함 수

Update – 2018.05

Contents

- 함수 선언과 호출
- 결과값이 가변인 함수
- 파라미터 초기값 설정
- 가변 인자 함수
- 전역 변수와 지역 변수
- 람다함수

함 수 선언과 호출

- 파이썬 함수의 구조

```
# 함수 정의
def 함수명(입력 인수):
    <수행할 문장1>
    <수행할 문장2>
    ...
```

```
# 함수 호출
함수명(입력 인수)
```

- 일반적인 함수

```
def sum(a, b):
    result = a + b
    return result
```

```
>>> a = sum(3, 4)
>>> print(a)
7
```

함수의 선언과 호출

- 함수의 종류

✓ 입력 값, 결과값이 존재하는 함수

✓ 입력 값도 결과값도 없는 함수

✓ 입력 값이 없는 함수

✓ 결과 값이 없는 함수

	Parameter 없음	Parameter 존재
반환 값 없음	함수 내의 수행문만 수행	인자를 사용, 수행문만 수행
반환 값 존재	인자없이, 수행문 수행 후 결과값 반환	인자를 사용하여 수행문 수행 후 결과값 반환

함수의 선언과 호출

- 입력 값이 없는 함수

```
>>> def say():
...     return 'Hi'
...
>>>
```

```
>>> a = say()
>>> print(a)
Hi
```

함수의 선언과 호출

- 입력값도 결과값도 없는 함수

```
>>> def say():
...     print('Hi')
...
>>>
```

```
>>> say()
Hi
```

함수의 선언과 호출

- 결과값이 없는 함수

```
>>> def sum(a, b):
...     print("%d, %d의 합은 %d입니다." % (a, b, a+b))
...
>>>
```

```
>>> sum(3, 4)
3, 4의 합은 7입니다.
```

결과값이 가변인 함수

- Return para1, para2 …

  : 쉼표(,)를 이용하여 결과값을 여러 개 지정할 수 있다.

  : 여러 개의 결과값은 튜플 형태로 저장된다.

```
>>> def sum_and_mul(a,b):
...     return a+b, a*b
```

```
>>> print(sum_and_mul(3,4))
```

>>> (7, 12)

파라미터 초기값

- 입력 인수에 초깃값 미리 설정하기

```
def say_myself(name, old, man=True):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % old)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
    print ('-'*50)
```

```
say_myself( " 홍길동", 20)
say_myself( " 신데렐라 " , 20, False)
```

```
나의 이름은 홍길동 입니다.
나이는 20살입니다.
남자입니다.
--------------------------------------------------
나의 이름은 신데렐라 입니다.
나이는 20살입니다.
여자입니다.
--------------------------------------------------
```

파라미터 초기값

- 함수 입력 인수에 초깃값을 설정할 때 주의할 사항

def say_myself(names, man=True, old=20):

```
def say_myself(name, man=True, old):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % old)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

결과 값 이상 발생
파라미터 순서 주의

가변 인자 함수

- 가변 입력 값 *args
  - 가변인자(Variable-length)란 개수가 정해지지 않은 변수를 함수의 파라미터로 사용
  - Asterisk(*) 기호를 사용하여 함수의 파라미터를 표시한다.
  - 가변 인자는 tuple 형태로 저장된다.
  - 일반 파라미터와 함께 사용할 경우에는 *args는 마지막에 위치해야 한다.

```
>>> def sum_many(*args):
...     sum = 0
...     for i in args:
...         sum = sum + i
...     return sum
...
>>>
```

가변 인자 함수

```
def cal(choice, *args):
    if choice == "Sum":
        result = 0
        for i in args:
            result += i
    elif choice == "Mul":
        result = 1
        for i in args:
            result *= i
    else:
        result = "오류 발생"
    return result
```

```
cal('Sum', 5,6,7)
>>> 18
cal('Mul', 5,6,7)
>>> 210
cal('Subtract',4,5,6)
>>> '오류발생'
```

가변 인자 함수

- 가변 입력 값 **kwargs
  - 딕셔너리 형태로 파라미터를 지정할 수 있다.

```
def myNum(**kwargs):
    print(kwargs)
    print('First value : {first}'.format(**kwargs))
    print('Second value : {second}'.format(**kwargs))

myNum(first=1, second=2)
```

Key=value 형태로 지정

```
{'first': 1, 'second': 2}
First value : 1
Second value : 2
```

가변 인자 함수

```
def my_num(**kwargs):
    print('num1 : {key1}'.format(**kwargs))
    print('num2 : {key2}'.format(**kwargs))
    print('num3 : {key3}'.format(**kwargs))
```

```
my_num(key1=1, key2=2, key3=3)
>>>  num1 : 1
.....num2 : 2
.....num3 : 3
```

Key=value 형태로 지정

전역 변수와 지역 변수

함수 안에서 선언된 변수의 효력 범위

```
# 전역 변수와 지역 변수
n = 1
def myFun(n):
    n = 10
    return n+1
print(myFun(10))
print(n)
```

전역변수

지역변수

전역 변수와 지역 변수

- 함수 안에서 함수 밖의 변수를 변경하는 방법
  : global 키워드 이용

```
m = 5
def myFun2():
    global m
    m = 10
    print(m)

myFun2()
print(m)
```

```
>>> 10
>>> 10
```

람다 함수

- 런타임에 생성해서 사용할 수 있는 익명 함수
- 쓰고 버리는 일시적인 함수
- 형식:
  lambda 인자리스트: 표현식

```
f = lambda x, y:
x + y
print(f(4, 4))
```

```
f = lambda x:
x**2
print(f(8))
```