# 리액트 훅 사용 주관식 문제 (10문제)

## 문제 1: useState 기초

현재 카운터의 값이 `0`입니다. 버튼을 클릭하면 카운터가 1씩 증가해야 합니다. 아래 코드에서 `???`로 표시된 부분을 채워주세요.

```jsx

import React from 'react';

function Counter() {

const [count, ???] = ???(???);

return (

<div>

<p>현재 카운터: {count}</p>

<button onClick={() => ???(count + 1)}>증가</button>

</div>

);

}

export default Counter;

```

\*\*답안:\*\*

```

setCount

useState

0

setCount

```

## 문제 2: useEffect 실행 조건

아래 코드에서 콘솔에 "마운트되었습니다!"가 출력되는 조건을 설명하고, 의존성 배열에 숫자 `1`을 추가하면 어떻게 되는지 설명하세요.

```jsx

import React, { useEffect, useState } from 'react';

function MyComponent() {

const [num, setNum] = useState(0);

useEffect(() => {

console.log('마운트되었습니다!');

}, []); // 빈 배열

return (

<div>

<button onClick={() => setNum(num + 1)}>클릭: {num}</button>

</div>

);

}

```

\*\*답안:\*\*

```

빈 배열은 컴포넌트가 처음 마운트 시 한 번만 useEffect가 실행됩니다.

의존성 배열에 1을 추가하면 useEffect는 마운트 시에만 실행되고, num 값이 변경되어도 "마운트되었습니다!"는 출력되지 않습니다.

(num과 1은 항상 같지 않으므로 의존성이 변경되었다고 감지되지 않음)

```

## 문제 3: useEffect 클린업

아래 코드에서 타이머를 사용하여 "카운트다운: 5"에서 0까지 카운트하고, 컴포넌트가 언마운트되거나 카운트가 0에 도달했을 때 타이머를 정리해야 합니다. `???`를 채워주세요.

```jsx

import React, { useEffect, useState } from 'react';

function Countdown() {

const [count, setCount] = useState(5);

useEffect(() => {

const timer = setInterval(() => {

???;

}, 1000);

return () => {

???;

};

}, [count]);

return <div>카운트다운: {count}</div>;

}

```

\*\*답안:\*\*

```

setCount(count - 1)

clearInterval(timer)

```

## 문제 4: useRef 사용법

아래 코드에서 입력 필드가 화면에 그려질 때 자동으로 포커스가 되도록 하려면 `???`를 어떻게 채워야 하나요?

```jsx

import React, { useRef, useEffect } from 'react';

function AutoFocus() {

const inputRef = ???;

useEffect(() => {

???.focus();

}, []);

return <input ref={???} type="text" placeholder="자동으로 포커스됩니다" />;

}

```

\*\*답안:\*\*

```

useRef(null)

inputRef.current

inputRef

```

## 문제 5: useContext 기본 사용

아래 코드에서 `ThemeContext`를 사용하여 테마를 가져오려면 `???`를 어떻게 채워야 하나요?

```jsx

import React, { createContext, useContext } from 'react';

const ThemeContext = createContext('light');

function App() {

return (

<ThemeContext.Provider value="dark">

<Child />

</ThemeContext.Provider>

);

}

function Child() {

const theme = ???(???);

return <div>현재 테마: {theme}</div>;

}

```

\*\*답안:\*\*

```

useContext

ThemeContext

```

## 문제 6: useState 함수형 업데이트

아래 코드에서 클릭 이벤트 핸들러에서 `???`를 사용하여 상태를 업데이트해야 합니다. 함수형 업데이트를 사용해서 작성하세요.

```jsx

import React, { useState } from 'react';

function DoubleCounter() {

const [count, setCount] = useState(0);

const handleClick = () => {

???((prev) => prev + 1);

???((prev) => prev + 1);

};

return (

<div>

<p>카운트: {count}</p>

<button onClick={handleClick}>두 번 증가</button>

</div>

);

}

```

\*\*답안:\*\*

```

setCount

setCount

```

\*\*설명:\*\*

함수형 업데이트를 사용하면 이전 상태 값을 기반으로 상태를 업데이트할 수 있습니다.

두 번의 setCount 호출이 모두 이전 값을 기준으로 계산되므로 최종적으로 2가 증가합니다.

