React Hooks

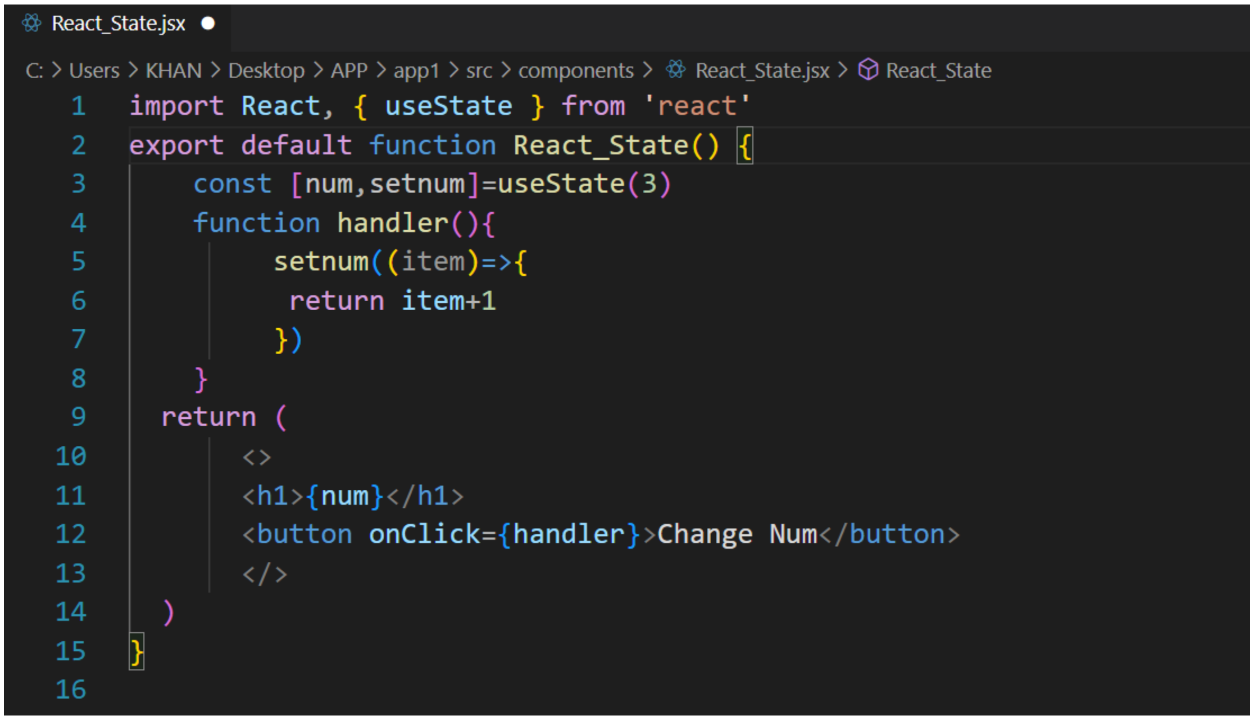
Hooks are **reusable functions** that **provide access to state** in **React Applications.** Hooks give access to states for functional components while creating a React application. It allows you to use state and other React features without writing a class.

React Hooks provide **functional components** with the **ability to use states and manage side effects**.

Types of hooks:

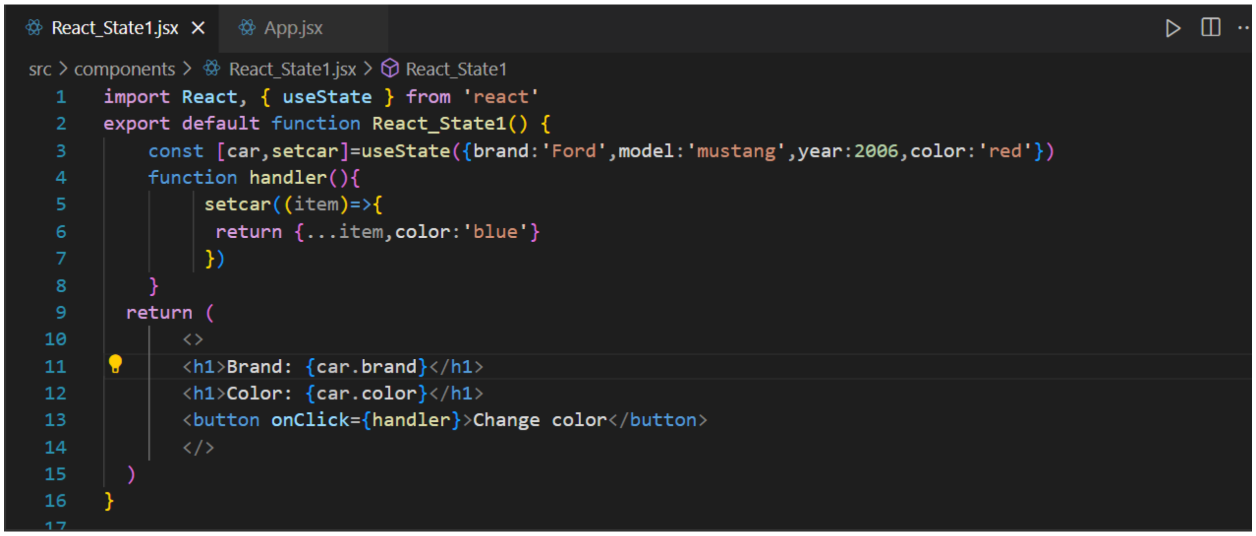
i) useState
ii) useEffect
iii) useRef
iv) useContext
v) useReducer
vi) useCallback
vii) useMemo
viii) Custom hooks

useState: The React useState Hook allows us to track state in a function component. State generally refers to data or properties that need to be tracking in an application.

