REACT HOOKS

Prof. Andrew Sheehan

Boston University/MET

Computer Science Dept.

HOOKS ARE COMPONENTS

WITHOUT CLASSES

TD;DR

PLEASE TAKE NOTE

You cannot use hooks with ES6 classes.

They do not work inside of your older, class-based components.

Hooks are meant to be used with components that were built with functions.

Sourced 25-Oct-2022: https://bit.ly/3TATAH3

Basic Hooks

useState
useEffect
useContext

Additional Hooks

useReducer
useCallback
useMemo
useRef
useImperativeHandle
useLayoutEffect
useDebugValue
useDeferredValue
useTransition
useId

HOOK REFERENCE

https://reactjs.org/docs/hooks-reference.html

WHY DID DAN ABRAMOV NEED TO CREATE HOOKS?

- *It's hard to reuse stateful logic between components*
- You wind up with '*wrapper hell*': components overlapping components – just to manage your state
- Hooks allow you to reuse stateful logic without changing your component hierarchy

HOOKS: MORE ABOUT: *WHAT THEY ARE*

Hooks are functions that let you “hook into” React state and lifecycle features from function components.

Functional Components

```
function MyDate() {
  const rightNow = new Date();

   return (
      <h1>{ rightNow }</h1>
   );
}


  function App() {
     return (
       <header>
          <i>The current date and time: </i>
          <MyDate/>
       </header>
     );
  }
```

Hooks started with version 16.8 (19-Feb-2019)

When you use a capital letter, you will know it's a React component.

Component must always start with a capital letter, while HTML tags must be lowercase

CAPITAL LETTERS

Sourced on 25-Oct-2022: "https://bit.ly/3N6dV4F"

JSX follows a very strict syntax

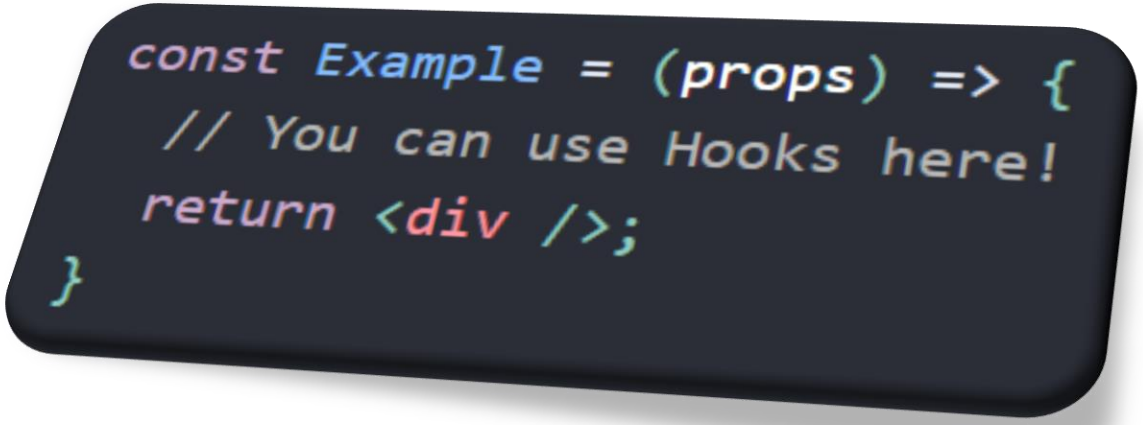
known as .xhtml in the past.