## 문제 7: useMemo 기본 사용

아래 코드에서 `expensiveValue`는 값비싼 계산을 수행합니다. `number`가 변경될 때만 재계산하도록 `???`를 사용하여 최적화하세요.

```jsx

import React, { useState, ??? } from 'react';

function ExpensiveComponent() {

const [number, setNumber] = useState(0);

const [toggle, setToggle] = useState(false);

const expensiveValue = ???(() => {

console.log('비용이 많이 드는 계산 실행');

return number \* number;

}, [???]);

return (

<div>

<p>결과: {expensiveValue}</p>

<button onClick={() => setNumber(number + 1)}>숫자 증가: {number}</button>

<button onClick={() => setToggle(!toggle)}>토글</button>

</div>

);

}

```

\*\*답안:\*\*

```

useMemo

useMemo

number

```

## 문제 8: useCallback 기본 사용

아래 코드에서 자식 컴포넌트에 전달되는 `handleClick` 함수를 메모이제이션하여 불필요한 재렌더링을 방지하세요. `???`를 채워주세요.

```jsx

import React, { useState, ??? } from 'react';

function Parent() {

const [count, setCount] = useState(0);

const handleClick = ???(() => {

console.log('클릭됨!');

}, []);

return (

<div>

<p>카운트: {count}</p>

<button onClick={() => setCount(count + 1)}>증가</button>

<Child onClick={handleClick} />

</div>

);

}

function Child({ onClick }) {

return <button onClick={onClick}>클릭</button>;

}

```

\*\*답안:\*\*

```

useCallback

useCallback

```

## 문제 9: useEffect로 API 호출

아래 코드에서 컴포넌트가 마운트될 때 API를 호출하여 데이터를 가져오도록 `???`를 채워주세요.

```jsx

import React, { useState, useEffect } from 'react';

function UserProfile() {

const [user, setUser] = useState(null);

const [loading, setLoading] = useState(true);

???(() => {

fetch('https://api.example.com/user')

.then(res => res.json())

.then(data => {

???(data);

???(false);

});

}, []);

if (loading) return <div>로딩 중...</div>;

return <div>사용자: {user?.name}</div>;

}

```

\*\*답안:\*\*

```

useEffect

setUser

setLoading

```

## 문제 10: useContext와 Provider

아래 코드에서 `UserContext`를 사용하여 사용자 정보를 공유하도록 `???`를 채워주세요.

```jsx

import React, { createContext, useContext, useState } from 'react';

const UserContext = ???(null);

function App() {

const [user, setUser] = useState({ name: '홍길동', age: 25 });

return (

<UserContext.??? value={{ user, setUser }}>

<Profile />

</UserContext.???>

);

}

function Profile() {

const { user } = ???(???);

return (

<div>

<p>이름: {user.name}</p>

<p>나이: {user.age}</p>

</div>

);

}

```

\*\*답안:\*\*

```

createContext

Provider

Provider

useContext

UserContext

```

---

## 보너스 문제: useEffect와 상태 업데이트

아래 코드에서 버튼을 클릭하면 상태가 업데이트되지만, useEffect 내부의 콘솔 로그가 무한히 반복됩니다. 어떻게 수정해야 하나요?

```jsx

import React, { useEffect, useState } from 'react';

function ProblemComponent() {

const [value, setValue] = useState(0);

useEffect(() => {

console.log('값이 변경되었습니다:', value);

setValue(value + 1); // 문제의 코드

});

return <button onClick={() => setValue(value + 1)}>클릭: {value}</button>;

}

```

\*\*답안:\*\*

```

useEffect의 두 번째 인자로 의존성 배열을 추가해야 합니다.

// 수정 방법 1: 의존성 배열 추가

useEffect(() => {

console.log('값이 변경되었습니다:', value);

}, [value]); // value가 변경될 때만 실행

// 또는 수정 방법 2: useEffect 내부의 setValue 제거

useEffect(() => {

console.log('값이 변경되었습니다:', value);

}); // 컴포넌트가 렌더링될 때마다 실행 (setValue 제거)

```

---

## 채점 기준

| 문제 | 배점 | 주요 평가 기준 |

|------|------|----------------|

| 1 | 10점 | useState의 기본 사용법 이해 |

| 2 | 10점 | useEffect의 의존성 배열 이해 |

| 3 | 10점 | useEffect 클린업 함수 이해 |

| 4 | 10점 | useRef의 기본 사용법 이해 |

| 5 | 10점 | useContext의 기본 사용법 이해 |

| 6 | 10점 | useState 함수형 업데이트 이해 |

| 7 | 10점 | useMemo의 기본 사용법 이해 |

| 8 | 10점 | useCallback의 기본 사용법 이해 |

| 9 | 10점 | useEffect API 호출 패턴 이해 |

| 10 | 10점 | useContext와 Provider 조합 이해 |

\*\*총점: 100점\*\*