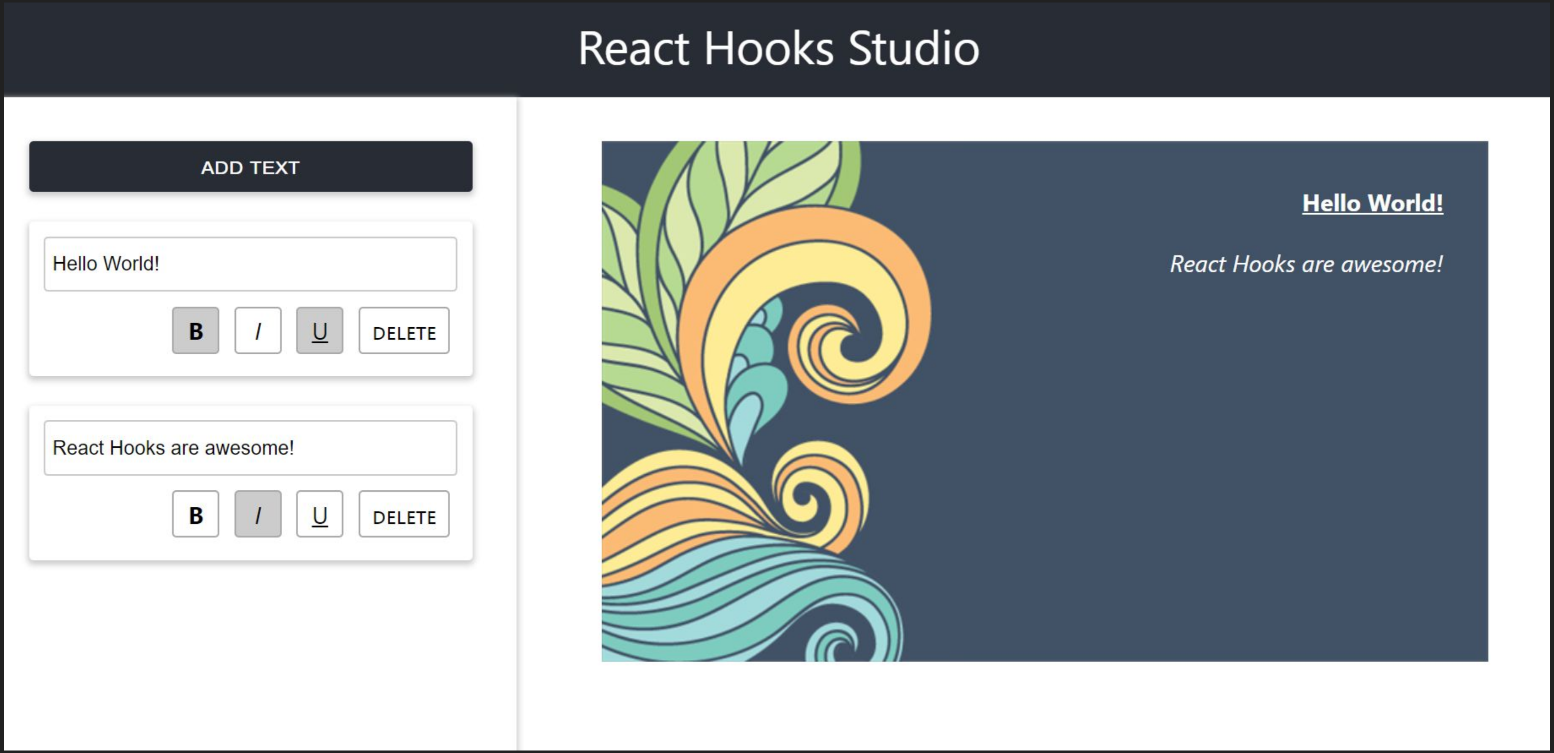
# React hooks lifecycle - function component

version 16.8.x



Made with <3 by Gal Margalit. feel free to contribute and share wavez/react-hooks-lifecycle

## Demo - Saving the document to Local Storage


React Hooks Studio

ADD TEXT

Hello World!

B I U DELETE

React Hooks are awesome!

