교재 113~124

1. 함수의 기본

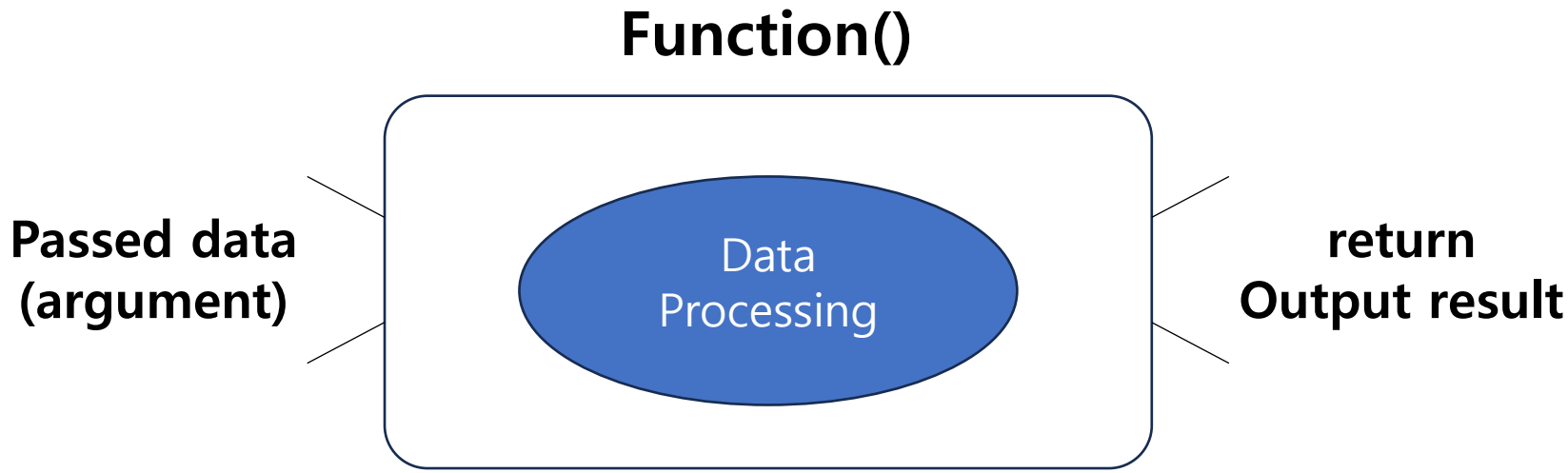
- 함수란
- 함수의 장점과 단점
- 한수의 사용법
- 매개 변수와 인수
- 함수 결과 반환_return
- 네임스페이스와 유효범위(Scope)

부성순 bakpak@empas.com

함수란

함수(Function)

- 특정한 작업을 수행하는 코드의 집합
- 데이터(인수)를 전달받아 처리하고, 그 결과를 반환 함



함수의 장점과 단점

➢ 함수의 장점

- 코드의 중복을 최소화할 수 있다.
- 코드를 재사용할 수 있다.
- 전체 프로그램을 모듈로 나눌 수 있어 개발 과정이 체계적이며 유지보수가 쉬워진다.

  * 모듈(module): 프로그램을 구성하는 구성 요소, 관련된 데이터와 함수를 하나로 묶은 단위를 의미함

➢ 함수의 단점

- 함수 호출할 때 마다 운영체제(OS)가 함수에 사용되는 지역 변수들의 준비와 인수(argument)전달 등을 처리해야 하므로 부담이 발생한다.
- 따라서 너무 작은 단위의 함수를 구성하여 자주 호출하는 경우 성능에 문제가 발생할 수 있다.

함수의 사용법

- 함수의 생성은 **def**라는 키워드를 입력하고 함수의 이름을 작성한 후 **():**으로 선언
- 함수의 시작과 끝을 들여쓰기만으로 구분하며 시작과 끝을 명시해 주지 않음

함수 선언

```
def 함수명( ):
    코드
```

예: 함수 선언

```
def hello( ):
    print('Hello world!' )
```

함수 호출

```
함수명( )
```

예: 함수 호출

```
hello( )
```

예: 실행 결과 'Hello world!'

