

# [2022-11-18] 2반 실습 파일

---

## 오늘의 Takeaway~

가변하는 값  
관리:  
**useState!!**

극단적인  
상황까지  
생각

**Divide and  
Conquer** 큰  
문제를 -> 작은  
문제로

혹의 최적화 가이드  
라인?  
**component**가  
반드시 필요한  
렌더링 만을  
진행하는가?

### 학습 목표

- React를 구성하는 props, hooks, state, 이벤트 처리 등의 개념 학습
  - JSX 스타일로 Props, State를 사용하여 UI를 표현하는 방법을 학습
  - React Hooks에 대해서 알아보고 어떤 방식으로 사용되는지 이해하고 학습
- 

### 문제풀이

#### 1. Counter 구현하기

##### useState hook?

useState를 사용하게되면 state 변수 및 상태값을 업데이트 시킬 수 있는 setter 함수를 리턴하게 됩니다. 아래와 같은 \*구조 할당 방식으로 쉽게 state 변수 및 setter 함수를 선언 + 초기화 시켜줄 수 있습니다.

```
| const [<<상태변수>>, <<setter함수>>] = useState(<<initial state>>);
```

세터함수 네이밍 컨벤션 (명명법)?

함수 앞에 set- 접두어 및 + state 명을 뒤에 lowerCamelCase 형태로 명명한다.

(\*구조 할당 방식(==구할당==Destructuring assignment): 배열이나 객체의 속성을 해체하여 그 값을 개별 변수에 담을 수 있게 하는 JavaScript 표현식입)

+구조 할당 방식 예제 (실습시간에 만든 빌딩 건물 예제!!)

```
// 구조 할당 방식 예제
const building = {
  level1: {
    room101: "엽기떡볶이",
    room102: "스타벅스"
  },
  level2: {
    room201: "코인노래방",
    room202: "명랑핫도그"
  }
}

const {
  level1: {room101, room102},
  level2: {room201, room202},
} = building;

// 구할당이라는 개념이 왜 계속 리액트에 등장합니까??!!
// props를 디셔너리 (오브젝트)를 바로 출력하려고 해보면 에러발생 하기 때문에
// 오브젝트.<<keyname>>으로 꺼내주어야 하는데요,
// 매번 object에서 꺼내는 코드가 귀찮으면 구할당을 사용합니다

console.log(room102)

// 구할당이 실제 사용되는 예시!
// 유저가 개인 프로필 페이지에서 name, email, username, location을 업데이트하는 post 요청을 보냈을 때~
// 그 데이터를 이렇게 데이터 받아올 수도 있겠죠?
const {
  session: {
    user: { _id },
  },
  body: { name, email, username, location: {street, country} },
} = req;

// 그.런.데
// 구할당을 안쓰고 바로 꺼낸다고 할 때...
console.log(building.level3.room301)

// 추가 설명: Optional chaining. ?. 문법 설명 (점 찍는게 2개 이상일 때 잘 쓰임)
// undefined 값에 없는 키를 찾으려고 하면 에러 발생
// 에러 발생하면 그 이후 코드 하나도 작동하지 않음
// 대신 undefined 뒤 ?.로 키를 찾게되면,
console.log(building.level3?.room301) // 에러가 출력 ㄴㄴ but undefined 할당

// 언제 쓸 수 있을까! location 정보는 선택일 때 유저가 location을 기입하지 않은 상태로 street를 꺼내려하면 에러 발생
// 이런식으로 에러 발생 안시키고 넘어갈 수 있음당
// 자바스크립트는 에러가 생기면 그 다음코드는 실행이 안돼요 그래서 나중에 쓸 일이 있을 겁니다
```

```
const street = req.body.location?.street
```

### **생각로작> useState를 사용함을 몰라도 다음과 같은 생각이 든다면 아주 굿!**

숫자가 변하네? → 값이 가변할 때는 그 값을 state로 관리해줄 수 있지 않을까? & 그 state를 쉽게 업데이트를 시킬 수 있는 도구가 뭐가 있었지? → useState hook을 사용할 수 있지 않을까?