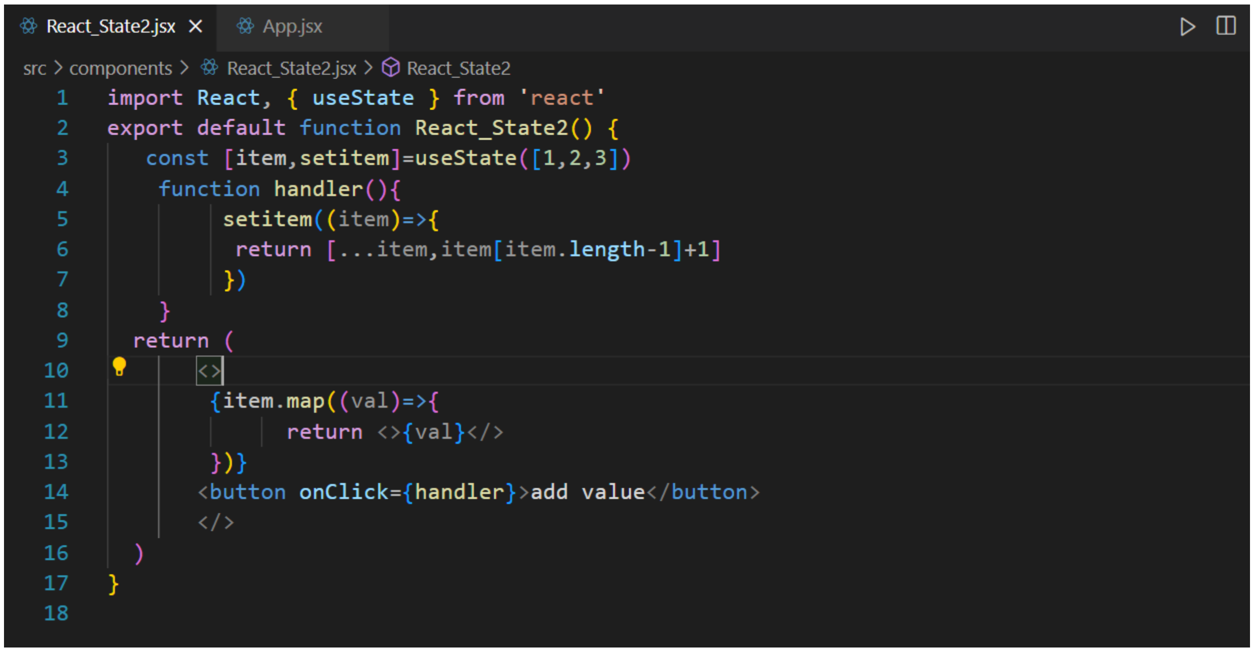

React_State.jsx
C: > Users > KHAN > Desktop > APP > app1 > src > components > React_State.jsx > React_State

```
import React, { useState } from 'react'
export default function React_State() {
    const [num,setnum]=useState(3)
    function handler(){
         setnum((item)=>{
          return item+1
         })
    }
  return (
        <>
        <h1>{num}</h1>
        <button onClick={handler}>Change Num</button>
        </>
  )
}

```

Updating Objects and Arrays in State

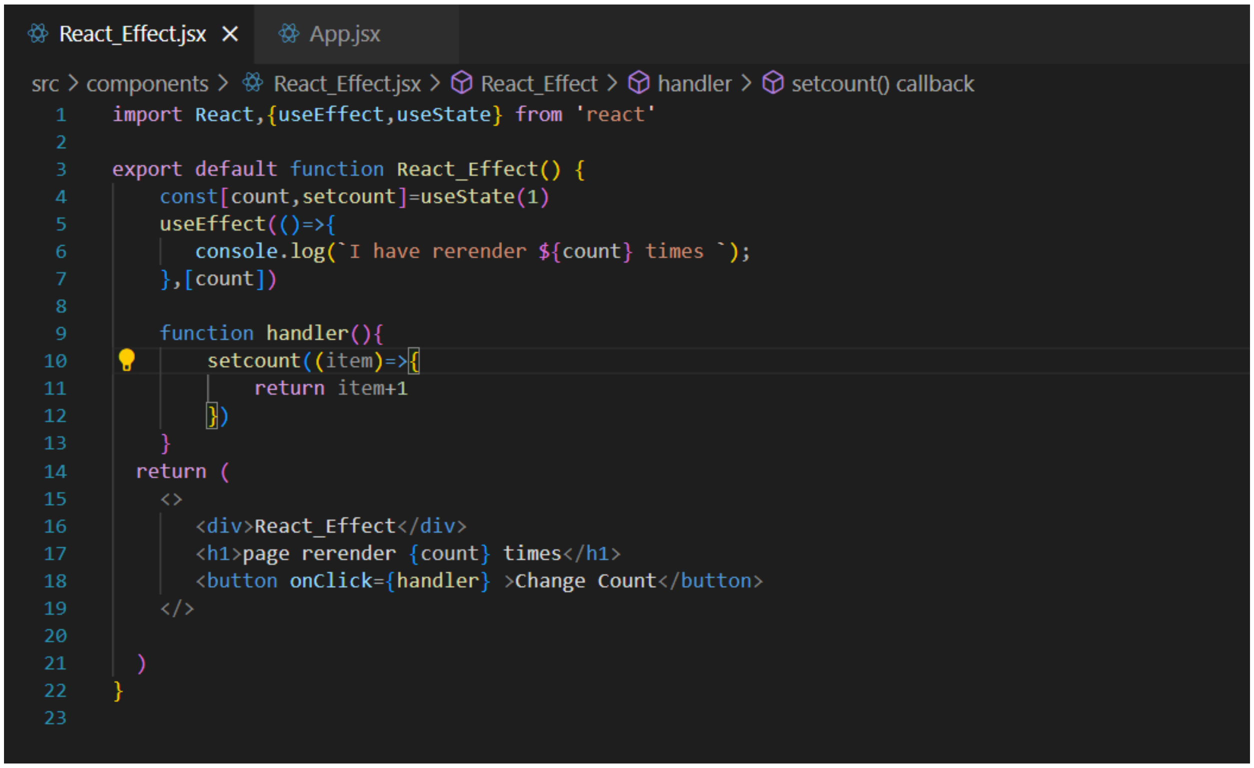


```jsx
import React, { useState } from 'react'
export default function React_State1() {
    const [car,setcar]=useState({brand:'Ford',model:'mustang',year:2006,color:'red'})
    function handler(){
        setcar((item)=>{
         return {...item,color:'blue'}
        })
    }
  return (
      <>
      <h1>Brand: {car.brand}</h1>
      <h1>Color: {car.color}</h1>
      <button onClick={handler}>Change color</button>
      </>
  )
}
```



