**React Hooks**

React Hook은 리액트 16.8버전에서 새롭게 등장한 개념으로 **React 함수형 컴포넌트에서 상태(state)나 생명주기(lifecycle) 기능을 사용할 수 있도록 해주는 JavaScript 함수이다.** 클래스형 컴포넌트를 사용할 때 필요했던 this.state, this.setState, componentDidMount 등의 기능을 함수형 컴포넌트에서도 사용할 수 있게 해준다.

**공식문서 :https://react.dev/reference/react/hooks**

**☞**Built-in React Hooks

|  |  |
| --- | --- |
| **State Hooks** | useState |
| useReducer |
| **Context Hooks** | useContext |
| **Ref Hooks** | useRef |
| **Effect Hooks** | useEffect |
| **Performance Hooks** | useMemo |
| useCallback |

**useEffect**

React 컴포넌트의 사이드 이펙트를 제어하는 Reack Hook

|  |
| --- |
| **사이드 이펙트(Side Effect)란?**  우리말로는 “부작용” 이라는 뜻!  리엑트에서는**“부수적인 효과”,”파생되는 효과”** 로 해석한다. |

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 그래픽이(가) 표시된 사진

자동 생성된 설명

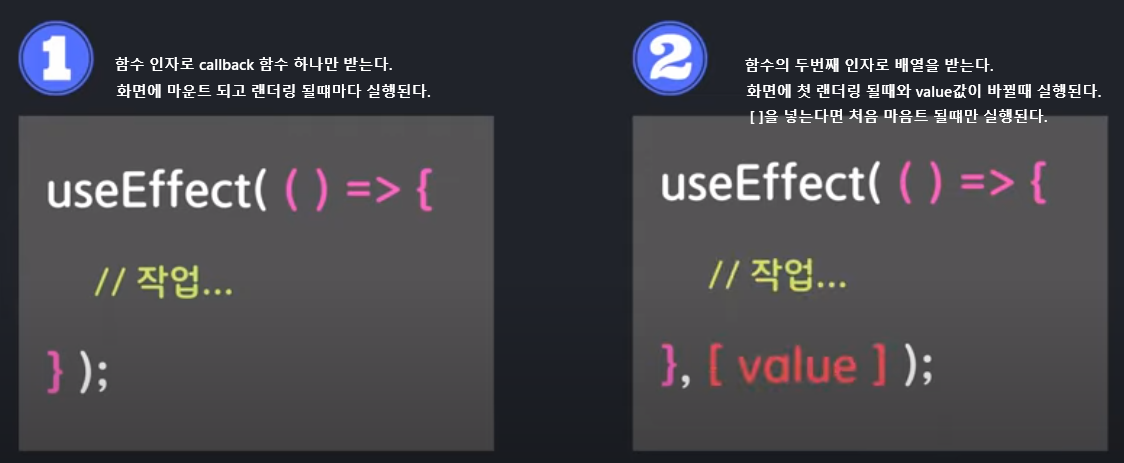
텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

