과제 3 - 장바구니

2 상태(state)와 컴포넌트

상태(state)의 종류

프론트엔드에서 상태란?

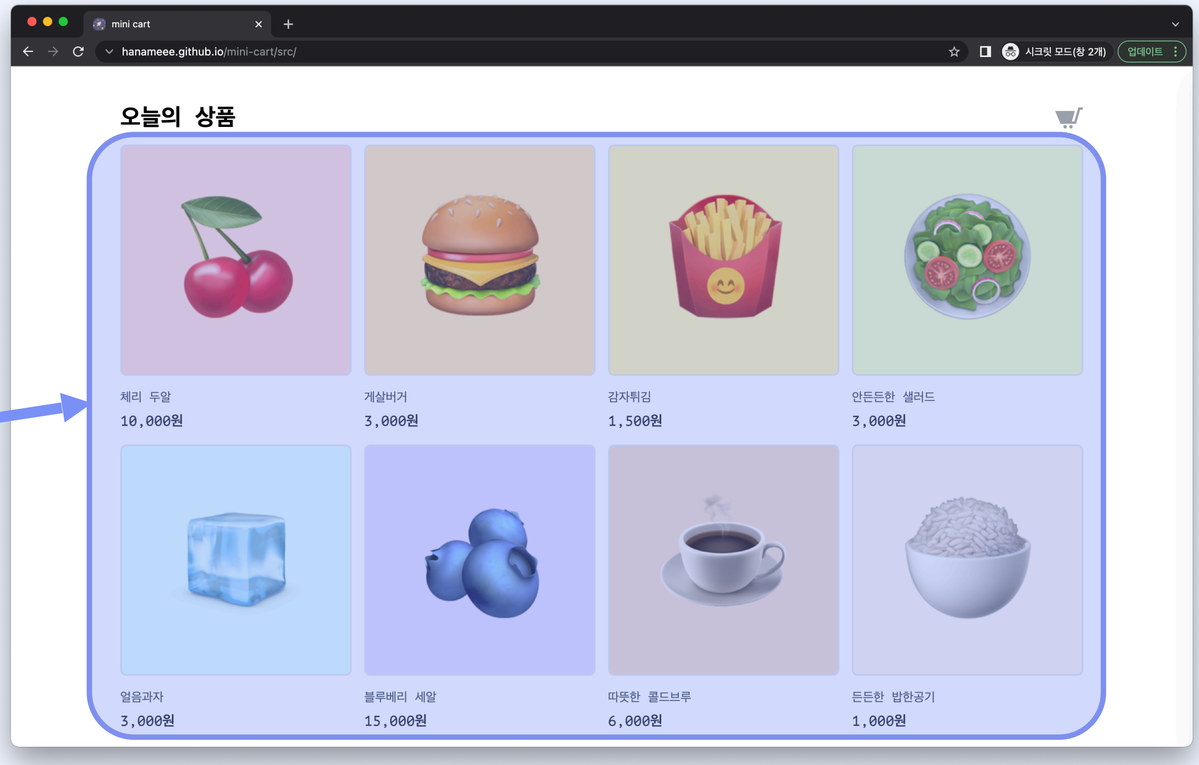
UI를 구성하는 가변적인 데이터

상태(state)의 종류

프론트엔드에서 상태란?

UI를 구성하는 가변적인 데이터

```
[
  {
    "id": 1,
    "imgSrc": "asset/cherry.png",
    "name": "체리 두알",
    "price": 10000
  },
  {
    "id": 2,
    "imgSrc": "asset/hamburger.png",
    "name": "게살버거",
    "price": 3000
  },
  {
    "id": 3,
    "imgSrc": "asset/fries.png",
    "name": "감자튀김",
    "price": 1500
  }
]
```



상태 - UI 동기화

상태를 UI와 동기화 시키는 법

상태

```
[
  {
    "id": 1,
    "imgSrc": "asset/cherry.png",
    "name": "체리 두알",
    "price": 10000,
    "count": 1
  }
]
```

유저 상호작용 등으로
상태 변화 발생

```
[
  {
    "id": 1,
    "imgSrc": "asset/cherry.png",
    "name": "체리 두알",
    "price": 10000,
    "count": 2
  }
]
```

UI

UI = state를
어떻게 유지할까?

