AN INTRODUCTION TO HOOKS WITH REACT NATIVE

GOAL

- Understand what react hooks are, and what problems they solve
- Use hooks in a simple react native application.

WHAT ARE REACT HOOKS

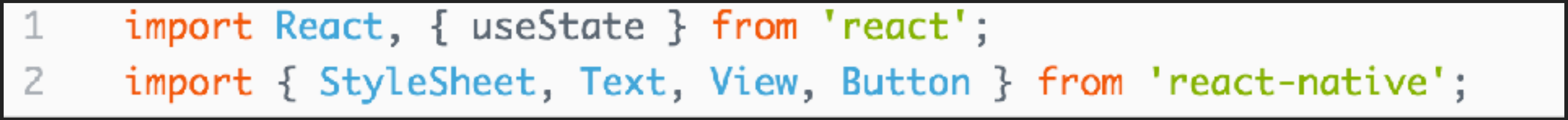
- Hooks are functions that let you “hook into” React state and lifecycle features from functional components. Hooks don’t work inside classes – they let you use React without classes. -React Documentation

WHEN WOULD I USE A HOOK?

- If you write a functional component and realize you need to add some state to it, previously you had to convert it to a class. Now you can use a Hook inside the existing functional component.

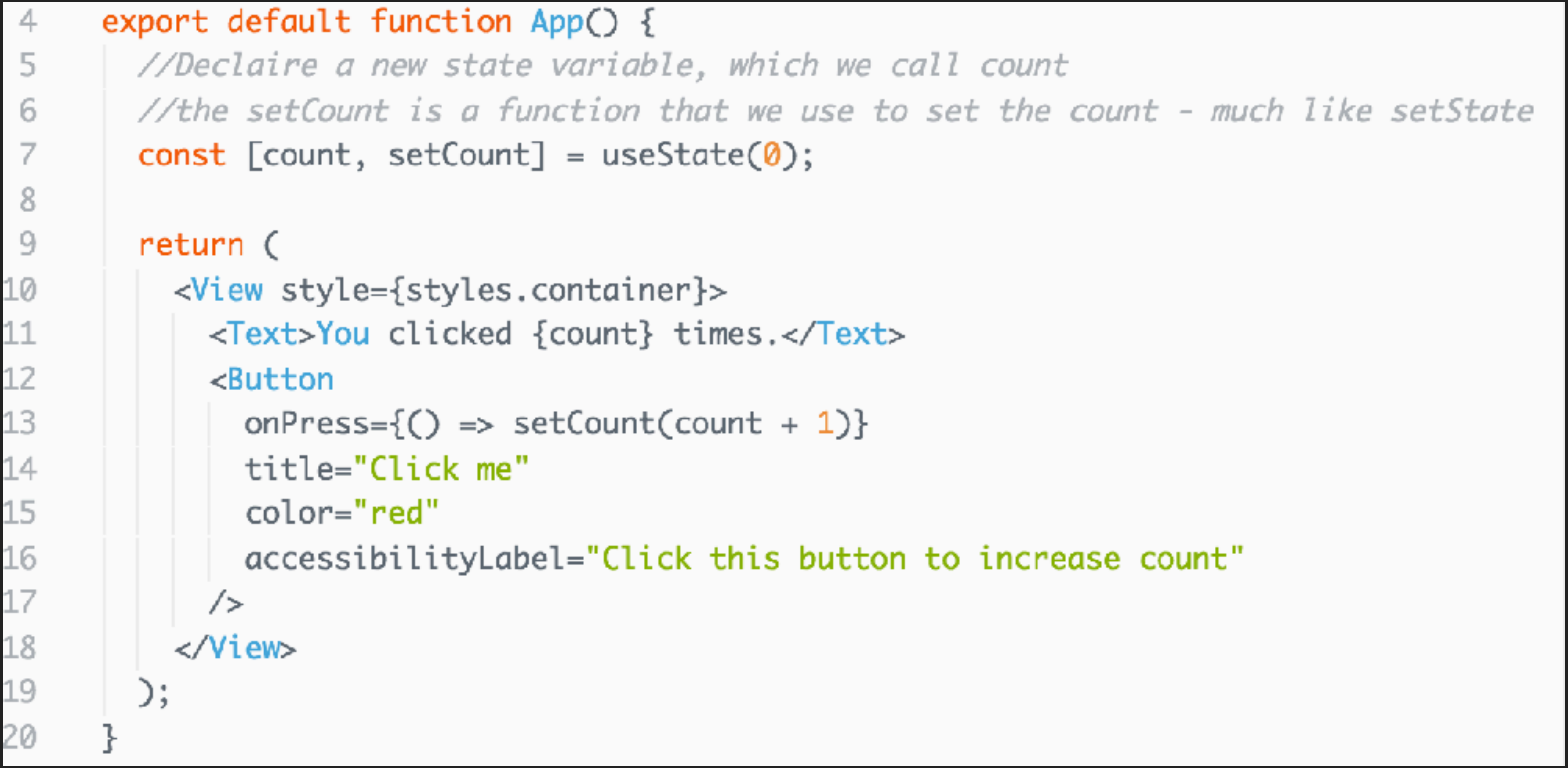
AWESOME! BUT WHAT DOES THAT MEAN?

