Function & Module

Contents

성적처리 예제

P대학교 Y교수는 학생들 성적 처리를 고민하고 있다.

학생들 성적처리를 컴퓨터를 활용하여 처리하기 위해서는 컴퓨터가 처리할 수 있도록 데이터를 입력해야 하고 입력된 데이터를 처리해야 할 기능에 따라 분류하는 것이 필요하다.

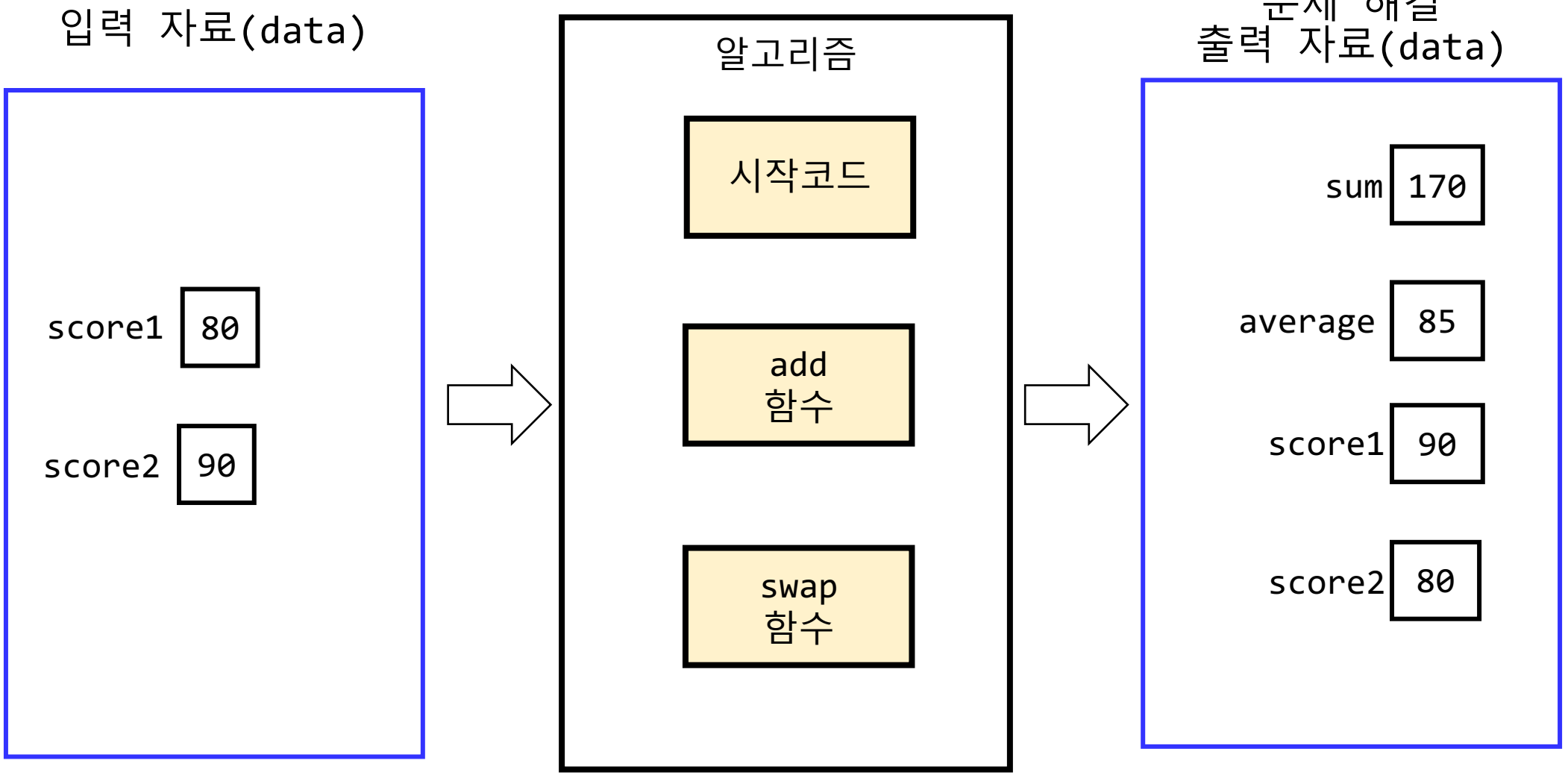
데이터 : 점수

처리기능 : 총점, 정렬(두 수의 교환)

학생이 여러 명인 경우 총점과 정렬을 해야하는 기능이 여러 번 필요하게 되는데, Y 교수는 같은 일을 오랜 시간 동안 반복하는 것을 싫어한다.

