**React.memo를 사용하면 컴포넌트에서 리렌더링이 필요한 상황에서만 리렌더링을 한다.**

-사용법은 그냥 감싸주면 됨.

//CreateUser.js

const CreateUser = ({ username, email, onChange, onCreate }) => {

return (

…

);

};

export default React.memo(CreacteUser);

//UserList.js

…

const User = React.memo(function User({ user, onRemove, onToggle }) {

return (

…

);

});

…

export default React.memo(UserList);

-위 코드까지하면 input을 수정할 때 UserList는 리렌더링 되지 않음

-> 하지만 User를 수정하면 모든 User들이 리렌더링되고, CreateUser도 리렌더링 됨.

-> users 배열이 바뀔 때마다 onCreate, onToggle, onRemove가 새로 만들어지기 때문이다.

->useCallback할 때 deps 배열에 users를 넣었기 때문.

-> deps에서 users를 지우고, 함수들에서 현재 useState로 관리하는 users를 참조하지 않으면 됨

-> useState할 때 배웠던 함수형 업데이트를 하면 된다.

//App.js

…

const onChange = useCallback(e => {

const { name, value } = e.target;

setInputs(inputs => ({

…inputs,

[name]:value

}));

}, []);

…

const onCreate = useCallback(() =. {

…

setUsers(users => users.concat(user));

…

}, [username, email]);

…

**기타**

1. useCallback, useMemo, React.memo는 컴포넌트의 성능을 실제로 개선할 수 있는 상황에서만 써라
   1. 예를들어 User 컴포넌트의 b, button의 onClick 으로 설정해준 함수들은 해당 함수들을 useCallback으로 재사용한다고 리렌더링을 막을 수 있는 것이 아니다.
   2. 렌더링 최적화하지 않을 컴포넌트에 React.memo를 사용하는건 필요없는 props비교를 불러온다
2. React.memo에서 두번째 파라미터에 propsAreEqual이라는 함수를 사용하여 특정값들만 비교를 할 수 있다.

export default React.memo(

UserList,

(prevProps, nextProps) => prevProps.users === nextProps.users

);

* 잘못사용하면 의도치않은 버그가 발생함.
* 예를들어 함수형 업데이트로 전환을 안했는데 이렇게 users만 비교를 하게 되면 onToggle과 onRemove에서 최신 users배열을 참조하지 않게 된다.