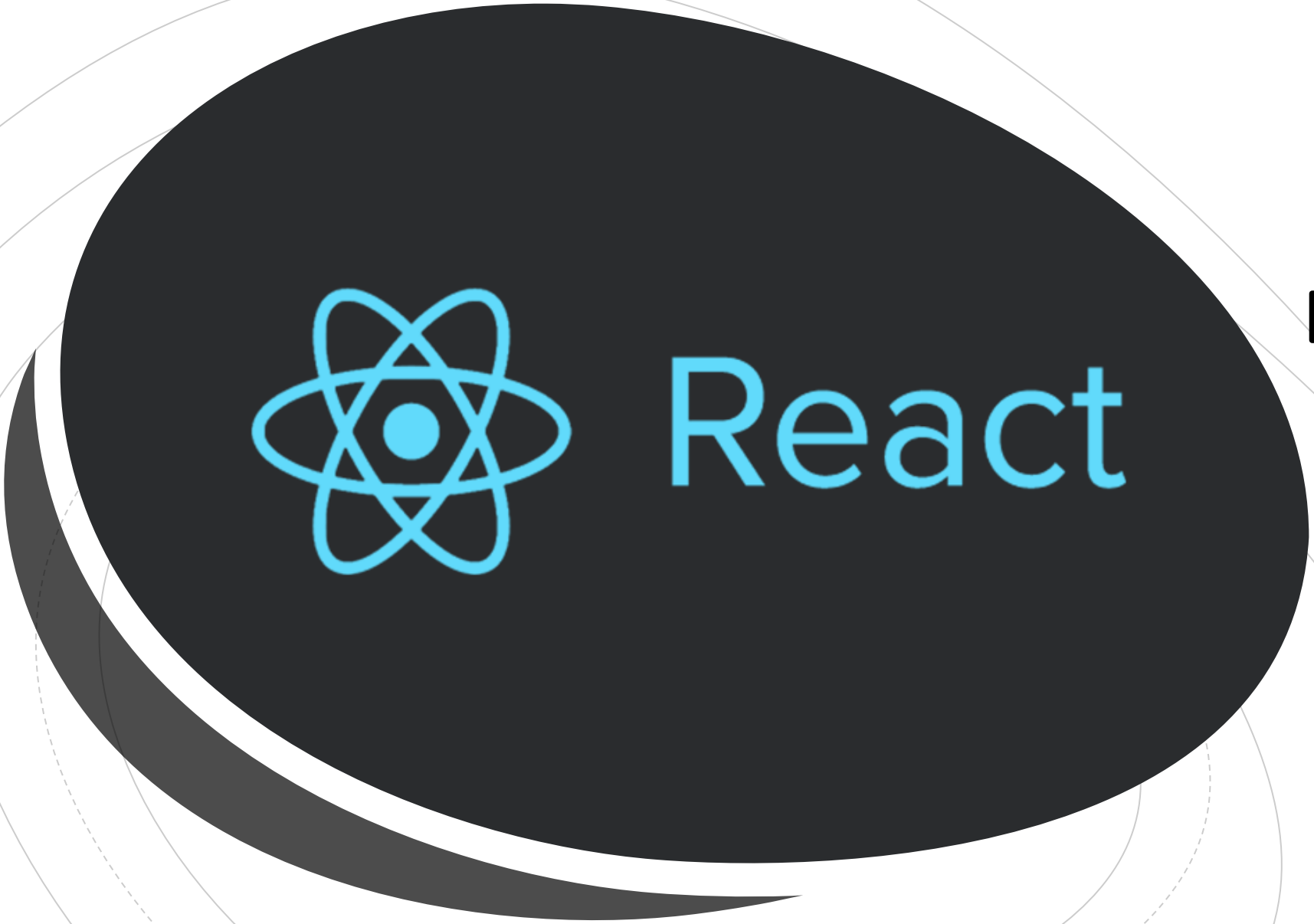

Full Stack Development - I

COMP 3123

Introduction to Hooks

- When React 16.8 was released officially in early February 2019, it shipped with an additional API that lets you use state and other features in React without writing a class.
- This additional API is called **Hooks** and they're becoming popular in the React ecosystem, from open-sourced projects to being used in production applications.

Web Reference Link

Introduction to Hooks

- React Hooks are in-built functions that allow React developers to use **state** and **lifecycle** methods inside **functional components**.
- Hooks allow developers to use state in functional components but under the hood, Hooks are much more powerful than that.
- They allow React Developers to enjoy the following benefits:
  - Improved code reuse;
  - Better code composition;
  - Better defaults;
  - Sharing non-visual logic with the use of custom hooks;
  - Flexibility in moving up and down the components tree.

Web Reference Link

APIs for the built-in Hooks in React

Basic Hooks

- useState
- useEffect
- useContext

Additional Hooks

- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

Web Reference Link

Advantages of React Hooks

- With Hooks, you can **reuse logic** between your components without changing their architecture or structure.
- When components become **larger** and carry out many operations, it becomes **difficult** to understand in the long run.
- Hooks solve this by allowing you **separate** a particular single component into various smaller functions based upon what pieces of this separated component are related (such as setting up a subscription or fetching data), rather than having to force a split based on lifecycle methods.
- React Hooks solves this problem by allowing developers to use the best of React features without having to use classes.

