REACT HOOK

BROCHILL LEARNING OVERVIEW

Overview

WRITING DOCS TO VISIT AND? UNDERSTAND HOOKS WHEN FORGOT OR NEED TO REVISIT THE CONCEPT ONCE AGAIN IN FUTURE

SHORT NOTES

components	May 28, 2024 import {hookName} from 'react';
useState()	**const {count,setCount} = useState(number)** **** always returns two parameters one variable two function** whenever the count is changed, setCount rerenders the component automatically --- **best way to use is using prev val** setCount(prev=>prev+1) here prev val is previous val and we just perform op on previous value which is used to update correct manner --- **best way to use is object const {userNAme,setUSerName} = useState({fName='',lName=''})** to update we spread/copy whole prev obj and update specific key value pair setUserName(prev=>{...prev,fname/lName:'someNAme'})
useEffect()	**used to perform sideEffects in functional components** **useEffect(function)** useEffect runs a function given to it whenever a render happens when no variable given as 2nd argument --- **useEffect(function,[stateVariable])** in order to run function for specific state change in site we give useEffect 2nd argument as array of state variables --- **useEffect clean up** **useEffect(()=>{** **code to run when state changes** **return ()=>{cleanup code }** **},[stateVarArray])** usually useEffect will return a function from the function which runs when state changes this function runs and cleans up the effect its imp
	dependency array is a array given 2nd arg to useEffect Empty dependency means to run function on first render

dependency array	any variable given in dependency array means run the function in useEffect whenever those variables changes their value --- dependency array is just there to say to keep track of the variables but not to say render then it just keeps track of values in variable and when changed then it rerenders const {count,setCount} = useState(0) const runn = ()=> setCount(count+1) here we are not tracking the count so we keep track in useEffect useEffect(()=>{ const interval = setInterval()runn,1000 return ()=>clearInterval(interval) },[count]) renders whenever count changes --- if we keep track of them in other place we dont need to keep track of them here const {count,setCount} = useState(0) const runn = ()=> setCount(prevCount = prevCount+1) here we are tracking the count so we keep track in useEffect useEffect(()=>{ const interval = setInterval()runn,1000 return ()=>clearInterval(interval) } []) renders whenever count changes
	const {age,setAge} = useState(25) **const {salary,setSalary} = useState(50000)** **const incAge = ()=> setAge(age+1)** **const incSal = ()=>setSalary(salary+10000)** **<Count text=”age” count={age}/>** **<Button handleClick={incAge}/>** **<Count text=”salary” count={salary}/>**

useCallback	**<Button handleClick={incSal}/>** when page is rendered and state is changed for age though it rerenders salaryButton too coz functions are not the same even though values of salary are same coz when rerendered function reference changes --- even when we use react.memo to avoid rerendering when props/state is not changed the function reference changes so the memo thinks they are diffrent and allows components to rerender --- this hook stores the function in cache and its dependencies whenever the salary changes or dependencies change it will return the memorized version of a callback fxn which will be changed only when dependencies change here callback means the fxn sent to children from parent **it checks if functions are equal and renders if not** **jus wrap setAge in useCallback** const incSal = useCallback(()=>setSalary(salary+10000)) like this and we can send incsal asusual

Relevant Resources

youtube codevolution

https://www.youtube.com/playlist?list=PLC3y8-rFHvwisvxhZ135pogtX7_Oe3Q3A

MAIN POINTS

	*call hooks only at toplevel of the component & dont call hooks inside loops,conditions or neasted functions*
	*only call hooks from react component function not from regural js functions*
	*practice more on dependencies on useEffect*