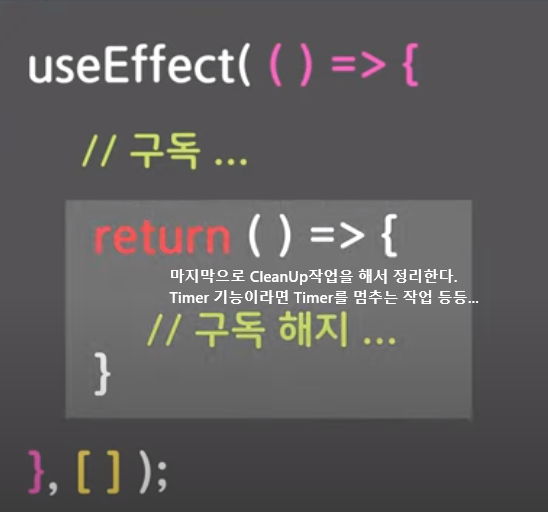
자동 생성된 설명

**React 컴포넌트 라이프 사이클**

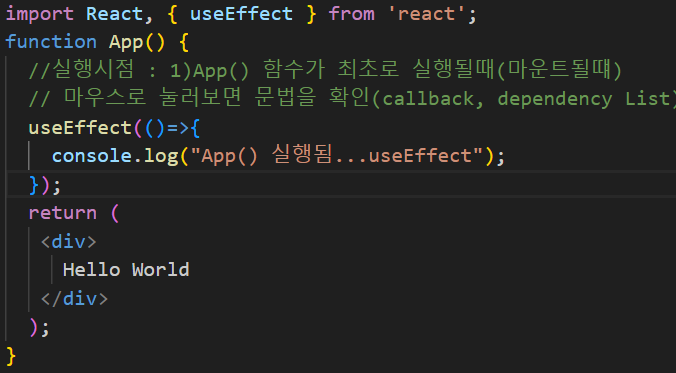
스크린샷, 텍스트, 폰트, 그래픽이(가) 표시된 사진

자동 생성된 설명



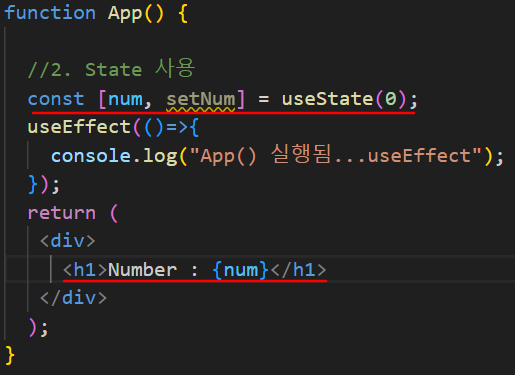


● Step1.useEffect ... **최초 랜더링 (마운트)**





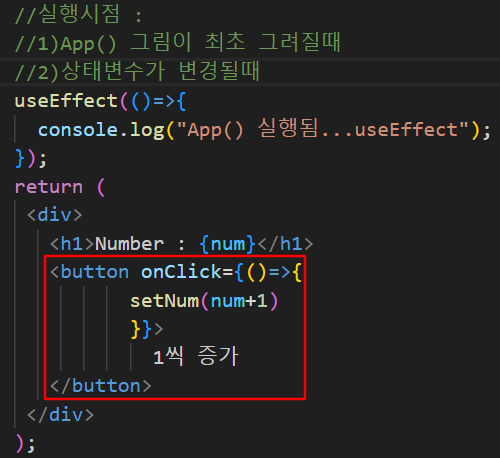


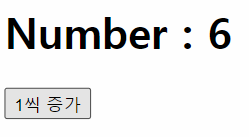




1번 실행됨을 확인

**● Step2 버튼 추가하고 값 변경**



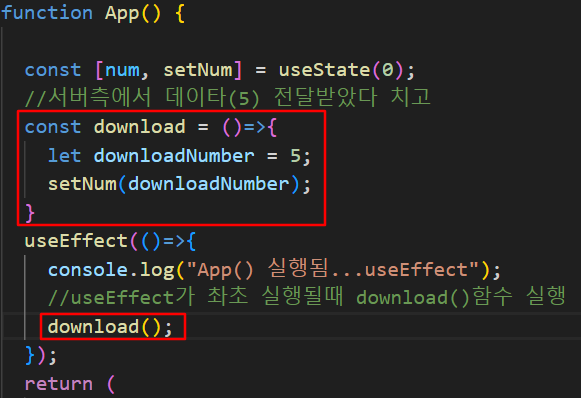


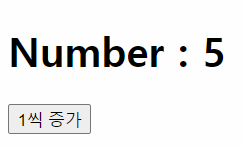


버튼 누를 때 마다useEffect가 실행된다.

상태변수가 실행될때 App이 실행되고 return이 동작하면서 다시 랜더링된다.

● **Step3 Dependency List**







▶ App 실행(useEffect실행됨)-->그안에 있는 download() 함수 실행됨 -->상태값을 5로 변경

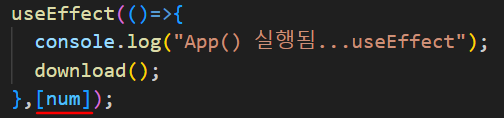
--> 화면에 5가 뜬다.//

▶ 버튼을 클릭하면 5에서 6으로 변경되어야 하는데 6으로 변경되자마자 다시 5로 뜬다. 왜?

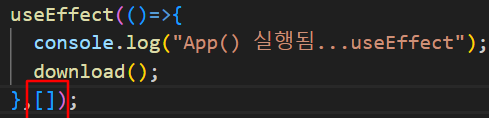
-->setNum(num+1) 에서 순간적으로 6이 되지만 상태값이 변경되었기에 다시 useEffect가 실행되고 다시 5로 값을 되돌려 놓기 때문이다.

▶ 처음 랜더링 될 때useEffect는 실행되어야 하지만상태값이변경될때마다 호출되지 않도록 한다.

-->useEffect()함수의 두번째 인자값을 사용해야 한다.

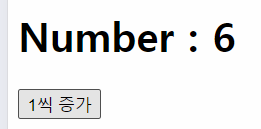
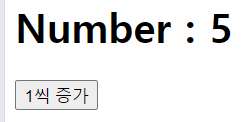