매개 변수와 인수(1)

- 매개 변수(parameter) : 함수에 입력으로 전달된 값을 받는 변수
- 인수(argument): 함수를 호출할 때 전달하는 입력 값을 의미

함수 선언

```
def add(x, y):
    sum=x + y

    return sum
```

매개변수: x, y

함수 호출

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

인수: 3, 4

주의점

- 매개변수와 인수가 여러 개면 쉼표로 구분
- 매개변수와 인수의 개수는 같아야 함(다르면 에러 발생)

매개 변수와 인수(2)

- 함수의 호출과 인수 전달

함수 선언

```
def add(x, y):
    print(x + y)
```

함수 호출

```
add(1, 2)
add(1.0, 2.0)
add('abc ' , 'def')
add([1,2,3], [4,5,6])
add((1,2,3), (4,5,6))
```

실행 결과

```
3
3.0
abcdef
[1, 2, 3, 4, 5, 6]
(1, 2, 3, 4, 5, 6)
```

함수 결과 반환_return(1)

➢ return 없는 함수

```
def hi():
    print('안녕')
```

```
hi()
```

안녕

```
x = hi()
print(x)
```

안녕
None

X = hi() → hi함수가 실행되어 ' 안녕'이 출력되지만, return이 없기 때문에 x에는 None 반환됨

➢ return 반환 값

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

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

7

X = add(3,4) → add함수가 실행되어 3과 4가 더해진 값 7이 X에 반환 됨

부성순 bakpak@empas.com

함수 결과 반환_return(2)

➢ **return 반환값1, 반환값2....**

```
def add_sub(a, b):
    return a+b, a-b
```

```
x= add_sub(5,3)
print(x)
```

```
(8, 2)
```

함수의 결과값이 튜플로 반환

```
x, y = add_sub(5,3)
print('x=', x, 'y=',y)
```

```
x= 8 y= 2
```

x, y 각 변수에 8, 2가 각각 반환 (unpacking 됨)

➢ **Return 반환값 생략**

return: 값 반환 or 함수 실행 종료

```
def chek_negative(num):
    if num < 0:
        return
    return num
```

```
chek_negative(3)
```

```
3
```

```
chek_negative(-3)
```

chek_negative(3)은 3을 반환하지만,
chek_negative(-3)은 num이 0보다 작으므로 return이 실행되어 아무것도 반환하지 않고 함수를 빠져나옴

부성순 bakpak@empas.com

2. 매개 변수와 인수

- 매개변수 정의와 인수 전달
- 위치 인수
- 기본값 위치 인수
- 가변적 위치 인수
- 키워드 인수
- 가변적 키워드 인수
- 주의: 매개변수 혼합 - 초기값 매개변수의 위치
- 주의: 매개변수 혼합 - 위치 인수와 가변적 위치 인수 혼합
- 주의: 매개변수 혼합 – 위치 인수와 가변적 키워드 인수 혼합
- 주의: 매개변수 혼합 – 가변적 위치인수와 가변적 키워드 인수의 혼합

부성순 bakpak@empas.com

매개변수(parameter)와 인수 (1)

➢ **매개변수 형식 정의와 인수 전달**

구분	형식	설명
함수 선언 (매개변수형식 정의)	def func(arg)	고정 매개변수(위치매개변수) 선언
	def func(arg=value)	고정 매개변수(위치매개변수)에 기본값 설정
	def func(*varargs)	가변적 위치 매개변수 선언 (가변적 매개변수: 매개변수 개수가 정해지지 않음)
	def func(**kwargs)	가변적 키워드 매개변수 선언
	def func(arg,*other)	고정 매개변수와 가변적 위치 매개변수 사용 선언 (arg 먼저 지정하고 *other이 선언, 순서 중요)
	def func(*arg,**kwargs)	가변적 위치 매개변수와 가변적 키워드 매개변수 사용 (*arg가 먼저 오고 **kwargs가 나중에 옴, 순서 중요)
	def func(*, name=value)	* 표시 이후에는 카워드 매개변수만 사용하도록 선언하며, 기본값 설정
함수 호출 (인수 정의)	func(arg1, arg2...)	고정 위치 인수 설정, 위치 인수로 전달
	func(arg=value)	키워드 인수, 이름으로 매개변수에 연계
	func(*iterable)	iterable(예: 문자,리스트, 튜플 등) 객체로 인수를 전달, interable 객체의 각 개체들이 위치인수로 연계됨
	func(**dict)	키워드:값 으로 표현된 항목들을 dict 자료형 인수로 전달되면, 인수에 포함된 키워드를 사용하여 키워드 매개변수에 전달 됨