- Lets see an example and explain what's going on. Lets create a simple counter application.
- First lets make sure we import the ability to use the useState hook



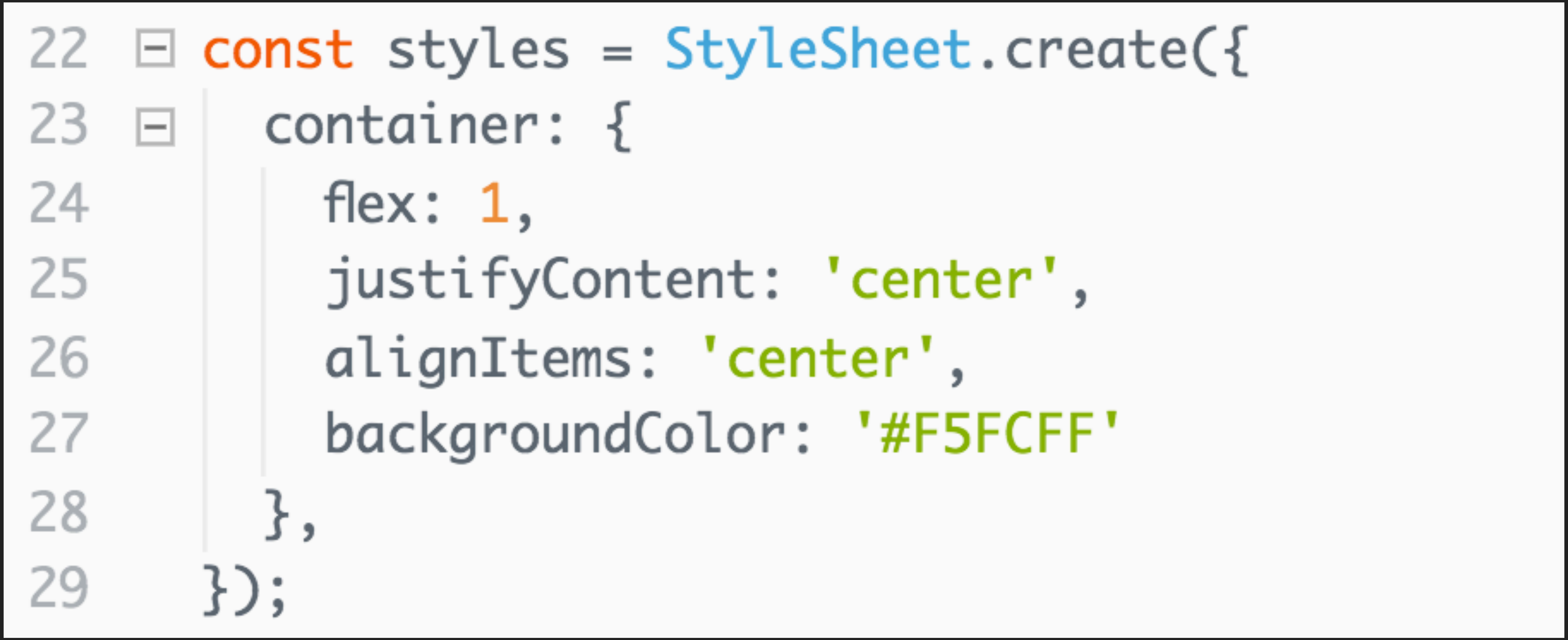
```
import React, { useState } from 'react';
import { StyleSheet, Text, View, Button } from 'react-native';
```