상태변수 num이 변경될때마다useEffect실행



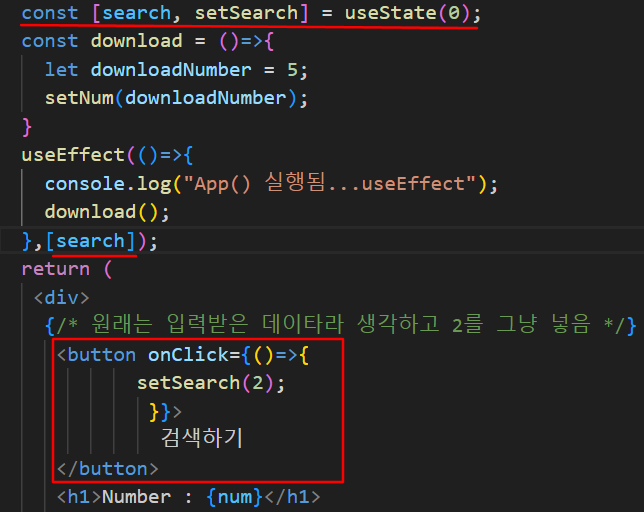
[ ]빈 배열은 어디에도 의존하지 않는다는 뜻

최초에 랜더링 될 때 만 useEffect()함수 실행



5에서 1씩 증가해서 다음 값은 6을 출력한다

5) 또다른 상태값을 추가

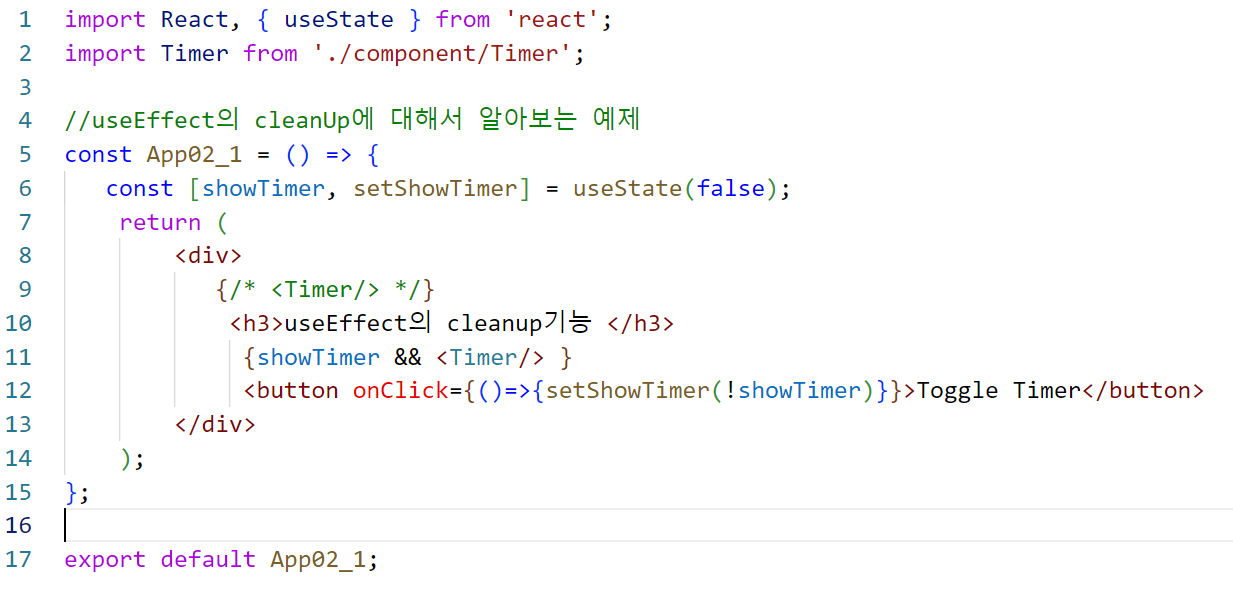


[search] search상태값에 의존적으로 작성

useEffect() 함수가 실행된다. 이때 출력숫자가

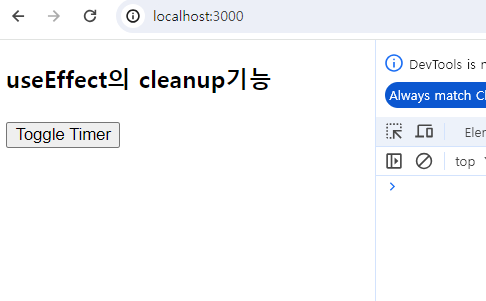
다시 5로 바뀌는것을 주목

**● Step4useEffect 정리작업**





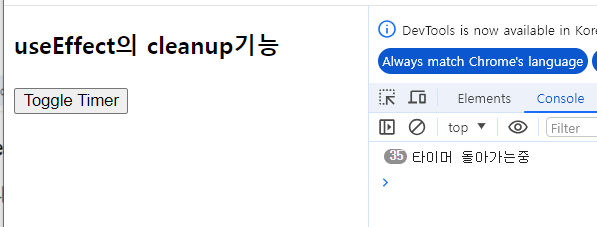
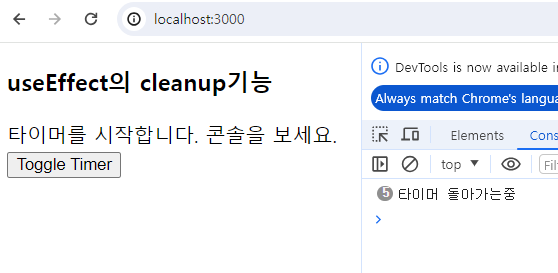
**실행화면**



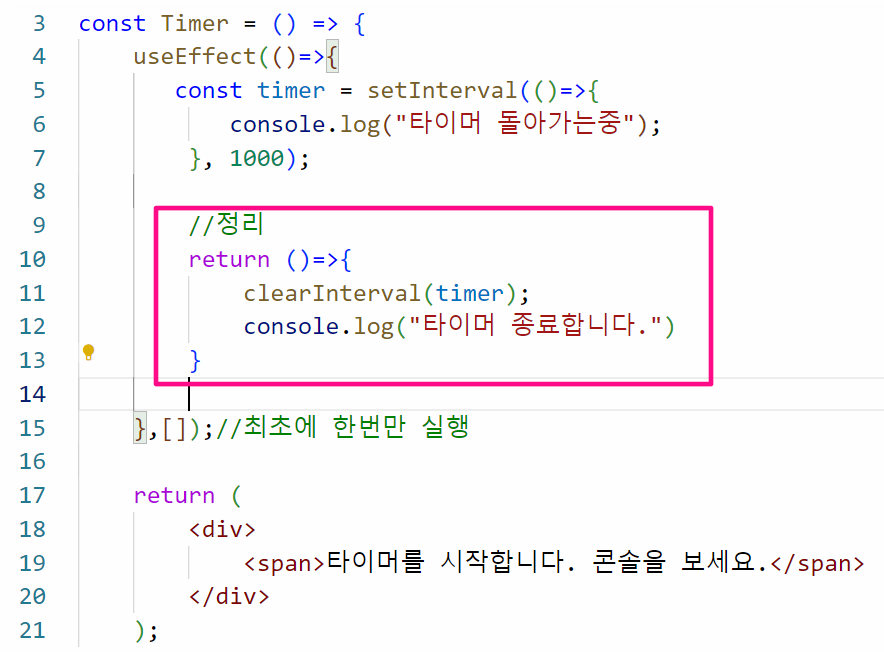
“Toggle Timer” 버튼을 클릭하면 <Timer>가 mount되고 , 다시 클릭하면 unmount된다.

**그러나**,

unmount되어도 돌고 있는 setInterval은 계속 실행중이다. 이것을 정리 해줘야 한다.



useEffect 함수안에서 return 함수를 작성하여 정리작업을 한다.



**useMemo**

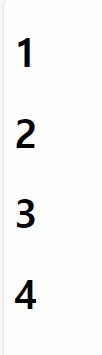
useMemo의 Memo는 Memoization(메모이제이션)을 뜻하는데 기존에 수행한 연산의 결과값을 어딘가에 저장해 두고 동일한 입력이 들어오면 저장해둔 값을 재활용하는 프로그래밍 기법이다.(캐싱기법)

React에서 성능 최적화를 위해 사용하는 훅으로 주로 **값 계산**에 드는 비용이 클 때, 해당 값이 **변경되지 않으면 다시 계산하지 않고 이전 값을 재사용**하도록 한다. 주로 **리렌더링 성능을 최적화**하려는 목적으로 사용된다.

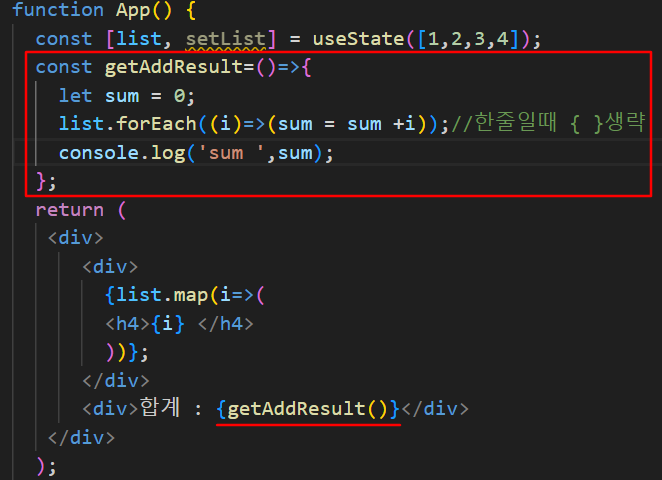
● Step1.

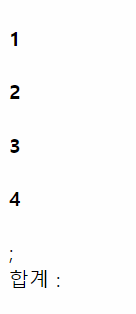
1) list 상태값 출력





2) sum 상태값 콘솔 출력

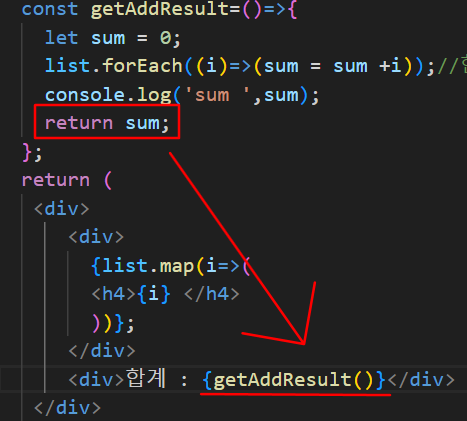


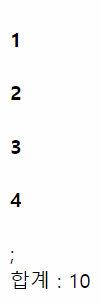




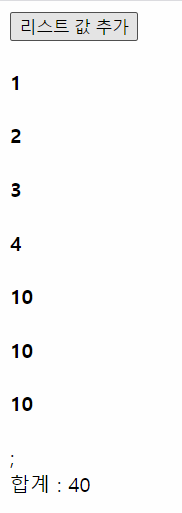
콘솔창에는 10이라고 출력, 화면에도 동일한 값이 출력되려면 getAddResult() 함수가 리턴해야함