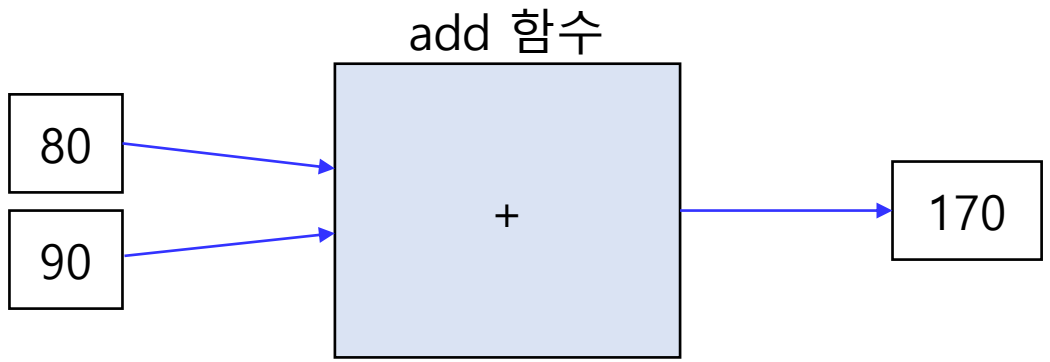
같은 기능의 일을 계속 반복하여 코딩하는 것보다 만들어 놓은 기능을 재사용할 수 있는 좋은 방법을 찾고 있다. 함께 해결 방법을 찾아봅시다!

문제 해결
problem solving



함수의 개념

- 함수(function)는 독립적으로 수행하는 프로그램 단위로 특정 작업을 수행하는 명령어들의 모음에 이름을 붙인 것
- 프로그램에서 반복적으로 수행되는 기능을 함수로 만들어 호출
- 함수는 작업에 필요한 데이터(매개변수)를 전달받을 수 있으며, 작업이 완료된 후작업의 결과를 반환할 수 있음



함수의 필요성

- 함수는 문제 해결의 방법
  - 주어진 문제를 작은 문제, 즉 여러 함수로 나누어 생각할 수 있으므로 함수를 만드는 것은 문제 해결의 하나의 방법
- 함수 이용의 장점
  - 함수로 구성된 프로그램은 함수 단위로 구성되어 있어, 읽기 쉽고, 이해하기 쉬움
  - 이미 정의된 함수는 여러 번 호출이 가능하므로 소스의 중복을 최소화하여 프로그램의 양을 줄이는 효과

함수 실행 과정

1. 함수 정의:
   - 함수를 정의하기 위해 def 키워드를 사용
   - 함수의 이름을 지정하고 필요한 매개변수를 정의한 후, 콜론(:)으로 끝남
   - 함수의 내용은 들여쓰기(indentation)로 구분
2. 함수 호출:
   - 함수를 호출하기 위해 함수의 이름과 ( )연산자 안에 필요한 인자(argument)를 넣어서 호출
   - 함수가 호출되면 호출된 함수 내의 코드가 실행
3. 반환값 활용 (선택적):
   - 함수가 값을 반환해야 할 경우, return 문을 사용하여 반환할 값을 지정
   - 반환된 값은 함수 호출문 자체가 해당 값을 갖게 됨

```
# 함수 정의

Def 함수이름(매개변수1, 매개변수2, …):
    # 함수의 내용
    # …
```

```
#함수 호출

함수이름(인자1, 인자2, ...)
```

```
# 함수 정의(반환값)

def 함수이름(매개변수1, 매개변수2, ...):
    # 함수의 내용
    # ...
    return 반환값
```

함수 정의와 함수 호출 예시

```
# 함수 정의
def greet(name) :
    greeting = "안녕하세요, " + name + "님!"
    return greeting

# 함수 호출
result = greet("Kim")

print(result)
```

```
# 실행 결과

안녕하세요, Kim님!
```

함수 매개변수와 반환 값

- 입력 값과 반환 값이 없는 함수

```
def say():
    print("hello") # None 반환
```

- 입력 값은 없고 반환 값이 있는 함수

```
def say():
    return "hello"
```

함수 예시

정수의 거듭제곱값을 계산하여 반환하는 함수 작성 (** 연산자 사용가능)

```
def power(x, y):
        result = 1
        for i in range(y):
                result = result * x
        return result

print(power(10, 2))
```

```
100
```

함수를 이용할 때 주의할 점

- 파이썬 인터프리터는 함수가 정의되면 함수 안의 문장들은 즉시 실행하지 않음
- 함수 정의가 아닌 문장들은 즉시 실행

무엇이 문제인가?

```
print(power(10, 2))

def power(x, y):
        result = 1
        for i in range(y):
                result = result * x
        return result
```

```
print(power(10, 2))
NameError: name 'power' is not defined
```

함수 작성 예시

- 정수의 거듭제곱값을 계산하여 반환하는 함수 작성 (** 연산자 사용가능)
- main() 함수 호출 활용

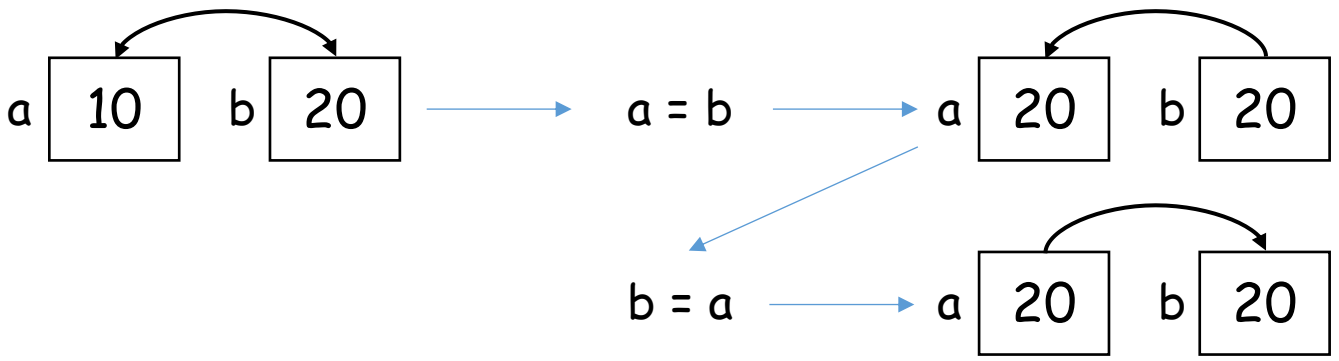
```
def main():
        print(power(10, 2))

def power(x, y):
        result = 1
        for i in range(y):
                result = result * x
        return result

main()
```

```
100
```