- Then lets build out our functional component.



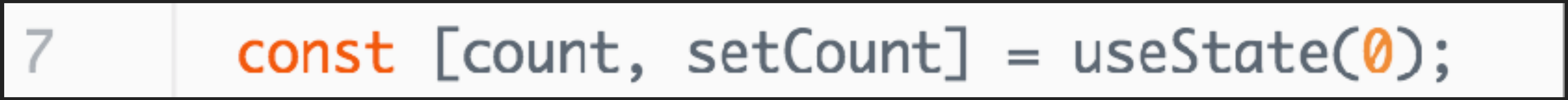
```
export default function App() {
  //Declaire a new state variable, which we call count
  //the setCount is a function that we use to set the count - much like setState
  const [count, setCount] = useState(0);

  return (
    <View style={styles.container}>
      <Text>You clicked {count} times.</Text>
      <Button
        onPress={() => setCount(count + 1)}
        title="Click me"
        color="red"
        accessibilityLabel="Click this button to increase count"
      />
    </View>
  );
}
```

- Then lets add the styling. Remember that this goes below our functional component code block.



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF'
  },
});
```

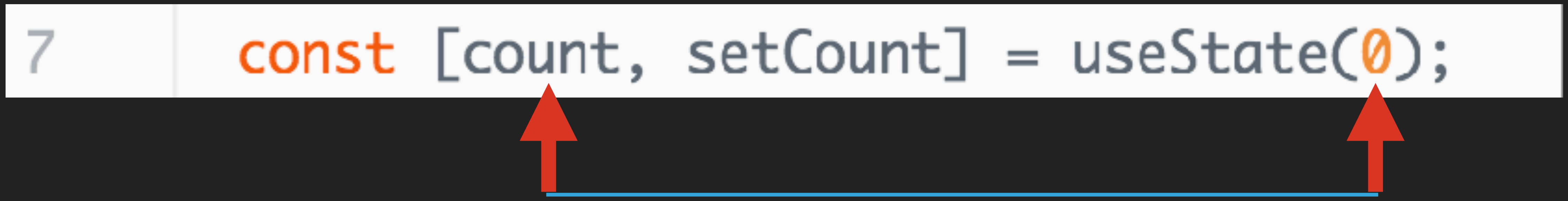
- Lets take a further look into how this useState hook is working.