상태 - UI 동기화

방식 A

+ 버튼에 onClick 이벤트가 발생하면 아래 작업을 수행한다.

1. 갯수 DOM을 select 한다.
2. 갯수 DOM의 innerHTML을 변경한다. (1 -> 2)
3. 가격 DOM을 select 한다.
4. 가격 DOM의 innerHTML을 변경한다. (10,000 -> 20,000)

```
$price.innerHTML = parseInt($price.innerHTML) + 10000
$count.innerHTML = parseInt($count.innerHTML) + 1
```

상태 - UI 동기화

방식 B

+ 버튼에 onClick 이벤트가 발생하면 아래 작업을 수행한다.

1. 장바구니 정보를 담고 있는 객체를 수정한다.

```
[
  {
    "id": 1,
    "imgSrc": "asset/cherry.png",
    "name": "체리 두알",
    "price": 10000,
    "count": 1
  }
```

```
[
  {
    "id": 1,
    "imgSrc": "asset/cherry.png",
    "name": "체리 두알",
    "price": 10000,
    "count": 2
  }
```

장바구니 정보가 담긴 객체가 수정되면 아래 작업을 수행한다.

1. 장바구니 객체를 바탕으로 UI를 다시 그린다. (HTML 재생성)

```
$target.innerHTML = state.map(
  ({price, count}) ⇒
  `<div>
    <li>${price*count}원</li>
    <li>${count}</li>개
   </div>`
)
```

명령형 프로그래밍과 선언형 프로그래밍

방식 A = 명령형(절차적) 방식

어떤 일을 **어떻게** 할 것인가

+ 버튼에 onClick 이벤트가 발생하면 아래 작업을 수행한다.

1. 갯수 DOM을 select 한다.
2. 갯수 DOM의 innerHTML을 수정한다. (1 -> 2)
3. 가격 DOM을 select 한다.
4. 가격 DOM의 innerHTML을 수정한다. (10,000 -> 20,000)

프론트엔드 웹 개발에서 일반적으로 명령형 방식은,

- 상태 정보가 **DOM**에 산발적으로 저장되어 있다.
- 상태 관리가 곧 DOM 조작을 의미한다. (분리 X)
- DOM을 조작하는 코드가 여러 곳에 산발적으로 존재한다.

방식 B = 선언적 방식

무엇을 할 것인가

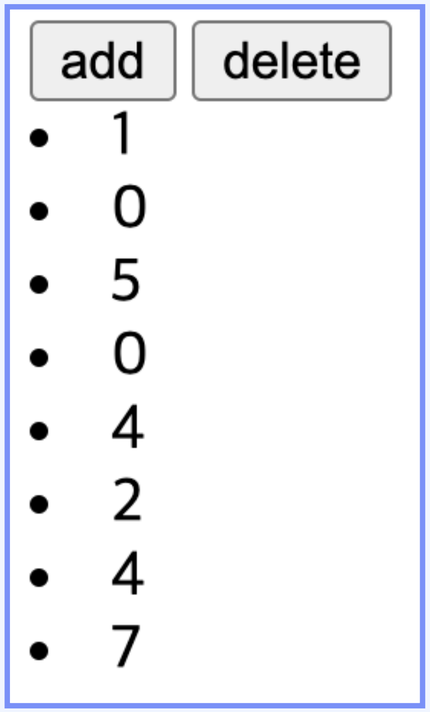
+ 버튼에 onClick 이벤트가 발생하면 아래 작업을 수행한다.
1. 장바구니 정보를 담고 있는 객체를 수정한다. (count += 1)

장바구니 정보가 담긴 객체가 수정되면 아래 작업을 수행한다.
2. 수정된 객체를 바탕으로 UI를 다시 그린다.

프론트엔드 웹 개발에서 일반적으로 선언적 방식은,

- 상태 정보는 DOM이 아닌 **JS 내의 값**으로 존재한다.
- 상태 관리 기능과, DOM 조작 기능이 분리되어 있다.
- DOM은 개별적으로 조작되지 않으며, 변경된 상태에 맞게 다시 그려진다.
- 선언적 방식으로 코드를 작성하기 위해서는, 상태 변화를 감지해 DOM을 조작하는 (명령형) 부분이 **추상화** 되어 있어야 한다.

