Assignment Solution

(https://github.com/100xdevs-cohort-2/assignments.git)

- **use-memo()**

Assignment 1:

```
import { useState } from "react";

// In this assignment, your task is to create a component that performs an
expensive calculation (finding the factorial) based on a user input.
// Use useMemo to ensure that the calculation is only recomputed when the
input changes, not on every render.

export function Assignment1() {
    const [input, setInput] = useState(0);
    // Your solution starts here
    const expensiveValue = 0;
    // Your solution ends here

    return (
        <div>
            <input
                type="number"
                value={input}
                onChange={(e) => setInput(Number(e.target.value))}
            />
            <p>Calculated Value: {expensiveValue}</p>
        </div>
    );
}
```

Solution:

```
import { useMemo, useState } from "react";
```

```
// In this assignment, your task is to create a component that performs an expensive calculation (finding the factorial) based on a user input.
// Use useMemo to ensure that the calculation is only recomputed when the input changes, not on every render.

export function Assignment1() {
    const [input, setInput] = useState(0);
    // this will currently changes only when input changes , if there were more state variables this question would have maked more sense
    // Your solution starts here

    // wrapping expensive operation inside useMemo this will only run when input changes
    const expensiveValue = useMemo(()=>{
        let value = 1;
        for(let i =1; i<= input;i++){
            expensiveValue = expensiveValue* i;
        }
        return value;
    },[input]);
    // Your solution ends here

    return (
        <div>
            <input
                type="number"
                value={input}
                onChange={(e) => setInput(Number(e.target.value))}
            />
            <p>Calculated Value: {expensiveValue}</p>
        </div>
    );
}
```

Assignment 2:

Goal of memo is to optimize the filtering process

```
import React, { useEffect, useState } from "react";
```

```
// In this assignment, you will create a component that renders a large list of sentences and includes an input field for filtering these items.
// The goal is to use useMemo to optimize the filtering process, ensuring the list is only re-calculated when necessary (e.g., when the filter criteria changes).
// You will learn something new here, specifically how you have to pass more than one value in the dependency array

const words = ["hi", "my", "name", "is", "for", "to", "random", "word" ];
const TOTAL_LINES = 1000;
const ALL_WORDS = [];
for (let i = 0; i < TOTAL_LINES; i++) {
    let sentence = "";
    for (let j = 0; j < words.length; j++) {
        sentence += (words[Math.floor(words.length * Math.random())])
        sentence += " "
    }
    ALL_WORDS.push(sentence);
}

// logic to generate 1000 random numbers

export function Assignment2() {
    const [sentences, setSentences] = useState(ALL_WORDS);
    const [filter, setFilter] = useState("");

    const filteredSentences = sentences.filter(x => x.includes(filter))
// expensive operation

    return <div>
        <input type="text" onChange={(e) => {
            setFilter(e.target.value)
        }}></input>
        {filteredSentences.map(word => <div>
            {word}
        </div>)}
    </div>
}
```

Solution:

```jsx
import React, { useEffect, useState ,useMemo} from "react";

// In this assignment, you will create a component that renders a large list of sentences and includes an input field for filtering these items.
// The goal is to use useMemo to optimize the filtering process, ensuring the list is only re-calculated when necessary (e.g., when the filter criteria changes).
// You will learn something new here, specifically how you have to pass more than one value in the dependency array

const words = ["hi", "my", "name", "is", "for", "to", "random", "word" ];
const TOTAL_LINES = 1000;
const ALL_WORDS = [];
for (let i = 0; i < TOTAL_LINES; i++) {
    let sentence = "";
    for (let j = 0; j < words.length; j++) {
        sentence += (words[Math.floor(words.length * Math.random())])
        sentence += " "
    }
    ALL_WORDS.push(sentence);
}

export function Assignment2() {
    const [sentences, setSentences] = useState(ALL_WORDS);
    const [filter, setFilter] = useState("");
    const filteredSentences = useMemo(()=>{
        return sentences.filter(x => x.includes(filter));
    },[sentences,filter])

    return <div>
        <input type="text" onChange={(e) => {
            setFilter(e.target.value)
        }}></input>
        {filteredSentences.map(word => <div>
            {word}
        </div>)}
    </div>
}
```