문제 해결 : 성적 처리를 위한 주요 함수 만들기

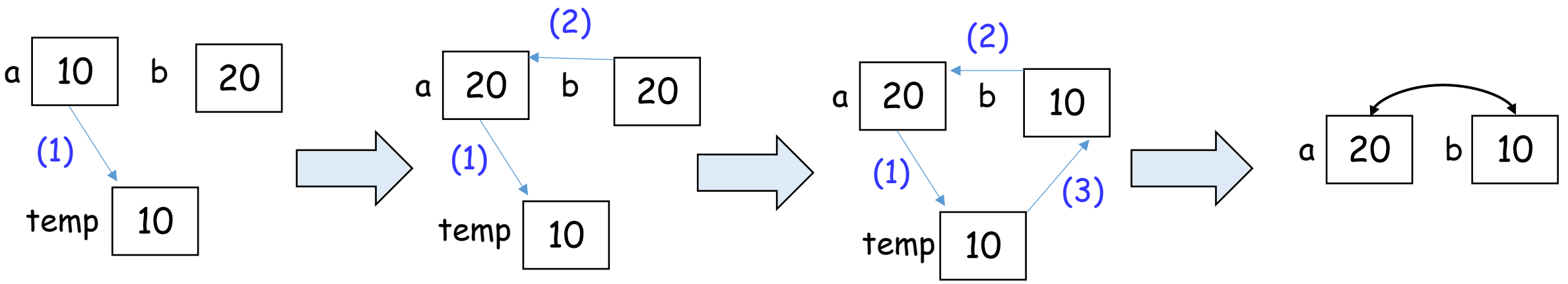
- 성적 처리 특성 분석
  - 처리해야할 작업 중 주요 기능 분석
  - 총점 계산하기 (add())
  - 총점을 사용하여 평균 계산하기
  - 총점을 반영하여 성적순으로 정렬하기
  - 정렬을 위해 두 변수의 값을 서로 바꾸는 함수 필요(swap())

생각 하기 : 두 개의 변수 값을 서로 바꾸려면 ? (Python)



swap( )

