

4. 액션에서 계산 빼내기

- 테스트하기 쉽고 재사용성이 좋은 코드를 만들기 위한 함수형 기술에 대해 알아보자.
- 액션에서 계산을 빼내는 방법을 배운다.

? 함수형 프로그래밍에서 클래스 변수는 액션인가?

- > 함수로 구현했다고, 함수형 프로그래밍을 적용한 것이 아니다.
- > 핵심 개념인 액션과 계산의 분리 여부가 함수형 프로그래밍이 적용했는지를 구분한다.

A. MegaMart.com에 오신 것을 환영합니다.

```
var shopping_cart = [];  
var shopping_cart_total = 0;  
  
function add_item_to_cart(name, price){  
    shopping_cart.push({  
        name: name,  
        price: price  
    });  
    calc_cart_total();  
}  
  
function calc_cart_total(){  
    shopping_cart_total = 0;  
    for(var i = 0; i < shopping_cart.length; i++){  
        var item = shopping_cart[i];  
        shopping_cart_total += item.price;  
    }  
    set_cart_total_dom();  
}
```

- add_item_to_cart()
 - 전역변수인 `shopping_cart`에 값을 추가
- calc_cart_total()
 - 전역변수인 `shopping_cart`의 값을 읽고, `shopping_cart_total`에 값을 추가

B. 무료 배송비 계산하기

a. 새로운 요구사항

- 장바구니에 넣으면 합계가 20\$가 넘는 제품의 구매 버튼 옆에 무료 배송 아이콘을 표시하자 !

b. 절차적인 방법으로 구현하기

- 구매 버튼에 무료 배송 아이콘 표시

```
function update_shipping_icons() {  
    var buy_buttons = get_buy_buttons_dom();  
  
    for(var i = 0; i < buy_buttons.length; i++){  
        var button = buy_buttons[i];  
        var item = button.item;  
  
        if(item.price + shopping_cart_total >= 20)  
            button.show_free_shipping_icon();  
        else  
            button.hide_free_shipping_icon();  
    }  
}
```

- DOM에서 읽음

- DOM을 바꿈

- 합계 금액이 바뀔 때마다 모든 아이콘을 업데이트

```
function calc_cart_total(){  
    shopping_cart_total = 0;  
    for(var i = 0; i < shopping_cart.length; i++){  
        var item = shopping_cart[i];  
        shopping_cart_total += item.price;  
    }  
    set_cart_total_dom();  
    update_shipping_icons();  
}
```

- 전역변수를 바꿈

- 전역변수를 읽음

C. 세금 계산하기

a. 요구사항

- 장바구니의 금액 합계가 바뀔 때마다 세금을 다시 계산

```
function update_tax_dom(){  
    set_tax_dom(shopping_cart_total * 0.10)  
}
```

```
function calc_cart_total(){  
    shopping_cart_total = 0;  
    for(var i = 0; i < shopping_cart.length; i++){  
        var item = shopping_cart[i];  
        shopping_cart_total += item.price;  
    }  
    set_cart_total_dom();  
    update_shipping_icons();  
    update_tax_dom();  
}
```

- 전역변수를 읽음

D. 테스트하기 쉽게 만들기

코드가 바뀔 때마다 아래와 같은 테스트를 만들어야 한다.

1. 브라우저 설정하기
2. 페이지 로드하기
3. 장바구니에 제품 담기 버튼 클릭
4. DOM이 업데이트될 때까지 기다리기
5. DOM에서 값 가져오기
6. 가져온 문자열 값을 숫자로 바꾸기
7. 예상하는 값과 비교하기

```
function update_tax_dom() {  
    set_tax_dom(shopping_cart_total * 0.10);  
}
```

테스트를 더 쉽게 하려면 다음 조건일 필요하다.

- DOM 업데이트와 비즈니스 규칙은 분리되어야 한다.
- 전역변수가 없어야 한다.