Assignment 3:

```
import React, { useState, useMemo } from 'react';
// You have been given a list of items you shopped from the grocery store
// You need to calculate the total amount of money you spent

export const Assignment3 = () => {
    const [items, setItems] = useState([
        { name: 'Chocolates', value: 10 },
        { name: 'Chips', value: 20 },
        { name: 'Onion', value: 30 },
        { name: 'Tomato', value: 30 },
        // Add more items as needed
    ]);

    // Your code starts here
    const totalValue = 0;
    // Your code ends here
    return (
        <div>
            <ul>
                {items.map((item, index) => (
                    <li key={index}>{item.name} - Price:
${item.value}</li>
                ))}
            </ul>
            <p>Total Value: {totalValue}</p>
        </div>
    );
};
```

Solution:

```
import React, { useState, useMemo } from 'react';
// You have been given a list of items you shopped from the grocery store
// You need to calculate the total amount of money you spent

export const Assignment3 = () => {
    const [items, setItems] = useState([
```

```
        { name: 'Chocolates', value: 10 },
        { name: 'Chips', value: 20 },
        { name: 'Onion', value: 130 },
        { name: 'Tomato', value: 30 },
        // Add more items as needed
    ]);

    // Your code starts here
    // reducer
    const totalValue = useMemo(()=>{
        let totalValue = 0;
        for(let i=0; i<items.length; i++){
            totalValue = totalValue + items[i].value;
        }
        return totalValue;
    },[items])

    // Your code ends here
    return (
        <div>
            <ul>
                {items.map((item, index) => (
                    <li key={index}>{item.name} - Price:
${item.value}</li>
                ))}
            </ul>
            <p>Total Value: {totalValue}</p>
        </div>
    );
};
```

useCallback()

Assignment 1/Solution:

```
import { useCallback, useState } from "react";
```

```jsx
// Create a counter component with increment and decrement functions. Pass these functions to a child component which has buttons to perform the increment and decrement actions. Use useCallback to ensure that these functions are not recreated on every render.

export function Assignment1() {
    const [count, setCount] = useState(0);

    // Your code starts here
    const handleIncrement = useCallback(()=> {
        // setCount(count => count+1)
        // setCount(count + 1)

        setCount(function(currentCount){
            return currentCount + 1;
        })
        // if we are using state vairble then we have to use them as dependencies
        // but its
    },[])

    const handleDecrement= useCallback(()=>{
        setCount(count => {
           return count -1;
        })
    },[])
    // Your code ends here

    return (
        <div>
            <p>Count: {count}</p>
            <CounterButtons onIncrement={handleIncrement} onDecrement={handleDecrement} />
        </div>
    );
};

const CounterButtons = ({ onIncrement, onDecrement }) => (
    <div>
        <button onClick={onIncrement}>Increment</button>
```

```
        <button onClick={onDecrement}>Decrement</button>
    </div>
);
```

Assignment 2:

```
import React, { useState, useCallback } from 'react';

// Create a component with a text input field and a button. The goal is to
display an alert with the text entered when the button is clicked. Use
useCallback to memoize the event handler function that triggers the alert,
ensuring it's not recreated on every render.
// Currently we only have inputText as a state variable and hence you
might not see the benefits of
// useCallback. We're also not passing it down to another component as a
prop which is another reason for you to not see its benefits immediately.

export function Assignment2() {
    const [inputText, setInputText] = useState('');

    // Your code starts here
    function showAlert() {

    }
    // Your code ends here

    return (
        <div>
            <input
                type="text"
                value={inputText}
                onChange={(e) => setInputText(e.target.value)}
                placeholder="Enter some text"
            />
            <Alert showAlert={showAlert} />
        </div>
    );
};
```

```
function Alert({showAlert}) {
    return <button onClick={showAlert}>Show Alert</button>
}
```

Solution

```
    const showAlert = useCallback(()=> {
        alert(inputText);
    },[inputText])
```

useRef()

