Managing State

Computed Value - Derived State

`React`에서는 같은 의미로 쓰이죠

다른 state혹은 props를 기반으로 계산될 수 있는 값이라면, 별도의 state로 저장하지 않는 편이 좋습니다

렌더링에 직접 영향을 끼치거나, state/props에 따라 변하지 않는 데이터라면 state로 관리해야겠죠

State의 보존과 초기화

해당 섹션에는 `isFancy`를 사용하여 `Counter`컴포넌트의 스타일을 바꾸는 예제가 존재합니다.

렌더/UI 트리에서의 위치가 변하지 않는 `Counter` 컴포넌트는 렌더링 위치가 동일하기에 state가 유지됨을 알 수 있죠

`isFancy`에 따라 `key`를 바꿔주면 어떻게 될까요?

```
<Counter key={isFancy ? 'fancy' : 'plain'} isFancy={isFancy}>
```

이러면 이제 React의 렌더 과정에서 해당 컴포넌트는 isFancy값이 변하면 리렌더링이 일어나고 state가 초기화됩니다.
다만 주의할 점은, 정말 필요한 경우에만 사용해야 한다는 점이겠네요

초기 상태로 되돌려야 할 때, useEffect를 사용하는 방법보다 간단한 방법으로 사용할 수 있겠습니다.

useReducer

알고는 있지만 거의 사용하지 않는 리듀서에 대해 간단하게

- `(state, action) => newState`
- Redux의 영향을 받아 도입되었다

useState와 useReducer

useState	useReducer
간단한 상태	복잡한 상태
독립적인 상태	연관된 상태들
적은 양의 상태 업데이트	많은 이벤트가 상태를 변경

- useReducer의 reducer 함수는 `switch` 문을 쓰는 것이 일반적
- 리듀서 내에서 비동기 로직을 처리하지 말 것

Context

- Context를 생성 -> Provider에 value 주입 -> useContext를 통한 value 사용
- `use`를 이용한 컨텍스트 사용 방식이 현재 테스트 단계