#### ▼ 문제 정답

```
// App.js

import React from 'react';
// Counter 함수 import
import Counter from './components/Counter.js'

function App() {
  return (
    <div>
      // Counter 함수 컴포넌트 삽입
      <Counter />
    </div>
  );
}

export default App;
```

```
// components/Counter.js

// react 라이브러리에서 useState 불러오기
import React, { useState } from 'react' // 지난 강의 비교: export default & 보통 export
import "../index.css"

function Counter(){
  // useState 함수 (초기값 0) 의 결과를 counter, setCounter 변수에 할당
  // useState는 맨처음 초기값을 넣어주면서 컴포넌트가 첫 렌더링될 때 한번 실행됩니다
  const [counter, setCounter] = useState(0)

  const increaseCounter = () => {
    setCounter(counter => counter + 1)
  }
  const decreaseCounter = () => {
    // counter 값을 1 줄이도록 코드 작성
    setCounter(counter => counter - 1) // 이전 값 참조 state 업데이트
  }
}
```

```

    return(
      <div className="counter">
        <h2> 카운터 </h2>
        <div>{ counter }</div>
        <button onClick={increaseCounter}>증가하기</button>
        // 함수 객체를 전달하되, 호출은 하지 않습니다.
        // increaseCounter(); 형태로 전달해주면 렌더링 될때마다 실행이 됩니다. 그게 싫어 함수 그대로 전달해줍니다!
        // vanilla.js 쓸 때는 onClick="함수()" 이렇게 문자열로 바꿔 전달해주는데 단지 표현 방식의 차이입니다.
        <button onClick={decreaseCounter}>감소하기</button>
      </div>
    );
  }

  export default Counter

```

## ▼ 문제 확장

### 추가 태스크: 만약 카운터가 state값이 0으로 초기화되기 직전 “카운터가 생성되었습니다!”를 출력하고 싶다면? [시간 남으면 진행]

아래 코드처럼 상태값을 초기화하기 전 이렇게 추가하고싶은 코드가 있다면 상태값을 반환하는 함수가 실행된 형태 (함수();)로 useState에 넘겨줄 수 있습니다.

```

const App = () => {

  const [count, setCount] = useState(countInitial());
  ...
  function countInitial(){
    console.log("카운터가 생성되었습니다!")
    return 0;
  }
  ...
}

```

### \*\*데이터 타입에 따른 이전 상태값 참조! (다음 강의 설명 예정)[시간 남으면 진행]

```

useState((prev)=> prev+1); // number

useState((prev)=> {...prev, newID: newValue}) // object

useState((prve)=> [...prev, newValue]) // array

```

## 2. 사이드 효과로 웹페이지 타이틀 변경하기

### useEffect hook?

useEffect의 구조는 다음과 같습니다. (참고할 수 있는 영상: [useEffect with cleanup](#))

- **이펙트 함수**: 마운트(어디에 놓다)와 같은 개념으로, 리액트에서는 컴포넌트가 처음 돔에 로드될 때/되기 직전(마운트될 때) 혹은 의존값이 업데이트될 때 실행됩니다.
- **클린업 함수**: 언마운트(제거된다)와 같은 개념으로, 리액트에서는 컴포넌트가 돔에서 제거될 때/되기 직전(언마운트될 때) **뒷정리해주는** 함수라고 생각해주시면 됩니다.
- **의존값**: 이 값을 기준으로 변경사항을 감지하겠다!!

```
useEffect(() => {  
  (이펙트 함수)  
  return {  
    (클린업 함수)  
  };  
}, [의존값]);
```

예제...

```
import React from 'react'  
  
// strictmode를 꺼줘야만 작동하는데...왜 그런지는 저도 공부해보겠습니다...  
  
const Welcome = ({ display }) => {  
  useEffect(() => {  
    alert("안녕"); // 마운드 되기 직전  
    return () => {  
      alert("바이바이"); // 언마운트 되기 직전  
    };  
  }, [display]);  
  return <h1>Hola</h1>;  
};  
  
export default function App() {  
  const [display, setDisplay] = useState(true);  
  const handleClick = () => {  
    setDisplay((display) => !display);  
  };  
  return (  
    <div className="App">  
      {display ? <Welcome display={display} /> : null}  
      <button onClick={handleClick}>Button</button>  
    </div>  
  );  
}
```

위 코드는 단순히 마운트되는 순간 언마운트되는 순간을 보여주기 위한 코드이고,  
실제 사용되는 경우로는...

