React

useMemo

복잡한 계산 결과, 객체 참조, 상수 배열/객초 초기화

복잡한 계산 결과

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

function ProductDisplay({ products, filterText }) {
 // filterText나 products가 변경될 때만 필터링을 다시 수행
 const filteredProducts = useMemo(() => {
   console.log('상품 필터링 중...'); // 불필요한 재렌더링 시에는 실행 안됨
   return products.filter(product =>
     product.name.includes(filterText)
   );
 }, [products, filterText]);
 return (
   <div>
     {filteredProducts.map(p => <div key={p.id}>{p.name}</div>)}
   </div>
 );
}
```

```
function App() {
 const [text, setText] = useState('');
 const allItems = [{id: 1, name: '사과'}, {id: 2, name: '바나나'}, {id: 3,
name: '딸기'}];
 // 다른 상태가 변경되어 App 컴포넌트가 재렌더링돼도 ProductDisplay는 필터링을 다시
하지 않음
 return (
   <>
     <input value={text} onChange={e => setText(e.target.value)} />
     <ProductDisplay products={allItems} filterText={text} />
   </>
 );
}
```

객체 참조

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

const UserInfo = memo(({ user }) => {
 console.log('UserInfo 렌더링됨:', user.name);
 return <p>{user.name} ({user.age})</p>;
});

function App() {
 const [name, setName] = useState('김철수');
 const [age, setAge] = useState(30);
 const [count, setCount] = useState(0); // 다른 상태
```

```
// name 또는 age가 변경될 때만 새로운 user 객체 생성
 const memoizedUser = useMemo(() => {
   console.log('새로운 user 객체 생성 중...');
   return { name, age };
 }, [name, age]);

 return (
   <>
     <input value={name} onChange={e => setName(e.target.value)} />
     <input type="number" value={age} onChange={e =>
setAge(Number(e.target.value))} />
     <button onClick={() => setCount(count + 1)}>카운트:
{count}</button>
     <UserInfo user={memoizedUser} />
   </>
 );
}
```

상수 배열/객체 초기화(일반적 사용)

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

function MyComponent() {
 // 이 배열은 컴포넌트 마운트 시 한 번만 생성되고 이후
재사용됨
 const staticOptions = useMemo(() => {
   console.log('staticOptions 생성됨');
   return [
     { id: 1, label: '옵션 A' },
     { id: 2, label: '옵션 B' },
     { id: 3, label: '옵션 C' },
   ];
 }, []); // 빈 의존성 배열
```

```
return (
   <div>
     <h3>정적 옵션 목록:</h3>
     <ul>
       {staticOptions.map(option => (
         <li key={option.id}>{option.label}</li>
       ))}
     </ul>
   </div>);}
function App() {
 const [toggle, setToggle] = useState(false);
 return (
   <>
     <button onClick={() => setToggle(!toggle)}>토글</button>
     {toggle && <MyComponent />} {/* 토글 시 MyComponent 렌더링 */}
   </>);}
```

useRef

DOM 요소에 직접 접근 (가장 일반적인 사용 사례)
렌더링에 영향을 주지 않는 값 저장
이전 prop 또는 상태 값 저장

DOM 요소에 직접 접근 (가장 일반적인 사용 사례)

```jsx
import React, { useRef } from 'react';

function FocusInput() {
 const inputRef = useRef(null); // input 요소에 연결할 ref 생성

 const handleFocusClick = () => {
   // current 속성을 통해 DOM 요소에 접근하여 focus() 메서드 호출
   if (inputRef.current) {
     inputRef.current.focus();
   }
 };
 const handleChangePlaceholder = () => {
   // current 속성을 통해 DOM 요소에 접근하여 placeholder 변경
   if (inputRef.current) {
     inputRef.current.placeholder = "새로운 플레이스홀더!";
   }
 };
```

```
 return (
   <div>
     <input type="text" ref={inputRef} placeholder="여기에 입력하세요" />
     <button onClick={handleFocusClick}>인풋에 포커스</button>
     <button onClick={handleChangePlaceholder}>플레이스홀더 변경</button>
   </div>
 );
}

export default FocusInput;
```

렌더링에 영향을 주지 않는 값 저장

```jsx
import React, { useState, useRef, useEffect } from 'react';
function Timer() {
 const [count, setCount] = useState(0);
 const intervalRef = useRef(null); // 타이머 ID를 저장할 ref
 useEffect(() => {
   // 컴포넌트가 마운트될 때 타이머 시작
   intervalRef.current = setInterval(() => {
     setCount(prevCount => prevCount + 1);
   }, 1000);