E. 재사용하기 쉽게 만들기

- 결제팀과 배송팀이 우리 코드를 사용하려고 한다.
 - 장바구니 정보를 전역변수에서 읽어오지만, 결제팀과 배송팀은 데이터베이스에서 장바구니 정보를 읽어와야 한다.
 - 결과를 보여주기 위해 DOM을 직접 바꾸고 있지만, 결제팀은 영수증을, 배송팀은 운송장을 출력해야 한다.
- 개발팀 제나의 제안
 - **전역변수**에 의존하지 않아야 한다.
 - DOM을 사용할 수 있는 곳에서 실행된다고 가정하면 안 된다.
 - 함수가 **결괏값**을 **리턴**해야 한다.

F. 액션과 계산, 데이터를 구분하기

```
var shopping_cart = [];
var shopping_cart_total = 0;

function add_item_to_cart(name, price){
    shopping_cart.push({
        name: name,
        price: price
    });
    calc_cart_total();
}

function calc_cart_total(){
    shopping_cart_total = 0;
    for(var i = 0; i < shopping_cart.length; i++){
        var item = shopping_cart[i];
        shopping_cart_total += item.price;
    }
    set_cart_total_dom();
    update_shipping_icons();
    update_tax_dom();
}
```

Diagram illustrating the separation of actions and calculations in the code:

- ACTION** (red box) points to `shopping_cart.push()` in `add_item_to_cart`.
- ACTION** (red box) points to `calc_cart_total()` in `add_item_to_cart`.
- ACTION** (red box) points to `shopping_cart_total = 0;` in `calc_cart_total`.

```
function update_shipping_icons() {
    var buy_buttons = get_buy_buttons_dom();

    for(var i = 0; i < buy_buttons.length; i++){
        var button = buy_buttons[i];
        var item = button.item;

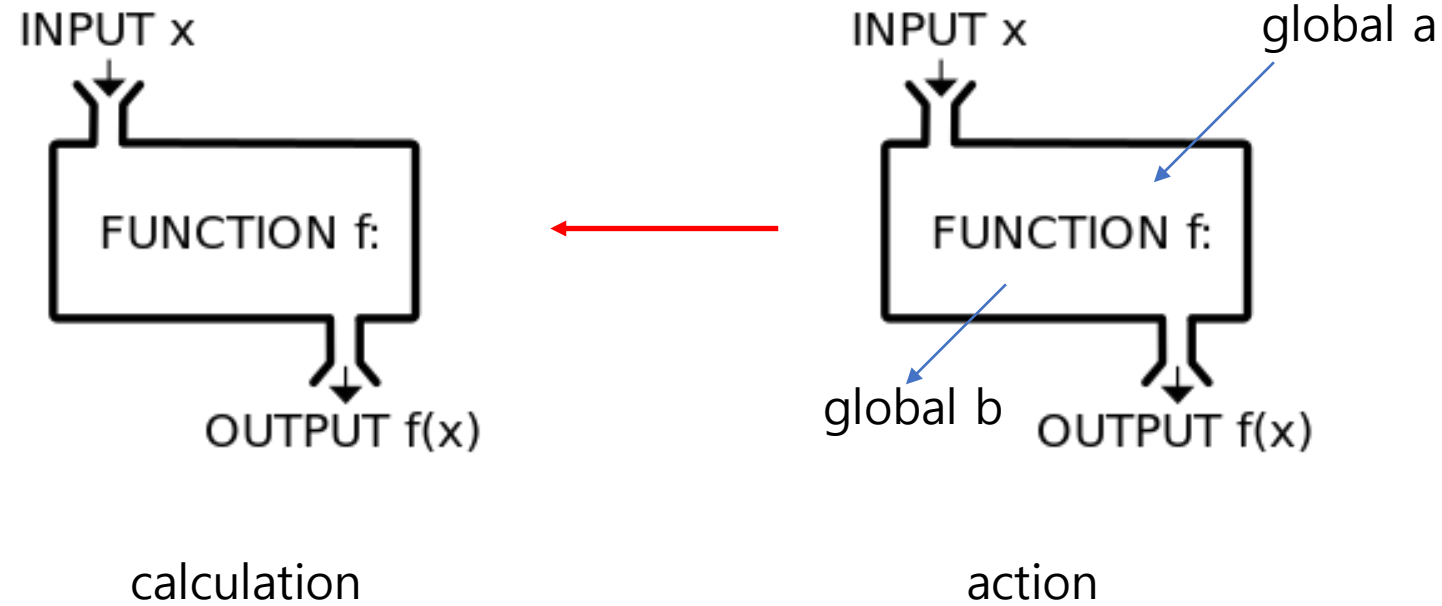
        if(item.price + shopping_cart_total >= 20)
            button.show_free_shipping_icon();
        else
            button.hide_free_shipping_icon();
    }
}

function update_tax_dom(){
    set_tax_dom(shopping_cart_total * 0.10)
}
```

