

K-NET DEVELOP #6

Hyungyu Kim

K-NET

2024-05-28

시작에 앞서서...

- Q. for문을 이용하면 join같은 함수가 필요없지 않나요?
- A. join() 등의 내장 함수는 최적화가 잘 되어 있어서 for문보다 실제로 더 빠릅니다.
(파이썬의 'time 모듈'을 이용해 실제로 확인 가능합니다.)
(time 모듈은 파이썬의 import문을 이용해 불러올 수 있습니다.)
- A. 또한 join()은 for문보다도 코드 작성자의 의도를 더 명시적으로 보여줍니다.
(사용자의 의도를 담은 새로운 함수를 여러분도 만들 수 있습니다.)

시작에 앞서서...

```
1  import time
2
3  alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
4  example = [alphabet[i % 26] for i in range(1000000)]
5
6  start = time.time()
7  result = "".join(alphabet)
8  end = time.time()
9  print(f"join 이용: {end - start}s")
10
11 start = time.time()
12 result = ""
13 for char in example:
14     result += char
15 end = time.time()
16 print(f"for문 이용: {end - start}s")
```

join 이용: 0.0001392364501953125s

for문 이용: 0.51656174659729s

지난 시간에 이어서...

- 변수, 배열, 리스트, 딕셔너리, 조건문, 반복문, 함수, ...
- 이번 시간에는 함수를 메인 주제로 하되, 모듈을 곁다리에 놓고 이야기하겠습니다.
- 지난 시간까지가 문장과 문법이었다면 이번 시간은 한 개의 문단과 글을 쓰는 법에 해당합니다.
- 문단은 함수(와 클래스)에 대응되고, 글은 모듈에 대응됩니다.
- 함수는 반복문과 함께 이번 스터디의 메인 주제입니다.

막간을 이용한 홍보: 집합(Set)

- 리스트와 비슷하지만 중복된 것을 같은 것으로 취급하는 자료구조입니다.
- 수학에서 말하는 집합과 비슷하고, 코딩 테스트나 실무에서 유용합니다.
(리스트에서 중복을 삭제하고 싶을 때 `set()`으로 감싸면 좋음)
- 딕셔너리처럼 중괄호로 감싸서 사용합니다.
- `set`과 관련된 연산: 원소 추가/제거, 합집합/교집합/차집합, `in` 연산자, 부분집합 등...

```
1  A = [1, 2, 1, 2, 3, 1, 2, 3, 4]
2
3  A = set(A)
4
5  print(A)
```

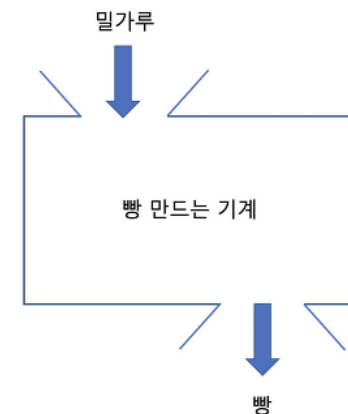
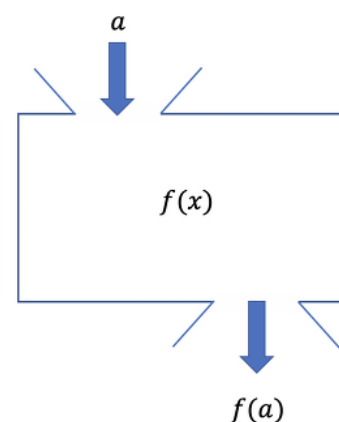
```
{1, 2, 3, 4}
```

막간을 이용한 홍보: 집합(Set)

- 참고: set은 mutable 객체이며, 리스트나 딕셔너리처럼 수정이 가능합니다.
- 이뮤터블(immutable) 객체: int, float, str, bool, tuple, ... → call by value와 관련
- 뮤터블(mutable) 객체: list, set, dict, ... → call by reference와 관련
- 곧 나올 call by assignment를 간략히 설명할 때 필요한 내용입니다.