3)화면에 sum이 출력됨

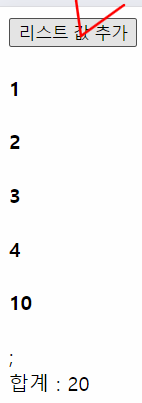


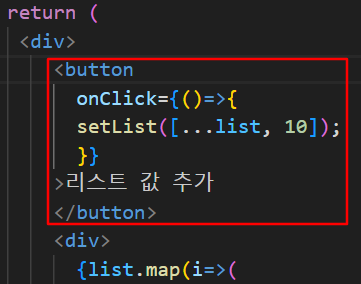


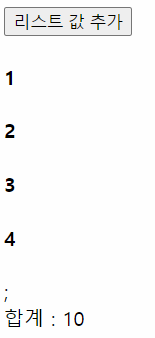
4) 버튼 클릭하면 list 상태값에 값 하나 추가



가장쉬운 전개연산자를 사용한다.



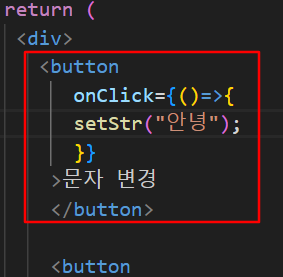


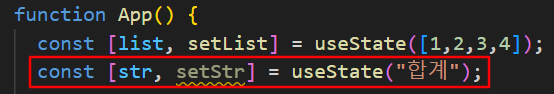


버튼 클릭하면 상태값 list가 변경됨으로 return이 다시 실행(재랜더링)...getAddResult() 함수가 다시 실행된 결과이다.

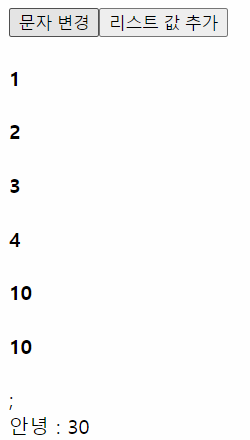
● Step2.

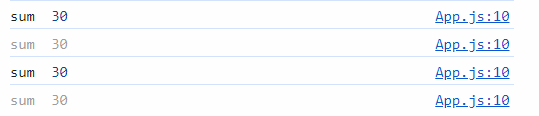
1) 문자변경 버튼을 추가하고 실행결과 확인











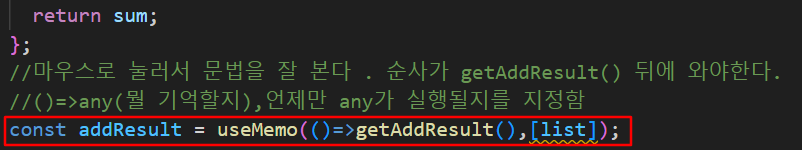
문자변경 버튼을 클릭하면 “안녕”으로 상태값이 변경됨으로 다시 return부분이 재랜더링된다 -->getAddResult() 함수가 다시 실행됨으로 필요하지 않은 연산작업을 수행한다!!

리스트 추가버튼을 클릭했을 때 getAddResult() 함수가 실행.

문자변경 버튼을 클릭했을때는getAddResult() 함수 실행을 막아둠.

2) useMemo() 함수 사용법

|  |
| --- |
| const memoizedValue = useMemo(() => {  // 값 계산  return 값;  }, [의존성 배열]);  **첫 번째 인자**: 값을 계산하는 함수. 이 함수는 의존성 배열이 변경될 때만 실행된다.  **두 번째 인자**: 의존성 배열. 배열 안에 있는 값이 변경될 때만 useMemo 안의 계산 함수가 실행된다. 의존성 배열이 빈 배열([])이면, useMemo는 컴포넌트가 처음 렌더링될 때만 값을 계산하고, 이후에는 재계산하지 않는다. |





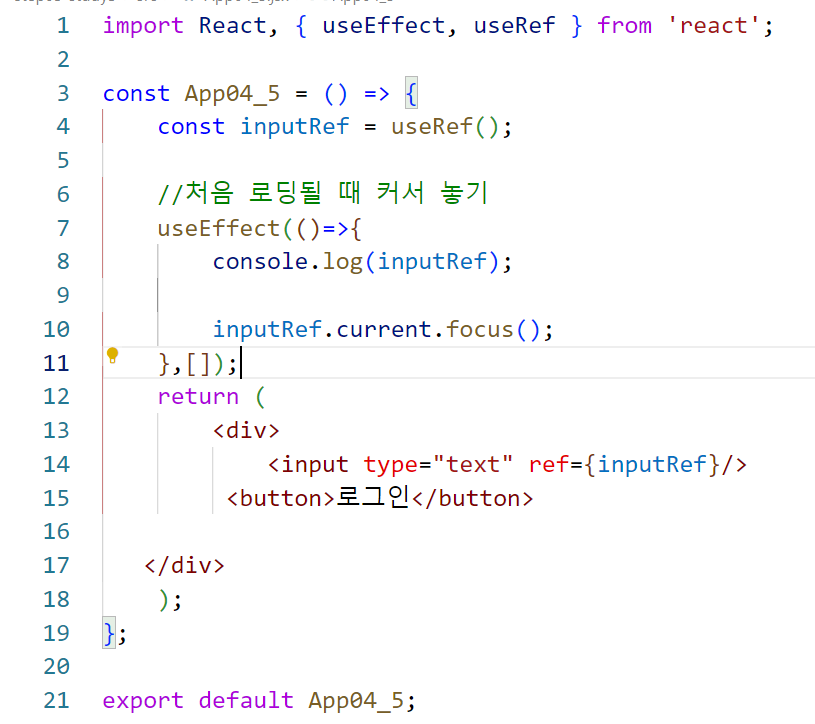
문자변경 버튼을 클릭하면 getAddResult() 함수가 실행되지 않는다.