Diagram illustrating the separation of actions and calculations in the code:

- ACTION** (red box) points to `get_buy_buttons_dom()` in `update_shipping_icons`.
- ACTION** (red box) points to `set_tax_dom()` in `update_tax_dom`.

G. 함수에는 입력과 출력이 있다



- Input $x \Rightarrow$ 명시적 입력
 - Output $f(x) \Rightarrow$ 명시적 출력
 - Global $a \Rightarrow$ 암묵적 입력
 - Global $b \Rightarrow$ 암묵적 출력
-) \rightarrow 부수 효과

H. 액션에서 계산 빼내기

```
function calc_cart_total(){
    shopping_cart_total = 0;
    for(var i = 0; i < shopping_cart.length; i++){
        var item = shopping_cart[i];
        shopping_cart_total += item.price;
    }
    set_cart_total_dom();
    update_shipping_icons();
    update_tax_dom();
}
```

서브루틴 추출하기
(extract subroutine)

```
function calc_cart_total(){
    calc_total();
    set_cart_total_dom();
    update_shipping_icons();
    update_tax_dom();
}

function calc_total(){
    shopping_cart_total = 0;
    for(var i = 0; i < shopping_cart.length; i++){
        var item = shopping_cart[i];
        shopping_cart_total += item.price;
    }
}
```

암묵적 출력 제거

```
function calc_cart_total(){
    shopping_cart_total = calc_total();
    set_cart_total_dom();
    update_shipping_icons();
    update_tax_dom();
}

function calc_total(){
    var total= 0;
    for(var i = 0; i < shopping_cart.length; i++){
        var item = shopping_cart[i];
        total += item.price;
    }
    return total;
}
```

암묵적 입력 제거


```
function calc_cart_total(){
    shopping_cart_total = calc_total(shopping_cart);
    set_cart_total_dom();
    update_shipping_icons();
    update_tax_dom();
}

function calc_total(cart){
    var total= 0;
    for(var i = 0; i < cart.length; i++){
        var item = cart[i];
        total += item.price;
    }
    return total;
}
```

H. 액션에서 계산 빼내기

```
function add_item_to_cart(name, price){  
    shopping_cart.push({  
        name: name,  
        price: price  
    });  
    calc_cart_total();  
}
```

서브루틴 추출하기
(extract subroutine)



```
function add_item_to_cart(name, price){  
    add_item(name, price);  
    calc_cart_total();  
}  
  
Function add_item(name, price){  
    shopping_cart.push({  
        name: name,  
        price: price  
    });  
}
```

암묵적 입력 제거



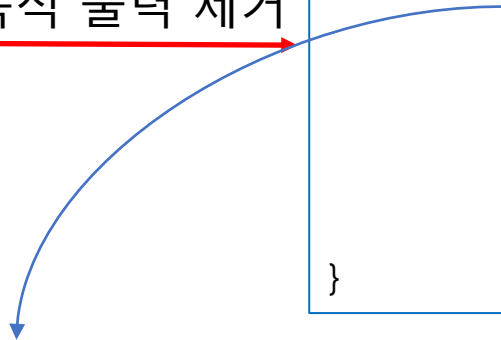
```
function add_item_to_cart(name, price){  
    add_item(shopping_cart, name, price);  
    calc_cart_total();  
}  
  
Function add_item(cart, name, price){  
    cart.push({  
        name: name,  
        price: price  
    });  
}
```