함수는 코드 여러 줄을 묶은 것

- 프로그래밍에서의 함수는 보통의 경우 2가지 의미를 가집니다.
 1. 입력과 출력이 있는 (수학적 의미의) 함수
 2. 코드(계산 과정) 여러 줄을 묶어놓은 것
(재사용과 관리가 용이하고 가독성에 유리함)
- 실제 수학의 함수처럼 동작하고 호출할 수 있지만 사용 예시가 미묘하게 다릅니다.
- 프로그래밍에서의 함수를 잘 쓰고 싶다면 2번의 의미에 집중해주시기 바랍니다.



함수는 코드 여러 줄을 묶은 것

- 함수 사용, 함수 호출 예시 (`def`, `return` 등의 키워드에 주목하세요!)
- 참고: 아래 함수에서 `a`, `b` 모두는 정수, 실수, 리스트, 문자열, 튜플 등이 될 수 있습니다.



```
# 함수 예시 - 너무 쉬운가요?
```

```
def add(a, b):  
    return a + b
```

```
# 함수 호출
```

```
result = add(3, 4)  
print(result)
```


함수는 코드 여러 줄을 묶은 것

- 함수 사용, 함수 호출 예시

```
1 # 함수 호출 예시
2
3 example = fibo(10)
4 print(example)
```

55

```
# 함수 예시
```

```
def fibo(num):
```

```
    '''
```

피보나치 수열의 num번째 항을 구하는 함수입니다.

지난 주차에서 보여드린 코드를 함수로 묶은 것입니다.

num은 0부터 시작이며 0번째 항, 1번째 항은 모두 1, 1입니다.

```
    '''
```

```
    if num == 0: return 1
```

```
    if num == 1: return 1
```

```
    a, b = 1, 1
```

```
    for _ in range(num-2):
```

```
        res = a + b
```

```
        a, b = b, res
```

```
    return res
```

모듈, 패키지, 라이브러리? (간단히만)

- **모듈**: 여러분이나 다른 사람이 미리 만든 함수, 변수, 클래스를 모아놓은 **파이썬 코드 파일**입니다.
- 모듈의 함수는 아까 전 슬라이드의 1번 의미, 2번 의미를 주로 모두 가지고 있습니다.
- 맨 처음 살펴본 코드의 `time`도 모듈이고, 지난 주 if문의 예시에 있던 러시아안 룰렛 게임 코드의 `os`, `random`도 모듈입니다.
- 모듈은 파이썬에 기본적으로 탑재된 **내장 모듈**과, 따로 설치해주어야 하는 **외장 모듈**로 나뉩니다.
- 참고: `time`, `os`, `random`, `math` 등은 내장 모듈입니다.


모듈, 패키지, 라이브러리? (간단히만)

- **패키지**: 모듈의 집합입니다. (예시: numpy, pandas, ...)
- **라이브러리**: 여러 패키지와 모듈을 집합한 것입니다. (예시: matplotlib, ...)
- 라이브러리는 **재사용 가능한 코드**를 좀 더 포괄적으로 칭하는 용어입니다.
- 모듈이 모이면 패키지가 되고, 패키지가 모이면 라이브러리가 됩니다.
- 모듈, 패키지, 라이브러리는 **import**문을 이용해 불러올 수 있습니다.

```
└─ my_app
   ├── main.py
   ├── package1
   │   ├── module1.py
   │   └── module2.py
   └── package2
       ├── __init__.py
       ├── module3.py
       ├── module4.py
       └── subpackage1
           └── module5.py
```