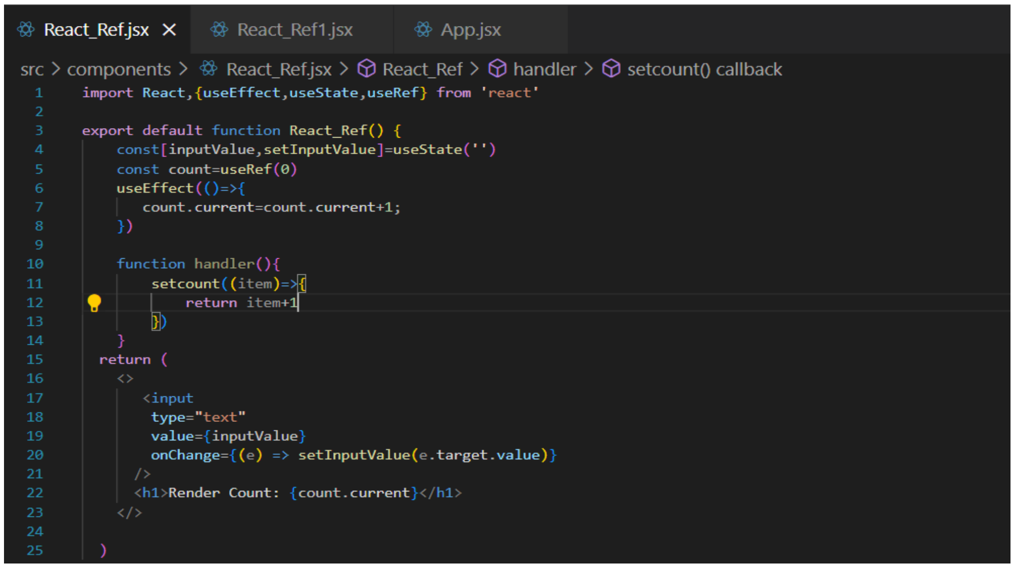
```jsx
import React, { useState } from 'react'
export default function React_State2() {
   const [item,setitem]=useState([1,2,3])
    function handler(){
        setitem((item)=>{
         return [...item,item[item.length-1]+1]
        })
    }
  return (
      <>
       {item.map((val)=>{
            return <>{val}</>
       })}
      <button onClick={handler}>add value</button>
      </>
  )
}
```

useEffect: The useEffect Hook allows you to perform side effects in your components.Some examples of side effects are: fetching data, directly updating the DOM, and timers.


React_Effect.jsx
App.jsx
src > components > React_Effect.jsx > React_Effect > handler > setcount() callback