**이펙트 함수: 이벤트 리스너를 붙여준(addEventListener) 특정 DOM이 화면에서 제거될 때 (언마운트 될 때)**

**클린업 함수: 이벤트 리스너를 제거하는(removeEventListener) 코드...**

왜 필요하나요?

없어진 html태그에 이벤트 리스너를 달아주면 warning 발생!! 또 메모리 누수를 막기 위해서 사용. 모던 브라우저는 Garbage collector (쓰레기 수거자...?)가 removeEventListener를 보고 불필요한 이벤트 리스너를 삭제해줍니다.

**이펙트 함수: 혹은 setInterval**

**클린업 함수: clearInterval 등에 사용됩니다**

왜 필요하나요?

같은 원리로 위 Garbage collector에게 알려줍니다.

**생각로직> useEffect를 사용함을 몰라도 다음과 같은 생각이 든다면 아주 굿!**

값 A가 상태값이 변함에 따라 업데이트 되었으면 좋겠네? → 그 상태값을 의존값으로 하는 useEffect 혹은 사용하면 되지 않을까?

(팁 혹은 최적화 설명 및 3가지 접근법 미리 설명하고 투표하기.)

## ▼ 문제 정답

```
// App.js

import React from 'react';
// Counter 함수 import
import Counter from './components/Counter.js'

function App() {
  return (
    <div>
      // Counter 함수 컴포넌트 삽입
      <Counter />
    </div>
  );
}

export default App;
```

```
// components/Counter.js

// react 라이브러리에서 useState 불러오기
import React, { useState, useEffect } from 'react';
import '../index.css';

function Counter() {
  // useState 함수 (초기값 0) 의 결과를 counter, setCounter 변수에 할당
  const [counter, setCounter] = useState(0);

  const increaseCounter = () => {
    // counter 값을 1 늘리도록 코드 작성
    setCounter(counter => counter + 1);
  };
  const decreaseCounter = () => {
    // counter 값을 1 줄이도록 코드 작성
    setCounter(counter => counter - 1);
  };

  useEffect(() => {
    document.title = `현재 카운터 숫자: ${counter}`;
  }); // **Hook의 최적화 가이드라인?

  return (
    <div className="counter">
      <h2> 카운터 </h2>
      <div>{counter}</div>
      <button onClick={increaseCounter}>증가하기</button>
      <button onClick={decreaseCounter}>감소하기</button>
    </div>
  );
}

export default Counter;
```

## ▼ 문제 확장

### **\*\*Hook의 최적화 가이드라인?**

혹을 최적화하기 위해서는 Component가 반드시 필요한 리렌더링만을 진행하는가? 라는 질문이 필요합니다.

#### **Case 1. deps (의존값)이 없다면?**

deps 파라미터를 생략한다면, 컴포넌트가 리렌더링 될 때마다 호출이 됩니다.

컴포넌트가 언제 리렌더링 되는가? 내부 state나 받는 props가 하나라도 업데이트될 때 리렌더링 됩니다.

따라서 선언된 상태값이 예제에서처럼 counter 하나가 아닌 여러개인 경우 렌더링(관련 없는 state가 업데이트) 할 때마다 사용되어 성능에 좋지 않습니다.

```
useEffect(() => {  
  document.title = `현재 카운터 숫자: ${counter}`;  
});
```

### Case 2. deps (의존값)에 특정 값을 넣는다면?

deps에 특정 값을 넣게 된다면, 컴포넌트가 처음 마운트 될 때에도 호출이 되고, 지정한 값이 바뀔 때에도 호출이 됩니다. counter가 변경될 때만 이펙트 함수가 작동하므로 Case 1보다 성능에 더 좋습니다.

```
useEffect(() => {  
  document.title = `현재 카운터 숫자: ${counter}`;  
}, [counter]);
```