import문으로 모듈 불러오기

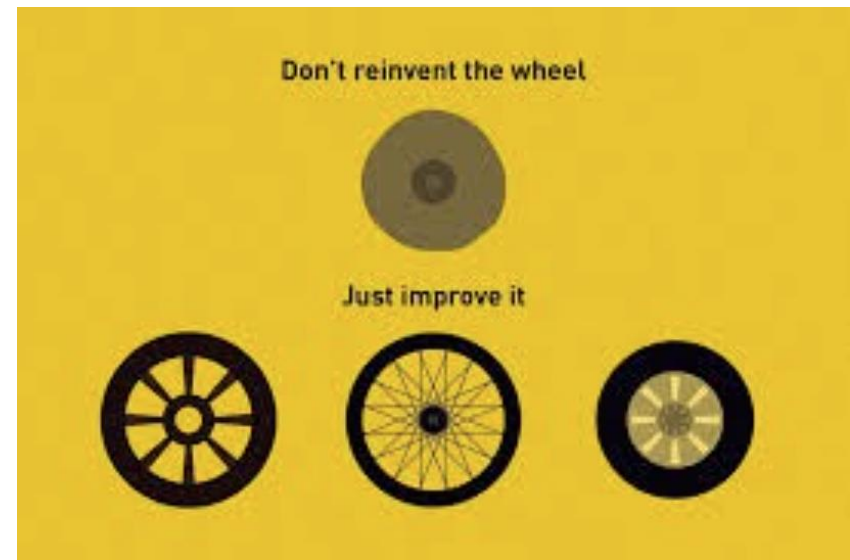
- `import` 모듈 (코드의 어디든지 쓸 수 있습니다.)
- `from` 모듈 `import` 이름 (모듈에서 무언가만 가져오고 싶을 때)
- `from` 모듈 `import *` (함수 전부 가져오기)
- `as` 키워드: 간략히 줄인 별칭을 쓸 때 좋습니다. (아래 예시에서 `pandas` – `pd` 관계)



```
import os
import pandas as pd
from matplotlib import pyplot as plt
```

모듈의 존재 의의

- 프로그래밍 고수들이 만들어놓은 것을 잘 써먹으면 좋기 때문입니다.
- 내가 예전에 만든 것을 재활용할 때도 용이하기 때문입니다.
- 깃허브에서 볼 수 있는 파이썬 오픈소스 프로젝트는 상당수가 패키지 혹은 라이브러리라고 볼 수 있습니다.
- “(굴러가는 자동차를 만들기 위해) 바퀴를 재발명하지 말라!”
- 모듈 이야기는 여기까지만 하겠습니다...



함수의 존재 의의

- 함수를 써서 코드를 세부 기능 단위로 분리하면 코드가 깔끔해집니다.
- 예시: 2부터 1000까지 소수가 몇 개인지 출력하는 프로그램
- 코드가 조금 복잡합니다.
- 참고: break는 안쪽의 for문 1개만 빠져나오는 역할만 합니다.

```
1  cnt = 0 # 한 개씩 개수 세기
2  for num in range(2, 1001):
3      is_prime = True
4      # 2~1000에 대해 소수 여부 검사
5      for j in range(2, num): # 중첩 반복문!
6          if num % j == 0:
7              is_prime = False
8              break
9      if is_prime:
10         cnt += 1
11
12  print(cnt)
```


함수의 존재 의미

- 함수를 써서 코드를 세부 기능 단위로 분리하면 코드가 깔끔해집니다.
- 예시: 2부터 1000까지 소수가 몇 개인지 출력하는 프로그램
- 새로운 함수를 도입했더니 코드가 이렇게나 깔끔해졌죠?

```
1 def is_prime(num): # 소수 판별 함수 정의 (출력이 True 혹은 False)
2     for j in range(2, num):
3         if num % j == 0:
4             return False
5     return True
6
7 # 개수 세기 - 함수를 이용하였더니 중첩 반복문의 형식을 피할 수 있게 됨
8 cnt = 0
9 for num in range(2, 1001):
10     if is_prime(num):
11         cnt += 1
12 print(cnt)
```

변수명, 함수명, 파일명, ...

