useRef() HOOK

01. useRef

- DOM 요소에 직접 접근하거나, 값을 유지하는 데 사용
- 렌더링 주기에 영향을 미치지 않으며, 값이 변경되어도 컴포넌트가 재렌더링되지 않음

```
import { useRef } from 'react';
export default function TextInput() {
  const inputEl = useRef(null);
  const focusInput = () => {
    inputEl.current.focus();
       };
  return (
    <div>
      <input ref={inputEl} type="text" />
      <button onClick={focusInput}>Focus the input</button>
    </div>
  );
}
```

01. useRef

- Ref객체는 current속성 하나만을 가지며 매핑되는 대상을 저장
- DOM 요소 직접 제어: 특정 DOM 요소에 접근하여 포커스 설정, 스크롤 제어, 애니메이션 실행 등의 작업을 할 때 사용.
- 값의 유지 및 추적: 컴포넌트가 다시 렌더링되더라도 유지해야 하는 값을 저장하고 추적.
- 상태 변경 없이 값 업데이트: 상태 변경이 아닌 단순한 값 업데이트를 수행하여 컴포넌트가 불필요하게 재렌더링되지 않도록 함.
- useRef 는 동기적으로 동작, 내용 변경시 바로 적용

01. useRef

```
import { useRef, useState } from 'react';

function ClickTracker() {
  const clickCount = useRef(0);

  const handleClick = () => {
   clickCount.current += 1;
   console.log(`Button clicked ${clickCount.current} times`);
  };

  return (
    <div>
        <button onClick={handleClick}>Click Me</button>
    </div>
  );
}

export default ClickTracker;
```

01. useRef

```
import { useRef } from 'react';

function RefExample() {
  const myRef = useRef(0);

  const handleClick = () => {
    console.log('Before:', myRef.current); // 이전 값 출력
    myRef.current += 1;                    // 값 즉시 변경
    console.log('After:', myRef.current);  // 변경된 값 출력
  };

  return <button onClick={handleClick}>Increase Ref</button>;
}

export default RefExample;
```

useEffect Hook

01. useEffect

- 사이드 이펙트(side effects)를 처리하는 데 사용
  - 컴포넌트의 렌더링 과정 외에 발생하는 작업
  - 데이터 가져오기(비동기 요청), DOM 조작, 구독(subscription), 타이머 설정
- useEffect()의 기본 구조
  - useEffect(사이드 이펙트 처리 함수, 의존성 배열)
  - useEffect(()=>{……}, [….])
  - 사이드 이펙트 처리 함수
    - React가 Dom을 업데이트한 이후에 실행
    - 의존성 배열의 내용에 따라 실행
    - 의존성 배열이 비어 있을 때: 컴포넌트 마운트시 한번만 실행
    - 의존성 배열에 특정값이 포함되었을 때 : 배열 안에 포함된 값이 변경될 때마다 실행
    - 의존성 배열을 생략했을 때 : 컴포넌트가 랜더링 될때 마다 실행

01. useEffect

컴포넌트의 생명주기

마운트(mount)

- 리액트가 해당 컴포넌트의 렌더링을 완료하고, 브라우저의 DOM에 반영되었을 때

업데이트(update)

- 상태나 props의 변경이 발생할 때마다 다시 렌더링

언마운드(unmount)

- 컴포넌트가 화면에서 사라지거나, 라우팅 등으로 다른 컴포넌트로 교체될때

01. useEffect

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

export default function HighlightComponent() {
  useEffect(() => {
    const element = document.getElementById("highlight");
    if (element) {
      element.style.backgroundColor = "yellow";
    }
  }, []);
  return (
    <div>
      <p id="highlight">배경색이 노란색으로 바뀝니다.</p>
    </div>
  );
}
```

01. useEffect

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

export default function DelayedMessage() {
  const [visible, setVisible] = useState(false);

  useEffect(() => {
    const timer = setTimeout(() => {
      setVisible(true);
    }, 3000); // 3초 후 표시
  }, []);

  return (
    <div>
      {visible ? <p>3초 후에 나타나는 메시지</p> :
       <p>기다리는 중...</p>}
    </div>
  );
}
```

01. useEffect

useEffect의 리턴값으로 정리(cleanup) 함수를 반환

- 컴포넌트가 언마운트되거나, useEffect가 다시 실행되기 직전에 실행
- 구독 해제, 타이머 제거, 메모리 누수 방지 등의 역할

```
import { useEffect, useState } from "react";
export default function TimerCounter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setCount((prev) => prev + 1);
    }, 1000); // 1초마다 카운트 증가

    return () => clearInterval(interval);
    // 언마운트 시 인터벌 제거
  }, []);

  return <p>경과 시간: {count}초</p>;
}
```

01. useEffect

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

export default function ExampleComponent() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log(`Count value updated to: ${count}`);
  }, [count]); // count가 변경될 때만 실행됨

  return (
    <div>
      <p>{count}</p>
      <button onClick={() => setCount(count + 1)}>
                    Increase Count</button>
    </div>
  );
}
```

01. useEffect

```
import { useEffect } from 'react';

export default function ExampleComponent() {
  useEffect(() => {
    console.log('Component mounted');
    // 마운트 시 한 번만 실행

    return () => {
      console.log('Component unmount');
       // 언마운트 시 실행
    };
  }, []);
 // 빈 배열: 마운트 시 한 번 실행, 언마운트 시 정리 함수 실행

  return <div>Example Component</div>;
}
```

01. useEffect

```
import { useEffect } from 'react';

export default function TimerComponent() {
  useEffect(() => {
    const timer = setInterval(() => {
      console.log('Timer tick');
    }, 1000);

    return () => {
      clearInterval(timer); // 타이머 해제 (정리 함수)
      console.log('Timer stopped');
    };
  }, []);  // 마운트 시 한 번 실행

  return <div>Timer is running</div>;
}
```

01. useEffect

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

export default function AutoFocusInput() {
  const inputRef = useRef(null);
  useEffect(() => {
    // 컴포넌트가 마운트되었을 때 실행
    if (inputRef.current) {
      inputRef.current.focus(); }
  }, []); // 빈 배열: 마운트시에만 실행됨

  return (
    <div>
      <input ref={inputRef} type="text"
             placeholder="자동으로 포커스됩니다" />
    </div>
  );
}
```

01. useEffect

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

export default function CounterComponent() {
  const [count, setCount] = useState(0);
  const previousCountRef = useRef(0);
  // 이전 count 값을 저장하는 useRef

  useEffect(() => {
  console.log(`Current count: ${count}, Previous count:
              ${previousCountRef.current}`);
  previousCountRef.current = count;
  }, [count]); // count가 변경될 때마다 실행

  return (
    <div>
      <p>Current Count: {count}</p>
      <button onClick={() => setCount(count +1)}>
        Increase Count</button>
    </div>
  );}
```

01. useEffect

```
import { useState, useEffect } from 'react';
export default function TimerComponent() {
  const [count, setCount] = useState(0);
  useEffect(() => {
    const interval = setInterval(() => {
      console.log(`Count: ${count}`);
    }, 1000);

    return () => {
              clearInterval(interval); // 기존 타이머 정리
              console.log('Cleanup function executed');
    };
  }, [count]); // count가 변경될 때마다 useEffect 실행

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count+1)}>
              Increase Count</button>
    </div>
  );}
```