함수(function)

함수는 파이썬에서 코드를 재사용하고 구조화하기 위해 사용하는 수단으로, 같은 작업을 반복적으로 수행하거나 비슷한 코드를 한번이상 실행할때 사용합니다. 프로그램의 함수는 수학의 함수와 유사한 부분이 있습니다.

일반적으로 공통적으로 사용되는, 한번이상 반복되는 코드를 함수로 만듭니다. 또는 복잡한 수식의 경우도 식별 가능한 이름의 함수를 만들어 주는 것이 좋습니다.

1. 함수의 정의

```
# 함수의 선언 - 보통의 형태는 식별자()
def 함수명():
    실행할 코드

# 함수의 실행(=호출)

def print_twice(n):
    print('안녕하세요')
    print('안녕하세요')
    .....지정 횟수만큼 반복...

print_twice(10)
'''
안녕하세요
안녕하세요
안녕하세요
안녕하세요
...반복...
'''

# 매개변수 - 함수내로 값을 전달, 함수명(매개변수)
def 함수이름(매개변수):
    실행할 코드

def 함수이름(매개변수1,매개변수2):
    실행할 코드

def 함수이름(매개변수, 매개변수, ...):
    실행할 코드

def print_n_times(value, n):
    for i in range(n):
        print(value)

print_n_times('안녕하세요', 5)
'''
안녕하세요
안녕하세요
안녕하세요
안녕하세요
```

```
안녕하세요
'''

# 에러발생
print_n_times('안녕하세요')
# print_n_times() missing 1 required positional argument: 'n'
print_n_times('안녕하세요',5,10)
# print_n_times() takes 2 positional arguments but 3 were given

# 가변매개변수 - 원하는 만큼의 매개변수
def 함수이름(매개변수1, 매개변수2, ... *가변 매개변수):
    실행할 코드

def print_n_times(n, *values):
    for i in range(n):
        for value in values:
            print(value)
        print()

print_n_times(3, 'hello', 'python', 'world')
'''
hello
python
world

hello
python
world

hello
python
world
'''

# 기본 매개변수 - 기본값 지정 / 기본 매개변수 뒤 일반 매개변수 사용시 에러
def print_n_times(value, n=2):
    for i in range(n):
        print(value)

print_n_times('안녕하세요')
'''
안녕하세요
안녕하세요
'''

# 기본 + 가변 매개변수 (순서주의) ===> 키워드 매개변수
# 가변 매개변수는 *values, *args, *python 등 임의의 이름, 한번만 사용가능
# 기본 매개변수가 가변매개변수보다 앞에 오면 에러! (기본매개변수 의미가 없음)
def print_n_times(n=2, *values):
    for i in range(n):
        for value in values:
            print(value)
        print()

print_n_times('안녕하세요','반갑습니다','파이썬')
# range(정수) 가 아니라서 오류
# 'str' object cannot be interpreted as an integer

# 올바른 코드(가변 매개변수가 기본 매개변수보다 앞에 위치)
def print_n_times(*values, n=2):
    for i in range(n):
```

```
        for value in values:
            print(value)
        print()

# print_n_times('안녕하세요', '반갑습니다', '파이썬', 3)
print_n_times('안녕하세요', '반갑습니다', '파이썬')
print_n_times('안녕하세요', '반갑습니다', '파이썬', n=3)
'''
안녕하세요
반갑습니다
파이썬

안녕하세요
반갑습니다
파이썬

안녕하세요
반갑습니다
파이썬

안녕하세요
반갑습니다
파이썬

안녕하세요
반갑습니다
파이썬
'''

# 키워드 매개변수 - 매개변수 이름을 지정해서 입력하는 매개변수
def test(a, b=10, c=50):
    print(a+b+c)

test(10,20,30)
test(a=10,b=100,c=300)
test(c=10,a=100,b=200)
test(10,c=200)
'''
60      <-- 기본 형태(매개변수)
410     <-- 키워드 매개변수로 모든 매개변수를 지정
310     <-- 키워드 매개변수로 매개변수를 지정, 순서를 변경
220     <-- 키워드 매개변수로 일부 매개변수만 지정
'''

# 익명함수 또는 람다 함수 - 주로 데이터 분석에 사용, 메모리 절약 목적
# 함수객체를 변수에 담은 시점에서, 함수객체는 메모리에 올라가서 변수를 통해 자신이 호출되기를 기다리지만,
# 만약 단한번만 사용될 함수라면 불필요한 메모리가 낭비됨.
# 값을 반환하는 단순한, 한문장으로 이루어진 함수를 의미함

# 팩토리얼(factorial) : 자연수의 계승 또는 그 수보다 작거나 같은 모든 양의 정수의 곱
# n이 하나의 자연수일 때, 1에서 n까지의 모든 자연수의 곱을 n에 상대하여 이르는 말
# n! = n x (n-1) x (n-2) x ... x 1    ex> 5! = 5 x 4 x 3 x 2 x 1 = 120
# 재귀 - 원래 자리로 되돌아가거나 되돌아 옴, 재귀 종료 조건이 존재함.
# 재귀 함수는 잘못 사용하면 메모리를 과다하게 사용함.

# 1. for반복문
def for_factorial(x):
    result = 1
    for i in range(1,x+1):
        result *= i
    return result
```

```
print('5! : ', for_factorial(5))

# 2. 재귀함수
def recursive_func(x):
    if x > 1:
        return x * recursive_func(x-1)
    else:
        return 1
print('5! : ', recursive_func(5))

# 3. math module
import math
print('5! : ', math.factorial(5))


# 함수를 매개변수로 전달 - map(), filter()
# map(함수, 리스트) -  리스트의 요소를 함수에 넣고 리턴된 값으로 새로운 리스트를 구성
# filter(함수, 리스트) - 리스트의 요소를 함수에 넣고 리턴된 값이 True인 것으로 새로운 리스트를 구성
# iterator - 여러개의 데이터를 일렬로 관리하는 데이터 타입 (list, tuple, strings)
# map object, filter object <---> 제너레이터(generator)

def make_double(num):
    return num * 2
list_a = [1, 2, 3, 4, 5]
# 반환은 리스트가 아님!, 필요하면 list() 함수를 사용해서 형 변환
result = map(make_double, list_a)
print(list(result))
# list(), tuple() - 캐스팅(casting) / 형변환
# 한번 더 출력해보세요!
print(list(result))

def num_even(num):
    if num % 2 == 0:
        return True
tuple_a = (1, 2, 3, 4, 5, 6, 7, 8, 9)
final = filter(num_even, tuple_a)
print(tuple(final))
print(tuple(final))

# 일반함수
def power(item):
    return item * item
def under_3(item):
    return item < 3
list1 = [1, 2, 3, 4, 5]
out1 = map(power, list1)
print('map 함수 실행결과')
print(out1)
out2 = filter(under_3, list1)
print('filter 함수의 실행 결과')
print(out2)

# 위 코드를 다시 람다 함수로 표현
# lambda 매개변수 : 리턴값

power = lambda x: x * x
under_3 = lambda x: x< 3

list_input = [1, 2, 3, 4, 5]

output_a = map(power, list_input)
```

```
# output_a = map(lambda x : x*2, list_input)
print('map 함수의 실행 결과')
print('map(power, list_input) : ', output_a)
print('map(power, list_input) : ', list(output_a)
print()

output_b = filter(under_3, list_input)
# output_b = filter(lambda y : y<3, list_input)
print('filter함수의 실행 결과')
print('filter(power, list_input) : ', output_b)
print('filter(power, list_input) : ', list(output_b)
print()
```

