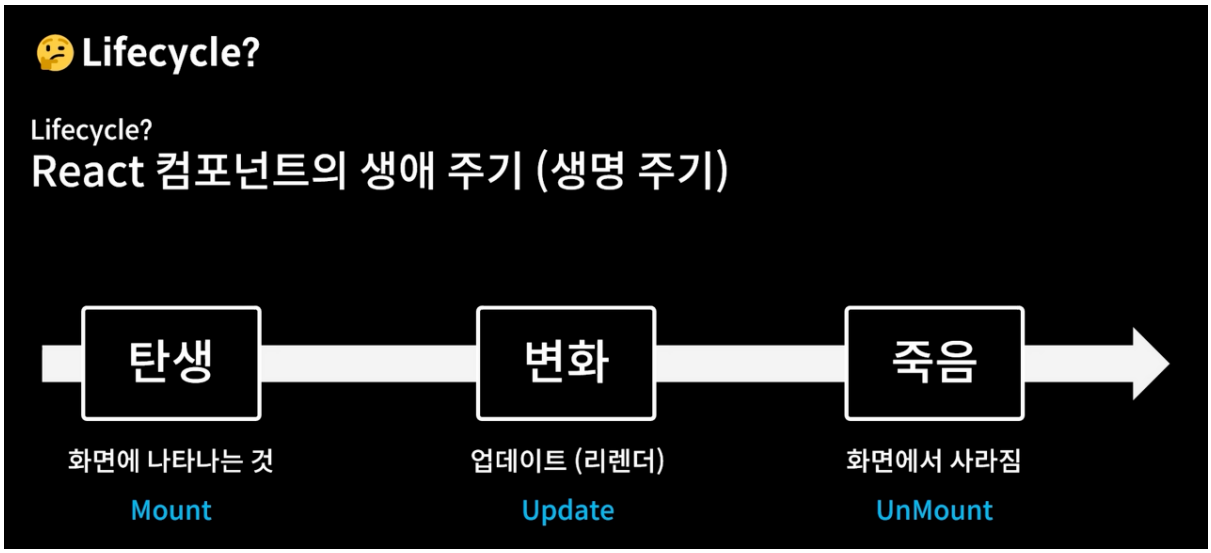
생명주기(LifeCycle),useEffect

LifeCycle(생애주기==생명주기)

생애주기란?

일반적으로 시간에 흐름에 따라 탄생부터 죽음에 이르는 순간에 따른 단계적인 과정



한입크기로 잘라먹는 리엑트 참조

Mount란?

DOM이 생성되고 웹 브라우저상에 나타나는 것

Update란?

컴포넌트를 업데이트할 때

1. props가 바뀔 때
2. state가 바뀔 때
3. 부모 컴포넌트가 리렌더링될 때
4. this , forceUpdate로 강제로 렌더링을 트리거할 때

```
forceUpdate()란?
컴포넌트의 렌더링을 강제로 트리거하는 메소드
-----------------------------------
```

```
트리거란?
특정 이벤트를 처리하기 위한 코드를 동작시키는 것
예를 들어, 이벤트가 발생하면 다른 이벤트를 동작시키거나,
코드에서 특정 함수를 호출하여 다른 함수를 실행시키는 것
ex)총의 방아쇠라고 생각하면 이해하기 쉽다. 당겨야 총알이 나가기 때문이다.
```

UnMount란?

컴포넌트를 DOM에서 제거하는 것

Class React Component Only(아래 함수들은 class형 컴포넌트에서만 사용가능)

1. 마운트(Mounting)

- constructor()
  컴포넌트가 생성될 때 호출되며, 초기 상태(state)를 설정하거나 메소드를 **바인딩**하는 등의 작업을 수행합니다.

  ```
  바인딩이란?
  프로그램에 사용된 구성 요소의 실제 값 또는 프로퍼티를 결정짓는 행위
  ```

- getDerivedStateFromProps()
  컴포넌트가 생성될 때와, 프롭스(props)가 변경될 때 호출되며, 변경된 프롭스를 기반으로 새로운 상태(state)를 반환합니다.
- render()
  컴포넌트가 화면에 그려질 때 호출되며, JSX를 반환합니다.
- componentDidMount()
  컴포넌트가 화면에 그려진 후 호출되며, 외부 API 호출이나 이벤트 리스너 등의 작업을 수행합니다.

