**성능 최적화를 위해, 연산된 값을 useMemo라는 Hook을 사용하여 재사용해보자**

**active 값이 true인 사용자의 수를 세어서 화면에 렌더링 하는 countActiveUsers 구현**

App 컴포넌트에서 다음과 같이 countActiveUsers 라는 함수를 만들어보자.

//App.js

…

function conuntActiveUsers(users) {

console.log(‘활성 사용자 수를 세는 중…’);

return users.filter(user => user.active).length;

}

function App() {

…

const count = countActiveUsers(users);

return (

…

<div> 활성사용자 수 : {count}</div>

…

);

}

설명

1. countActiveUsers에서 console.log를 출력한 이유
   1. 호출될 때마다 우리가 알 수 있게 하기 위해
2. users.filter(user => user.active).length
   1. users의 원소 하나하나의 active가 참인 것들을 filter가 반환하고, 이 반환된 것의 length를 참조하는 것.

**위 코드의 치명적 단점**

1. 바로 users에 변화가 생길 때만 리렌더링 해야하는데, input 값이 바뀔 때에도 리렌더링 하게 됨.
   * + - useMemo를 사용하여 성능 최적화

**useMemo 사용한 코드**

//App.js

import React, { useRef, useState, useMemo } from ‘react’;

…

fucntion countActiveUsers(users) {

console.log(‘활성 사용자 수를 세는 중…’);

return users.filter(user => user.active).length;

}

function App() {

…

const count = useMemo(() => countActiveUsers(users), [users]);

return (

…

<div>활성사용자 수 : {count}</div>

…

);

}

설명

1. const count = useMemo(() => countActiveUsers(users), [users]);
   1. useMemo의 첫번째 파라미터에는 어떻게 연산할지 정의하는 함수를 넣어준다.
   2. 두번째 파라미터에는 deps 배열을 넣어주면 되는데, 이 배열 안에 넣은 내용이 바뀌면 우리가 등록한 함수를 호출해서 값을 연산해주고, 바뀌지 않으면 이전에 연산한 값을 재사용함.