REACT HOOKS

REACT HOOKS

Introduction to the new addition of the React library.

WHAT HOOKS ARE?

WHAT HOOKS ARE?

WHAT HOOKS CAN OFFER?

WHAT HOOKS ARE?

WHAT HOOKS CAN OFFER?

- Adds local state and lifecycle-hooks to functional components

WHAT HOOKS ARE?

WHAT HOOKS CAN OFFER?

- Adds local state and lifecycle-hooks to functional components
- Use all/most React features without using React built-in classes

WHAT HOOKS ARE?

WHAT HOOKS CAN OFFER?

- Adds local state and lifecycle-hooks to functional components
- Use all/most React features without using React built-in classes
- Allow you to reuse stateful logic without changing your component hierarchy

WHAT HOOKS ARE?

WHAT HOOKS CAN OFFER?

- Adds local state and lifecycle-hooks to functional components
- Use all/most React features without using React built-in classes
- Allow you to reuse stateful logic without changing your component hierarchy
- Let you split one component into smaller functions based on what pieces are related, rather than forcing a split based on lifecycle methods.

AND FOR WHAT REASONS HOOKS WHERE INVENTED?

AND FOR WHAT REASONS HOOKS WHERE INVENTED?

- Big problem with “wrapper hell” of components surrounded by layers of providers, consumers, higher-order components, render props, and other abstractions.

AND FOR WHAT REASONS HOOKS WHERE INVENTED?

- Big problem with “wrapper hell” of components surrounded by layers of providers, consumers, higher-order components, render props, and other abstractions.
- Complex components become hard to understand

AND FOR WHAT REASONS HOOKS WHERE INVENTED?

- Big problem with “wrapper hell” of components surrounded by layers of providers, consumers, higher-order components, render props, and other abstractions.
- Complex components become hard to understand
- Classes confuse both people and machines and they can be a large barrier to learning React.

FUNCTIONS VS REACT CLASSES

FUNCTIONS VS REACT CLASSES

- Functional components are much easier to read and test because they are plain JavaScript functions without state or lifecycle-hooks.

FUNCTIONS VS REACT CLASSES

- Functional components are much easier to read and test because they are plain JavaScript functions without state or lifecycle-hooks.
- It is easier to separate container and presentational components because you need to think more about your component’s state if you don’t have access to setState() in your component.

FUNCTIONS VS REACT CLASSES

- Functional components are much easier to read and test because they are plain JavaScript functions without state or lifecycle-hooks.
- It is easier to separate container and presentational components because you need to think more about your component’s state if you don’t have access to setState() in your component.
- File size is reduced if you are using functional components over classes.

HOW TO USE HOOKS?

HOW TO USE HOOKS?

- Hooks are now available with the release of React v16.8.0

HOW TO USE HOOKS?

- Hooks are now available with the release of React v16.8.0
- Only Call Hooks from React Functions

HOW TO USE HOOKS?

- Hooks are now available with the release of React v16.8.0
- Only Call Hooks from React Functions
- Only Call Hooks at the Top Level