2. 업데이트(Updating)

- getDerivedStateFromProps()
  컴포넌트가 업데이트될 때, 즉 프롭스(props)가 변경될 때 호출되며, 변경된 프롭스를 기반으로 새로운 상태(state)를 반환합니다.
- shouldComponentUpdate()
  컴포넌트가 업데이트되기 전에 호출되며, 업데이트를 할지 말지를 결정합니다. 기본적으로 true를 반환하며, 특정 조건에 따라 false를 반환하여 업데이트를 막을 수 있습니다.

- render()
  shouldComponentUpdate()가 true를 반환할 경우, JSX를 반환합니다.
- getSnapshotBeforeUpdate()
  컴포넌트가 업데이트되기 전에 호출되며, 변경된 내용을 캡처하고 반환합니다.
  이 메소드는 주로 컴포넌트의 이전 상태와 새로운 상태를 비교하여 변경된 내용을 파악하고, 이를 기반으로 추가적인 작업을 수행할 때 사용됩니다.
- componentDidUpdate()
  컴포넌트가 업데이트된 후 호출되며, DOM 조작이나 외부 API 호출 등의 작업을 수행합니다.

3. 언마운트(Unmounting)

- componentWillUnmount()
  컴포넌트가 화면에서 사라지기 전에 호출되며, 이벤트 리스너 해제 등의 작업을 수행합니다.
  ex)타이머를 중지하는 등의 작업을 수행할 수 있다.
  컴포넌트가 사용한 리소스를 해제하지 않으면 메모리 누수가 발생할 수 있으며, 이는 성능저하나 최악의 경우 브라우저 충돌 등의 문제를 발생시킬 수 있습니다.

또한, 에러(Error)가 발생할 경우에는 다음과 같은 메소드들이 호출

에러(Error) 처리

- static getDerivedStateFromError()
  자식 컴포넌트에서 에러가 발생할 경우 호출되며, 에러 상태(state)를 반환합니다.
- componentDidCatch()
  자식 컴포넌트에서 에러가 발생할 경우 호출되며, 에러 로그를 기록하거나 사용자에게 알림을 보여줄 수 있습니다.