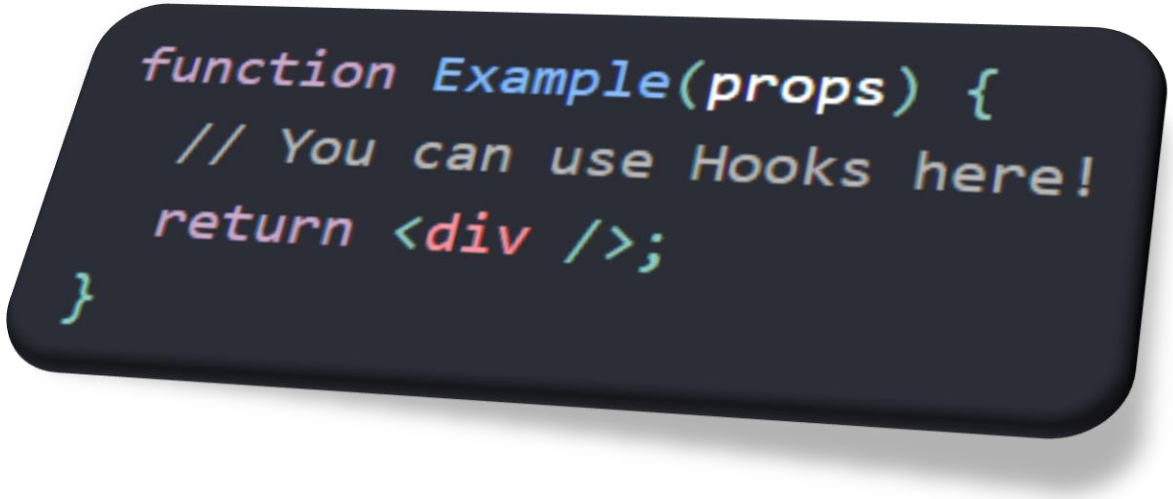
```
// examples
<br/> || <img/> || <link/>

// must close your end tags
<li>a list item</li>
```

REACT
JSX

FUNCTIONAL COMPONENTS

Should look very familiar to you.





useState(): a built-in React hook.

It creates local state for your component.

a) A variable you need (state)
b) A function to update it.

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

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

```
useState( initial value );
```

The initial state can be any valid type

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
  // ...
}
```

AS NEEDED..

useState();

Preserves state between function calls.

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

We declare a state variable called count, and set it to 0

React will remember its current value between re-renders, and provide the most recent one to our function

If we want to update the current count, we can call setCount(1234)

USING STATE

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

```
function showClickAttempts() {
  return (
    <span>{count}</span>
  );
}
```

```
function updateCount() {
  return (
    <div onclick={() => setCount(count++)}>Run</div>
  );
}
```

UPDATING STATE

STATE:
PUTTING IT ALL TOGETHER

```
1:  import React, { useState } from 'react';
2:
3:  function Example() {
4:    const [count, setCount] = useState(0);
5:
6:    return (
7:      <div>
8:        <p>You clicked {count} times</p>
9:        <button onClick={() => setCount(count + 1)}>
10:         Click me
11:       </button>
12:     </div>
13:   );
14: }
```

PROPS AND HOOKS

Your functional components can still use the props , if you declare it.

```
const MyButton = (props) => {
    return <Button isActive={props.isActive}>Click me!</Button>;
};
```

```
const MyCard = () => {
    return (
        <Wrapper>
            The button should be active
            <MyButton isActive={true} />
        </Wrapper>
    );
```

useEffect()

The useEffect() hook runs on the initial and subsequent renderings

USEEFFECT() REPLACES THESE LIFECYCLE METHODS

- componentDidMount
- componentDidUpdate
- componentWillUnmount

USEEFFECT()

Always remember to place your calls to useEffect() inside your component definition

It will have access to your state and props (due to JS Closures)

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

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

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

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

WHY CALL IT AN EFFECT?

Data fetching, configuring a subscription and changing the DOM in React components are all examples of side effects

If you're familiar with React class lifecycle methods, you can think of `useEffect` Hook as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` combined.

There are two common kinds of side effects: those that don't require cleanup and those that do

USE CASE:

NO CLEAN UP REQUIRED

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

Having multiple useEffect()

It is totally acceptable using useEffect() as much as you need in your Components

Having multiple useEffect()

If the data is related, then its normal to have them manipulated with one useEffect()

PLACEMENT

useEffect() functions will be executed in the same order as they are defined..

YOU CAN CONFIGURE EFFECTS TO EXECUTE ONLY IF A SPECIFIC STATE OR PROP IS UPDATED

Array of props or state

```
function SimpleUseEffect() {

  let [userCount, setUserCount] = useState(0);
  let [simpleCount, setSimpleCount] = useState(0);

  useEffect(() => {
    alert("Component User Count Updated...");
  }, [userCount]);
  useEffect(() => {
    alert("Component Simple Count Updated");
  }, [simpleCount]);

  return (
    <div>
      <b>User Count: {userCount}</b>
      <b>Simple Count: {simpleCount}</b>
      <input type="button" onClick={() => setUserCount(userCount + 1)}
value="Add Employee" />
      <input type="button" onClick={() => setSimpleCount(simpleCount +
1)} value="Update Simple Count" />
    </div>
  )
}
```

If userCount updates, useEffect() will execute, otherwise it will not