USESTATE

USESTATE

- Use React State within functional components

USESTATE

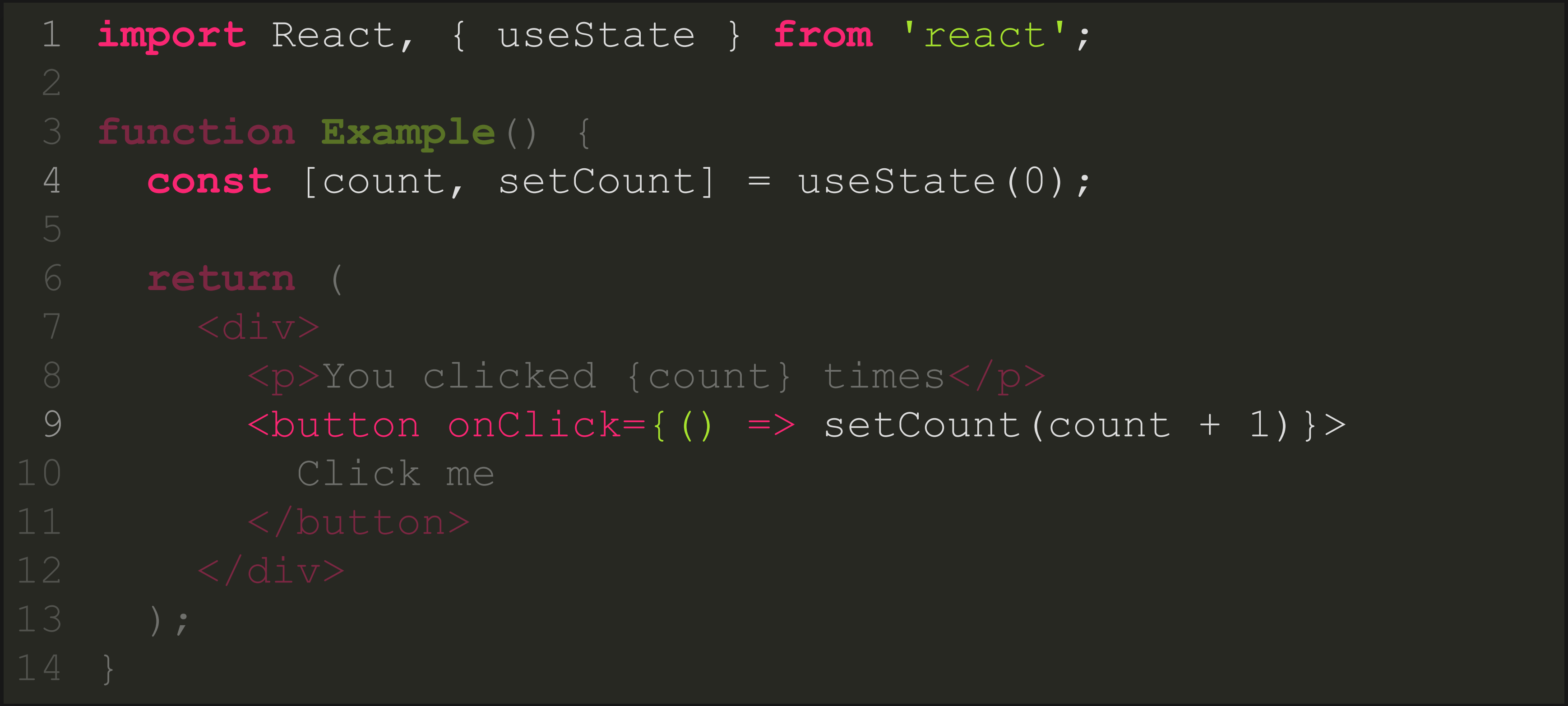
- Use React State within functional components
- Updating a state variable always replaces it instead of merging it

USESTATE

- Use React State within functional components
- Updating a state variable always replaces it instead of merging it
- React will remember its current value between re-renders, and provide the most recent one to our function