- 변수 이름처럼 함수 이름, 파일 이름도 직관적으로 잘 지어야 합니다.
- 잘 지으면 영어 문장 읽듯이 코드가 읽힙니다.
- 참고: 아래의 is_prime은 True 혹은 False만 출력하는 함수였습니다.



```
for num in range(2, 1001):  
    if is_prime(num):  
        cnt += 1
```


함수로 묶는 기준?

- 참고: 로직이 좀 복잡해지겠다 싶으면 그 부분을 함수로 묶어버리도록 합시다.
- 처음 연습할 땐 가능한 작은 단위로 다 쪼개보세요!
- 여러 번 하다 보면 어렵지 않다는 것을 알 수 있습니다.
- 참고: 함수의 컨셉에 대한 이해가 있으면 [남의 코드\(i.e. 라이브러리\)](#) 가져다쓰기도 즐겁습니다.

입출력이 없는 함수!

- 프로그래밍의 함수는 반드시 입출력이 있을 필요는 없습니다.
- 입력(input)이 없는 경우: 함수 정의 시 소괄호 안을 비워주면 됩니다.
- 출력(output)이 없는 경우: return 키워드는 생략해주면 됩니다. return을 함수 탈출 용도로만 써도 됩니다. (break와 비슷)

```
# 함수의 입출력이 모두 있음
def say_hi(name):
    return f"Hello, {name}!"
```

```
# 리턴 값이 없음
def say_hi(name):
    print(f"Hello, {name}!")
```

```
# 인풋이 없음
def say_hi():
    return "Hello World!"
```

```
# 입출력이 모두 없음
def say_hi():
    print("Hello World!")
```

매개변수와 인자

- 소괄호 안에 들어가는 name을 **파라미터(parameter)** 또는 **매개변수**라고 합니다.
- 실질적으로 매개변수를 통해 함수에 들어가게 되는 값들(아래 예시에서는 “John”)을 **인자(argument)**라고 합니다.

```
1 def say_hi(name):  
2     print(f"Hello, {name}!")  
3  
4 say_hi("John")
```

```
Hello, John!
```

return 키워드의 쓰임새

- 함수의 출력 값을 반환하는 용도 (그 이후에는 즉시 함수를 탈출)
- 그냥 그 자리에서 탈출
- 보통 함수의 출력을 “리턴 값”, “반환 값”이라고 자주 칭합니다.
- 참고: return문을 쓰지 않아도 코드를 모두 실행한 후에는 함수를 자동적으로 빠져나오게 됩니다.

return문의 사용 예시

- 예시 - return의 쓰임새에 주목하여 봐주세요. (함수의 반환 값)

```
def fibo(num):  
    if num == 0: return 1 # 1을 반환 후 바로 탈출  
    if num == 1: return 1 # 1을 반환 후 바로 탈출  
  
    # 위에서 함수가 탈출하지 못한 경우 (즉, num >= 2인 경우) 아래를  
    # 실행할 수 있게 된다.  
    a, b = 1, 1  
    for _ in range(num-2):  
        next = a + b  
        a, b = b, next  
    return next # num번째 항을 출력한 후 함수를 빠져나오기
```

return문의 사용 예시

- 예시 - return의 쓰임새에 주목하여 봐주세요. (break와 흡사)

```
# return과 break는 흡사합니다.

def up_down_game():
    import random
    secret_num = random.randint(100) # 0과 100 사이의 정수를 선택
    cnt = 1

    while True:
        answer = int(input("자연수 하나를 추측해보세요: "))
        if answer == secret_num:
            print(f"맞았습니다! 총 {cnt}번 시도하셨습니다.")
            return # 강제로 함수 빠져나가기
            # 이는 break와 흡사하지만 반복문 뿐만 아니라 함수 자체를 빠져나옴
        if answer > secret_num:
            print(f"다운! {cnt}번째 시도입니다.")
        if answer < secret_num:
            print(f"업! {cnt}번째 시도입니다.")
        cnt += 1
```

파이썬 리턴 값은 두 개, 세 개도 되나?

