

## 6. 변경 가능한 데이터 구조를 가진 언어에서 불변성 유지하기

## 이번 장에서 살펴볼 내용

1. 데이터가 바뀌지 않도록 하기 위해 카피-온-라이트를 적용합니다.
2. 배열과 객체를 데이터에 쓸 수 있는 카피-온-라이트 동작을 만듭니다.
3. 깊이 중첩된 데이터도 카피-온-라이트가 잘 동작하게 만듭니다.

# 모든 동작을 불변형으로 만들 수 있을까?

## 장바구니에 대한 동작

1. 제품 개수 가져오기
2. 제품 이름으로 제품 가져오기
3. 제품 추가하기
4. 제품 제품 이름으로 제품 빼기
5. 제품 이름으로 제품 구매 수량 바꾸기



## 동작을 읽기, 쓰기 또는 둘 다로 분류하기

### 읽기

데이터를 바꾸지 않고 정보를 꺼내는 것

### 쓰기

어떻게든 데이터를 바꾸는 동작

## 동작을 읽기, 쓰기 또는 둘 다로 분류하기

1. 제품 개수 가져오기
2. 제품 이름으로 제품 가져오기
3. 제품 추가하기
4. 제품 제품 이름으로 제품 빼기
5. 제품 이름으로 제품 구매 수량 바꾸기

## 동작을 읽기, 쓰기 또는 둘 다로 분류하기

- 쓰기 동작은 불변성 원칙에 따라 구현해야 한다.
- 불변성 원칙은 카피-온-라이트라고 한다.
- 자바스크립트는 변경 가능한 데이터 구조를 사용하기 때문에 불변성 원칙을 적용하려면 직접 구현해야 한다.
  - 하스켈, 클로저 같은 언어는 구현되어 있다.
- 읽으면서 쓰는 동작도 가능하다.

## 카피-온-라이트 원칙 세 단계

1. 복사본 만들기
2. 본사본 변경하기
3. 복사본 리턴하기

## 카피-온-라이트 원칙 세 단계

```
function addElementLast(array, elem) {  
  const newArray = array.slice(); // 1. 복사본 생성  
  newArray.push(elem); // 2. 변경  
  return newArray; // 3. 리턴  
}
```

`addElementLast()` 는 읽기일까, 쓰기일까?



## 쓰기를 하면서 읽기도 하는 동작은 어떻게 해야할까?

1. 함수를 분리하기
2. 값을 두 개 리턴하기

# 쓰면서 읽기도 하는 함수를 분리하기

```
Array.prototype.shift()
```

## 읽기

```
function getFirstElement() {  
  return array[0];  
}
```

## 쓰기

```
function dropFirstElement(array) {  
  const copiedArray = array.slice();  
  copiedArray.shift();  
  return copiedArray;  
}
```

## 값을 두 개 리턴하는 함수로 만들기

`Array.prototype.shift()`

```
function shift(array) {  
  const copiedArray = array.slice();  
  const firstElement = copiedArray.shift();  
  return {  
    firstElement,  
    array: copiedArray,  
  };  
}
```

## 불변 데이터 구조를 읽는 것은 계산이다

- 변경 가능한 데이터를 읽는 것은 액션이다.
- 쓰기는 데이터를 변경 가능한 구조로 만든다.
- 어떤 데이터에 쓰기가 없다면 데이터는 변경 불가능한 데이터다.
- 불변 데이터 구조를 읽는 것은 계산이다.
- 쓰기를 읽기로 바꾸면 코드에 계산이 많아진다.

## 애플리케이션에서는 시간에 따라 변하는 상태가 있다

교체  
1. 읽기  
2. 바꾸기  
3. 쓰기

```
shopping_cart = add_item(shopping_cart, shoes);
```

교체  
1. 읽기  
2. 바꾸기  
3. 쓰기

```
shopping_cart = remove_item_by_name(shopping_cart, "shirt");
```

## 불변 데이터 구조는 충분히 빠릅니다.

- 언제든지 최적화 할 수 있다.
- 가비지 콜렉터는 매우 빠르다.
- 생각보다 많이 복사하지 않는다.
- 함수형 프로그래밍 언어에는 빠른 구현체가 있다.

## 객체에 대한 카피-온-라이트

1. 복사본 만들기
2. 복사본 변경하기
3. 복사본 리턴하기

## 객체에 대한 카피-온-라이트

`Object.assign()`

```
function setPrice(item, newPrice) {  
  const copiedItem = Object.assign({}, item); // 1. 복사  
  copiedItem.price = newPrice; // 2. 변경  
  return copiedItem; // 3. 리턴  
}
```



## 중첩된 쓰기를 읽기로 바꾸기

```
function setPriceByName(cart, name, price) {  
  const copiedCart = cart.slice();  
  for (let i = 0; i < copiedCart.length; i += 1) {  
    if (copiedCart[i].name === name)  
      copiedCart[i] = setPrice(copiedCart[i], price);  
  }  
  return copiedCart;  
}
```

- 요소를 직접 변경한다는 것은 불변 데이터가 아니라는 것이다.
- 최하위부터 최상위까지 중첩된 데이터 구조의 모든 부분이 불변형이어야 한다는 것이다.

## 요점 정리

- 함수형 프로그래밍에서 불변 데이터가 필요하다.
  - 계산 과정에서 변경 가능한 데이터에 쓰기를 할 수 없기 때문.
- 카피-온-라이트는 데이터를 불변형으로 유지할 수 있는 원칙이다.
- 카피-온-라이트는 값을 변경하기 전에 얇은 복사를 한다.
  - 이렇게 하면 통제할 수 있는 범위에서 불변성을 구현할 수 있다.