매개변수(parameter)와 인수 (2) :위치 인수

➢ **위치 인수(Positional Argument): 함수에 인수를 순서대로 넣는 방식**

```
def pos_args(x,y,z):
    return 3*x + 2*y +z
```

```
w=pos_args(1,2,3)
print(w)
```

```
10
```

pos_args(1,2,3) 함수 호출로 인해 위치 순서대로 x=1, y=2, z=3이 각각 지정 되고

w변수에는 3*1 + 2*2 +3 의 수식 결과 10이 반환됨

매개변수(parameter)와 인수 (3) : 기본값 위치 인수

➢ **기본값(default value)을 지정하는 위치 인수**

- 기본값을 가지며, 함수 호출시 생략 가능함

```
def volume(width=1, length=1, height=1):
    return width * length * height
```

```
print(volume())
print(volume(10))
print(volume(10,20))
print(volume(10,20,30))
```

인수로 값을 전달하지 않았으므로 매개변수는 모두 기본값(width=1, length=1, height=1)

width에만 10을 전달, 나머지는 기본값(width=10, length=1, height=1)

width에는 10, length에는 20 전달, height는 기본값 (width=10, length=20, height=1)

width에는 10, length에는 20, height=30 전달(width=10, length=20, height=30)

```
1
10
200
6000
```

매개변수(parameter)와 인수 (4) : 가변적 위치 인수

➢ **가변적 위치 인수(variable positional argument)**

- 인수의 개수가 정해지지 않은 가변적 인수일 때, 매개변수 이름 앞에 *를 표시
- 함수 내부에서 튜플 형태로 처리됨

```
def var_postargs(*args):
    print(type(args),args)
```

```
var_postargs(1)
var_postargs(1,2)
var_postargs(1,2,3)
```

```
<class 'tuple'> (1,)
<class 'tuple'> (1, 2)
<class 'tuple'> (1, 2, 3)
```

매개변수(parameter)와 인수 (5) : 키워드 인수

➢ 키워드 인수(Keyword argument)

- 함수 호출에서 전달되는 인수의 이름(키워드)을 명시
- 매개변수의 순서를 맞추지 않아도 키워드에 해당값을 전달 가능

```
def BMI(weight, height):
    print('몸무게:', weight)
    print('키:', height)
    print('BMI:',weight /((height/100)**2))
```

✓ 위치인수: weight, height 값을 순서대로 입력

```
BMI(50, 160)
```

```
몸무게: 50
키: 160
BMI: 19.531249999999996
```

✓ 키워드 인수: weight, height 값을 순서 상관 없음

```
BMI(weight=50, height=160)
```

```
몸무게: 50
키: 160
BMI: 19.531249999999996
```

```
BMI(height=160,weight=50)
```

```
몸무게: 50
키: 160
BMI: 19.531249999999996
```

매개변수(parameter)와 인수 (6) : 키워드 인수

➢ **키워드 인수: 딕셔너리 자료형으로 키워드 인수를 전달**

- 딕셔너리 앞에 **를 붙여서 매개변수에 전달 (딕셔너리 언패킹 사용)

```
def dict_args(x,y,z):
    return 3*x + 2*y +z
```

```
w=dict_args(**{'x':1, 'y':2, 'z':3})
print(w)
```

```
10
```

```
data={
    'x':1,
    'y':2,
    'z':3
}
w=dict_args(**data)
print(w)
```

```
10
```

매개변수(parameter)와 인수 (7) : 키워드 인수

➢ **키워드 인수: * 표시 다음에 키워드 인수만을 사용하는 함수**

- 함수 매개변수 형식 정의에서 *표시가 있으면, 그 다음은 키워드 인수만 사용해야 함

```
def keyword_only_args(x,*, y,z):
    return 3*x + 2*y +z
```

*표시 뒤의 y, z는 키워드 인수를 사용해야 함

✓ **y, z를 키워드 인수로 전달**

```
w= keyword_only_args(1,y=2, z=3)
print(w)
```

```
10
```

```
w= keyword_only_args(1,**{'y':2, 'z':3})
print(w)
```

```
10
```