- 참고: 리턴 값은 두 개, 세 개 쓸 수 있는 것처럼 보이지만 사실은 한 개의 튜플입니다.

```
1 def add_sub_mul_div(a, b):  
2     if b != 0:  
3         return a+b, a-b, a*b, a/b  
4     else:  
5         return a+b, a-b, a*b, None
```

```
1 x = add_sub_mul_div(5, 4)  
2  
3 print(f"x == {x}; type(x) == {type(x)}; len(x) == {len(x)}")
```

```
x == (9, 1, 20, 1.25); type(x) == <class 'tuple'>; len(x) == 4
```

참고: docstring(독스트링)

- 함수나 클래스를 선언한 후 바로 아랫 줄에 함수 관련 설명을 적어줄 수 있습니다.
- `help(함수명)`을 통해 독스트링을 불러올 수 있습니다.

```
1  def is_prime(num): # 아까 그 함수
2      '''
3      num이 소수인지 판별하는 함수입니다.
4      여러분, 참고로 이 함수는 아까 전에 썼던 함수입니다.
5      '''
6      for j in range(2, num):
7          if num % j == 0:
8              return False
9      return True
10
11  help(is_prime)
```

```
Help on function is_prime in module __main__:
```

```
is_prime(num)
    num이 소수인지 판별하는 함수입니다.
    여러분, 참고로 이 함수는 아까 전에 썼던 함수입니다.
```


참고: docstring(독스트링)

- 팁: 프로그래머스 풀 때 긴 문제 이해 시에 용이합니다. (문제 지문 일부를 하나씩 독스트링에 복사하기...)

```
def solution(N, stages):  
    ...  
  
    stages에는 1 이상 N + 1 이하의 자연수가 담겨있다.  
    - 각 자연수는 사용자가 현재 도전 중인 스테이지의 번호를 나타낸다.  
    - N + 1 은 마지막 스테이지(N 번째 스테이지) 까지 클리어 한 사용자를 의미한다.  
    실패율이 같은 스테이지가 있다면 작은 번호의 스테이지가 먼저 오도록 하면 된다.  
    도달한 유저가 없는 경우 해당 스테이지의 실패율은 0 으로 정의한다.  
    ...  
  
    answer = []  
    for stage in range(1, N+1):  
        failure = stages.count(stage)
```

위치 인수와 키워드 인수

- 함수가 인수를 여러 개 받을 때 이 두 가지를 알면 파이썬에서 조금 더 편리할 수 있어요.
- 말로 설명하는 것보다 직접 보는 것이 더 이해하기 좋습니다.



```
def fibo(num, first, second):  
    '''  
    0번째 항이 first, 1번째 항이 second인 피보나치 수열의 num번째 항을 반환합니다.  
    '''  
    if num == 0: return first  
    if num == 1: return second  
    for _ in range(num-2):  
        next = first + second  
        first, second = second, next  
    return next
```

위치 인수와 키워드 인수

- 함수가 **인수를 여러 개** 받을 때 이 두 가지를 알면 파이썬에서 조금 더 편리할 수 있어요.
- 말로 설명하는 것보다 직접 보는 것이 더 이해하기 좋습니다.

```
1  # 위치 인수 - 숫자가 들어가는 순서가 중요
2  print(fibo(11, 0, 1))
```

55

```
1  # 키워드 인수 - 키워드 값 지정이 중요
2  print(fibo(num=11, first=0, second=1))
3  print(fibo(second=1, num=11, first=0))
```