암묵적 출력 제거













```
function add_item_to_cart(name, price){  
    shopping_cart = add_item(shopping_cart, name, price);  
    calc_cart_total();  
}  
  
Function add_item(cart, name, price){  
    var new_cart = cart.slice();  
    new_cart.push({  
        name: name,  
        price: price  
    });  
    return new_cart;  
}
```

카피-온-라이트(copy-on-write)



I. 전체 코드

```
var shopping_cart = [];   
var shopping_cart_total = 0;   
  
function add_item_to_cart(name, price) {   
    shopping_cart = add_item(shopping_cart, name, price);  
    calc_cart_total();  
}  
  
function calc_cart_total() {   
    shopping_cart_total = calc_total(shopping_cart);  
    set_cart_total_dom();  
    update_shipping_icons();  
    update_tax_dom();  
}  
  
function update_shipping_icons() {   
    var buttons = get_buy_buttons_dom();  
    for(var i = 0; i < buttons.length; i++){  
        var button = buttons[i];  
        var item = button.item;  
  
        if(gets_free_shipping(shopping_cart_total, item.price))  
            button.show_free_shipping_icon();  
        else  
            button.hide_free_shipping_icon();  
    }  
}
```

```
function update_tax_dom() {   
    set_tax_dom(calc_tax(shopping_cart_total));  
}  
  
function add_item(cart, name, price) {   
    var new_cart = cart.slice();  
    new_cart.push({  
        name: name,  
        price: price  
    });  
    return new_cart;  
}  
  
function calc_total(cart) {   
    var total = 0;  
  
    for(var i = 0; i < cart.length; i++){  
        var item = cart[i];  
        total += item.price;  
    }  
    return total;  
}  
  
function gets_free_shipping(total, item_price) {   
    return item_price + total >= 20;  
}  
  
function calc_tax(amount) {   
    return amount * 0.10;  
}
```

J. 요점 정리

- 액션은 암묵적인 입력 또는 출력을 가지고 있다.
- 계산은 암묵적은 입력이나 출력이 없어야 한다.
- 공유변수(i.e. 전역변수)는 일반적으로 암묵적 입력 또는 출력이 된다.
- 암묵적 입력은 인자로 바꿀 수 있다.
- 암묵적 출력은 리턴값으로 바꿀 수 있다.

5. 더 좋은 액션 만들기

- 액션에서 암묵적 입력과 출력을 줄여 설계를 개선하는 방법에 대해 알아보자.

A. 비즈니스 요구 사항과 설계를 맞추기

a. 비즈니스 요구 사항

장바구니에 담긴 제품을 주문할 때 무료 배송인지 확인하는 것

b. Code smell 제거

B. 비즈니스 요구 사항과 함수를 맞추기

```
function gets_free_shipping(total, item_price) {  
    return item_price + total >= 20;  
}
```



```
function gets_free_shipping(cart) {  
    return calc_total(cart) >= 20;  
}
```

```
function update_shipping_icons() {  
    var buttons = get_buy_buttons_dom();  
    for(var i = 0; i < buttons.length; i++){  
        var button = buttons[i];  
        var item = button.item;  
  
        if(gets_free_shipping(shopping_cart_total, item.price))  
            button.show_free_shipping_icon();  
        else  
            button.hide_free_shipping_icon();  
    }  
}
```

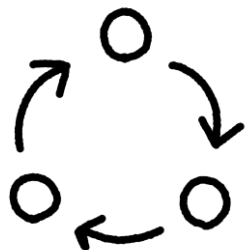


```
function update_shipping_icons() {  
    var buttons = get_buy_buttons_dom();  
    for(var i = 0; i < buttons.length; i++){  
        var button = buttons[i];  
        var item = button.item;  
        var new_cart = add_item(shopping_cart,  
                                item.name,  
                                item.price);  
  
        if(gets_free_shipping(new_cart))  
            button.show_free_shipping_icon();  
        else  
            button.hide_free_shipping_icon();  
    }  
}
```


Z. 쉬는 시간

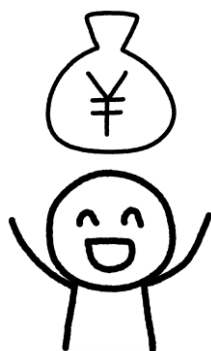
a. 코드 라인 수가 늘어났습니다. 그래도 좋은 코드인가요?