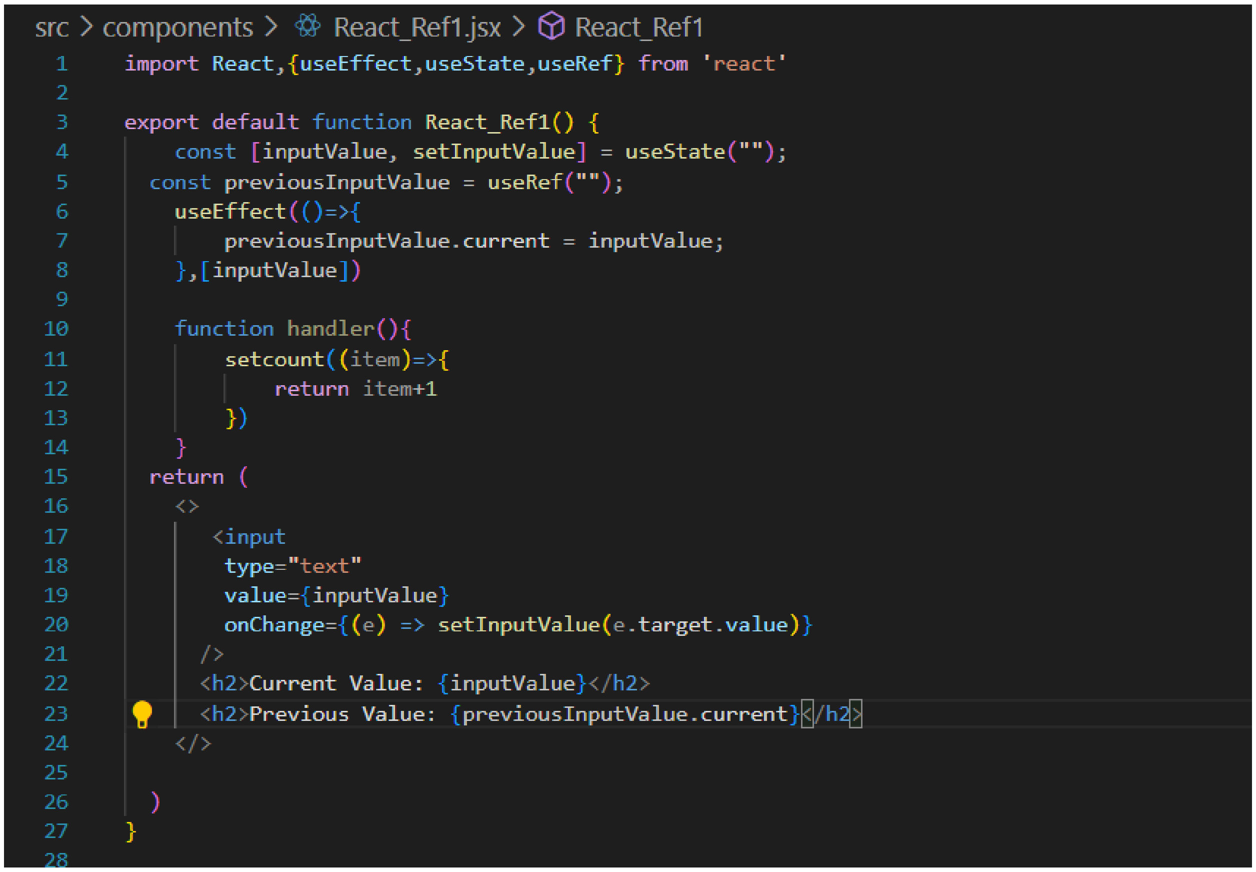
```jsx
import React,{useEffect,useState} from 'react'

export default function React_Effect() {
    const[count,setcount]=useState(1)
    useEffect(()=>{
       console.log(`I have rerender ${count} times `);
    },[count])

    function handler(){
        setcount((item)=>{
            return item+1
        })
    }
  return (
    <>
       <div>React_Effect</div>
       <h1>page rerender {count} times</h1>
       <button onClick={handler} >Change Count</button>
    </>

  )
}

```

userRef: The useRef Hook allows you to persist values between renders. It can be used to store a mutable value that does not cause a re-render when updated.


React_Ref.jsx
React_Ref1.jsx
App.jsx
src > components > React_Ref.jsx > React_Ref > handler > setcount() callback

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

export default function React_Ref() {
    const[inputValue,setInputValue]=useState('')
    const count=useRef(0)
    useEffect(()=>{
       count.current=count.current+1;
    })

    function handler(){
        setcount((item)=>{
            return item+1
        })
    }
  return (
    <>
       <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
      />
      <h1>Render Count: {count.current}</h1>
    </>

  )
```

Tracking State Changes



```
import React,{useEffect,useState,useRef} from 'react'

export default function React_Ref1() {
    const [inputValue, setInputValue] = useState("");
  const previousInputValue = useRef("");
    useEffect(()=>{
        previousInputValue.current = inputValue;
    },[inputValue])

    function handler(){
        setcount((item)=>{
            return item+1
        })
    }
  return (
    <>
       <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
      />
      <h2>Current Value: {inputValue}</h2>
      <h2>Previous Value: {previousInputValue.current}</h2>
    </>

  )
}

```