55

55

매개변수 기본값

- 매개변수를 생략할 시에도 자동적으로 값을 지정할 수 있게끔 하는 방법입니다.
- 이것 역시 예시를 통해 살펴보면 이해하기 쉽습니다.

```
1 def fibo(num, first=0, second=1):
2     '''
3     0번째 항이 first, 1번째 항이 second인 피보나치 수열의 num번째 항을 반환합니다.
4     first, second는 각각 0, 1이 기본 값입니다.
5     '''
6     if num == 0: return first
7     if num == 1: return second
8     for _ in range(num-2):
9         next = first + second
10        first, second = second, next
11    return next
12
13 fibo(10)
```

34

```
1 fibo(num=10)
```

34

지역 변수를 의식하면서 코딩하기

- 함수 내에서 선언한 변수들은 “지역 변수”들이라 함수 밖에서는 사용할 수 없습니다.
- 함수의 실행이 끝나면 함수 내의 지역 변수들은 사라집니다.

```
1 def fibo(num, first=0, second=1):
2     if num == 0: return first
3     if num == 1: return second
4     for _ in range(num-2):
5         third = first + second
6         first, second = second, third
7     return third
8
9 print(third)
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-36-86040d85d51a> in <cell line: 9>()
      7     return third
      8
----> 9 print(third)

NameError: name 'third' is not defined
```

지역 변수를 의식하면서 코딩하기

- 지역 변수는 함수의 안쪽 범위에서만 사용할 수 있습니다.
- 아래에서 함수 밖의 birthday와 함수 안의 birthday는 상관없습니다. (이름이 겹쳐도!)
- 참고: 이는 네임스페이스에 관한 내용이므로 관심있으신 분은 검색 바랍니다.

```
1 birthday = 2608
2
3 def my_mum():
4     birthday = 1111 # 그저 지역 변수일 뿐
5     print(f"birthday = {birthday}")
6
7 my_mum()
8 print(birthday)
```

```
birthday = 1111
2608
```

```
1 birthday = [8, 26]
2
3 def my_mum():
4     birthday = [11, 11]
5     print(f"birthday = {birthday}")
6
7 my_mum()
8 print(birthday)
```

```
birthday = [11, 11]
[8, 26]
```

충격 사실: 함수도 객체다!

- 파이썬의 모든 것은 객체이고, 함수 역시 메모리에 저장되는 객체입니다.
- 엥? 함수가 객체?
- 진짜인가 볼까요?

- !!!!!

```
1  type(fibo), type(print)  
  
(function, builtin_function_or_method)
```

- type()으로 감싸서 출력해보니 function이라고 뜹니다!
- 특히 print()는 “내장 함수”(builtin function)이라고 뜹니다.

함수도 객체다!

- 그래서 함수 매개변수의 인자로 함수도 받을 수 있습니다!
- C언어의 함수 포인터와 유사합니다.
- 아래에서 sorted() 함수의 key 파라미터를 함수 이름 key로 주었습니다!

```
1 def key(x): return x[1]
2
3 example = [("짱구", 100), ("철수", 98), ("유리", 104),
4            ("맹구", 102), ("훈이", 99)]
5
6 # example의 각 요소(튜플)의 1번째 요소(100, 98, ...)에 의해 정렬하기
7 sorted(example, key=key)
```

[('철수', 98), ('훈이', 99), ('짱구', 100), ('맹구', 102), ('유리', 104)]

lambda 함수 (익명 함수)

- 간단한 함수에 굳이 def나 이름 등을 붙이고 싶지 않을 때 씁니다.
- 함수를 한 줄로 압축할 수 있습니다. 미니멀리즘 실현 야무집니다.

```
def key(tup): # def 키워드랑 이름 붙여줘야 됨
    return tup[1]
```

```
def add(x, y): # def 키워드랑 이름 붙여줘야 됨
    return x+y
```

한 줄로 줄이는 대신 이 자체로는 이름이 없습니다.

```
lambda tup: tup[1]
lambda x, y: x+y
```

lambda 함수 (익명 함수)