**useRef**

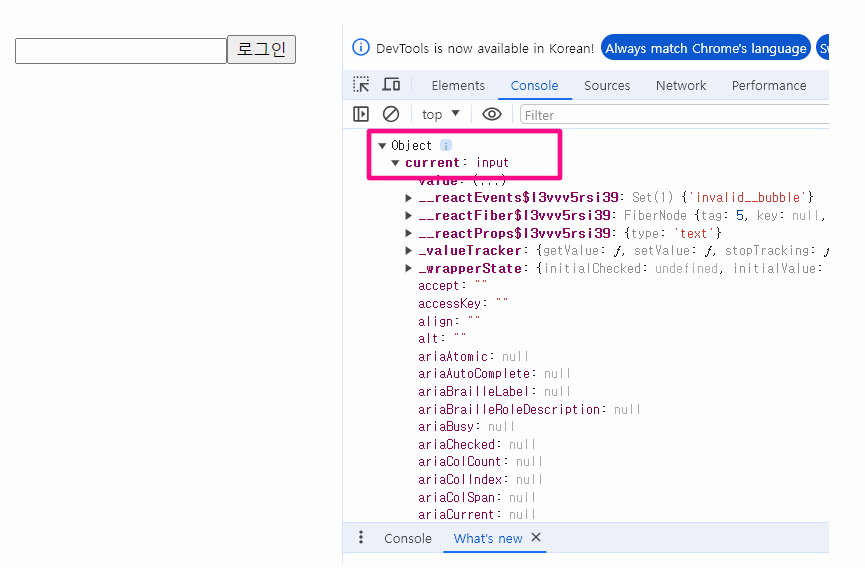
useRef는 React에서 값을 참조하거나 DOM 요소에 직접 접근할 수 있도록 해주는 훅이다. useRef는 렌더링 사이에 값을 유지할 수 있는 방법을 제공하며, 컴포넌트의 렌더링과 관계없이 값이 유지되므로 렌더링을 유발하지 않기 때문에 성능상 이점이 있다.

|  |
| --- |
| **사용법**  **const ref = useRef(initialValue);**  initialValue : useRef로 생성할 참조의 초기 값이다. initialValue는 기본값이 될 수 있으며, DOM 요소를 참조하는 경우에는 null로 초기화하는 경우가 많다.  useRef는 { current : T } 형태의 객체를 반환한다. T는 useRef에 전달한 initialValue의 타입이고 ref.current를 통해 값을 읽거나 수정할 수 있다. |

**text박스에 커서 놓기 - useRef를 이용하면 DOM요소에 쉽게 접근 할 수 있다.**

****

**실행결과**

****

**버튼 클릭시 alert() 출력**

****

**2) 저장공간**

- state의 변화가 생기면 다시 랜더링이 되고 컴포넌트 내부 변수들이 모두 초기화 되지만 **ref는 ref의 변화가 생겨도 랜더링이 되지 않고 변수들의 값이 유지가 된다.**

**결론적으로,**

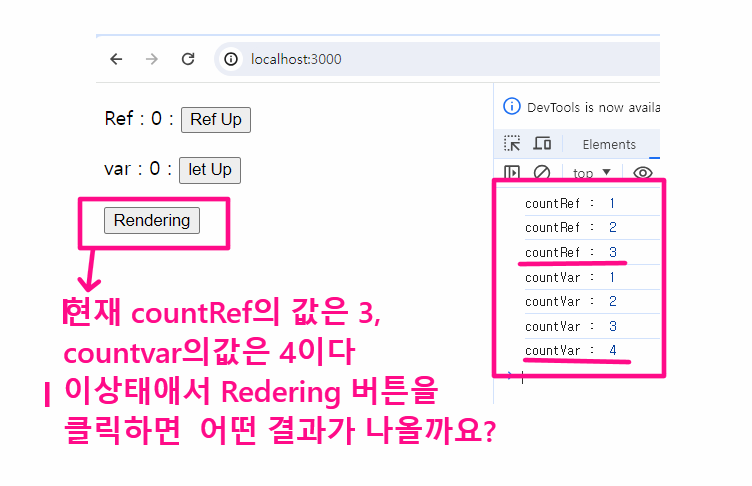
**useRef는 변화는 감지해야 하지만 그 변화가 랜더링을 감지하면 안 되는 경우에 사용하면 좋다.**