```
useEffect(() => {
    // your code here...
}, []); // pass in empty array to indicate it should only run 1-time.
```

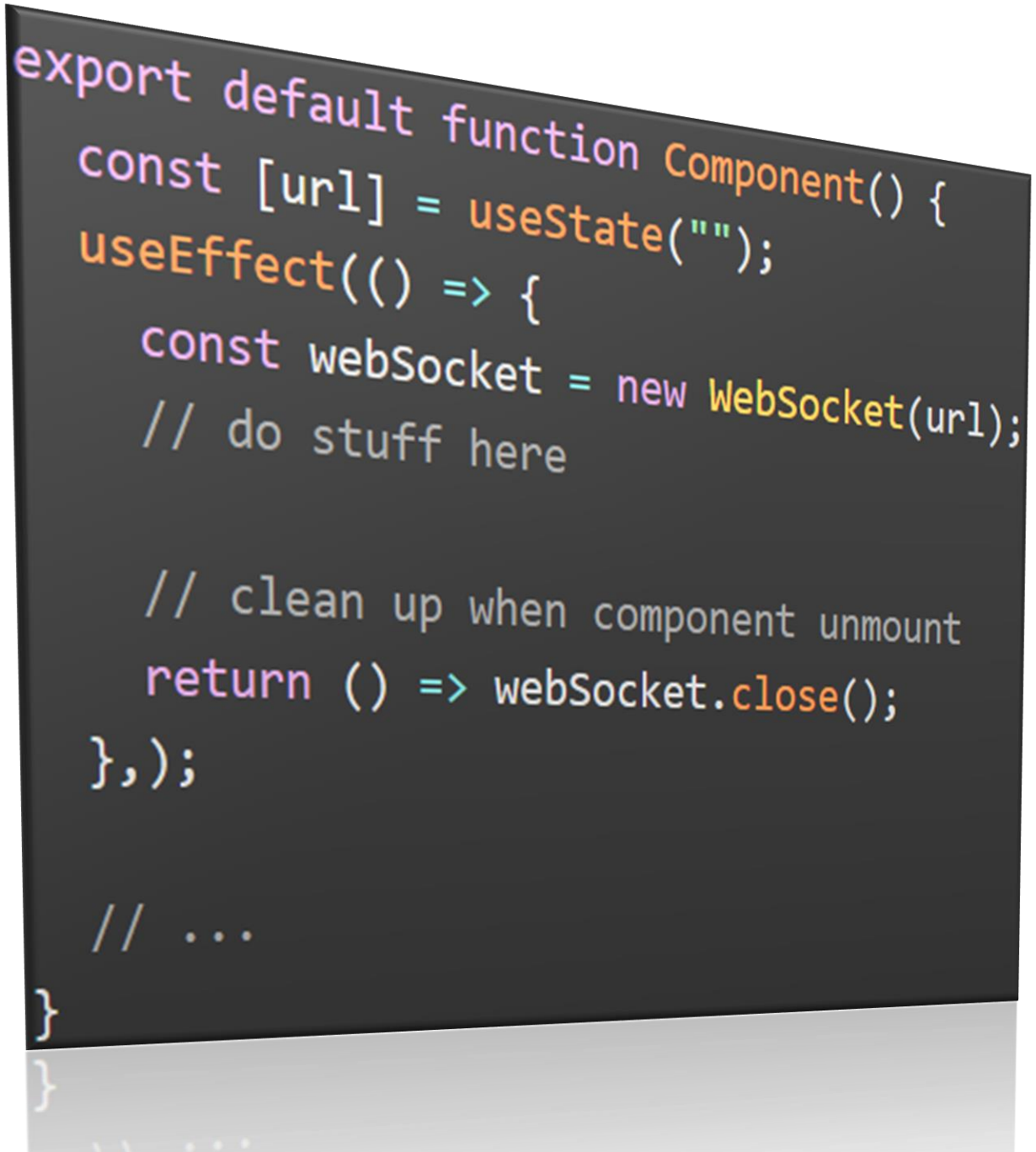
USEEFFECT(): JUST ONCE

CLEANUP

Returning a function from useEffect() will cancel data fetches or clear out memory allocations

useEffect();

Close the socket when done.
Clean up.



```
export default function Component() {
  const [url] = useState("");
  useEffect(() => {
    const webSocket = new WebSocket(url);
    // do stuff here

    // clean up when component unmount
    return () => webSocket.close();
  },);

  // ...
}
```