- 작은 함수는 이해하기 쉽다.
- 작은 함수는 응집력 있고 재사용하기 쉽다.



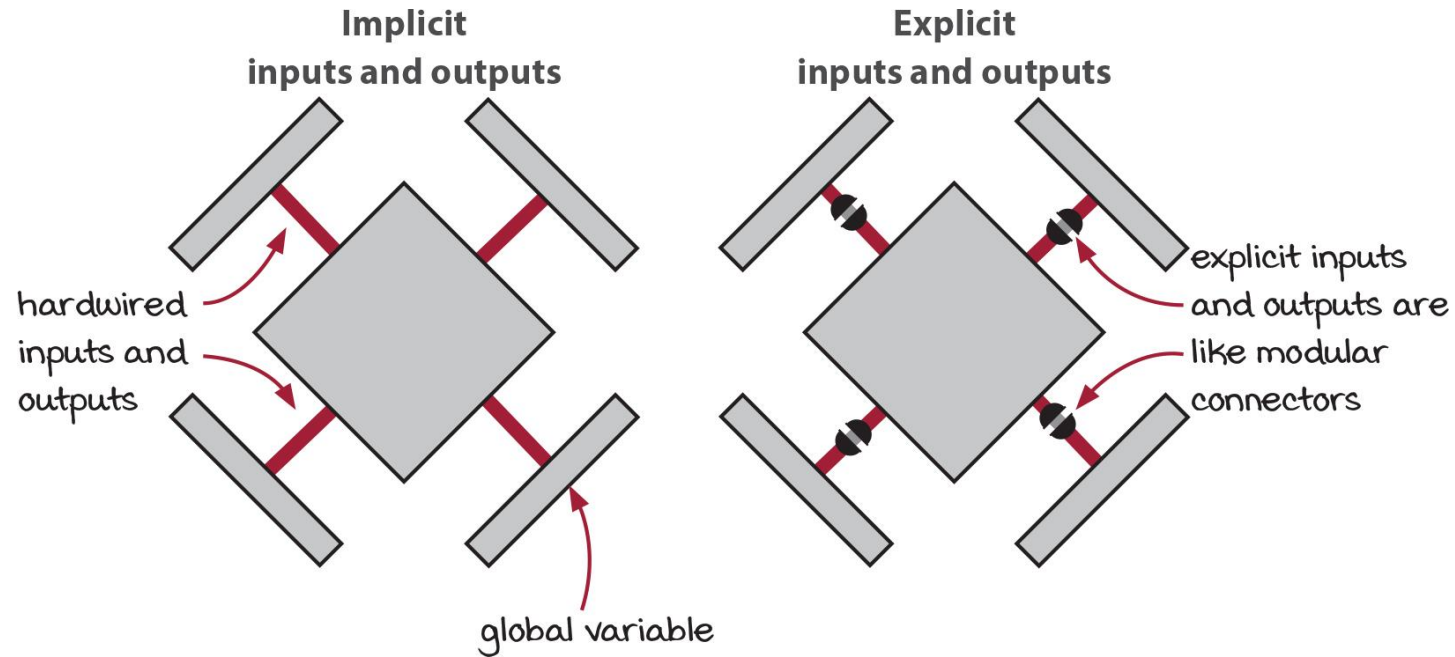
b. `add_item()` 함수를 부를 때마다 `cart` 배열을 복사합니다. 비용이 너무 많이 들지 않나요?