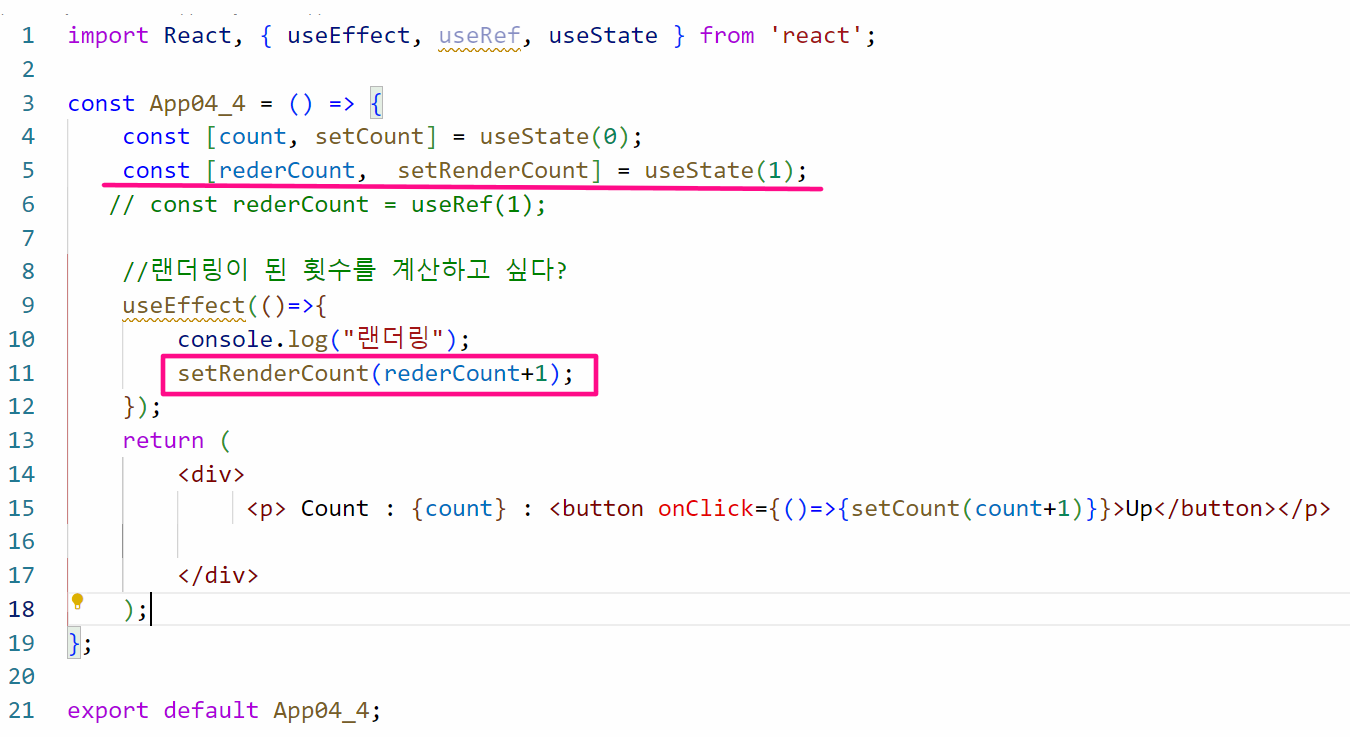
**☞ Ref와 let의 차이**

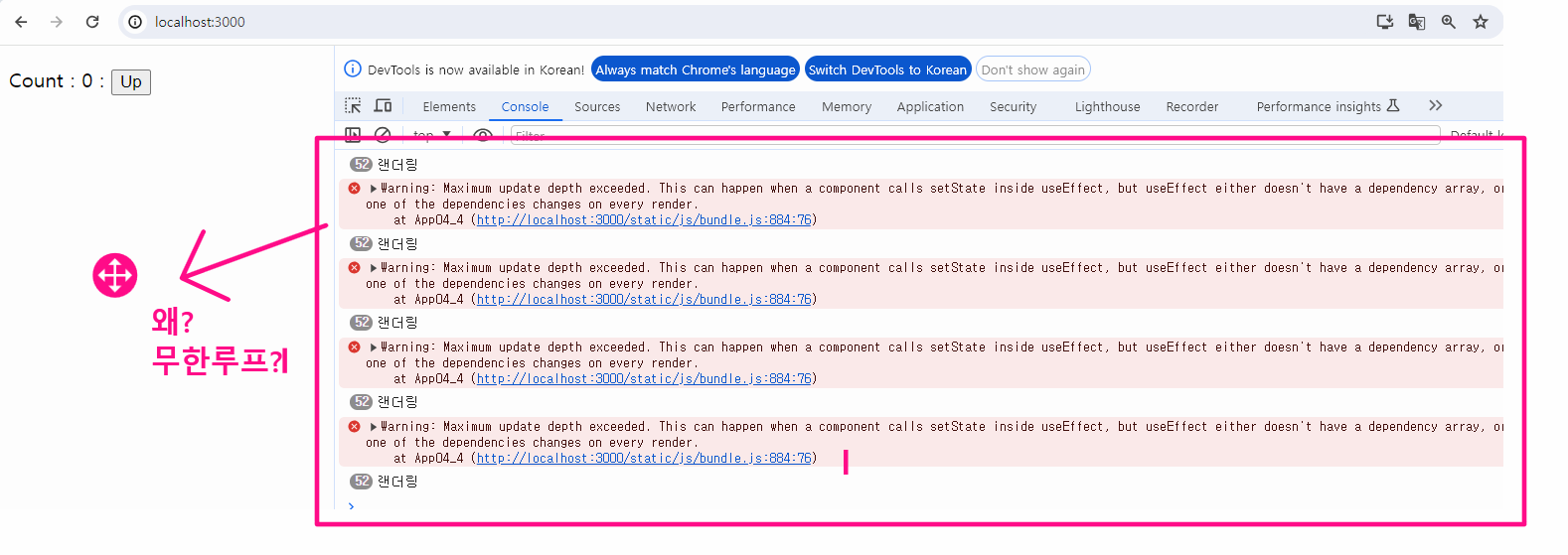
****

**실행결과**



**총 랜더링이 된 횟수를 계산하고 싶다면?**





**☞useState를 useRef로 변경해보자!**

****

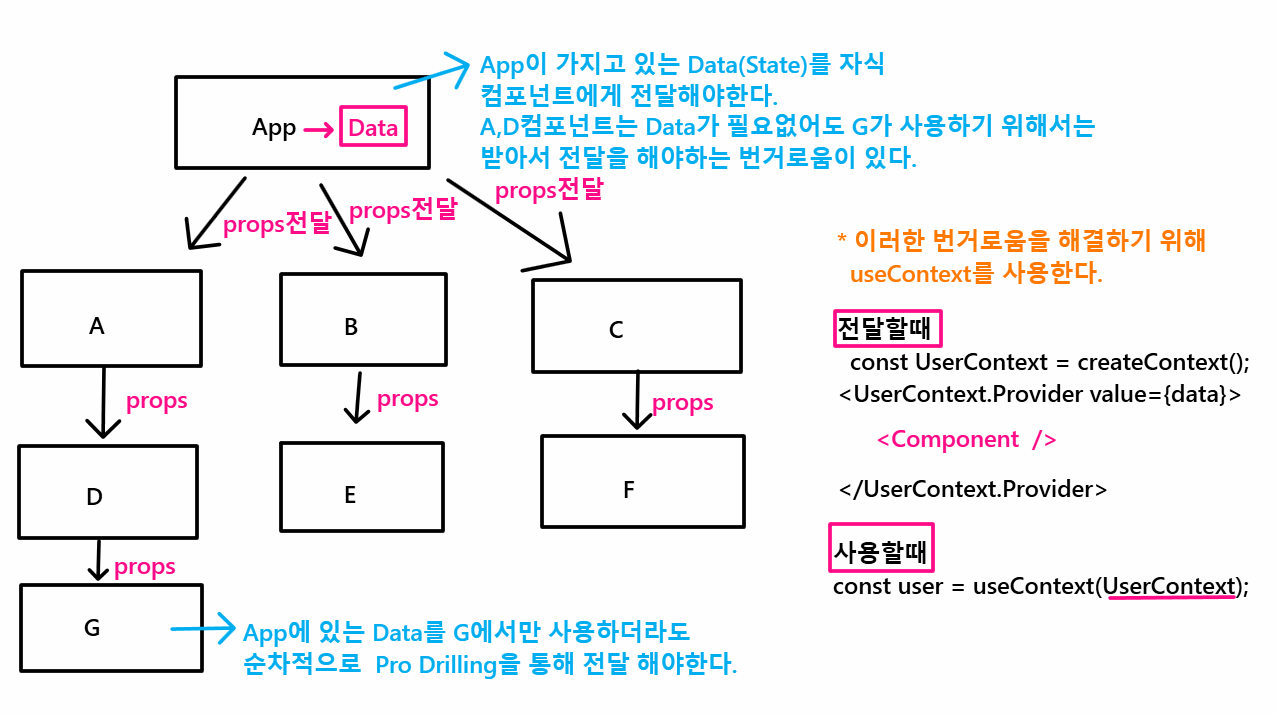
**useContext**

**https://ko.legacy.reactjs.org/docs/context.html**

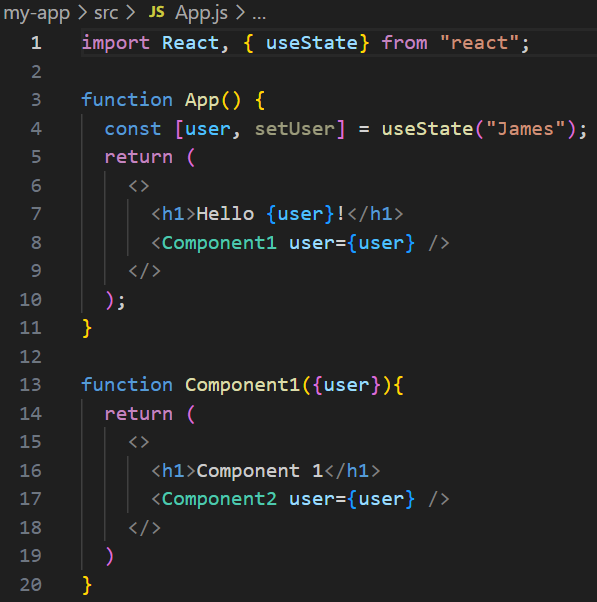
글로벌하게 상태를 관리하는 방법

useState만 사용하는 것보다 깊게 중첩된 컴포넌트 간의 상태를 더 쉽게 공유할 수 있다.

Context를 사용하면 컴포넌트를 재사용하기 어려워 질 수 있다.