a, b = b, a

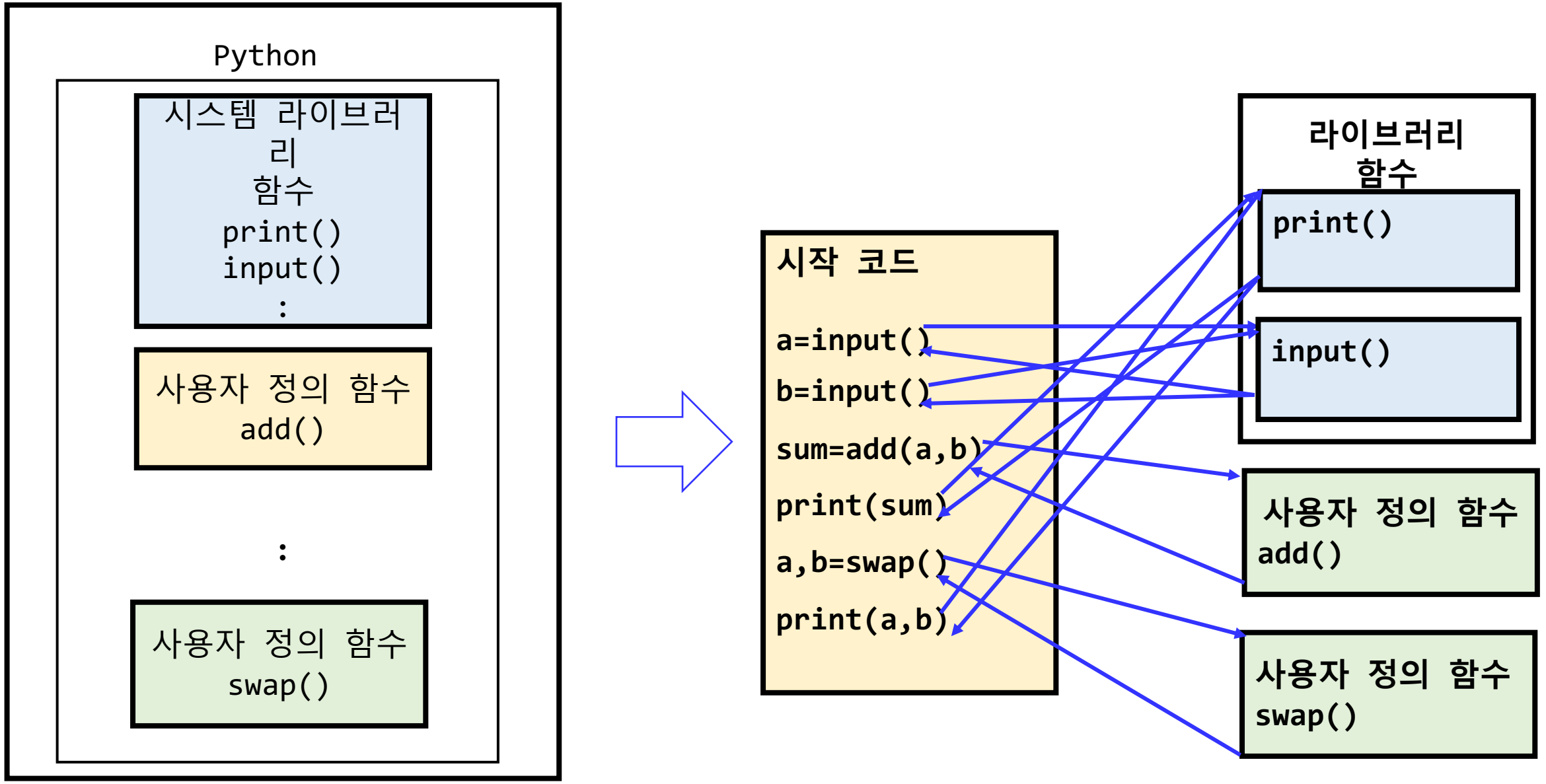


temp = a

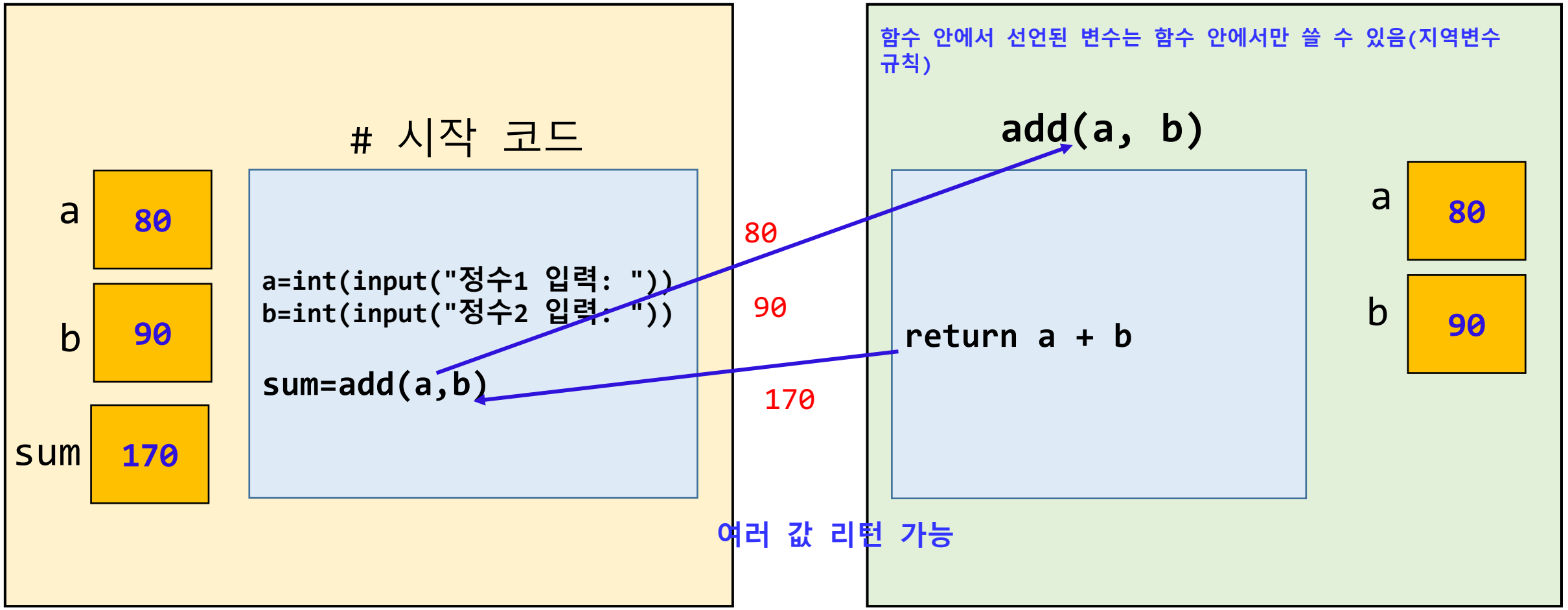
a = b

b = temp

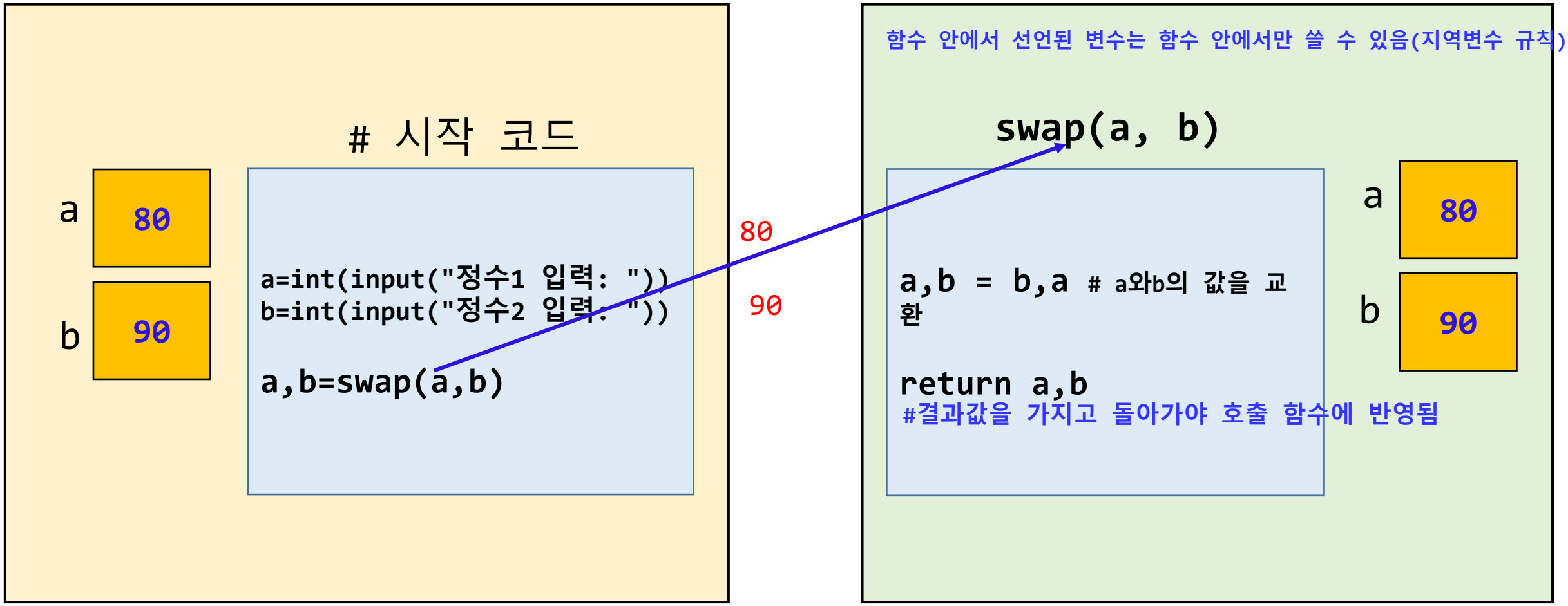
함수 호출 방법(Python)



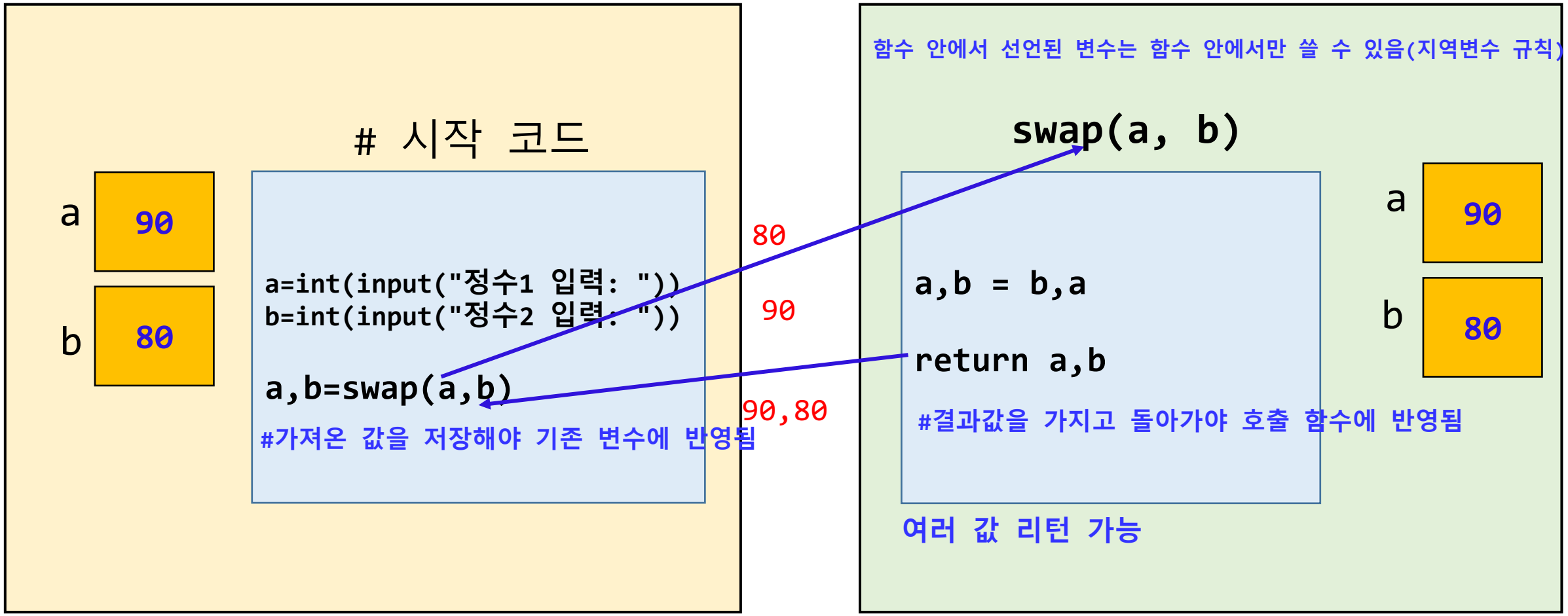
add() 함수 호출하기


함수 안에서 선언된 변수는 함수 안에서만 쓸 수 있음(지역변수 규칙)
시작 코드
a
80
b
90
sum
170
a=int(input("정수1 입력: "))
b=int(input("정수2 입력: "))
sum=add(a,b)
80
90
170
add(a, b)
return a + b
a
80
b
90
여러 값 리턴 가능