✓ **y, z를 위치인수로 전달하면 오류가 남**

```
w= keyword_only_args(1,2,3)
print(w)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[108], line 1
----> 1 w= keyword_only_args(1,2,3)
      2 print(w)

TypeError: keyword_only_args() takes 1 positional argument but 3 were given
```

매개변수(parameter)와 인수 (8) : 가변적 키워드 인수

➢ **가변적 키워드 인수:**

- 인수의 개수가 정해지지 않은 가변적 키워드 인수일 때, 매개변수 이름 앞에 **를 표시
- 함수 내에서는 딕셔너리 자료형으로 처리됨

```
def varkw_args(**kwargs):
    print(type(kwargs))
    print(kwargs)
```

**kwargs: 가변적 키워드 인수 지정

```
varkw_args(name='홍길동')
```

```
<class 'dict'>
{'name': '홍길동'}
```

```
varkw_args(name='홍길동', age=20,  add='서울')
```

```
<class 'dict'>
{'name': '홍길동', 'age': 20, 'add': '서울'}
```

```
data={
    'name':'홍길동',
    'age':20,
    'add':'서울'
}
varkw_args(**data)
```

**data: 딕셔너리 자료형으로 인수 전달

```
<class 'dict'>
{'name': '홍길동', 'age': 20, 'add': '서울'}
```

주의: 매개변수 혼합 - 초기값 매개변수의 위치

➢ 초기값 매개변수의 위치

- 초기값이 지정된 매개변수 다음에는 초기값이 없는 매개 변수가 올 수 없음

```
def p_info(name, add='비공개', age):
    print('이름:',name)
    print('나이:',age)
    print('주소:',add)
```

```
  Cell In[136], line 1
    def p_info(name, add='비공개', age):
                                   ^
SyntaxError: non-default argument follows default argument
```

p_info(name, add='비공개',age) → 초기값이 지정된 매개변수 뒤에 초기값 없는 매개 변수 올 수 없음 (오류)

```
def p_info(name, age, add='비공개'):
    print('이름:',name)
    print('나이:',age)
    print('주소:',add)
```

```
p_info('홍길동',30)
```

```
이름: 홍길동
나이: 30
주소: 비공개
```

p_info(name, age, add='비공개') → name, age 매개변수는 반드시 값을 전달해야하고, add는 값이 제공되지 않으면 기본값 '비공개'가 사용됨

주의: 매개변수 혼합 - 위치 인수와 가변적 위치 인수 혼합

➢ 위치 인수와 가변적 위치 인수 함께 사용

- 위치 인수 개수 만큼 매핑시킨 후, 나머지는 튜플로 전달
- 가변 매개 변수 뒤에는 일반 매개 변수 사용할 수 없다. (*args가 고정매개변수 a보다 먼저 오면 안됨)
- 가변 매개 변수는 하나만 사용할 수 있음

```
def var_postargs(a, *args):
    print('a=',a,'args=',args)
```

```
var_postargs(1)
var_postargs(1,2)
var_postargs(1,2,3,4,5)
```

```
a= 1 args= ()
a= 1 args= (2,)
a= 1 args= (2, 3, 4, 5)
```

주의: 매개변수 혼합 - 위치 인수와 키워드 인수 혼합

➢ 위치 인수와 키워드 인수 함께 사용

- 위치 인수, 키워드 인수 순서대로 배치되어야 함

```
def pos_kw(name, **kwargs):
    print('이름:', name)
    print('kwargs:', kwargs)
```

Nam ㄴe: 위치 인수로, 필수 값 제공
****kwargs: 키워드 인수로 여러 개의 키-쌍으로 값 제공**

```
pos_kw('홍길동')
```

```
이름: 홍길동
kwargs: {}
```

```
pos_kw('홍길동', age=20, addr='서울')
```

```
이름: 홍길동
kwargs: {'age': 20, 'addr': '서울'}
```

```
data={
    'age':20,
    'addr':'서울'
}
pos_kw('홍길동', **data)
```