B I U DELETE

**<u>Hello World!</u>**

*React Hooks are awesome!*

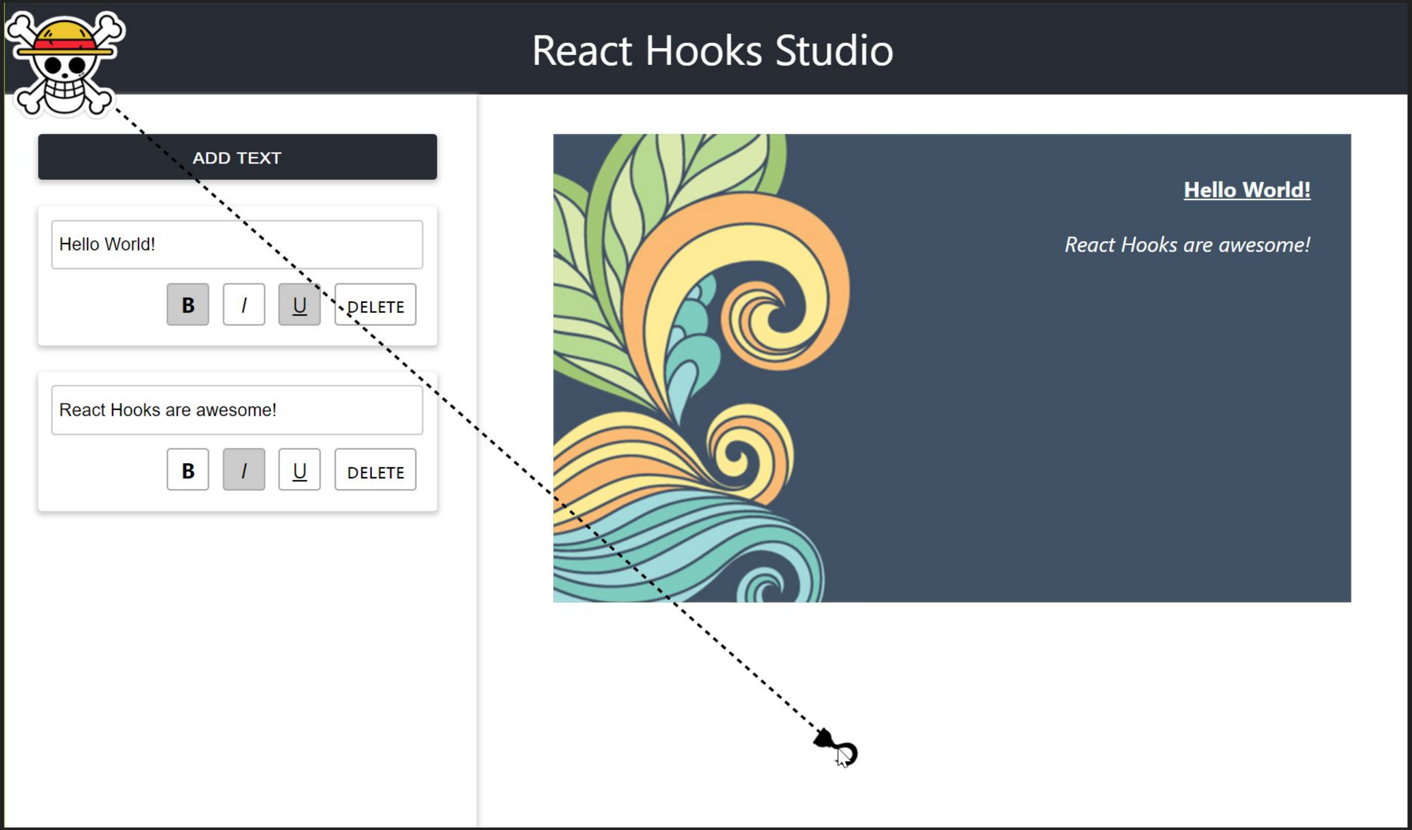
## Other Hooks!

## Other Hooks

useContext()
useReducer()
useCallback()
useLayoutEffect()
useRef()

## Custom Hooks

# Custom Hooks - Pirate Hook!



React Hooks Studio

ADD TEXT

Hello World!

B I U DELETE

React Hooks are awesome!

B I U DELETE

**<u>Hello World!</u>**

*React Hooks are awesome!*

The END