   // 컴포넌트가 언마운트될 때 타이머 정리
   return () => {
     if (intervalRef.current) {
       clearInterval(intervalRef.current);}};
 }, []); // 빈 의존성 배열로 컴포넌트 마운트/언마운트 시에만 실행
```

```
  const handleStopTimer = () => {
    if (intervalRef.current) {
      clearInterval(intervalRef.current);
      console.log('타이머 정지됨!');
      intervalRef.current = null; // 정지 후 intervalRef를 null로 설정하여 재시작
방지
    }
  };
  return (
    <div>
      <p>카운트: {count}</p>
      <button onClick={handleStopTimer}>타이머 정지</button>
    </div>
  );
}
export default Timer;
```

이전 prop 또는 상태 값 저장

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

function ValueTracker({ value }) {
 const prevValueRef = useRef(); // 이전 값을 저장할 ref

 // value prop이 변경될 때마다 prevValueRef.current에 현재 value를 저장
 useEffect(() => {
   prevValueRef.current = value;
 }, [value]); // value prop이 변경될 때마다 이 이펙트가 실행

 const prevValue = prevValueRef.current;
```

```
 return (
   <div>
     <p>현재 값: {value}</p>
     <p>이전 값: {prevValue !== undefined ? prevValue : '없음'}</p>
     {value !== prevValue && prevValue !== undefined && (
       <p style={{ color: 'blue' }}>값이 {prevValue}에서 {value}로 변경되었습니다!</p>
     )}
   </div>);}
function App() {
 const [myValue, setMyValue] = useState(0);
 return (
   <div>
     <button onClick={() => setMyValue(myValue + 1)}>값 증가</button>
     <ValueTracker value={myValue} />
   </div>);}
export default App;
```

useContext

사용자 테마 변경 (Context의 가장 흔한 사용 사례)
사용자 인증 상태 관리
다국어(언어) 설정 관리

사용자 테마 변경 (Context의 가장 흔한 사용 사례)

테마 변경 예제

현재 테마: light (클릭하여 변경)

이 단락은 현재 테마를 따릅니다.

테마 변경 예제

현재 테마: dark (클릭하여 변경)

이 단락은 현재 테마를 따릅니다.

```jsx
import React, { createContext, useContext, useState } from 'react';
// 1. Context 생성: 테마 Context
const ThemeContext = createContext(null); // 초기값 null
// 2. Theme Provider 컴포넌트
function ThemeProvider({ children }) {
 const [theme, setTheme] = useState('light'); // 'light' 또는 'dark'
 const toggleTheme = () => {
   setTheme(prevTheme => (prevTheme === 'light' ? 'dark' : 'light'));
 };
 // Provider의 value prop으로 theme과 toggleTheme 함수를 전달
 const contextValue = { theme, toggleTheme };
 return (
   <ThemeContext.Provider value={contextValue}>
     {children}
   </ThemeContext.Provider>
 );
}
```

```jsx
// 3. Context를 사용하는 컴포넌트 (소비자)
function ThemedButton() {
 // useContext 훅을 사용하여 ThemeContext의 현재 값(theme, toggleTheme)에 접근
 const { theme, toggleTheme } = useContext(ThemeContext);
 const buttonStyle = {
   backgroundColor: theme === 'light' ? 'white' : 'black',
   color: theme === 'light' ? 'black' : 'white',
   border: '1px solid gray',
   padding: '10px 20px',
   cursor: 'pointer',
 };
 return (
   <button style={buttonStyle} onClick={toggleTheme}>
     현재 테마: {theme} (클릭하여 변경)
   </button>
 );
}
```

```
// 4. Context를 사용하는 또 다른 컴포넌트 (소비자)
function ThemedParagraph() {
 const { theme } = useContext(ThemeContext);
 const paragraphStyle = {
   color: theme === 'light' ? 'black' : 'white',
   backgroundColor: theme === 'light' ? '#eee' : '#333',
   padding: '15px',
   borderRadius: '5px'
 };
 return <p style={paragraphStyle}>이 단락은 현재 테마를
따릅니다.</p>;
}
```

```
/ 최상위 컴포넌트: Provider로 감싸서 하위 컴포넌트에 Context 제공
function App() {
 return (
  <ThemeProvider>
   <div style={{ padding: '20px', display: 'flex', flexDirection: 'column',
                 gap: '20px', minHeight: '100vh' }}>
       <h1>테마 변경 예제</h1>
       <ThemedButton />
       <ThemedParagraph />
     </div>
   </ThemeProvider>
 );
}
export default App;
```

사용자 인증 상태 관리

인증 상태 예제

로그인해주세요.