두가지 방법 직접 실습해보기



조건

- [add]를 누르면 랜덤한 값의 가 추가된다.
- [delete]를 누르면 마지막 가 삭제된다.

명령형

선언형

https://codesandbox.io/s/sample-component-state-management-o2d7ev

컴포넌트 단위로 생각하기

명령형 프로그래밍에서는…

하나의 HTML에 모든 UI가 덩어리로 존재

기능 단위로 파일을 분리하기 어려움

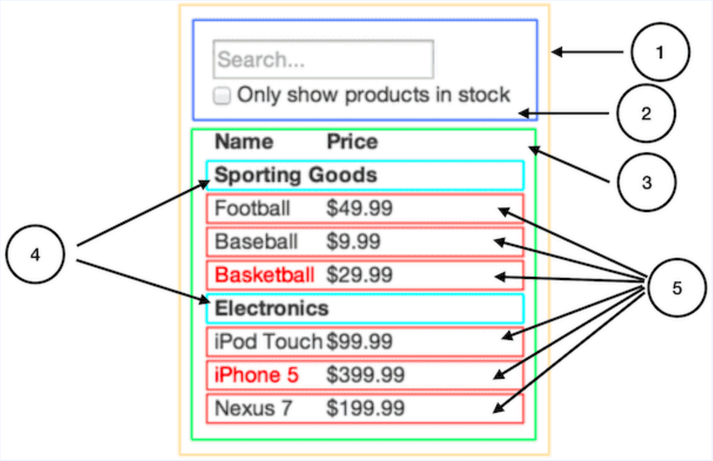
코드의 재사용성이 낮고, 유지보수가 어려움

선언적 프로그래밍에서는…

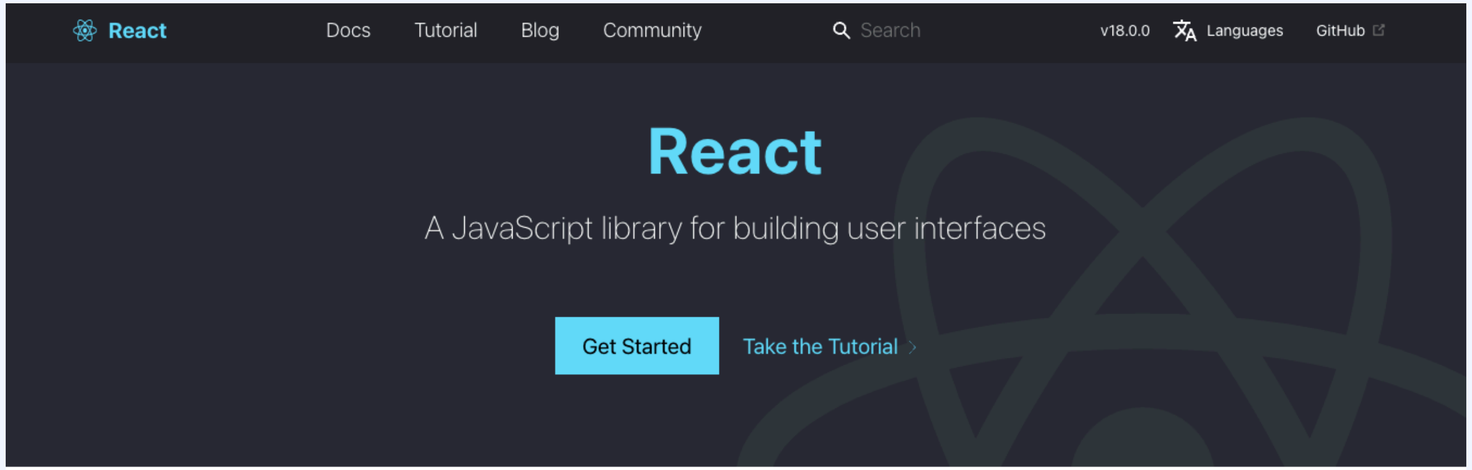
state를 바탕으로 **Javascript**가 UI를 그림

기능 단위로 JS 파일을 분리할 수 **있음**

코드의 **재사용성**이 높고, **유지보수**가 쉬움



컴포넌트



선언적 & 컴포넌트 기반 프로그래밍 = **모던 프론트엔드 어플리케이션**의 패러다임