## 3. 컴포넌트 합성

### ▼ 문제 정답

```
import React from 'react';  
import './App.css';  
  
//함수 컴포넌트를 사용하여 Subject컴포넌트를 정의합니다.  
function Subject(props) {  
  return <h3>React를 이해하기 위해서는 {props.name}을(를) 알아야 합니다.</h3>;  
} // **컴포넌트에서 props를 넘겨주는 방법  
  
//App 컴포넌트를 정의합니다.  
function App() {  
  // 여러가지 방법으로 풀어봅시다  
  // **Loop inside React JSX  
  return (  
    <div>  
      <Subject name="HTML" />  
      <Subject name="CSS" />  
      <Subject name="JavaScript" />  
    </div>  
  );  
}
```



```
//생성한 컴포넌트를 export하여 index.js에서 렌더링합니다.  
export default App;
```

## ▼ 문제 확장

### \*\*컴포넌트에서 props를 넘겨주는 방법

방법 1. {<<개별 props>>, <<개별 props2>> ...}

방법 2. props

### \*\*Loop inside React JSX

코딩을 할 때 항상 극단적인 경우까지 생각해보는게 큰 도움이 됩니다. 웹개발 언어 3개가 아닌 수백 개 수만개가 된다고 할 때, 모든 컴포넌트를 일일이 추가할 수는 없는 노릇입니다.

```
// 방법 1. 단순한 수기 입력  
return (  
  <div>  
    <Subject name="HTML" />  
    <Subject name="CSS" />  
    <Subject name="JavaScript" />  
  </div>  
);  
  
// 방법 2. JSX내 map을 사용하여 반복문을 돌려줍니다.  
// 단, 컴포넌트 배열을 렌더링했을 때 어떤 원소에 변동이 있었는지 알아내기 위해 key props를 필요로 합니다  
  
const name_lst = ['HTML', 'CSS', 'JavaScript']; // 변수명은 names같은 복수형을 지양해주세요  
return (  
  <div>  
    {name_lst.map(function (lang, idx) {  
      return <Subject name={lang} key={idx} />;  
    })}  
  </div>  
);  
  
// 방법 3. map + es6 최신 자바스크립트  
  
const name_lst = ['HTML', 'CSS', 'JavaScript'];  
return (  
  <div>  
    // 함수 결과값 표기 () == {return }  
    {name_lst.map((lang, idx) => (  
      <Subject name={lang} key={idx} />  
    ))}  
  </div>  
);
```

```
);
```

💡 **Tips. JSX 내 반복문은 map!**

## 4. React Hook 만들기

### Hook이란?

#### 추가설명자료

그냥 함수입니다. 단 state를 핸들하는...컴포넌트에서 재사용할 수 있는 함수를 분리해서 만들었습니다

종류로는 크게 State Hook (e.g. useState) / Effect Hook (e.g. useEffect) / Context Hook (e.g. useContext) 등등... 또 직접 만들 수 있는 Custom Hook이 있습니다.

단, 커스텀 훅을 작성하는 룰이 있습니다.

- use- 로 시작해야합니다
- 일반적인 함수에서 부르면 안돼요. 컴포넌트 & 커스텀 훅에서는 사용해도 괜찮습니다

커스텀 훅은 src > hooks 폴더 > 커스텀훅명.js 이런식으로 분리해서 사용할 수 있습니다

### 커스텀 훅 예시

#### 1. useFetch 훅 만들기

```
import { useState, useEffect } from "react";

const useFetch = (url) => {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch(ur
      .then((res) => res.json())
      .then((data) => setData(data));
    }, [url]);

  return [data];
};
```

```
export default useFetch; // 코드 소스:https://www.w3schools.com/react/react_customhooks.asp
```

## 2. useToggle 훅 만들

```
// 커스텀 토글 훅 만들기 전

import { useEffect, useState } from "react";
import "./styles.css";

export default function App() {
  const [toggle, setToggle] = useState(false);
  const [message, setMessage] = useState("Click");

  const handleToggle = () => {
    setToggle((toggle) => !toggle);
  };

  useEffect(() => {
    setMessage(toggle ? "Toggled" : "Click");
  }, [toggle]);

  return (
    <div className="App">
      <button onClick={handleToggle}>{message}</button>
    </div>
  );
}
```

```
// 커스텀 토글 훅 만들 후

import { useEffect, useState } from "react";
import "./styles.css";

const useToggle = (initialState = false) => {
  const [toggle, setToggle] = useState(initialState);
  const handleToggle = () => {
    setToggle((toggle) => !toggle);
  };
  return [toggle, handleToggle];
};

export default function App() {
  const [toggle, handleToggle] = useToggle(false);
  const [message, setMessage] = useState("Click");

  useEffect(() => {
    setMessage(toggle ? "Toggled" : "Click");
  }, [toggle]);

  return (
    <div className="App">
      <button onClick={handleToggle}>{message}</button>
    </div>
  );
}
```

```

    </div>
  );
}