Peter 로그인

인증 상태 예제

환영합니다, Peter님!

로그아웃

```
import React, { createContext, useContext, useState } from 'react';
const AuthContext = createContext(null);
function AuthProvider({ children }) {
 const [isLoggedIn, setIsLoggedIn] = useState(false);
 const [userName, setUserName] = useState('');
 const login = (name) => { setIsLoggedIn(true); setUserName(name);};
 const logout = () => { setIsLoggedIn(false);  setUserName(''); };
const authContextValue = { isLoggedIn, userName, login, logout };
 return (
   <AuthContext.Provider value={authContextValue}>
     {children}
   </AuthContext.Provider> );}
```

```
function UserGreeting() {
 const { isLoggedIn, userName } = useContext(AuthContext);
 return (
   <h3>
     {isLoggedIn ? `환영합니다, ${userName}님!` : '로그인해주세요.'}
   </h3>
 );
}
```

```
function AuthButtons() {
 const { isLoggedIn, login, logout } = useContext(AuthContext);
 const [inputName, setInputName] = useState('');
 const handleLogin = () => {
   if (inputName) {
     login(inputName);
     setInputName('');
   }
 };
```

```
return (
 <div>
   {isLoggedIn ? (
     <button onClick={logout}>로그아웃</button>
   ) : (
     <>
       <input
         type="text"
         value={inputName}
         onChange={(e) => setInputName(e.target.value)}
         placeholder="사용자 이름"
       />
       <button onClick={handleLogin}>로그인</button>
     </>
   )}
 </div>);}
```

```jsx
function App() {
 return (
   <AuthProvider>
     <div style={{ padding: '20px' }}>
       <h1>인증 상태 예제</h1>
       <UserGreeting />
       <AuthButtons />
       {/* 다른 어떤 컴포넌트도 AuthContext를 useContext하여 인증 상태에 접근 가능
*/}
     </div>
   </AuthProvider>
 );
}
export default App;
```

다국어(언어) 설정 관리

다국어 설정 예제

Hello

This is an example of language switching using Context API.

Change Language (한국어)

다국어 설정 예제

안녕하세요

이것은 Context API를 이용한 언어 전환 예제입니다.

언어 변경 (English)

```jsx
import React, { createContext, useContext, useState } from 'react';
const LanguageContext = createContext(null);
const translations = {
 en: {
   greeting: 'Hello',
   buttonText: 'Change Language',
   paragraph: 'This is an example of language switching using Context API.'
 },
 ko: {
   greeting: '안녕하세요',
   buttonText: '언어 변경',
   paragraph: '이것은 Context API를 이용한 언어 전환 예제입니다.'
 }
};
```

```
function LanguageProvider({ children }) {
 const [language, setLanguage] = useState('en'); // 'en' 또는 'ko'
 const toggleLanguage = () => {
   setLanguage(prevLang => (prevLang === 'en' ? 'ko' : 'en'));
 };
 // 현재 언어의 번역 텍스트와 언어 변경 함수를 Context로 제공
 const langContextValue = {
   texts: translations[language],
   toggleLanguage,
   currentLanguage: language
 };
 return (
   <LanguageContext.Provider value={langContextValue}>
     {children}
   </LanguageContext.Provider>
 );}
```

```jsx
function LanguageDisplay() {
 const { texts, toggleLanguage, currentLanguage } = useContext(LanguageContext);
 return (
   <div>
     <h2>{texts.greeting}</h2>
     <p>{texts.paragraph}</p>
     <button onClick={toggleLanguage}>
       {texts.buttonText} ({currentLanguage === 'en' ? '한국어' : 'English'})
     </button>
   </div>
 );
}
```

```jsx
function App() {
 return (
   <LanguageProvider>
     <div style={{ padding: '20px', border: '1px solid #ccc', borderRadius: '8px' }}>
       <h1>다국어 설정 예제</h1>
       <LanguageDisplay />
     </div>
   </LanguageProvider>
 );
}
export default App;
```