2. 함수의 반환값

```
# 일반적으로 함수의 반환값은 return
def add(a, b):
    return a + b

result = add(5, 10)
print('두 수의 합 : ', result)

# 그렇다면, 둘 이상의 반환값은?
def add_sub(a, b):
    return a+b, a-b

# 두개의 결과값을 반환받는 변수 result에 튜플이 반영됨
result = add_sub(5, 10)
print(result, type(result))

#언패킹
add_result, sub_result = add_sub(5, 10)
print(add_result, sub_result)

# 패킹(packing) - 인자로 여러개의 값을 하나의 객체로 합쳐서 받는 것, 매개변수(parameter)에 사용
print('가나다','abc','123)

def func(*args):
    print(args)
    print(type(args))

func(1,2,3,'a','가')
'''
(1, 2, 3, 'a', '가')
<class 'tuple'>
'''

# 언패킹(unpacking) - 여러개의 객체를 하나의 객체로 풀어주는 것, 인자(argument)에 사용
# 함수의 매개변수와 인자의 갯수가 같아야 함.

# case1
a,b = 1,2

def sum(a, b, c):
    return a+b+c
numbers = [1,2,3]
sum(numbers) # error
print(sum(*numbers))
```

```
'''
6
'''

#case2
sum(*'abc')
sum(*(4,5,6))
sum(*{'가','나','다'})
sum(*{'치킨':3, '피자':12, '음료':10})

'''
abc
15
나다가
치킨피자음료
'''

# divmod() - 몫, 나머지를 반환
a, b = 97, 40
x, y = divmod(a, b)
print(x, y)

# key, value 구조 딕셔너리
dict1 = {
    'name':'hong',
    'age': 30,
    'job': 'student'
}
for key, value in dict1:
    print(key, ':', value)
```

3. 좋은 코드란? (Python Coding Convention - PEP 8)

- 들여쓰기 공백 4칸만
- 한줄에 최대 79자 (모니터를 넘치지 않게)
- 연산자는 1칸 이상 띄우지 않기
- 주석은 항상 갱신, 불필요한 주석은 제거하기
- 소문자 l, 대문자 o, 대문자I 사용하지 않기 (혼선)
- 함수명은 소문자, snake case로 작성하기
- 함수의 이름에 기능, 역할이 명확하게 드러나게 하기
- 함수는 가능한 짧게 작성하기
- flake8 모듈 사용으로 테스트해보기

```
# flake8 모듈 설치
# 터미널 실행
pip install flake8
```

```
# flake8 파일이름.py
flake8 iterator.py

# 체크 결과
iterator.py:58:5: E265 block comment should start with '# '
iterator.py:62:1: E305 expected 2 blank lines after class or function definition, found 0
iterator.py:67:1: E302 expected 2 blank lines, found 0
iterator.py:71:1: E305 expected 2 blank lines after class or function definition, found 1
iterator.py:77:1: E302 expected 2 blank lines, found 1
iterator.py:82:1: E305 expected 2 blank lines after class or function definition, found 1
iterator.py:82:24: E231 missing whitespace after ','
iterator.py:84:14: E231 missing whitespace after ','
iterator.py:88:1: E302 expected 2 blank lines, found 1
iterator.py:93:1: E305 expected 2 blank lines after class or function definition, found 1
iterator.py:93:24: E231 missing whitespace after ','
iterator.py:100:1: E402 module level import not at top of file
iterator.py:102:12: E203 whitespace before ':'
iterator.py:103:19: W292 no newline at end of file
```