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

- Here we are using the destructuring assignment syntax. This expression allows us to unpack values from the useState hook. Since the useState Hook returns two values - the current state (which in this case is count) and the function used to update the current state (which in this case is setCount)

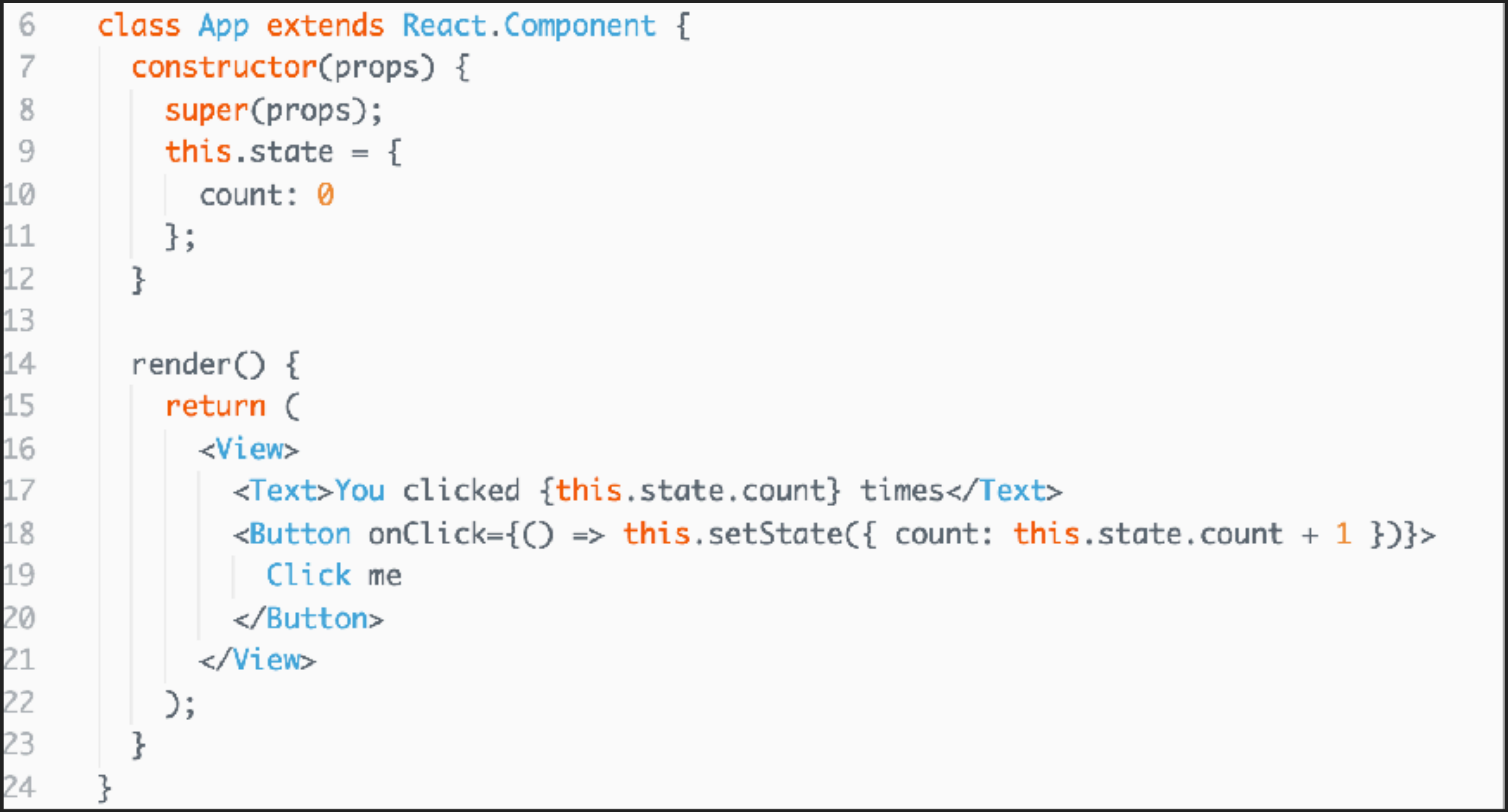
- Lets take a further look into how this useState hook is working.



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

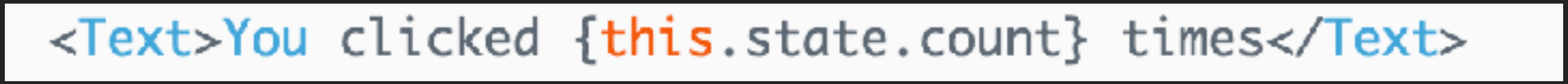
- The argument we're passing the useState Hook is what the count will be initialized to, which in this case is 0. Unlike with classes, the state doesn't have to be an object.

- Here is the equivalent of using a class and state.



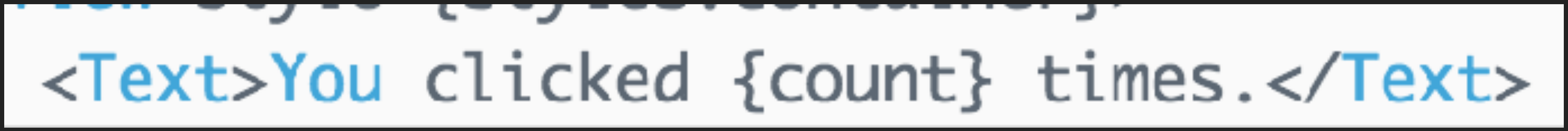
```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