****data: 딕셔너리 자료로 인수 전달**

```
이름: 홍길동
kwargs: {'age': 20, 'addr': '서울'}
```

주의: 매개변수 혼합 - 가변적 위치인수와 가변적 키워드 인수의 혼합

➢ **가변적 위치 인수(*args)와 가변적 키워드 인수(**kwargs) 함께 사용**

- 가변적 위치 인수(*args) 먼저 지정한 다음에 가변적 키워드 인수(**kwargs)가 지정되어야 함

```
def var_pos_key(*args, **kwargs):
    print('args=', args)
    print('kwargs=', kwargs)
```

```
var_pos_key(1,x=3)
```

```
args= (1,)
kwargs= {'x': 3}
```

```
var_pos_key(1,2,3,x=3,y=4)
```

```
args= (1, 2, 3)
kwargs= {'x': 3, 'y': 4}
```

네임스페이스와 유효범위(Scope) (1)

- **네임스페이스(namespace):**이름(변수, 함수, 클래스 등)이 저장되는 공간을 의미

Name Space	설명
지역(local)	• 함수 또는 클래스 메서드 내에서 정의된 이름. • 함수나 메서드가 호출될 때 생성되고, 호출이 종료되면 소멸. • locals()함수로 확인 가능
전역(global)	• 프로그램의 전체 범위에서 접근할 수 있는 변수 • 함수 외부에서 정의되며, 프로그램의 어느 곳에서나 접근 가능 • globals()함수로 확인 가능
내장(built-in)	• 파이썬 인터프리터에 의해 미리 정의된 변수. • dir(__builtins__)로 확인 가능 • 예: print(), abs(), len(), max() 등의 함수나 True, False, None 등

파이썬에서는 이름을 찾을 때, **지역 네임스페이스(Local) → 전역 네임스페이스(Global) → 내장 네임스페이스(Built-in)** 순서로 탐색함

네임스페이스와 유효범위(Scope) (2)

```
def func1():
    a=10
    print(f'func1()-> a={a}')

def func2():
    print(f'func2()-> a={a}')

a=20
func1()
func2()
```

지역변수

전역변수

func1함수 호출

func2함수 호출

```
func1()-> a=10
func2()-> a=20
```

- func1( ) → func1함수 내 지역변수 a를 찾아 출력함 (a=10)
- func2( ) → func2함수 내 지역변수 a를 먼저 찾는데 없으니 전역변수 a를 찾아 출력함 (a=20)

네임스페이스와 유효범위(Scope) (3)

Name Scole 지정선언	의미
Global	함수 내부에서 전역변수를 수정하고자 할 때, 해당 변수명 앞에 global 키워드를 사용
nonlocal	중첩된 함수 내부에서 부모 함수의 변수를 수정하고자 할 때, 해당 변수명 앞에 nonlocal 키워드를 사용

```
x=1
y=3
z=5
def func():
    x=10
    global y
    y=30
    z=50
    print(f'func() -> x={x},y={y},z={z}')
    def sub_func():
        nonlocal z
        z=100
        print(f'sub_func() -> x={x},y={y},z={z}')
    sub_func()
    print(f'func() -> x={x},y={y},z={z}')
func()
print(f'main -> x={x},y={y},z={z}')
```

x, y, z는전역변수

X 지역 변수

y전역 변수

z 지역 변수

z 부모함수 func()의 지역 변수

```
func() -> x=10,y=30,z=50
sub_func() -> x=10,y=30,z=100
func() -> x=10,y=30,z=100
main -> x=1,y=30,z=5
```

1 2 func( ) 함수 호출

X → 함수 내 x값 할당 (지역변수 **x=10**)
Y → global y 인해 전역변수 y값이 새로 할당 됨 (전역변수 **y=30**)
z → 함수 내 z값 할당 (지역변수 **z=50**)

3 4 sub_func( ) 함수 호출

X → 함수 내 x가 없어 부모함수에서 찾음 (지역변수 **x=10**)
Y → 함수 내 y가 없어 부모함수에서 찾음(전역변수 **y=30**)
Z → nonlocal z 로 인해 부모함수의 z값을 새로 할당 (부모지역변수 **z=100**)

5 X → (func)함수 내 x값 (지역변수 **x=10**)
Y → 함수 내 y 값(전역변수 **y=30**)
Z → sub_func( )실행으로 Z=100으로 수정 (부모지역변수 **z=100**)

6 X → 전역변수 **X=1**
Y → 전역변수 **Y=30** (FUNC()실행으로 Y=30수정 됨
Z → 전역변수 **Z=5**

부성순 bakpak@empas.com