**1) 컴포넌들 간에 Props를 통해서 데이타를 전달받는 코드**



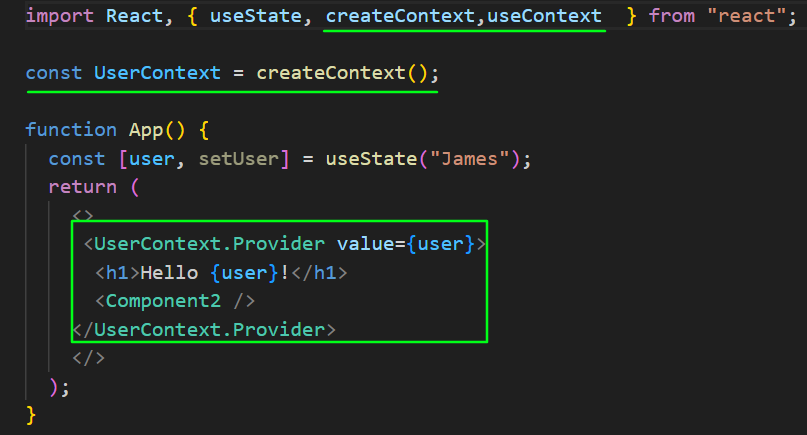


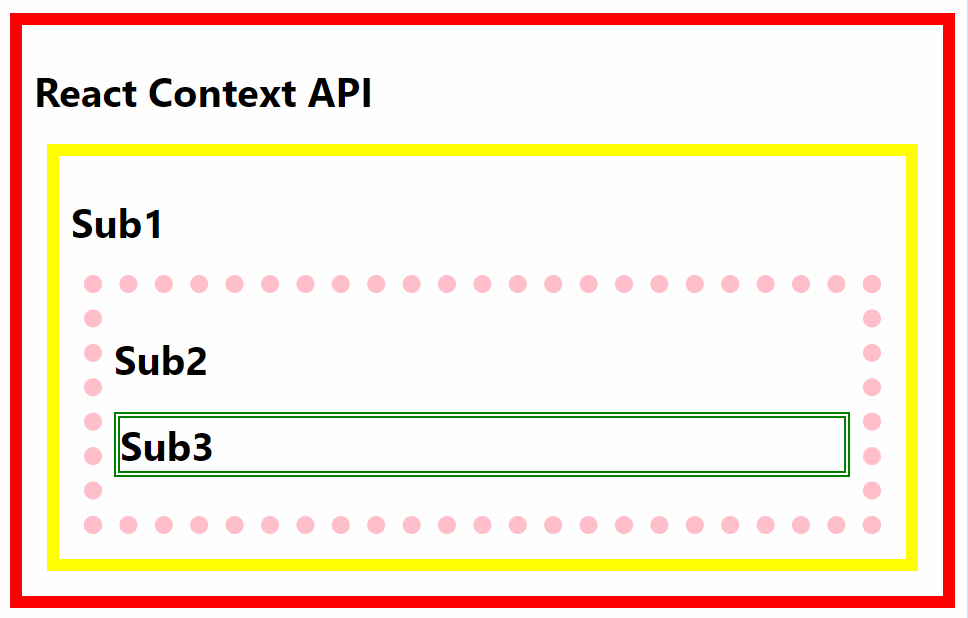
결과 확인



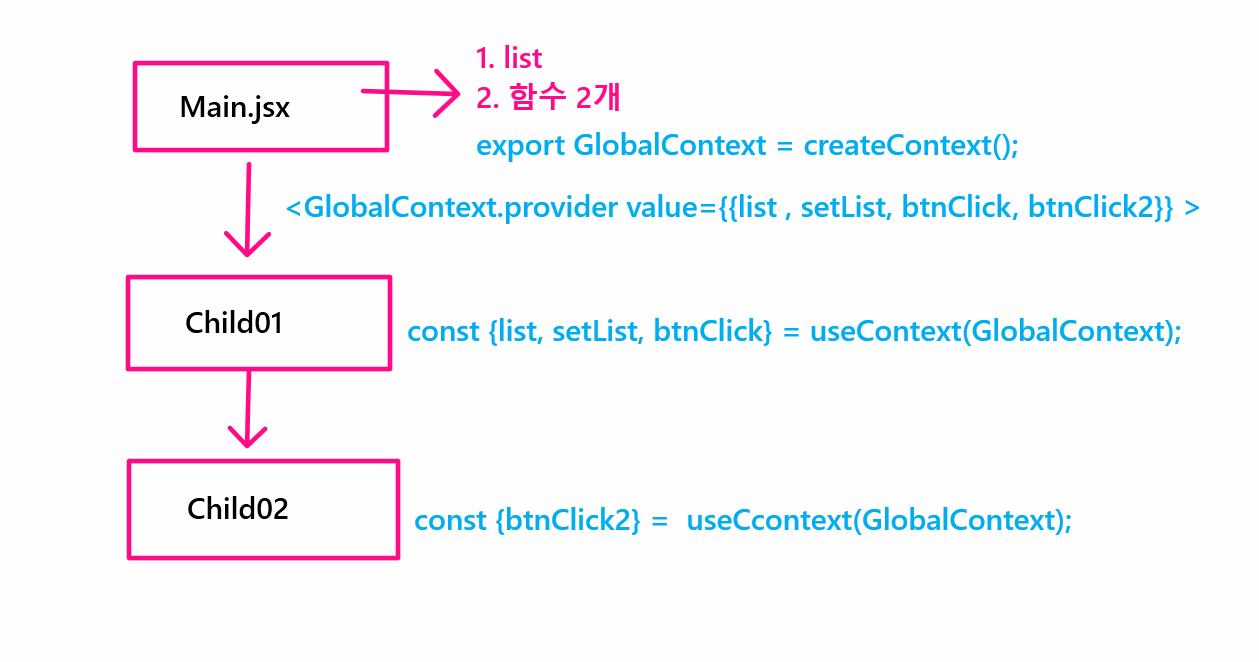
**2) Context 사용해서 코드 작성**



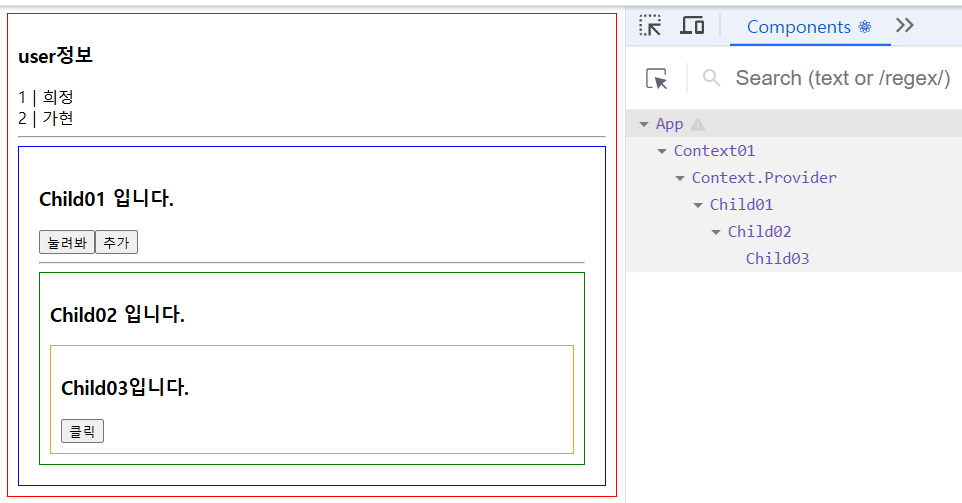