- 비용이 더 드는 것은 맞다.
- 최신 프로그래밍 언어의 런타임과 가비지 컬렉터는 불필요한 메모리를 효율적으로 잘 처리한다.
- 또, 복사본을 만들어 바꾸는 방법은 많은 장점이 있다. (6장, 7장 참고)



C. 암묵적 입력과 출력은 적을수록 좋다.

- 액션에서 모든 암묵적 입력과 출력을 없애지는 않더라도 암묵적 입력과 출력은 줄이면 좋다.
- 어떤 함수에 암묵적 입력과 출력이 있다면 다른 컴포넌트와 강하게 연결된 컴포넌트라고 할 수 있다.
- 암묵적 입력과 출력을 명시적으로 바꿔 모듈화된 컴포넌트로 만들 수 있다.



- 암묵적 입력과 출력이 있는 함수는 아무 때나 실행할 수 없기 때문에 테스트하기 어렵다.
- 모든 입력값을 설정하고 테스트를 돌린 후에 모든 출력값을 확인해야 한다.
- 암묵적 입력과 출력을 줄이면 테스트하기 쉽고 재사용하기 좋다.

D. 암묵적 입력과 출력 줄이기

```
function update_shipping_icons() {  
    var buttons = get_buy_buttons_dom();  
    for(var i = 0; i < buttons.length; i++){  
        var button = buttons[i];  
        var item = button.item;  
        var new_cart = add_item(shopping_cart,  
                                item.name,  
                                item.price);  
  
        if(gets_free_shipping(new_cart))  
            button.show_free_shipping_icon();  
        else  
            button.hide_free_shipping_icon();  
    }  
}
```

```
function update_shipping_icons(cart) {  
    var buttons = get_buy_buttons_dom();  
    for(var i = 0; i < buttons.length; i++){  
        var button = buttons[i];  
        var item = button.item;  
        var new_cart = add_item(cart,  
                                item.name,  
                                item.price);  
  
        if(gets_free_shipping(new_cart))  
            button.show_free_shipping_icon();  
        else  
            button.hide_free_shipping_icon();  
    }  
}
```

```
function calc_cart_total() {  
    shopping_cart_total = calc_total(shopping_cart);  
    set_cart_total_dom();  
    update_shipping_icons();  
    update_tax_dom();  
}
```

```
function calc_cart_total() {  
    shopping_cart_total = calc_total(shopping_cart);  
    set_cart_total_dom();  
    update_shipping_icons(shopping_cart);  
    update_tax_dom();  
}
```

E. 액션을 계산으로 (전역변수 읽는 부분을 인자로 바꾸자)

```
function add_item_to_cart(name, price){  
    shopping_cart = add_item(shopping_cart, name, price);  
    calc_cart_total();  
}
```

```
function calc_cart_total() {  
    shopping_cart_total = calc_total(shopping_cart);  
    set_cart_total_dom();  
    update_shipping_icons(shopping_cart);  
    update_tax_dom();  
}
```

```
function update_tax_dom(){  
    set_tax_dom(calc_tax(shopping_cart_total));  
}
```

```
function add_item_to_cart(name, price){  
    shopping_cart = add_item(shopping_cart, name, price);  
    calc_cart_total(shopping_cart);  
}
```

```
function calc_cart_total(cart) {  
    var total = calc_total(cart);  
    set_cart_total_dom(total);  
    update_shipping_icons(cart);  
    update_tax_dom(total);  
    shopping_cart_total = total;  
}
```

```
function update_tax_dom(total){  
    set_tax_dom(calc_tax(total));  
}
```

F. 코드 다시 살펴보기




- 함수형 원칙을 더 적용할 수 있는 부분이 있는지 살펴보는 것도 중요하지만 중복이나 불필요한 코드가 있는지도 살펴봐야 한다.



```
function add_item_to_cart(name, price){
    shopping_cart = add_item(shopping_cart, name, price);
    calc_cart_total(shopping_cart);
}




function calc_cart_total(cart) {
    var total = calc_total(cart);
    set_cart_total_dom(total);
    update_shipping_icons(cart);
    update_tax_dom(total);
    shopping_cart_total = total;
}
```


```
function add_item_to_cart(name, price){
    shopping_cart = add_item(shopping_cart, name, price);
    var total = calc_total(shopping_cart);
    set_cart_total_dom(total);
    update_shipping_icons(shopping_cart);
    update_tax_dom(total);
}
```


