**HOOKS [src : youtube/codeevolution]**

Hooks are just functions.

Hooks must be called only at the top level. Not inside loops/nested functions.

Hooks should be used only inside React functional components.  
   
 usestate hook

1.usestate hook

Why ?   
You can use **state** in functional components also.

How?

import React, { useState } from 'react'

import ReactDOM from 'react-dom'

function App() {

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

    let btnStyle = {

        width:'100px',

        height:'25px'

    }

    return (

        <div>

            <p>You have clicked {count} times</p>

            <button onClick={ () => setCount(count+1) }   
 style={btnStyle}>  
 Click</button>

        </div>

    )

}

ReactDOM.render(<App />, document.getElementById('root'));

2.usestate hook with previous state

What ? how to set state based on the previous state value

The setstate(count+1)

Why ? If you want to update state based on the previous state value.

How ?  
  
  
  
The problem : Even though Inc5 is clicked and you want to increment 5 times : the end result will have the count incremented by only 1 – no use even if the ***setCount(count+1)*** code is executed 5 times

function App() {

    const initialCount = 0;

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

    let btnStyle = {   width:'100px',  height:'25px'    }

    const Inc5 = ()=>{

        for (let index = 0; index < 5; index++) {

            setCount(count+1)

            // This setCount will be operating

// in the previous 'count' value.

            // 5 times on the same old value

        }

        //Not sure but : After this function is executed :   
 // The value of oldvalue incremented by 1 is displayed.

    }

    return (

        <div>

            <p>You have clicked {count} times</p>

            <button onClick={ () =>setCount(count+1) }         
 style={btnStyle}>  
 Increment</button>

            <button onClick={ () => setCount(count1)}  
        style={btnStyle}>  
 Decrement</button>

            <button onClick={ () => setCount(initialCount) }   
  style={btnStyle}>  
 Reset</button>

            {/\* <button onClick={ () => {setCount(count + 5)} }  style={btnStyle}>Incrementby5</button>

            The above line will work \*/}

            <button onClick={Inc5}    
 style={btnStyle}>  
 Inc5</button>

        </div>

    )

}

ReactDOM.render(<App />, document.getElementById('root'));

The Solution : Should be adopted for the increment and decrement buttons as well

import React, { useState } from 'react'

import ReactDOM from 'react-dom'

function App() {

…

    const Inc5 = ()=>{

        for (let index = 0; index < 5; index++) {

            setCount((prevCount)=>{

                return prevCount+1;

            })

        }

    }  
…

3.usestate hook with object {} as a state variable

Why ? : useState hook setter function does not automatically merge and update the object.

From the below code, when the function setName is called with an object the state object is overwritten

How ?

Problem :

const App = () => {

    const [name,setName] =   
 useState({firstName : '',lastName : ''});

    return (

        <div>

            <input type='text' value={name.firstName}

                onChange={(e)=>{  
 setName({firstName:e.target.value})}  
 }  
 />  
   
 <br/>

            <input type='text' value={name.lastName}

                onChange={(e)=>{  
 setName({lastName:e.target.value})}  
 }  
 />

            <h2>Firstname : {name.firstName}</h2>

            <h2>LastName : {name.lastName}</h2>

            {JSON.stringify(name)}

        </div>

    )

}

Solution:

const App = () => {

    const [name,setName] =   
 useState({firstName : '',lastName : ''});

    return (

        <div>

            <input type='text' value={name.firstName}

                onChange={(e)=>  
 setName({...name,firstName:e.target.value})  
 }   
 />  
 <br/>

            <input type='text' value={name.lastName}

                onChange={(e)=>  
 setName({...name,lastName:e.target.value})  
 }  
 />

            <h2>Firstname : {name.firstName}</h2>

            <h2>LastName : {name.lastName}</h2>

            {JSON.stringify(name)}

        </div>

    )

}

4.usestate hook with array of objects as a state variable

const App = () => {

    const [items,setItems] = useState([]);

    const AddItem = (e)=>{

        setItems([...items,

            {  
 id:items.length,  
 value:Math.floor(Math.random() \* 10) + 1  
 }

         ])