Two cases useRef are quite useful

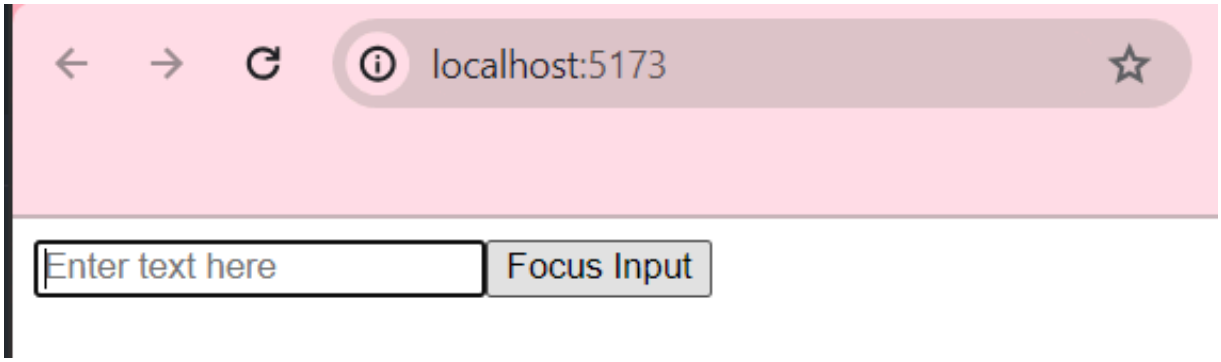
1. Getting access to DOM element
2. When we want to have to access to a variable accross renders which is not state variable

```
import { useEffect } from "react";

// Create a component with a text input field and a button. When the component mounts or the button is clicked, automatically focus the text input field using useRef.

export function Assignment1() {

    useEffect(() => {

    }, []);

    const handleButtonClick = () => {

    };

    return (
        <div>
            <input type="text" placeholder="Enter text here" />
            <button onClick={handleButtonClick}>Focus Input</button>
```

```
        </div>
    );
};
```

Solution:

```
import { useEffect,useRef } from "react";
// Create a component with a text input field and a button. When the
component mounts or the button is clicked, automatically focus the text
input field using useRef.
export function Assignment1() {
    const inputRef = useRef();

    useEffect(() => {
        inputRef.current.focus()
    }, [inputRef]);

    const handleButtonClick = () => {
        inputRef.current.focus()
    };
    return (
        <div>
            <input type="text" placeholder="Enter text here"
ref={inputRef}/>
            <button onClick={handleButtonClick}>Focus Input</button>
        </div>
    );
};
```



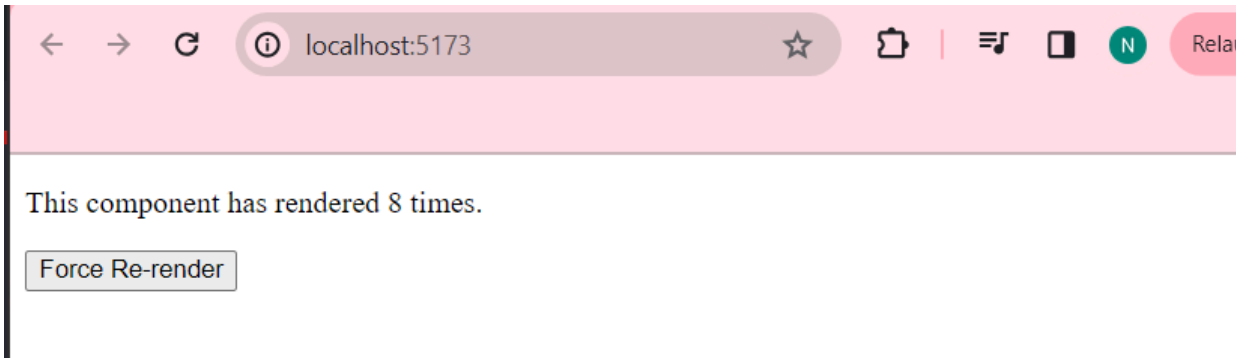
Assignment 2:

```jsx
import React, { useState, useCallback } from 'react';

// Create a component that tracks and displays the number of times it has
been rendered. Use useRef to create a variable that persists across
renders without causing additional renders when it changes.

// Goal is to forcefully render the component
//  We want to put that the component has been re-rendered is ' x ' times.
export function Assignment2() {
    const [, forceRender] = useState(0);
// if we dont need the first variable we can keep it empty

    const handleReRender = () => {
        // Update state to force re-render
        forceRender(Math.random());
    };

    return (
        <div>
            <p>This component has rendered {0} times.</p>
            <button onClick={handleReRender}>Force Re-render</button>
        </div>
    );
};
```

Method 1: worse way is to track it by using state variable:

In this method we are not able to track the correct no. of re-renders

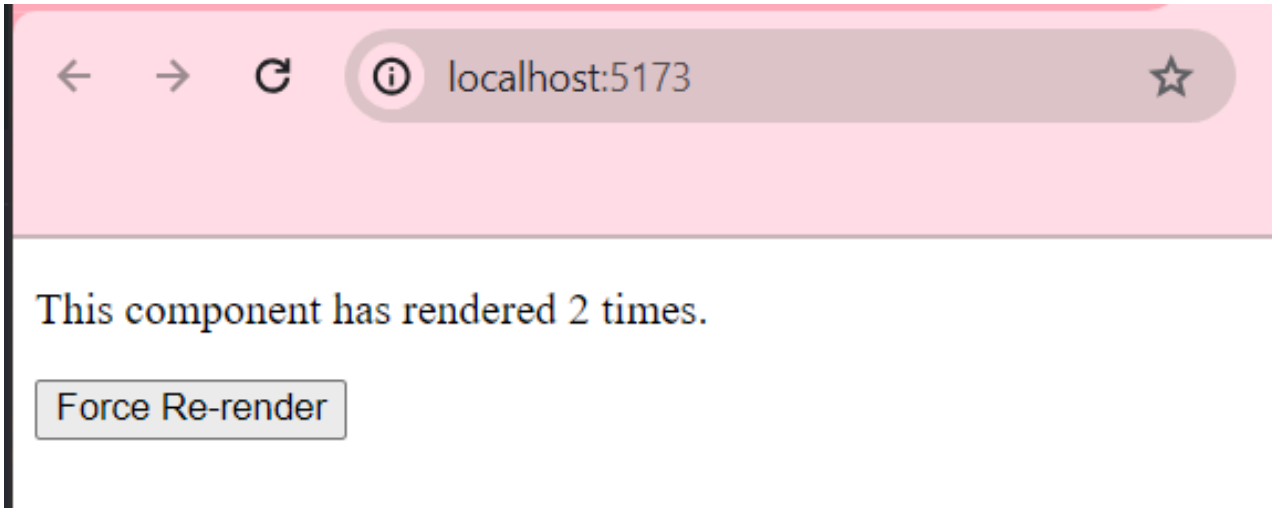
```jsx
import React, { useState, useCallback } from 'react';

// Create a component that tracks and displays the number of times it has
been rendered. Use useRef to create a variable that persists across
renders without causing additional renders when it changes.

// Goal is to forcefully render the component
//  We want to put that the component has been re-rendered is ' x ' times.
export function Assignment2() {
    const [count, setCount] = useState(0);
    const [numberOfTimesReRendered, setNumberOfTimesReRendered] =
useState(0);
// if we dont need the first variable we can keep it empty

    const handleReRender = () => {
        // Update state to force re-render
        setCount(count+1);
        setNumberOfTimesReRendered(numberOfTimesReRendered+1);
    };

    return (
        <div>
            <p>This component has rendered {0} times.</p>
            <button onClick={handleReRender}>Force Re-render</button>
        </div>
    );
};
```



Method 2: Use global variable

```
import  { useState } from 'react';

// Create a component that tracks and displays the number of times it has been rendered. Use useRef to create a variable that persists across renders without causing additional renders when it changes.

// Goal is to forcefully render the component
//  We want to put that the component has been re-rendered is ' x ' times.
let numberOfTimesReRendered = 0;
export function Assignment2() {
    const [count, setCount] = useState(0);
// if we dont need the first variable we can keep it empty

    const handleReRender = () => {
        // Update state to force re-render
        setCount(count+1);
    };
    numberOfTimesReRendered=numberOfTimesReRendered+1;

    return (
        <div>
            <p>This component has rendered {numberOfTimesReRendered} times.</p>
            <button onClick={handleReRender}>Force Re-render</button>
        </div>
    );
}
```

React in strict mode will re-render one more time



We should not have any variable like thes outside component lifecycle.

Method 3:

Here comes another use case of useRef()

```
import  { useState,useRef } from 'react';

// Create a component that tracks and displays the number of times it has
been rendered. Use useRef to create a variable that persists across
renders without causing additional renders when it changes.

// Goal is to forcefully render the component
//  We want to put that the component has been re-rendered is ' x ' times.

export function Assignment2() {
    const [count, setCount] = useState(0);
// if we dont need the first variable we can keep it empty
    const numberOfTimesReRendered = useRef(0);
    // its reference is kept constant
    // now its part of React lifecycle
    // it can be passed to the child

    const handleReRender = () => {
        // Update state to force re-render
        setCount(count+1);
    };
```

```
    numberOfTimesReRendered.current=numberOfTimesReRendered.current+1;

    return (
        <div>
            <p>This component has rendered
{numberOfTimesReRendered.current} times.</p>
            <button onClick={handleReRender}>Force Re-render</button>
            {/* either put a button reference or a number */}
        </div>
    );
};
```