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

- When we want to read the state in a class component we used this.state.count



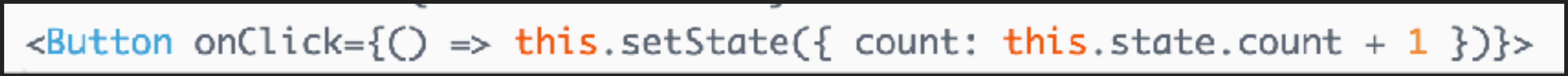
```
<Text>You clicked {this.state.count} times</Text>
```

- When we want to read the state in a functional component using Hooks we simply call the value. In our example it will be the variable count.



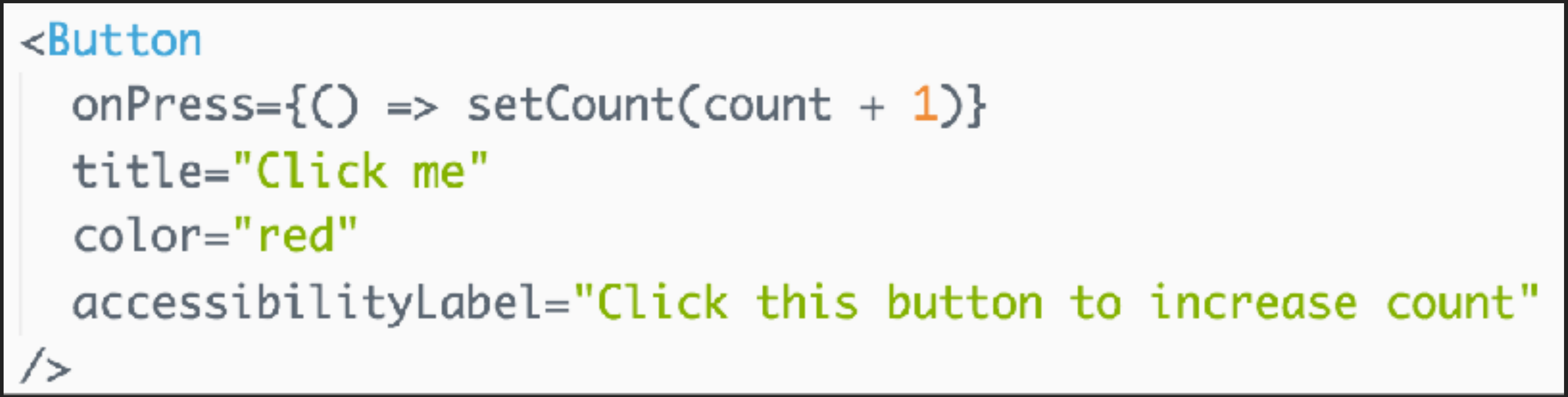
```
<Text>You clicked {count} times.</Text>
```