    }

    return (

        <div>

            <button onClick={AddItem}>  
 Add Items</button>

            <ul>

                {  
 items.map(item =>   
 (<li key={item.id}>{item.value}</li>))  
 }

            </ul>

        </div>

    )

}

useEffect hook

**Why** ? Setting up subscriptions/timer,updating DOM,fetching data from an api endpoint.

Initial render : componentDidMount – executed only once

When state changes upon button click : component must be updated : componentDidUpdate.

componentWillUnmount : suppose you have set up a timer in the componentDidMount you want to destroy it in the componentWillUnmount.

The code will look as shown below.

class App extends React.Component{

    constructor(props){

        super(props);

        this.state={

            count:0

        }

    }

    componentDidMount(){

        document.title = `${this.state.count} - times clicked`;

        this.timer = setInterval(() => {

            console.log('Logging this every two seconds');

        }, 2000);

    }

    componentDidUpdate(){

        document.title = `${this.state.count} - times clicked`

    }

    componentWillUnmount(){

        clearInterval(this.timer)

    }

    onClickHandler = (e)=>{

            this.setState({

                count : this.state.count  + 1

            })

    }

    render(){

        return(

            <div>

                <button onClick={this.onClickHandler}>Click</button>

            </div>

        )

    }

}

Whats wrong with the above code :

* componentDidMount and componentDidUpdate has the same code separated in two functions
* related code is also in two different function, code related to timer is in componentDidMount and componentWillUnmount.
* Unrelated code is put together : componentDidMount has code related to updating the DOM and has also code that sets up a timer.

What you can achieve with the effect hook : no code repeat + grouping of similar code.

**Why ? T**he effect hook helps you perform side effects, with out repeating any code and grouping code with similar functionality.

What ? This is a close replacement for componentDidmount,ComponentDidUpdate,ComponentWillunmount methods.

How to use the EffectHook.

* The useEffect hook takes a parameter which gets executed after every time the component renders i.e after probably every time the state changes.
* You can conditionally run the effects also

const App = ()=>{

    const [count,setCount] = useState(0)

    const [inputT,setinputT] = useState('')

    const clickHdlr = (e)=>{

        setCount((prevCount)=>{

            return prevCount+1;

        })

    }

    const textChangeHdlr = (e)=>{

        setinputT(e.target.value)

    }

    //useEffect generally runs after first render +

    //after EVERY update

    // conditionally run useEffect   
 // i.e only when the count value changes

    // only execute this if only state.count changes

    // ignore for other state variables

    useEffect(()=>{

        console.log('use effect called');

        document.title = `${count} - clicks`

    },[count]);  
 //Specify either ‘state’ or ‘props’ you need to watch for

    return(

        <div>

            <button onClick={clickHdlr}>CLICK</button>

            <input type='text'

                value={inputT}

                onChange={textChangeHdlr}></input>

        </div>

    )

}

1.Run the effect only once + cleaning .

const MouseComponent = ()=>{

    const [pos,setPos] = useState({x:0,y:0})

    const logMousePosition = (e)=>{

        console.log('mouse moved - handler executed')

        setPos({x:e.clientX,y:e.clientY})

    }

    useEffect(()=>{

        console.log('mounting component + Running Effect')

        window.addEventListener('mousemove', logMousePosition)

        // clean by returning this function -

        // if this component is not rendered the effect should be removed

        return ()=>{

            console.log('unmounting component')

            window.removeEventListener('mousemove',logMousePosition)

        }

    },[]);

    // This particular effect does not depend on any state or

    // props – only called in initial render

    return( <div>X : {pos.x}<br/> Y:  {pos.y} </div> )

}

const App = ()=>{

    const [visible,setVisibility] = useState(false)

    return(

        <div>

        <button onClick={(e)=>{setVisibility(!visible)}}>  
 Click</button> <br/>

        {visible && <MouseComponent/>}

    </div>);

}

The clean up code can be cancelling subscriptions,clearing timers, or removing event handlers.