G. 계산 분류하기

- 장바구니 구조를 알아야 하면 C라고 표시 
- 제품에 대한 구조를 알아야 하면 I라고 표시 
- 비즈니스 규칙에 대한 함수라면 B라고 표시 

```
function add_item(cart, name, price) {    
    var new_cart = cart.slice();  
    new_cart.push({  
        name: name,  
        price: price  
    });  
    return new_cart;  
}
```

```
function calc_total(cart) {     
    var total = 0;  
    for(var i = 0; i < cart.length; i++){  
        var item = cart[i];  
        total += item.price;  
    }  
    return total;  
}
```

```
function gets_free_shipping(cart){   
    return calc_total(cart) >= 20;  
}
```

```
function calc_tax(amount){   
    return amount * 0.10;  
}
```

H. 설계는 엉켜있는 코드를 푸는 것이다

- 함수는 인자로 넘기는 값과 그 값을 사용하는 방법을 분리한다.
- 분리된 것은 언제든지 쉽게 조합할 수 있다.

a. 재사용하기 쉽다.

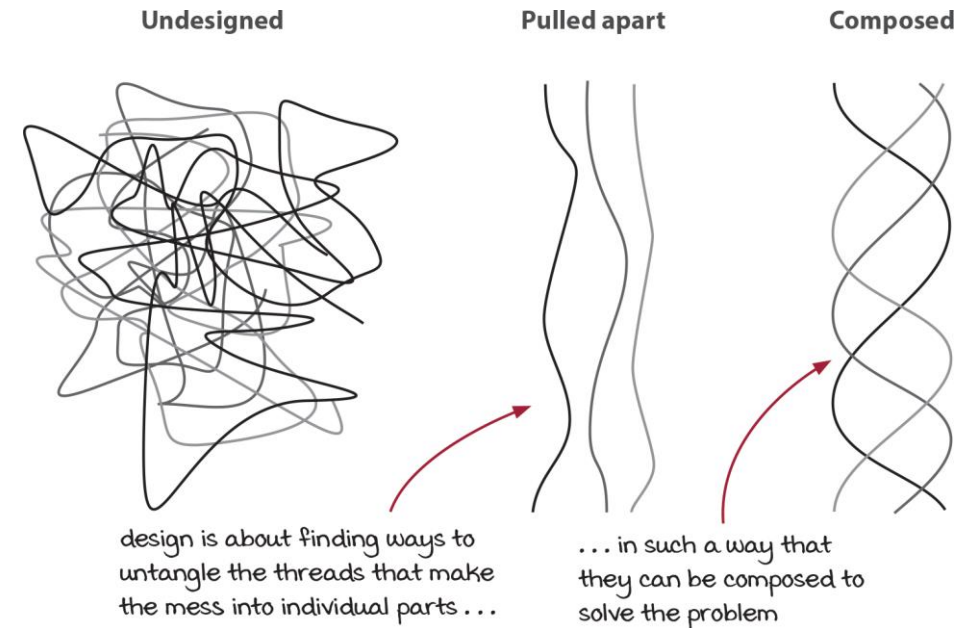
- 함수는 작으면 작을수록 재사용하기 쉽다.
- 하는 일도 적고 쓸 때 가정을 많이 하지 않아도 된다.

b. 유지보수하기 쉽다.

- 작은 함수는 이해할 수 있고 유지보수하기 쉽다.

c. 테스트하기 쉽다.

- 한 가지 일만 하기 때문에 한 가지만 테스트하면 된다.
- 함수에 특별한 문제가 없어도 꺼낼 것이 있다면 분리하는 것이 좋다.
 - 그렇게 하면 더 좋은 설계가 된다.



