SetInterval
커스터마이징
UseEffect
초단위 움직임

커스텀 훅
컴포넌트가 바뀔 때마다 그 부분만 업데이트를 시켜야 하기 때문에 useState를 사용하는데, 중복되는 코드는 하나로 묶어서 import 시키는 게 편하다.
그럴 때 사용하는 게 커스텀 훅인데, 유저가 입력하거나, 변경을 요청하면 그 때 이용된다.
코드를 변경할 때도 커스텀훅에서만 변경하면 되기 때문에 수정도 간단하다.
커스텀 훅은 앞에 use를 사용한다. useEffect가 커스텀 훅의 대표적인 예이다.

useEffect는 처음 렌더링이 되고 무조건 한 번 실행되는데, 원하는 변수가 변경 됐을 때, 그 부분만 업데이트 해준다.
SetInterval는 Hooks의 하나인데, 일정시간마다 함수가 실행되도록 처리한다. 그 예로, 1초마다 변수가 1씩 늘어나게 하는 등으로 업데이트 해주는 것이다.

그러나 JS 코드를 삽입하는 식으로 setInterval을 사용하면 자잘한 오류가 생긴다.

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

function App() {
    const [fontState, setFontState] = useState(5);

    setInterval(() => {
        setFontState(fontState + 1);
    }, 1000);
    // 글자 크기를 1씩 늘려주는데, 그 크기는 1000까지.

    return (
        <>
        <div style = {{ fontSize : fontState + "px"}}>Hello</div>
        </>
        // Hello의 글자 크기를 점점 늘려준다.
    );
}

export default App;
// 근데 이런 방법을 사용하면 에러가 뜬다. 렌더링이 계속 되기 때문.
```

그래서 useEffect를 사용하면, 또 setInterval을 사용하는 의미가 없어진다.

```jsx
import React, { useState } from "react";

function App() {
    const [fontState, setFontState] = useState(5);
    // fontState의 초기값 5

    useEffect(() => {
        setInterval(() => {
            setFontState(fontState + 1);
        }, 1000);
    }, []);
    //렌더링 이후, 단 한 번만 실행시킨다. 그럼 무한으로 렌더링되는 걸 막아줄 수 있다.
    // 그러나 변경되지 않고, 초기값 5로만 유지된다. 즉, setInterval이 의미가 없는 것.

    return (
        <>
        <div style = {{ fontSize : fontState + "px"}}>Hello</div>
        </>
    );
}

export default App;
```

그럼 왜 계속 렌더링이 되는가? useEffect를 왜 사용해야 하는가?
그 전에 라이프 사이클을 알아야 한다. 라이프 사이클에는 component 3가지가 있는데,

1. componentDidMount() : 첫 렌더링 때, 실행되는 함수 선언
2. componononetWillUnmount() : 위 동작, componentDidMount를 닫는 느낌으로, 같이 사용한다. Didmount를 정리하는 목적이다.
3. componentDidUpdate : 리랜더링 후에 실행된다.

즉, 첫 렌더링 -> componentDidMount -> 변경 사항 -> 렌더링 -> componentDidUpdate -> 부모가 현 컴포넌트 제거 -> componentWillUnmount
이런 사이클이 되는 것이다. 그러나 Hooks에는 라이프 사이클이 없어, useEffect로 비슷하게 구현해야 한다.
위와 같은 오류가 뜨는 이유는, 라이프 사이클이 작동하지 않기 때문이고, 이를 위해서 useEffect를 사용한다.
근데, 또 useEffect를 사용하려면 useRef를 먼저 불러와야 한다.
(useRef은 반드시 함수 안에서 사용해야 한다.)

그래서 세 번째에서 useRef를 사용하는 것이다.
그럼 useRef란 무엇인가.
useRef는 앞에서 말했다시피 커스텀 훅의 일종인데, 컴포넌트 안에서 관리하는 변수인데, useState와의 차이점은, useState는 값이 바뀔 때마다 렌더링 해야 할 때 사용하고, 그 반대면 useRef를 사용한다. 반대로 생각하면, useState는
위를 이해하려면 리액트의 특징 3가지를 봐야 한다.

1. 함수형 컴포넌트는 그냥 함수이다. 즉, 함수형 컴포넌트는 단지 JSX를 반환하는 함수이다.
2. 컴포넌트가 렌더링 된다는 것은 누군가가 그 함수(컴포넌트)를 호출하여 실행되는 뜻이다.

함수가 실행될 때마다 내부에 선언되어 있던 표현식도 매번 다시 선언된다.
3. 컴포넌트는 자신의 State나 부모에게서 받은 props가 변경될 때만 리렌더링 된다.
그래서 useState와 다르게, 별도 특정 데이터를 가지게 하고, 이 데이터들을 리렌더링 없이 관리하고 싶을 때 useRef를 사용한다.

그럼 세 번째 방법으로, useRef와 useEffect를 사용하여 setInterval를 오류없이 실행시켜보자.

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

function App() {
    const [fontState, setFontState] = useState(5); //초기화
    const fontRef = useRef(5);

    useEffect(() => {
        setInterval(() => {
            setFontState((fontRef.current +=1));
        }, 1000);
    }, []);

    return (
        <>
        <div style = {{ fontSize : fontState + "px"}}>Hello</div>
        </>
    );
}

export default App;

/*
Hello의 fontSize 는 fontState 이고, const로 fontState를 정의한다.
fontState는 5로 초기화되고, setFontState값이 변할 때마다 fontState가 변한다.
setFontState는 useRef 렌더링되지 않는 값, 5로 초기화되고, 1씩 늘어나서 1000까지 계속된다.
```

출처
https://falaner.tistory.com/52
https://haesoo9410.tistory.com/168
https://ko-de-dev-green.tistory.com/71?category=865864