- When we want to update the state in a class component we use this.setState() to update the count.



```
<Button onClick={() => this.setState({ count: this.state.count + 1 })}>
```

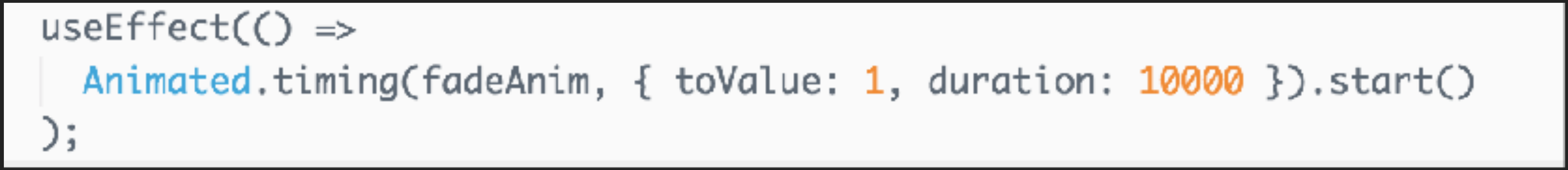
- When we want to update the state in a functional component using Hooks we simply call the function we assigned in our hook. In our example it will be the function setCount().



```
<Button
  onPress={() => setCount(count + 1)}
  title="Click me"
  color="red"
  accessibilityLabel="Click this button to increase count"
/>
```

- Now that we have a good understanding of the useState hook. Lets have a look at the useEffect hook.
- This Effect Hook allows us to perform side effects in functional components.
- You can think of the useEffect Hook as componentDidMount, componentDidUpdate, and componentWillUnmount combined into one.
- Data fetching, setting up a subscription, and manually changing the DOM in React components are all examples of side effects. Whether or not you're used to calling these operations 'side effects', you've used them in your components before.

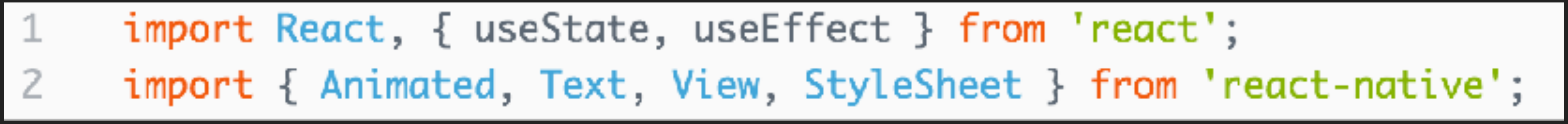
- So lets see this in action. Lets use some animations to show how to use this useEffect Hook.



```
useEffect(() =>
  Animated.timing(fadeAnim, { toValue: 1, duration: 10000 }).start()
);
```