1. add_item()을 분리해 더 좋은 설계 만들기

1. 배열을 복사

```
function add_item(cart, name, price) {  
  var new_cart = cart.slice();  
  new_cart.push({  
    name: name,  
    price: price  
  });  
  return new_cart;  
}
```

2. Item 객체를 만듦

3. 복사본에 item을 추가

4. 복사본을 리턴

```
add_item(shopping_cart, "shoes", 3.14);
```

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

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

- 어떤 배열이나 항목에도 쓸 수 있는 이름 (재사용할 수 있는 유틸리티(utility)함수)

```
function make_cart_item(name, price) {  
  return {  
    name: name,  
    price: price  
  };  
}
```

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

```
add_item(shopping_cart, make_cart_item("shoes", 3.14));
```

- Item 구조만 알고 있는 함수와 cart 구조만 알고 있는 함수로 분리
- cart와 item을 독립적으로 확장가능
- Copy-on-write를 구현한 부분은 함께 두는 것이 좋음 (1, 3, 4)

J. 계산 분류하기

```
function add_item(cart, name, price) {  
    var new_cart = cart.slice();  
    new_cart.push({  
        name: name,  
        price: price  
    });  
    return new_cart;  
}
```



```
function calc_total(cart) {  
    var total = 0;  
    for(var i = 0; i < cart.length; i++){  
        var item = cart[i];  
        total += item.price;  
    }  
    return total;  
}
```



```
function gets_free_shipping(cart){  
    return calc_total(cart) >= 20;  
}
```



```
function calc_tax(amount){  
    return amount * 0.10;  
}
```



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



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



```
function make_cart_item(name, price){  
    return {  
        name: name,  
        price: price  
    };  
}
```



```
function calc_total(cart){  
    var total = 0;  
    for(var i = 0; i < cart.length; i++){  
        var item = cart[i];  
        total += item.price;  
    }  
    return total;  
}
```











```
function gets_free_shipping(cart){  
    return calc_total(cart) >= 20;  
}
```









```
function calc_tax(amount){  
    return amount * 0.10;
```



K. 작은 함수와 많은 계산

```
var shopping_cart = [];   
  
function add_item_to_cart(name, price){   
    shopping_cart = add_item(shopping_cart, name, price);  
    var total = calc_total(shopping_cart);  
    set_cart_total_dom(total);  
    update_shipping_icons(shopping_cart);  
    update_tax_dom(total);  
}  
  
function update_shipping_icons(  {  
    var buttons = get_buy_buttons_dom();  
    for(var i = 0; i < buttons.length; i++){  
        var button = buttons[i];  
        var item = button.item;  
        var new_cart = add_item(,  
                                item.name,  
                                item.price);  
  
        if(gets_free_shipping(new_cart))  
            button.show_free_shipping_icon();  
        else  
            button.hide_free_shipping_icon();  
    }  
}  
  
function update_tax_dom(){   
    set_tax_dom(calc_tax());  
}
```

```
function add_element_last(array, elem) {   
    var new_array = array.slice();  
    new_array.push(elem);  
    return new_array;  
}  
  
function add_item(cart, item) {   
    return add_element_last(cart, item);  
}  
  
function make_cart_item(name, price){   
    return {  
        name: name,  
        price: price  
    };  
}  
  
function calc_total(cart){   
    var total = 0;  
    for(var i = 0; i < cart.length; i++){  
        var item = cart[i];  
        total += item.price;  
    }  
    return total;  
}  
  
function gets_free_shipping(cart){   
    return calc_total(cart) >= 20;  
}  
  
Function calc_tax(amount){   
    return amount * 0.10;
```

결론

- 액션은 데이터 구조에 대해 몰라도 된다.
- 재사용할 수 있는 유용한 인터페이스 함수가 많이 생겼다.

요점 정리

- 암묵적 입력과 출력은 인자와 리턴값으로 바꿔 없애는 것이 좋다.
- 설계는 엉켜있는 것을 푸는 것이다. 풀려있는 것은 언제든지 다시 합칠 수 있다.
- 엉켜있는 것을 풀어 각 함수가 하나의 일만 하도록 하면, 개념을 중심으로 쉽게 구성할 수 있다.