함수형 프로그래밍 2

함수 리팩토링 & 불변성 보장

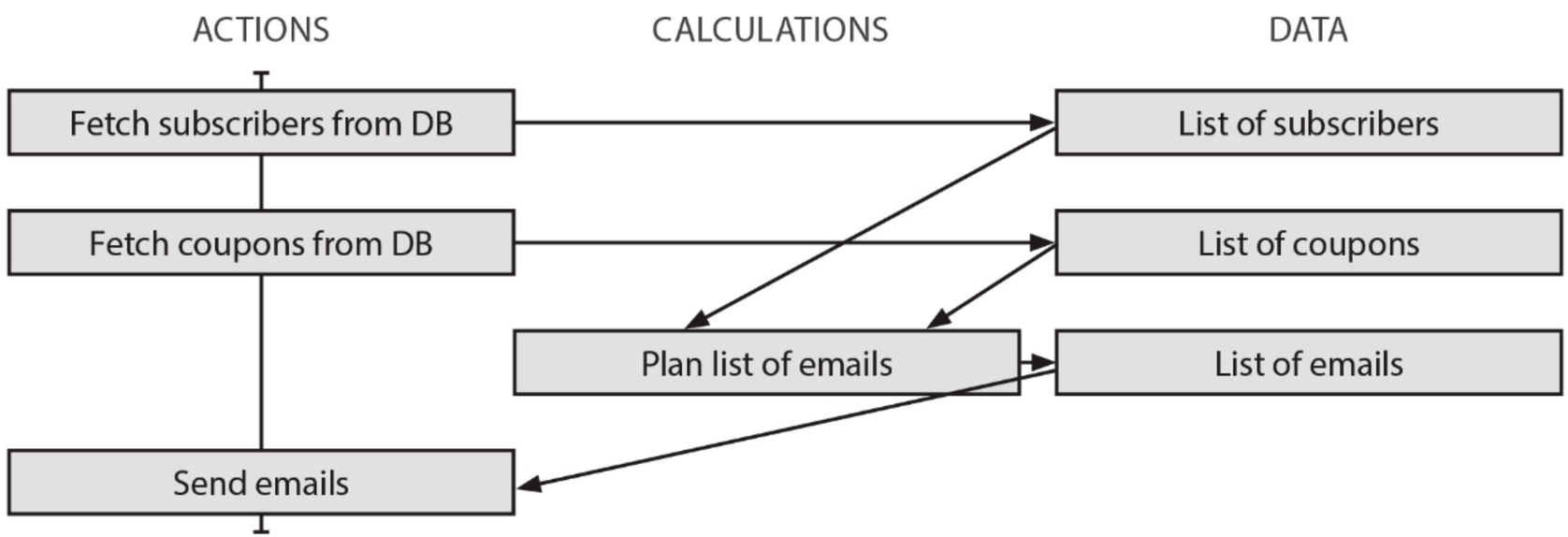
2023.07.02

액션, 계산, 데이터

함수형 프로그래머는 코드를 데이터, 계산, 액션으로 분류해서 생각한다

- 액션 : 실행 시점, 순서에 의존하는 동작 ex) 부수효과가 있는 함수, 가변값 읽기
- 계산 : 입력값을 처리해서 출력을 하는 동작 ex) 순수함수
- 데이터 : 이벤트에 대한 사실을 기록한 것 ex) 유저 입력에 대한 기록, DB에서 읽어온 값

액션, 계산, 데이터


ACTIONS
CALCULATIONS
DATA
Fetch subscribers from DB
Fetch coupons from DB
Plan list of emails
Send emails
List of subscribers
List of coupons
List of emails