swap() 함수 호출하기


a
80
b
90
# 시작 코드
a=int(input("정수1 입력: "))
b=int(input("정수2 입력: "))
a,b=swap(a,b)
80
90
함수 안에서 선언된 변수는 함수 안에서만 쓸 수 있음(지역변수 규칙)
swap(a, b)
a,b = b,a # a와b의 값을 교환
return a,b
#결과값을 가지고 돌아가야 호출 함수에 반영됨
a
80
b
90

swap() 함수 호출 후 결과



함수 호출 예제(Python)

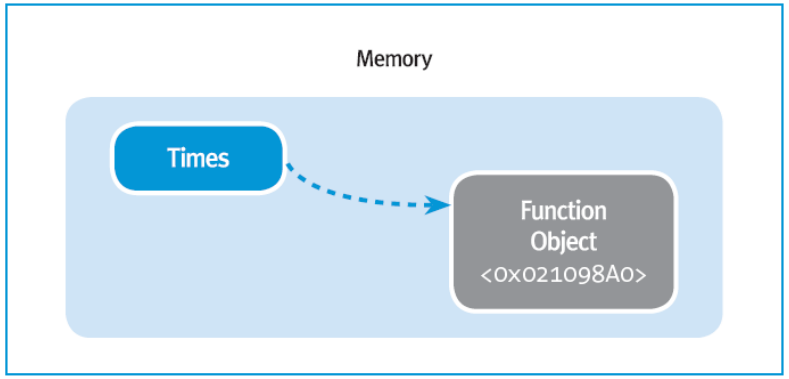
```
# 함수
def add(a,b):
      return a+b
def swap(a,b):
      a,b = b,a
      return a,b

# 시작코드
a=int(input("정수1 입력: "))
b=int(input("정수2 입력: "))
sum=add(a,b)
average=sum/2
print("두 수의 합 : ", sum)
print("두 수의 평균 : ", average)
a,b=swap(a,b)
print("두 수의 교환 : ", a,b)
```

```
정수1 입력: 80
정수2 입력: 90
두 수의 합 :  170
두 수의 평균 :  85.0
두 수의 교환 :  90 80
```


19

함수를 선언하면...

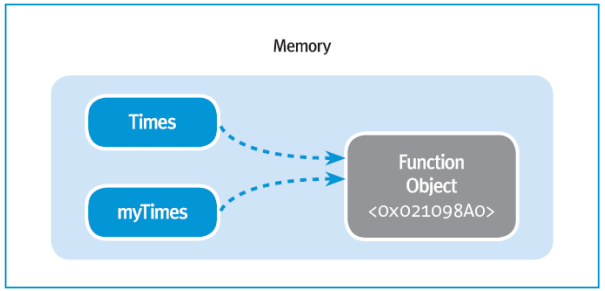
- 메모리에 함수 객체가 생성
- 함수 객체를 가리키는 레퍼런스가 생성



- 함수 레퍼런스를 통해서 함수를 사용
  - 함수 레퍼런스는 다른 변수에 할당 할 수 있음

```
>>> myTimes = Times
```

```
>>> r = myTimes(10, 10)
>>> r
100
```



return

- 함수를 종료시키고 호출한 곳으로 돌아가게 함
- return은 어떠한 객체로 돌려줄 수 있음
  - 여러 개의 값을 튜플로 묶어서 값을 전달 할 수 있음

```
>>> def swap(x, y):
        return y, x

>>> swap(1, 2)
(2,1)
```

- return을 사용하지 않거나 return만 적을 때도 함수가 종료
  - 리턴값으로 None을 리턴

```
>>> def setValue(newValue):
        x = newValue        <-- 반환 값이 없는 경우

>>> retval = setValue(10)

>>> print(retval)
None
```

함수 호출 시 parameter 값 갱신 기준(Python)

- Immutable(값 객체)
  - 변수에 저장된 것을 값으로 인식하며 변수는 치환이 가능하지만 변경은 안됨
  - 문자열은 임의적으로 값 객체로 정의

- Mutable(참조 객체)
  - 변수에 저장된 것은 객체의 요소(값)들이 저장된 참조이므로 실제 값들이 변경이 가능
  - 함수에 매개변수로 전달할 경우 참조만 넘어가므로 요소들이 변경이 가능

변할 수 있는 (Mutable)	변할 수 없는 (Immutable)
리스트형(list) 사전형(dict) 집합형(set)	숫자형(numbers) 문자열형(string) 튜플형(tuple)

함수 호출 방법(Python)