- 예시: 아까 sorted 예시에서 lambda 함수로 바꾸어서 실제로 자주 사용합니다.
- 아까와 같이 간단한 함수를 lambda 함수로 바꿔서 자주 씁니다.

```
1 example = [("짱구", 100), ("철수", 98), ("유리", 104),  
2           ("맹구", 102), ("훈이", 99)]  
3  
4 # lambda로 쓴 함수 역시 객체입니다. 람다 식 전체를 객체로 생각해도 괜찮습니다.  
5 sorted(example, key=lambda x: x[1])
```

```
[('철수', 98), ('훈이', 99), ('짱구', 100), ('맹구', 102), ('유리', 104)]
```

lambda 함수 (익명 함수)

- lambda 함수가 요긴한 타이밍: map, filter (요긴해서 정말 자주 쓰임)
- map(함수, 리스트)를 쓰면 리스트의 각 원소에 함수를 씌워줄 수 있습니다.
- filter(함수, 리스트)를 쓰면 리스트에서 조건에 맞는 원소만 필터링해줄 수 있습니다.
- A = [1, 2, 3, 4]일 때,
- list(map(func, A)) == [func(1), func(2), func(3), func(4)]
- list(filter(lambda x: x%2==0, A)) == [2, 4]

lambda 함수 (익명 함수)

- map, filter와 lambda 사용 예시
- 참고: map은 매핑하기 역할 (각 원소에 함수값을 취해주기)
- 참고: filter은 걸러내기 역할 (조건에 맞는 것만 걸러내기)

```
1 example = [("짱구", 100), ("철수", 98), ("유리", 104),  
2           ("맹구", 102), ("훈이", 99)]  
3 list(map(lambda x: x[1], example))
```

```
[100, 98, 104, 102, 99]
```

```
1 example = [("짱구", 100), ("철수", 98), ("유리", 104),  
2           ("맹구", 102), ("훈이", 99)]  
3 list(filter(lambda x: x[1] >= 100, example))
```

```
[('짱구', 100), ('유리', 104), ('맹구', 102)]
```

참고: call by assignment


- 요약: 파이썬에서 immutable 객체는 call by value, mutable 객체는 call by reference가 적용됩니다. (값에 의한 호출, 참조에 의한 호출)
- 잘 보면, 함수 func1가 정수 값의 원본을 못 바꾸고 있습니다.
(call by value – 정수 값의 복사본이 전달)
- 근데 func2는 리스트 객체의 원본을 바꿔버립니다.
(call by reference – 리스트의 “참조” 혹은 “레퍼런스”가 전달)

```
1 def func1(x):  
2     x += 1  
3  
4 def func2(list):  
5     list.append(1)  
6  
7 a = 0  
8 b = [0, ]  
9  
10 func1(a)  
11 func2(b)  
12  
13 print(a) # 안 바뀜  
14 print(b) # 바뀜
```

```
0  
[0, 1]
```

참고: 타입 힌트

- 파이썬에서도 함수의 매개변수와 출력 형식을 적어줄 수 있습니다.
- 깃허브의 많은 오픈소스 코드들을 보면 타입 힌트를 적어주고 있습니다.
- 그러나 이는 “힌트”일 뿐, 아래 예시에서 name의 자료형이 str(문자열)이 아닌 자료형을 인자로 주더라도 타입 힌트가 에러를 내지는 않습니다.



```
def say_hi(name: str) -> str:  
    return 'Hello ' + name
```

참고: 재귀 함수

- 함수 내부에서 자기 자신을 호출하는 기법입니다.
- 피보나치 수열을 구하는 함수는 재귀를 이용하면 코드가 깔끔합니다.
- 대신 여기서는 실행 시간은 반복문보다 더 느립니다. (경우에 따라 다름)
(“시간 복잡도”로 생각하셔도 되고, `time` 모듈을 불러와서 직접 알아보셔도 좋고…)
- 탈출 조건이 있어야 빠져나올 수 있습니다.

```
1 def recursive_fibo(n):  
2     if n == 0: return 1  
3     elif n == 1: return 1  
4     return recursive_fibo(n-1) + recursive_fibo(n-2)
```

```
1 recursive_fibo(10)
```

참고: *args, **kwargs?