액션, 계산, 데이터

최대한 데이터로 표현하고 액션은 최소화 해야한다

- 계산은 순수함수이다
- 순수함수는 테스트하기 쉽고, 따라서 유지보수하기 쉽다
- 또한 잘 분리된 계산은 재사용성이 뛰어나다

함수 리팩토링

액션에서 계산을 추출하면 액션의 크기와 개수를 줄일 수 있다

계산 추출

1. 코드를 선택하고 빼낸다
2. 암묵적 입력과 출력을 찾는다.
3. 입력은 인자로 바꾸고 출력은 리턴값으로 바꾼다

함수 리팩토링

계산 추출

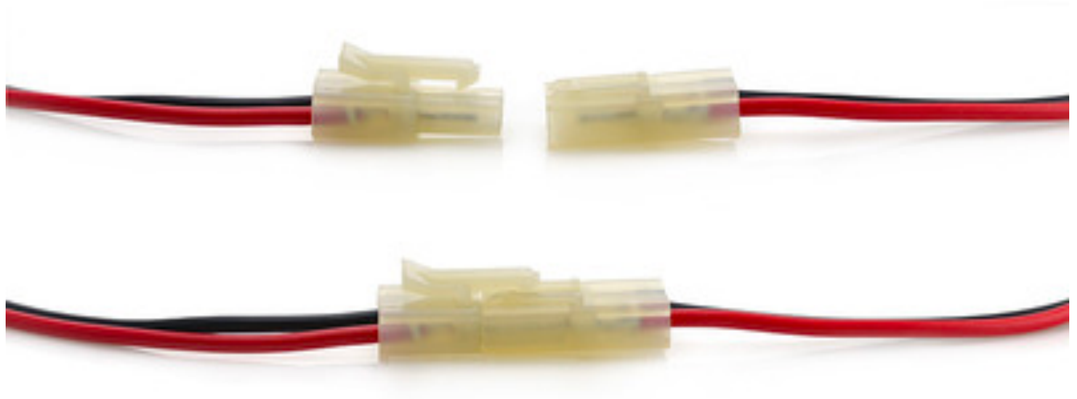
```
function update_tax_dom() {
  set_tax_dom(shopping_cart_total * 0.10); // shopping_cart_total 는 전역 변수이다
}
```

```
function update_tax_dom() {
  set_tax_dom(calc_tax(shopping_car_total)); // shopping_cart_total 는 전역 변수이다
}

function calc_tax(total) {
  return total * 0.1;
} // 순수함수 (계산)

// 이제 calc_tax 는 순수 함수이다. 이 함수는 전역 변수를 사용하지 않고,
// 인자로 전달된 값을 사용한다. 이제 이 함수는 테스트하고 재사용하기 쉽다.
```

함수 리팩토링

암묵적인 입출력

액션에서 계산을 분리했다고 다가 아니다.
원칙적으로 함수에는 암묵적인 입력/출력이 없는 것이 좋다.



함수 리팩토링

함수 분리

함수를 사용하면 관심사를 자연스럽게 분리할 수 있다.
잘분리된 함수는 계층형 설계의 기반이 된다.

함수 리팩토링

작은 함수로 분리하기

```
function add_item(cart, name, price) {
  // 1. 배열을 복사한다
  const new_cart = cart.slice();

  // 3. 복사본에 아이템을 추가한다
  new_cart.push(
    // 2. 아이템 객체를 생성한다
    {
      name: name,
      price: price,
    }
  );

  // 4. 복사본을 반환한다
  return new_cart;
}
```

```
function make_cart_item(name, price) {
  // 2. 아이템 객체를 생성한다
  return {
    name: name,
    price: price,
  };
}

function add_item(cart, item) {
  // 1. 배열을 복사한다
  const new_cart = cart.slice();

  // 3. 복사본에 아이템을 추가한다
  new_cart.push(item);

  // 4. 복사본을 반환한다
  return new_cart;
}

add_item(shopping_cart, make_cart_item("apple", 1000));
```

함수 리팩토링

일반화 시키기

```
function add_item(cart, item) {
  const new_cart = cart.slice();
  new_cart.push(item);
  return new_cart;
}
```

```
function add_item(cart, item) {
  return add_element_last(cart, item);
}

function add_element_last(array, elem) {
  var new_array = array.slice();
  new_array.push(elem);
  return new_array;
}
```

함수 리팩토링

계산 분류하기

```
// 배열 유틸리티
function add_element_last(array, elem) {
  var new_array = array.slice();
  new_array.push(elem);
  return new_array;
}

// 카트에 대한 동작
function add_item(cart, item) {
  return add_element_last(cart, item);
}

// 아이템에 대한 동작
function make_item(name, price) {
  return {
    name: name,
    price: price,
  };
}
```

```
// 카트에 대한 동작 + 아이템에 대한 동작 + 비즈니스 로직
function calc_total(cart) {
  var total = 0;
  for (var i = 0; i < cart.length; i++) {
    var item = cart[i];
    total += item.price;
  }
  return total;
}

// 비즈니스 로직
function gets_free_shipping(cart) {
  return calc_total(cart) >= 100000;
}

// 비즈니스 로직
function calc_tax(amount) {
  return amount * 0.1;
}
```

불변성 보장

이런 과정이 왜 필요할까?

```
function add_item(cart, item) {
  return add_element_last(cart, item);
}

function add_element_last(array, elem) {
  var new_array = array.slice();
  new_array.push(elem);
  return new_array;
}
```