파이썬의 리스트는 객체의 참조가 함수로 전달
전달된 객체를 참조하여 변경 시 호출자에 영향을 주나,
새로운 객체를 만들 경우 호출자에게 영향을 주지 않음

```
def spam(eggs):
        eggs.append(1)        # 기존 호출자 ham 객체에 [1] 추가
        eggs = [2, 3]         # 새로운 eggs 객체 생성
        print(eggs)           # [2, 3]

# 시작 코드
ham = [0]                     # list data type (mutable:변경 가능)
spam(ham)                     # 함수 호출
print(ham)                    # [0, 1]
```

Function – Python (swap ex1)

```python
def sum (a,b):
    return a+b

def swap01(a,b):
    a,b = b,a
    print(a,b) # 20 10

def swap02(list1):
    list1[0],list1[1] = list1[1],list1[0]
    print(list1) # [90 80]

a=10; b=20

result = sum(a,b)
print(result) # 30

print(a,b) # 10 20
swap01(a,b)
print(a,b) # 10 20

list1=[80,90]
print(list1) # [80 90]
swap02(list1)
print(list1) # [90 80]
```

```
30
10 20
20 10
10 20
[80, 90]
[90, 80]
[90, 80]
```

Function – Python (swap ex2)

```python
def sum (a,b):
    return a+b

def swap01(a,b):
    a,b = b,a
    print(a,b) # 20 10
    return a,b

def swap02(list1):
    list1[0],list1[1] = list1[1],list1[0]
    print(list1) # [90 80]

a=10; b=20

result = sum(a,b)
print(result) # 30

print(a,b) # 10 20
a,b = swap01(a,b)
print(a,b) # 20 10

list1=[80,90]
print(list1) # [80 90]
swap02(list1)
print(list1) # [90 80]
```

```
30
10 20
20 10
20 10
[80, 90]
[90, 80]
[90, 80]
```

람다(lambda) 함수

- 익명 함수로서 간단한 함수를 한 줄로 작성
- 람다 함수는 lambda 키워드를 사용하여 정의되며, 함수의 이름 없이 매개변수와 표현식만으로 구성
- 람다 함수는 간단한 함수를 한 줄로 작성할 때 유용하며, 주로 함수를 인자로 받는 함수나 정렬, 필터링 등의 작업에서 많이 활용
- 람다 함수는 익명 함수이므로 함수의 이름을 정의할 필요가 없고, 간결하고 가독성이 좋은 코드를 작성

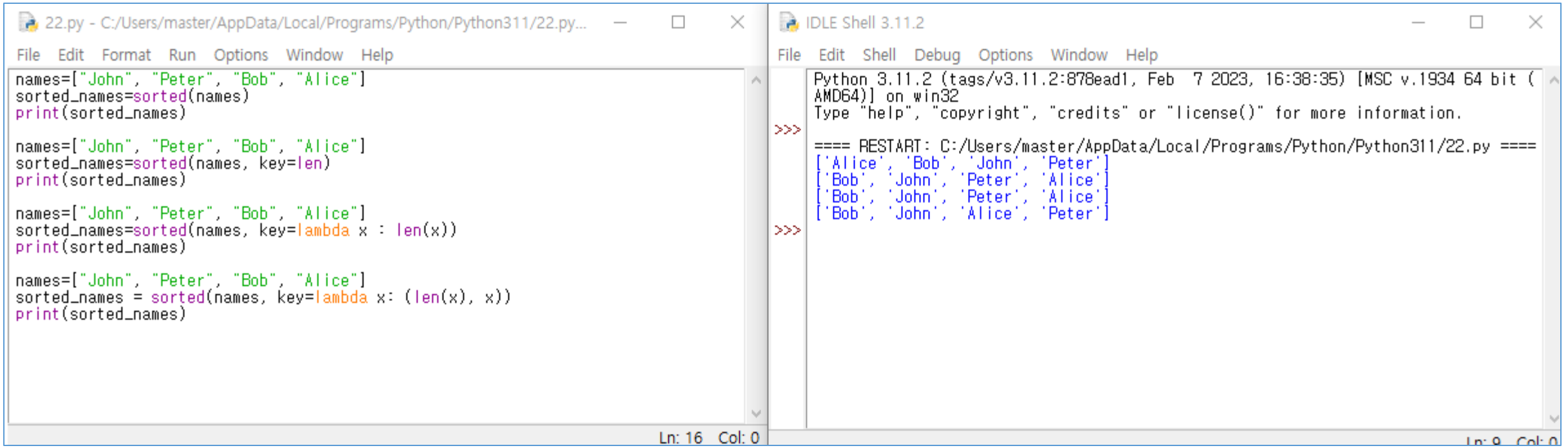
```
# 람다 함수 사용 예제 1: 두 수의 합 구하기

sum_func = lambda a, b: a + b

result = sum_func(3, 5)

print(result)  # 출력: 8
```

람다(lambda) 함수 예제

- 리스트 원소의 문자열 길이 기준으로 정렬
- sorted 함수의 key 매개변수에 람다 함수를 사용하여 문자열의 길이를 기준으로 정렬
- names 리스트를 sorted 함수에 전달하고, key 매개변수로 람다 함수를 사용하여 각 문자열의 길이를 반환하여 정렬 기준으로 사용

```
# 람다 함수 사용 예제 2: 문자열 길이 기준으로 정렬
names = ['John', 'Alice', 'Peter', 'Bob']
sorted_names = sorted(names, key=lambda x: len(x))
print(sorted_names)  # 출력: ['Bob', 'John', 'Alice', 'Peter']
```