- 파이썬은 포인터가 없으면서, 코드를 읽는데 왜 저런 별표시가 있는지 당황하신 분???
- 가변 인자(입력의 개수를 자유롭게 받고 싶은 경우)를 받을 때 사용됩니다.
- 가령 `print()`도 가변 인자를 받습니다. (인자 개수 자유)

```
sample_args.py > ...
1  import json
2
3  def greet(*users):
4      for user in users:
5          print(f'Welcome {user}')
6
7  def main():
8      greet('Fred', 'Harry', 'Tom')
9
10 if __name__ == '__main__':
11     main()
```

```
1  print(1) # 가변 인자
2  print(1, 2)
3  print(1, 2, 3)
4  print(1, 2, 3, 4)
5  print(1, 2, 3, 4, 5)
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```


참고: *args, **kwargs?

- 저 포인터 닮아서 보기 싫은 별표를 리스트 앞에 달아주면 (*args) 가변 인자가 됩니다.
- 별표 두 개(**kwargs)는 keyword arguments의 줄임말로 딕셔너리와 관계 있습니다.
- 중요한 포인트와 간단한 예시만 보여드리고 설명은 여기까지만...

```
1 args = [1, 2, 3, 4, 5, 6, 7]
2
3 print(1, 2, 3, 4, 5, 6, 7)
4 print(*args) # 가변 인자
```

```
1 2 3 4 5 6 7
1 2 3 4 5 6 7
```

```
1 def fibo(num, first=0, second=1):
2     if num == 0: return first
3     if num == 1: return second
4     for _ in range(num-2):
5         third = first + second
6         first, second = second, third
7     return third
8
9 kwargs = {'num': 10, 'first': 1, 'second': 1}
10 fibo(**kwargs)
```

55

화석 김현규의 끈대질 시간...

- 말씀드렸듯 오늘은 비공식적인 마지막 시간입니다!
- 파이썬이 안 맞으셨던 분들께 한두마디 마지막 말씀 올리겠습니다...
- 유형 1 - 그저 프로그래밍이 잘 안 맞으셨던 분들께
 괜찮습니다. 쉽게 떠오르는 간단한 코드부터 짜보거나 읽어보는 걸 추천해요.
- 유형 2 - 다른 언어는 괜찮지만 파이썬이 잘 안 맞으셨던 분들께
 파이썬이 항상 좋은 건 아닙니다. 파이썬으로부터는 이점만 챙기자구요.
 (코딩 테스트 등)

이 질문에 답을 찾아보세요

- 우리가 직접 유연하고 재사용성이 좋은 모듈(프로그램의 부품)을 만들려면 파이썬 파일 내에서 함수, 변수들을 어떻게 관리해야 할까요?
- 힌트: 다음 시간에 다룰 **클래스**와 관계 있습니다.
- 다음 시간은 시험 바로 코앞이죠?
- 오늘은 비공식적인 마지막 시간이고, 다음 시간은 **자유롭게** 참석하는 것으로 합시다.
(다음 시간에 딱 1명 오더라도 저는 클래스에 대한 설명을 하겠습니다)

우리 스터디에서 다룰 주요 주제

- 파이썬 조감도
- 변수와 리스트
- 문자열, 튜플, 딕셔너리
- 조건문과 반복문
- 함수와 모듈
- 클래스와 상속
- 파이썬을 이용한 문제 해결 (간단한 알고리즘 문제들)
- 파이썬 라이브러리로 할 수 있는 일 (이미지 처리, 엑셀 데이터 처리, 그래프 그리기, ...)

다음 시간에 다룰 주제

- 클래스와 상속 – 시간 되시는 분만 오세요! 시험이 더 중요하니까요.

