Hooks

1. useMemo

- Memoization is essentially just caching. Imagine a complex function that is slow to run which takes a as an argument. aba argument a ko value same input pathauda kheri ta output ni same ta aauni ho. aba yesto condition ko lagi function lai speed up garna chahi just cached value use garda sajilo hunxa, instead of recomputing the whole thing.
- useMemo ko syntax chahi just like useEffect hook ho
    - const result = useMemo(() ⇒ {
      return slowFunction(a)
      }, [a])
    - yo example syntax maa, aba a ko value maa kei change aayo vani slowFunction lai run garni, natra just cached value run garda pugihalxa
    - Aba jastai ko yo muni ko example maa
        - harek choti kei changes aayo, number maa vayeni yaa just theme maa vayeni, whole thing rerender vairakhxa
        - testo vo vane ta aba slow function chaleko chalei garni vayo ni ta, double number vanni variable maa value halnu ko lagi (even when theme matra change garda, light mode to dark mode garda ni whole thing rerender huda full computation chahirakhyo)
        - aba testo nahos vanerw doubleNumber variable wala chahi useMemo maa hanerw rakhni
        - yesto garexi aba theme matra change garda ta argument ‘a’ maa kei change nai aayeko xaina, so cached value matra use huni vo

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

export default function App() {
  const [number, setNumber] = useState(0)
  const [dark, setDark] = useState(false)
  const doubleNumber = useMemo(() => {
    return slowFunction(number)
  }, [number])
  const themeStyles = {
    backgroundColor: dark ? 'black' : 'white',
    color: dark ? 'white' : 'black'
  }

  return (
    <>
      <input type="number" value={number} onChange={e => setNumber(parseInt
      (e.target.value))} />
      <button onClick={() => setDark(prevDark => !prevDark)}>Change Theme</button>
      <div style={themeStyles}>{doubleNumber}</div>
    </>
  )
}

function slowFunction(num) {
  console.log('Calling Slow Function')
  for (let i = 0; i <= 1000000000; i++) {}
  return num * 2
```

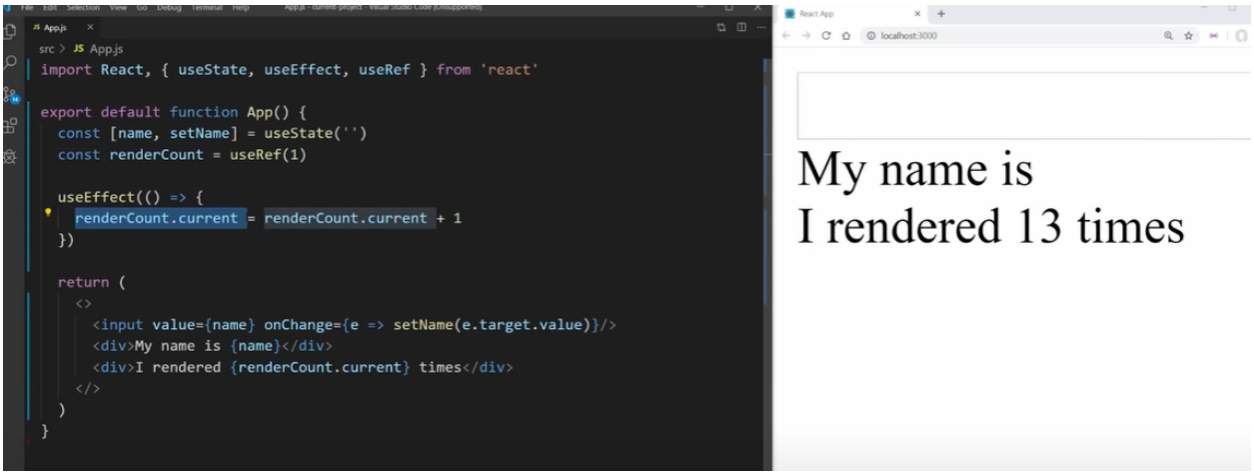
- aba memoization jaile chahi use garna hunna, cuz jaile use garyo vane ta performance overheads ra storage overheads aaihalxa
- useMemo for referential equality
  - when comparing 2 varibles (in case of objets and arrays), JS compares reference instead of actual values.
  - eg:

```jsx
useEffect(() => {
  console.log('Theme Changed')
}, [themeStyles])
```

- yo useEffect chalauda harek render maa themeStyles jun xa tyo chahi naya object banxa...same dekhey ni referential equality anusar diff objects hun. So, aba doubleNumber vanni chalauda ni yo chahi run huni vo, since doubeNumber le rerender handinxa, ra theme change navayeni naya themeStyles vanni obj le garerwTheme Changed vanerw dekhauni vo console maa
- so do smth like this

2. useRef

- useState jastai ho, tarw useState maa kei changes aauda kheri full rerender hunthyo. But ref doesnot cause component to reupdate when it gets changed
- returns one single value unlike state
- returns an object that has a single property “current” whose value is equal to useRef maa j pass garyo tehi. Yo value chahi persists between different renders.
- jati change gareyni, it never causes our component to rerender
- aba yo muni ko example maa input field maa value = {name} garya xa, and name is a useState. So e.target.value bata setName garda kheri harek choti rerender huni vo state change vara
  Then. useEffect maa chahi rerender huda chalauni banaxa, no dependecy array le garda... so harek rerender maa ref ko current value will increment by one and we can see the thing



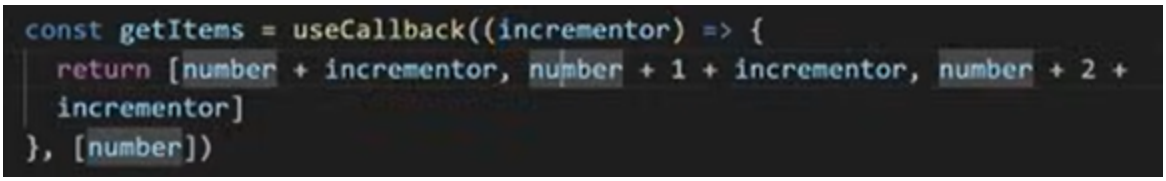
- ***mainly used to reference elements inside HTML***
- aba yo muni ko example maa ni inputRed vanni useRef bata banayo...since ref = {inputRef} garerw euta input element lai chahi reference gariraxa, aba muni ko button click huda focus vanni func chalxa jasle chahi inputRef.current, which is just that input html element, tesma .focus() vanni method lagayera focus gardinxa.

- In a class component, we can use class variables and such in order to persist values between rerenders. but in functional components, useRef le matra persist garna dinxa.

3. useCallback

- useMemo jastai ho tarw useMemo chahi is used to cache values, tarw useCallback can be used to directly cache functions.

- useMemo maa chahi getItems nai vayeko vaye just return value matra getItems maa as a cache store hunthyo. Tara hamro useCallback ko case maa whole function is stored as cache in getItems.
- aba useMemo le value matra lini vara parameters pass garna mildaian thyo function maa. Tarw useCallback maa testo hunna, and allows parameters too



- aba function nai store garni vayexi, nachahiyesamma function getItems() chahi change huni vayena. That is why referential equality pani maintained huni vayo.
- so, overall ma chahi function creation nai slow vo vane yo use garnu payo... Yesto condition khassai chahi aaudaina, so yo dherai use huni chahi referential equality problem aauda kheri ho, when you use some other hooks like useEffect.

React.memo

- useMemo ra useCallback bahek yo ni euta memoization technique ho.
- a component wrapped in React.memo will not re-render unless the props change (so, it acts as React.PureComponent)

```
React.memo(function Component(props) {
  // Do something
})
```