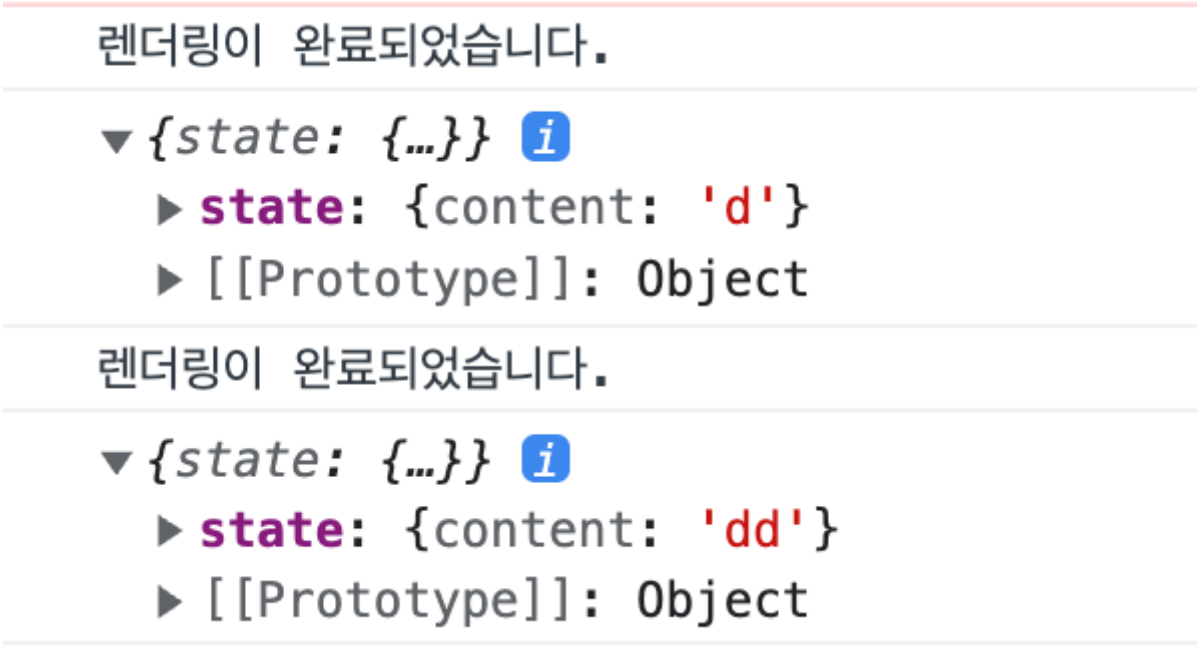
함수형컴포넌트

useEffect

`useEffect`는 리액트 컴포넌트가 렌더링될 때마다 특정 작업을 수행하도록 설정할 수 있는 Hook이다.

클래스형 컴포넌트의 `componentDidUpdate`와 `componentDidMount`를 합친 형태로 보아도 무방하다.

```
useEffect(() => {
    console.log('렌더링이 완료되었습니다.')
    console.log({
      state
    })
  })
```



`state`가 변경되서 렌더링 될때마다 콘솔에 찍히는걸 볼 수 있다.

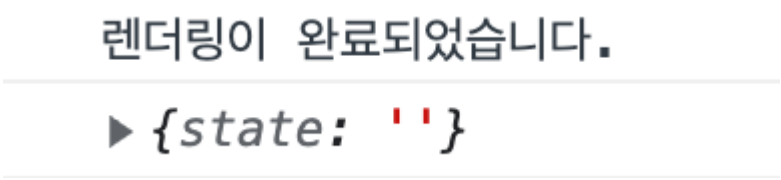
1. 마운트될 때만 실행

```
useEffect(()=>{
console.log('마운트될 때만 실행됩니다.')
},[]);
```

화면에 맨 처음 렌더링될 때만 실행하고, 업데이트될때는 실행하지 않으려면 함수의 두 번째 파라미터로 비어 있는 배열을 넣어주면 된다.

```
useEffect(() => {
    console.log('렌더링이 완료되었습니다.')
    console.log({
      state
```

```
    })
  }, [])
```



처음에 렌더링 될때만 콘솔에 찍히고 이후에 `state`값이 변경이 되더라도 콘솔이 추가로 찍히지는 않는다.

2. 특정 값이 업데이트될 때만 실행하고 싶을 때

`useEffect`의 두 번째 파라미터로 전달되는 배열 안에 검사하고 싶은 값을 넣어 주면 된다.

```
useEffect(() => {
    console.log('렌더링이 완료되었습니다.')
    console.log({
      title
    })
  }, [title])
```

이렇게 하면 `state`값이 변경될때만 변경되고 `state`가 아닌걸로 렌더링이 될 경우 콘솔에 찍히지 않는다.

3. 뒷정리하기

   컴포넌트가 언마운트되기 전이나 업데이트되기 직전에 어떠한 작업을 수행하고 싶다면 `useEffect`에서 뒷정리 함수를 반환해 주어야한다.

   두 번째 파라미터 배열에 무엇을 넣는지에 따라 실행되는 조건이 달라진다.

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

function MyComponent() {
  useEffect(() => {
    // 컴포넌트가 mount될 때 실행되는 코드

    return () => {
      // 컴포넌트가 unmount될 때 실행되는 코드
    };
  }, []);

  return <div>My Component</div>;
```

```
}

export default MyComponent;
```

여기서 `useEffect` hook의 두번째 인자로 빈 배열(`[]`)을 전달하면, 이펙트 함수는 컴포넌트가 `mount`될 때 한 번만 실행되고, 컴포넌트가 `unmount`될 때는 해당 함수가 반환하는 함수(clean-up 함수)만 실행된다.

따라서 반환되는 함수에서 `unmount` 시점에 필요한 로직을 작성하면 된다.

ex) 타이머를 종료하는 예시이다.

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

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

  useEffect(() => {
    const intervalId = setInterval(() => {
      setCount((prevCount) => prevCount + 1);
    }, 1000);

    return () => {
      clearInterval(intervalId);
    };
  }, []);

  return (
    <div>
      <p>Count: {count}</p>
    </div>
  );
}

export default MyComponent;
```

이 컴포넌트는 `mount`될 때 타이머를 시작하고, `unmount`될 때 타이머를 종료한다. 이렇게 하면 메모리 누수를 방지하고, 필요하지 않은 작업을 중단할 수 있다.