Rules of Hooks

Hooks are JavaScript functions, but they impose two additional rules:

- Only call Hooks at the **top level**. Don’t call Hooks inside **loops**, **conditions**, or **nested functions**.
- Only call Hooks from **React function components**. Don’t call Hooks from regular JavaScript functions. (There is just one other valid place to call Hooks — your own custom Hooks.)

useState

- The useState() hook allows React developers to update, handle and manipulate state inside functional components without needing to convert it to a class component.

useState

- It declares a “**state variable**”.
- This is a way to “**preserve**” some values between the function calls — useState is a new way to use the exact same capabilities that this.state provides in a class.
- What do we pass to useState as an **argument**? The only argument to the useState() Hook is the **initial state**.
- Unlike with classes, the state **doesn’t have to be an object**.
- We can keep a **number** or a **string** if that’s all we need.
- What does useState **return**? It returns a **pair of values**: the **current state** and a **function that updates it**.
- This is why we write **const [count, setCount] = useState()**. This is similar to this.state.count and this.setState in a class, except you get them in a pair.

```
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>
  );
}
```

Example- useState

Web Reference Link

useRef

- This hook allows us to **access a DOM element** imperatively.
- It allows **accessing** a DOM element and **storing** mutable information in the state without triggering a **re-render** of that component.
- In vanilla JavaScript, we work with DOM elements by calling **document.getElementById(),** passing in the id of a given element.
- With **refs** in React, we no longer need to do this.
- The **refs attribute** will literally reference that element and give us access to it’s methods.
- The **.current** property is what will be initialized and where the most recent information is stored.

```
import React, { useState, useRef } from 'react’
const Counter = () =>
{
        const [count, setCount] = useState(0)
        const counterEl = useRef(null)
        const increment = () => {
                setCount(count + 1) console.log(counterEl)
        }
        return (
        <>
        Count: <span ref={counterEl}>{count}</span>
         <button onClick={increment}>+</button> </> )
}
```

Example- useRef

Web Reference Link

```
function Form() {
  const textInput = useRef();
  const focusTextInput = () => textInput.current.focus();

  return (
    <input type="text" ref={textInput} />
    <button onClick={focusTextInput}>Focus the text input</button>
  );
}
```

Example- useRef

Web Reference Link

useEffect

- The Effect Hook, useEffect, adds the ability to perform side effects from a function component.
- It serves the same purpose as **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount** in React classes, but unified into a single API.

What does useEffect do?

- By using this Hook, you tell React that your component needs to do **something after render**.
- React will remember the function you passed (we'll refer to it as our "effect"), and call it later after performing the DOM updates.
- In this effect, we set the document title, but we could also perform data fetching or call some other imperative API.

useEffect

Why is useEffect called inside a component?

- Placing useEffect inside the component lets us access the count state variable (or any props) right from the effect.
- Hooks embrace JavaScript closures and avoid introducing React-specific APIs where JavaScript already provides a solution.

Does useEffect run after every render?

- Yes! By default, it runs both after the first render and after every update.
- Instead of thinking in terms of “mounting” and “updating”, you might find it easier to think that effects happen “after render”.
- React guarantees the DOM has been updated by the time it runs the effects.

```
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>
  );
}
```

Example-useEffect

Web Reference Link

```jsx
import React, {useState, useEffect} from 'react';
function App() {
    //Define State
    const [name, setName] = useState({firstName: 'name', surname: 'surname'});
    const [title, setTitle] = useState('BIO');

    //Call the use effect hook
    useEffect(() => {
      setName({FirstName: 'Shedrack', surname: 'Akintayo'})
    }, [])//pass in an empty array as a second argument

    return(
        <div>
            <h1>Title: {title}</h1>
            <h3>Name: {name.firstName}</h3>
            <h3>Surname: {name.surname}</h3>
        </div>
    );
};
export default App
```

Example-useEffect

Other React Hooks Available

`useCallback`	This hook returns a callback function that is memoized and that only changes if one dependency in the dependency tree changes.
`useMemo`	This hook returns a memoized value, you can pass in a “create” function and also an array of dependencies. The value it returns will only use the memoized value again if one of the dependencies in the dependency tree changes.
`useRef`	This hook returns a mutable ref object whose `.current` property is initialized to the passed argument (`initialValue`). The returned object will be available for the full lifetime of the component.
`useImperativeHandle`	This hook is used for customizing the instance value that is made available for parent components when using refs in React.
`useLayoutEffect`	This hook similar to the `useEffect` hook, however, it fires synchronously after all DOM mutations. It also renders in the same way as `componentDidUpdate` and `componentDidMount` .
`useDebugValue`	This hook can be used to display a label for custom hooks in the React Dev Tools. It is very useful for debugging with the React Dev Tools.

Thank You