```

## ▼ 심화학습: useMemo, useCallback [시간 남으면 진행]

### 참고자료


지금까지 살펴본 컴포넌트는 state 값이나 props 값이 변경될 때마다 재-렌더링되어 왔습니다. 이는 즉 그 내부에 선언되었었던 표현식 (함수 & 변수)도 매번 선언된다는 의미입니다.

이미 선언되었던 변수를 생성하고 함수를 생성하는 불필요한 작업 또한 CPU를 무리시키고, 이는 혹은 최적화 목표: 컴포넌트가 반드시 필요한 리렌더링만을 진행하는가?를 벗어나게 됩니다.

결론부터 말씀드리면 useMemo를 활용하면 deps가 업데이트 될 때까지 함수가 리턴한 값을 저장하고 리턴할 수 있습니다.

useCallback은 deps가 업데이트 될 때까지 함수를 새로 만들지 않고 저장하여 재사용할 수 있게 해줍니다.

아래 링크로 가 예제를 확인해봅시다. [예제코드](#)

 **Tips.** 하위 컴포넌트에게 callback 함수를 props로 넘길 때, 상위 컴포넌트에서 useCallback 으로 함수를 선언하는 것이 유용!

## ▼ 문제 정답

```

import ReactDOM from 'react-dom';
import './index.css';
import * as serviceWorker from './serviceWorker';
import React, { useState } from 'react';

const User = () => {
  const [nickname, setNickname] = useState('');
  const updateNickname = event => {
    // nickname 변수에 event를 이용해 사용자가 입력한 값을 삽입하세요.

    const nickname = event.target.value;
    // 부모 태그를 알고 싶다면 event.target.parentNode
    // 여기에 변화를 주어도 반영되지 않습니다. dom selector를 통해 불러오고 변경해주어야합니다.

    // setNickname의 매개변수로 nickname을 넘겨주세요.
    setNickname(nickname); // **바뀐 state로 로직을 짜는 경우?

    return (
      <div>
        <label>{nickname}</label>
        <br />
      </div>
    );
  };
};

```

```

    <input value={nickname} onChange={updateNickname} /> // lowerCamelCase
  </div>
);
};

ReactDOM.render(<User />, document.getElementById('root'));

serviceWorker.unregister();

```

## ▼ 문제 확장

### **\*\*바뀐 state로 로직을 짜는 경우?**

state가 변형 된 후 아래 함수를 실행하고 싶지만 문제가 있습니다.

바뀐 state를 활용하는 로직을 짤 경우 자바스크립트 비동기 처리 특성 때문에 (1)이 완료되기도 전에 (2) 함수가 실행되어 업데이트가 반영되지 않을 수 있습니다. 그런 경우, (3), (4) 번처럼 `useEffect` or `async await`을 통해 비동기 처리해줍니다.

여기선 문제가 안됩니다만 추후 api로 데이터를 받아올 때, 받아오고 그 데이터를 또 json으로 변경해서 또 그 중에 뭐를 꺼내서 상태 변수를 업데이트 해준다거나... 시간이 오래걸리는 코드 같은 경우 아래 `useEffect`나 `async- await` 을 써줄 수 있습니다.

```

setNickname(nickname); // --- (1)

const checkName = () => {
  if (nickname === '송강') {
    alert('감사합니다');
  } else if (nickname === '송강호') {
    alert('죄송합니다');
  }
}
checkName(); // --- (2)

useEffect(()=>{
  checkName();
}, [nickname]) // --- (3)

await setNickname(nickname);
...

checkName(); // --- (4)

```