LET'S LOOK AT AN EXAMPLE WITH USESTATE HOOK

LET'S LOOK AT AN EXAMPLE WITH USESTATE HOOK

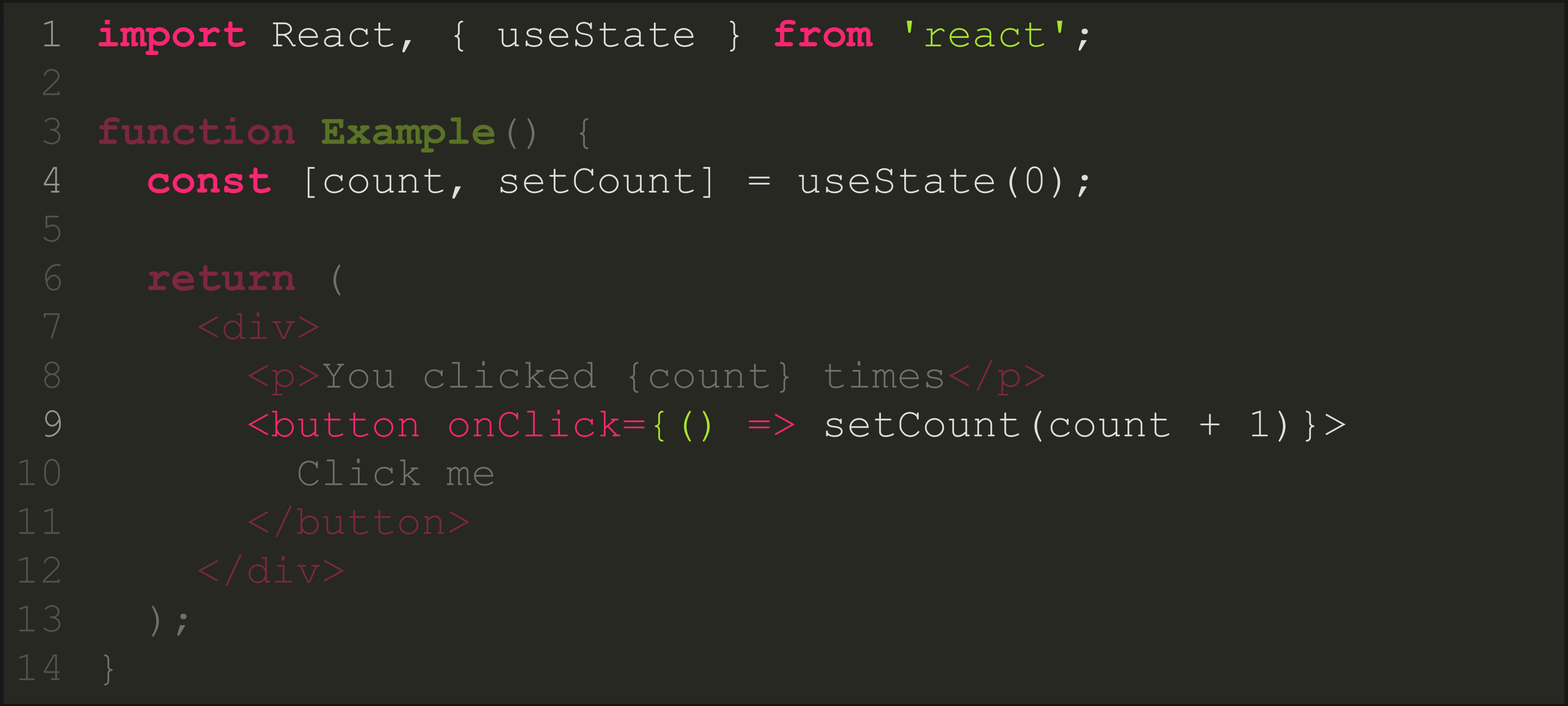


```
import React, { useState } from 'react';

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

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

LET'S LOOK AT AN EXAMPLE WITH USESTATE HOOK



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

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

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

It is quite simple and elegant

USEEFFECT

USEEFFECT

- useEffect Hook lets you perform side effects in function components

USEEFFECT

- useEffect Hook lets you perform side effects in function components
- Data fetching, setting up a subscription, and manually changing the DOM in React components are all examples of side effects

USEEFFECT

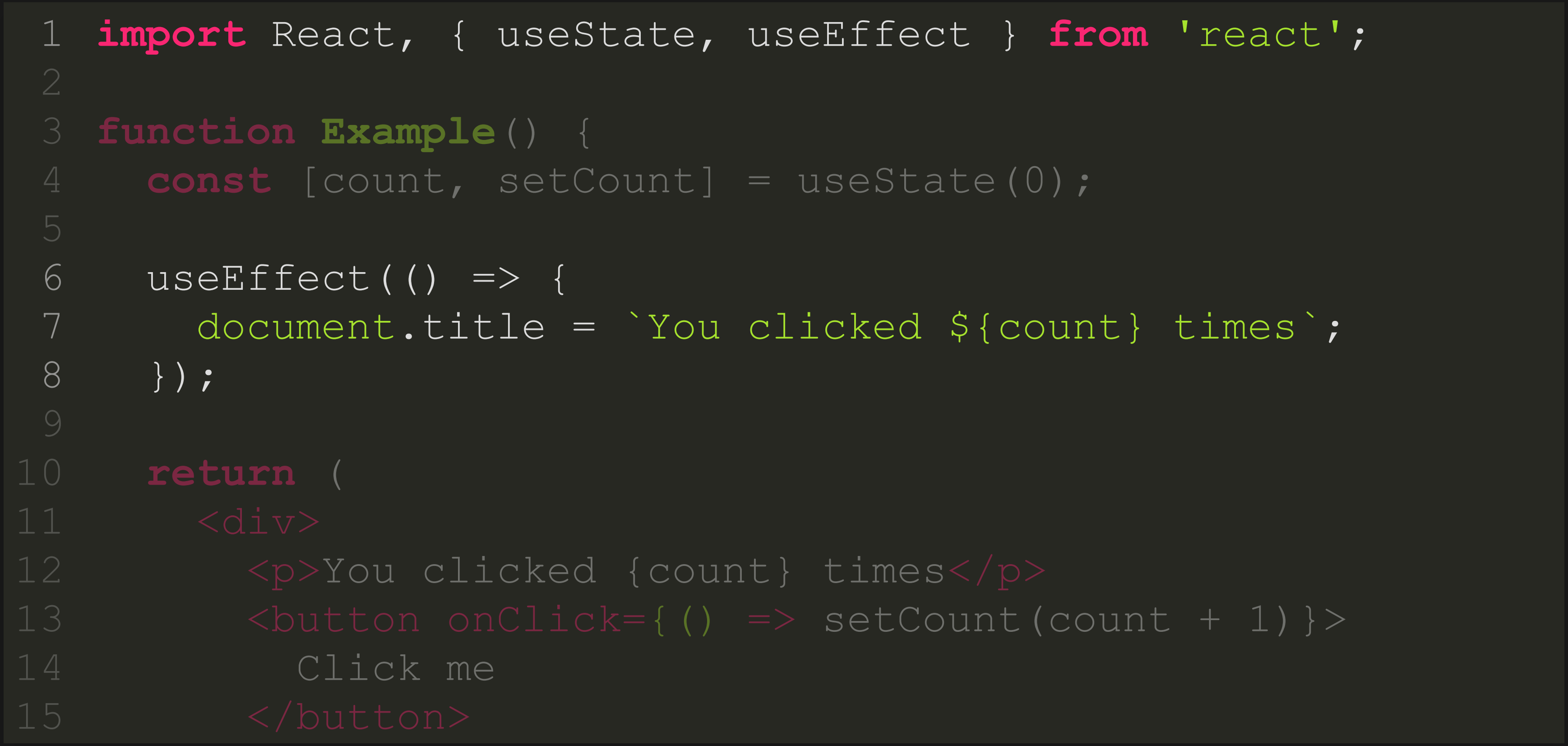
- useEffect Hook lets you perform side effects in function components
- Data fetching, setting up a subscription, and manually changing the DOM in React components are all examples of side effects
- Think of useEffect Hook as componentDidMount, componentDidUpdate, and componentWillUnmount combined

USEEFFECT

- useEffect Hook lets you perform side effects in function components
- Data fetching, setting up a subscription, and manually changing the DOM in React components are all examples of side effects
- Think of useEffect Hook as componentDidMount, componentDidUpdate, and componentWillUnmount combined
- You can use multiple useEffect() within a component.

NOW LET'S ADD USEEFFECT HOOK TO THE LAST EXAMPLE

NOW LET'S ADD USEEFFECT HOOK TO THE LAST 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>
```