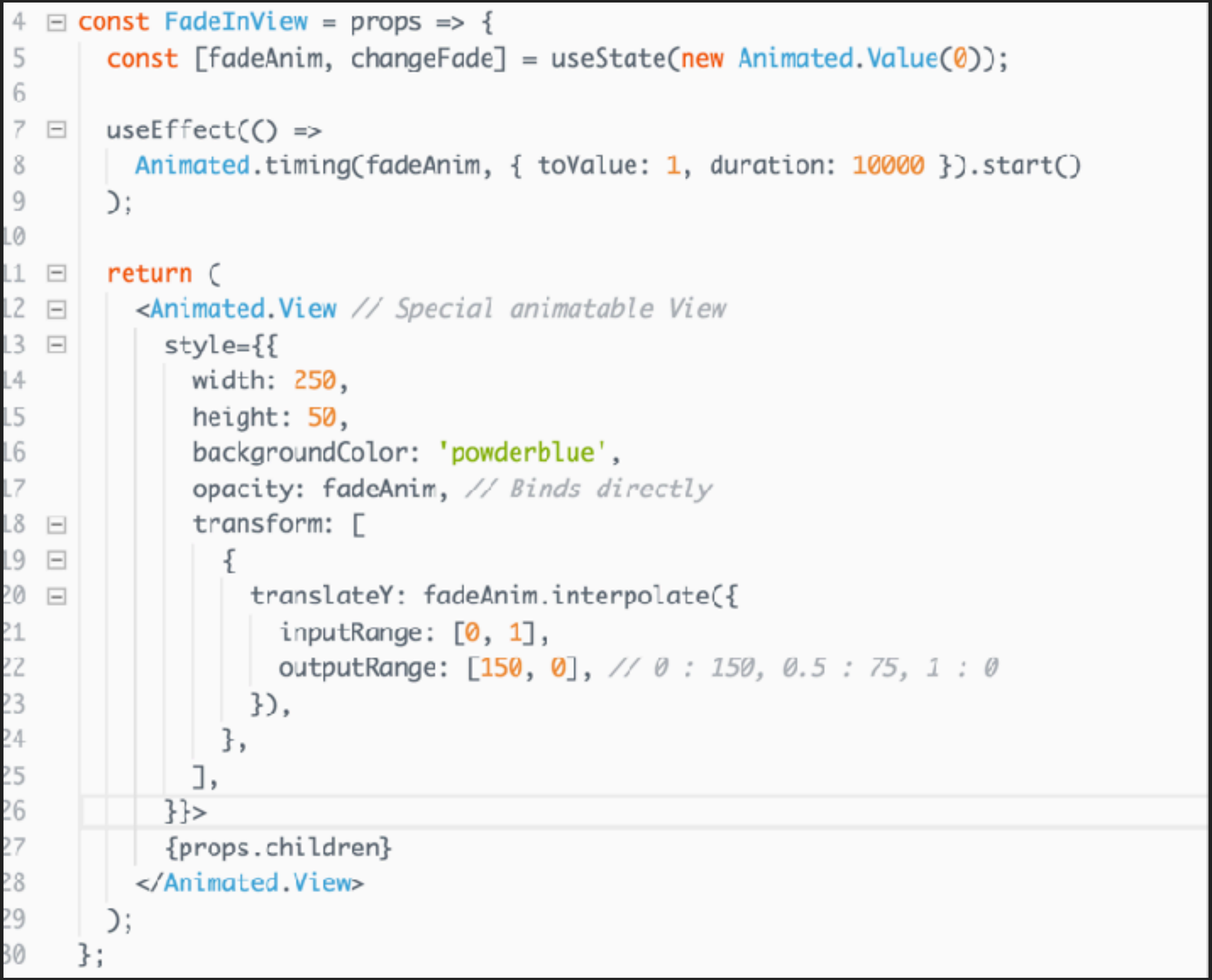
- In this example the useEffect Hook takes a function that will start a Animation timer that we start and runs for a duration of 10 seconds. We want to start the animation from 0 - 1 and have it take 10 seconds for that animation to fully finish.

- Lets start building this new animation app that uses both useState and useEffect Hooks.