람다(lambda) 함수 예제



변수의 유효범위

- 지역변수(local variable) : 함수 내에서만 사용
- 전역변수(Global variable) : 프로그램전체에서 사용

- 함수 내에 전역변수 사용시 global 키워드 사용

```
def add(x , y): # 더하기 함수
    global z
    print("In function- z : ", z) # 100
    z=200
    print("In function- z : ", z) # 200
    return x + y

# 시작
x = 10
y = 20
z = 100

print(" x : ", x)  # 10
print(" y : ", y)  # 20
print(" z : ", z)  # 100

sum = add(x, y) # 함수 호출

print("%d + %d = %d "% (x,y,sum))
print(" In program - z :", z) # 200
```

```
 x :  10
 y :  20
 z :  100
In function- z :  100
In function- z :  200
10 + 20 = 30
 In program - z : 200
```

변수의 유효범위

```
def calculate(x, y):
    global total          # global 선언하고 전역변수 변경 시 변경값이 전역에 반영
    total = x + y
    print("In Function")
    print("a:", str(a), "b:", str(b), "a+b:", str(a+b),"total :", str(total))
    # a: 5 b: 7 a+b: 12 total : 12
    return total

a = 5             # a와 b는 전역변수
b = 7
total = 0         # 전역변수 total

print("In Program")
print("a:", str(a), "b:", str(b), "a+b:", str(a+b))
# a: 5 b: 7 a+b: 12

sum = calculate (a,b)
print("After Calculation")
print("Total :", str(total), " Sum:", str(sum))
#Total : 12  Sum: 12
```

변수의 범위

전역변수는 함수에서 사용가능하나, 함수 내에서 전역변수에 새로운 값을 할당했을 경우, 새로운 지역변수가 생김

```
def calculate(x, y):
    total = x + y          # 새로운 값이 할당되어 함수내 total은 지역변수가 됨
    print("In Function")
    print("a:", str(a), "b:", str(b), "a+b:", str(a+b),"total :", str(total))
    # a: 5 b: 7 a+b: 12 total : 12
    return total

a = 5              # a와 b는 전역변수
b = 7
total = 0          # 전역변수 total

print("In Program")
print("a:", str(a), "b:", str(b), "a+b:", str(a+b))
# a: 5 b: 7 a+b: 12

sum = calculate (a,b)
print("After Calculation")
print("Total :", str(total), " Sum:", str(sum)) #지역변수는 전역변수에 영향 X
#Total : 0  Sum: 12
```

모듈

- 파이썬에서 **모듈(module)**이란 함수나 변수 또는 클래스 들을 모아 놓은 파일
- 모듈은 파이썬 문장들이 저장된 파일

모듈 작성하기

fibo.py

```
# 피보나치 수열 모듈

def fib(n):          # 피보나치 수열 출력
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n): # 피보나치 수열을 리스트로 반환
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

모듈 사용하기

```
>>> import fibo

>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

keyword 모듈

```
import keyword

name = input("변수 이름을 입력하시오: ")

if keyword.iskeyword(name):
    print (name, "은 예약어임.")
    print ("아래는 키워드의 전체 리스트임: ")
    print (keyword.kwlist)
else:
    print (name, "은 사용할 수 있는 변수이름임.")
```

```
변수 이름을 입력하시오: for
for 은 예약어임.
아래는 키워드의 전체 리스트임:
['False', 'None', 'True', 'and', 'as', 'assert', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
```

random 모듈

```
import random

# 1부터 10까지의 무작위 정수 생성
random_number = random.randint(1, 10)
print("무작위 정수:", random_number)

# 0부터 1 사이의 무작위 실수 생성
random_float = random.random()
print("무작위 실수:", random_float)

# 리스트 요소를 무작위로 섞기
my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print("무작위로 섞인 리스트:", my_list)

# 주어진 리스트에서 무작위로 하나의 요소 선택
my_list = [1, 2, 3, 4, 5]
random_element = random.choice(my_list)
print("무작위로 선택된 요소:", random_element)
```

```
무작위 정수: 7
무작위 실수: 0.5736247656549271
무작위로 섞인 리스트: [2, 5, 4, 1, 3]
무작위로 선택된 요소: 3
```

예제: 동전 던지기 게임

•random 모듈을 사용하여 아래와 같이 출력 되도록 프로그램 작성

```
동전 던지기를 계속하시겠습니까? (yes, no): yes
동전 결과: 앞면
동전 던지기를 계속하시겠습니까? (yes, no): yes
동전 결과: 뒷면
동전 던지기를 계속하시겠습니까? (yes, no): no
```

Solution

```
import random

def coin_toss():
    coin = random.choice(['앞면', '뒷면'])
    return coin

def play_game():
    while True:
        response = input("동전 던지기를 계속하시겠습니까? (yes, no): ")
        if response == 'yes':
            result = coin_toss()
            print("동전 결과:", result)
        else:
            break

play_game()
```