- Javascript에서는 객체 타입은 const 키워드를 사용해도 불변성이 보장 되지 않음
- 얕은 복사본을 생성해서 데이터의 불변성을 보장할 수 있음

불변성 보장

카피-온-라이트

```
// 카피-온-라이트
function add_element_last(array, elem) {
  // 1. 복사본을 만들기
  var new_array = array.slice();
  // 2. 복사본 바꾸기
  new_array.push(elem);
  // 3. 복사본 반환하기
  return new_array;
}
```

카피-온-라이트는 데이터를 불변형으로 유지 시켜주는 원칙이다.
함수형 프로그래밍에서 데이터를 변경할 때, 얕은 복사를 하고
복사본을 바꾼다음 반환하는 과정을 의미.

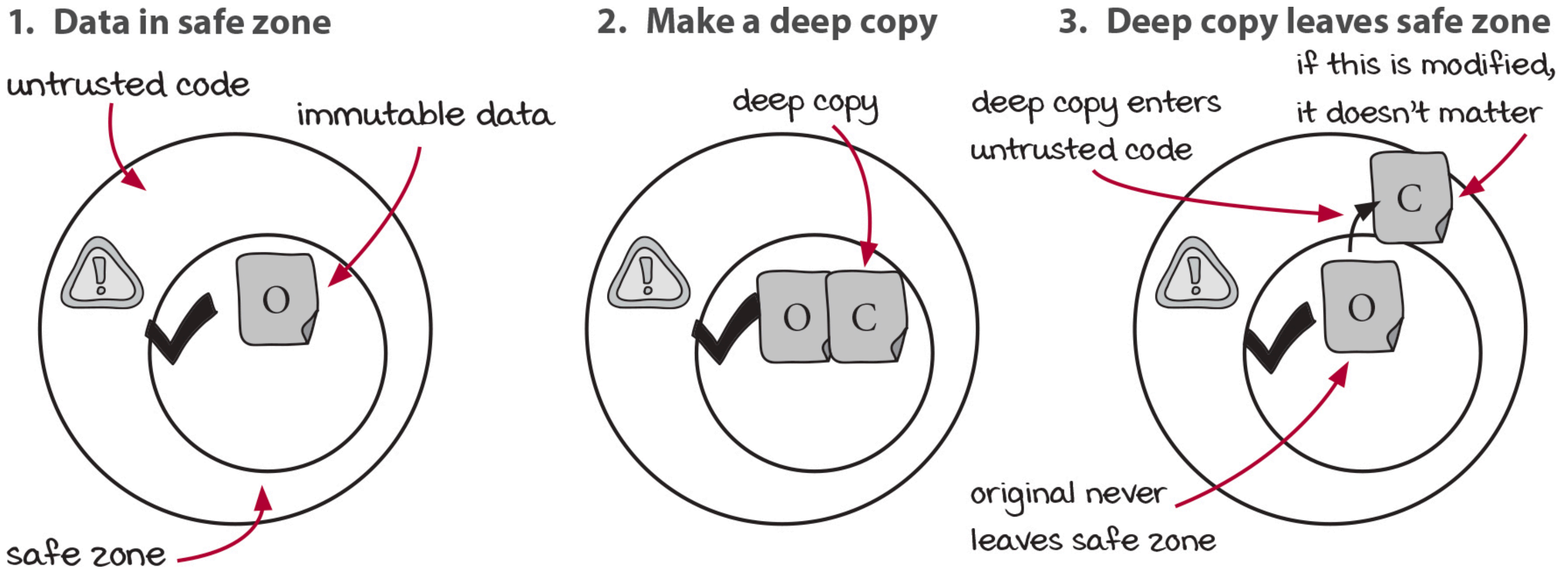
불변성 보장

카피-온-라이트를 사용하면 코드에 계산이 많아진다

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

불변성 보장

방어적 복사



방어적 복사는 믿을 수 없는 코드(데이터의 불변성이 보장돼지 않는코드)와 상호작용하는 함수형 코드에서 사용하는 기법이다.
데이터를 주고/받을 때 깊은 복사를 수행하는 것을 의미한다.

불변성 보장

방어적 복사

```
function addItemToCart(name, price) {
  const item = make_cart_item(name, price);

  shoppingCart = add_item(shoppingCart, item);

  const total = calc_total(shoppingCart);

  //중략 ...

  //blackFridayPromotion는 믿을 수 없는 코드입니다.
  const cartCopy = deepCopy(shoppingCart);
  blackFridayPromotion(cart_copy);
  shoppingCart = deepCopy(cart_copy);
}
```