****

**실습**

****

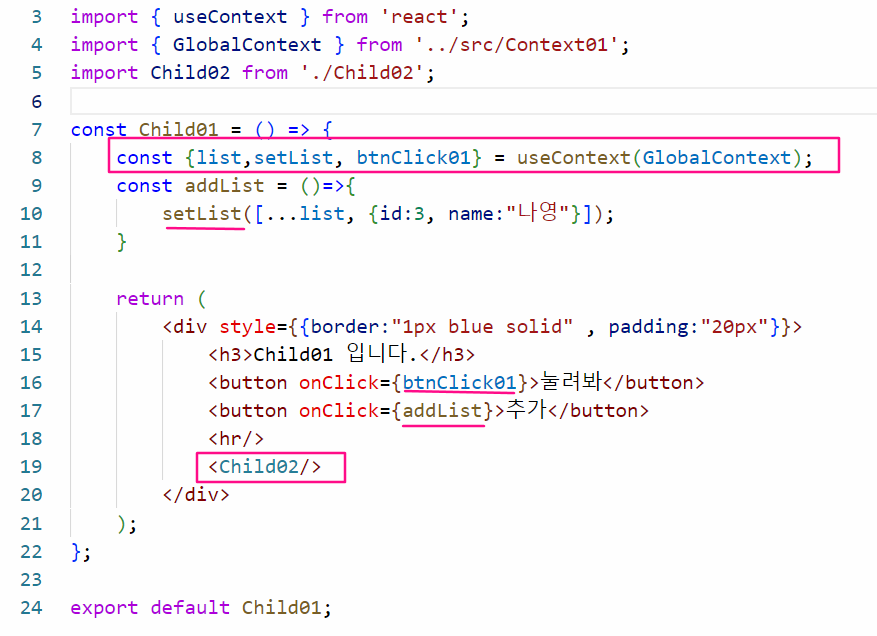
**실습화면**

****

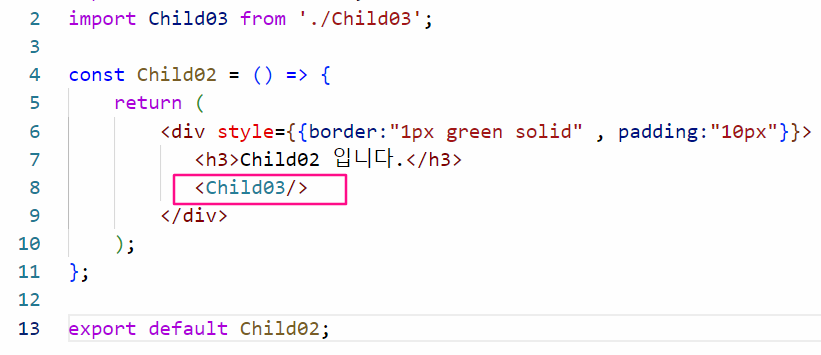
**Context01.jsx**

****

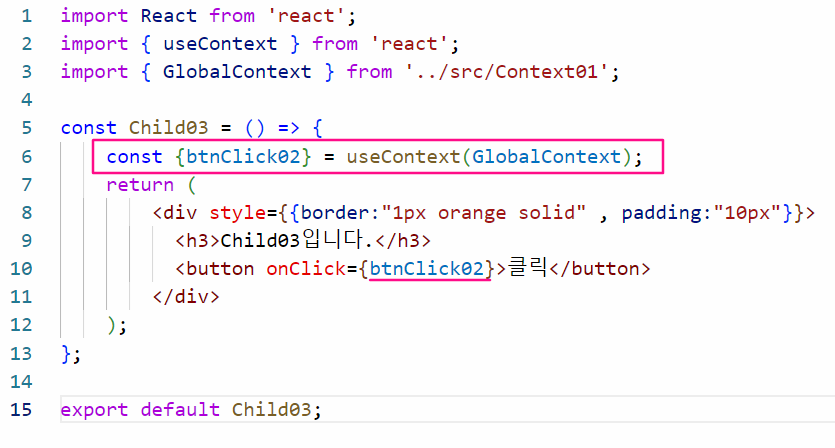
**Child01.jsx**

****

**Child02.jsx**

****

**Child03.jsx**

****