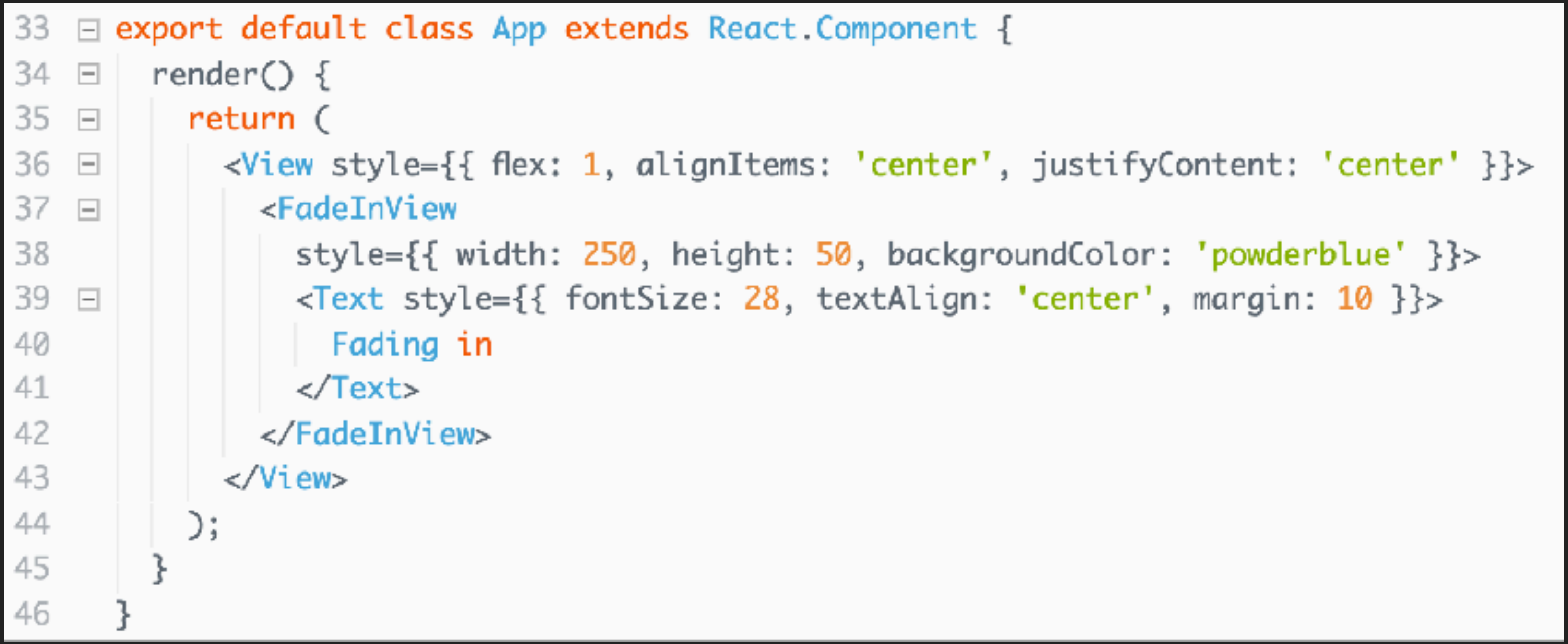
```
import React, { useState, useEffect } from 'react';
import { Animated, Text, View, StyleSheet } from 'react-native';
```

- Now lets create our FadeInView functional component and our APP class component



```
const FadeInView = props => {
  const [fadeAnim, changeFade] = useState(new Animated.Value(0));

  useEffect(() =>
    Animated.timing(fadeAnim, { toValue: 1, duration: 10000 }).start()
  );

  return (
    <Animated.View // Special animatable View
      style={{
        width: 250,
        height: 50,
        backgroundColor: 'powderblue',
        opacity: fadeAnim, // Binds directly
        transform: [
          {
            translateY: fadeAnim.interpolate({
              inputRange: [0, 1],
              outputRange: [150, 0], // 0 : 150, 0.5 : 75, 1 : 0
            }),
          },
        ],
      }}>
      {props.children}
    </Animated.View>
  );
};
```

- Lets put our App class component in the same file as well for this exercise



```
export default class App extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <FadeInView
          style={{ width: 250, height: 50, backgroundColor: 'powderblue' }}>
          <Text style={{ fontSize: 28, textAlign: 'center', margin: 10 }}>
            Fading in
          </Text>
        </FadeInView>
      </View>
    );
  }
}
```

- So if we think about what is going on in our program. We initialize our component by setting up the state. The App render() method executes, then our useEffect Hook executes and allows the animation to start.

- Try Practicing with the useEffect Hook by making an API call and populating a functional component's list with the data from the API.

- Resources:
- Animation in React Native
- React Hooks
- Array Destructuring