CREATE YOUR OWN HOOKS

CREATE YOUR OWN HOOKS

- Components and Hooks are just functions

CREATE YOUR OWN HOOKS

- Components and Hooks are just functions
- If you want to share reusable logic, just extract it to a separate function

CREATE YOUR OWN HOOKS

- Components and Hooks are just functions
- If you want to share reusable logic, just extract it to a separate function
- Hooks always start with a "use" keyword for linting purposes (useState, useEffect, useReducer etc.)

CREATE YOUR OWN HOOKS

- Components and Hooks are just functions
- If you want to share reusable logic, just extract it to a separate function
- Hooks always start with a "use" keyword for linting purposes (useState, useEffect, useReducer etc.)
- Using useState and useEffect you can already create useful custom hooks

SHOULD I SWITCH TO HOOKS?

SHOULD I SWITCH TO HOOKS?

- You don't need to rewrite all of your components with React Hooks

SHOULD I SWITCH TO HOOKS?

- You don't need to rewrite all of your components with React Hooks
- You can try Hooks in a few components without rewriting any existing code

SHOULD I SWITCH TO HOOKS?

- You don't need to rewrite all of your components with React Hooks
- You can try Hooks in a few components without rewriting any existing code
- Hooks are 100% backwards-compatible

SHOULD I SWITCH TO HOOKS?

- You don't need to rewrite all of your components with React Hooks
- You can try Hooks in a few components without rewriting any existing code
- Hooks are 100% backwards-compatible
- React Classes will still be supported in the future and there are no plans to remove them

SHOULD I SWITCH TO HOOKS?

- You don't need to rewrite all of your components with React Hooks
- You can try Hooks in a few components without rewriting any existing code
- Hooks are 100% backwards-compatible
- React Classes will still be supported in the future and there are no plans to remove them
- Best practices are still a "work in progress"

AND…

AND...

THAT'S ALL! THANK YOU FOR YOUR ATTENTION :)