2023.6.6

※ 함수

- 믹서와 비슷, 입력값을 가지고 어떤 일을 수행한 다음에 결과물을 내어놓는 것

```
def 함수 이름(매개변수):
    수행할 문장
    ...
    return 결과값
```

```
# a,b는 매개변수, 3,4는 인수
# 매개변수 : 함수에 입력으로 전달된 값을 받는 변수
# 인수 : 함수를 호출할 때 전달하는 입력값
def add(a,b):
    return a + b

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

```
# 입력값이 없는 함수
def say():
    return 'Hi'

a = say()
print(a)


# result
Hi
```

```
# 결과값이 없는 함수
def add(a,b):
    print("%d, %d의 합은 %d입니다." % (a, b, a+b))

add(3,4)


# result
3,4의 합은 7입니다.
```

```
# 입력값도 결과값도 없는 함수
def say():
    print('Hi')
```

```
say()


# result
Hi
```

- 매개변수 지정하여 호출하기

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

# 다음과 같이 매개변수를 지정하여 사용할 수 있음
result = add(a=3, b=7)
print(result)
10
```

- 입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?

```
# 여러 개의 입력값을 받는 함수 만들기
# *args는 임의로 정한 변수의 이름
def add_many(*args):
    result = 0
    for i in args:
         # *args에 입력받은 모든 값을 더한다
         result = result + i
    result result
```

→ 위에서 만든 add_many 함수는 입력값이 몇 개이든 상관이 없다. *args처럼 매개변수 이름 앞에 *을 붙이면 입력값을 전부 모아서 튜플로 만들어주기 때문이다. 만약 add_many(1,2,3)처럼 이 함수를 쓰면 args는 (1,2,3)이 된다.

- 함수의 결과값은 언제나 하나

```
def add_and_mul(a,b):
    return a+b, a*b

result = add_and_mul(3,4)
result = ??
# 결과값을 출력했을 때 결과값이 a+b, a*b 2개이고 result가 하나만 쓰였기에
# 오류가 발생할 것 같지만 a+b,a*b가 튜플 값 하나인(a+b,a*b)로 돌려주므로
# 에러 발생하지 않음

result = (7,12)
```

```
def add_and_mul(a,b):
    return a+b
    return a*b

# return을 2번 적으면 2개의 결과값을 돌려주지 않을까 싶지만
# 첫 번째 return문은 실행되는 한편 두 번째 return문은 실행되지 않음
```

- 매개변수에 초깃값을 미리 설정하기

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

→ 위와 같이 man=True 와 같이 함수의 매개변수에 초깃값을 미리 설정할 수 있다.

```
# 아래 2문장은 같은 값을 출력한다.
say_myself("홍길동", 27)
say_myself("홍길동", 27, True)


# result
나의 이름은 홍길동입니다.
나이는 27살입니다.
남자입니다.
```

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

say_myself("홍길동",27, True)


# result
SyntaxError: non-default argument follows default argument
```

→ 매개변수의 순서를 바꾸면 같은 값을 구할 때 에러 발생

→ 위 오류 메시지는 초깃값을 설정해놓은 매개변수 뒤에 초깃값을 설정해 놓지 않은 매개변수는 사용할 수 없다는 뜻

- 함수 안에서 선언한 변수의 효력 범위

```
a = 1
def vartest(a):
    a = a + 1
vartest(a)
print(a)
```

→ 위 코드에서 print(a)를 하면 2가 출력될 것 같지만 1이 출력된다. 함수 안에서 새로 만든 매개변수는 함수 안에서만 사용하는 '함수만의 변수'이기 때문이다.

→ 그렇다면 함수 안에서 함수 밖의 변수를 변경할 수는 없을까?

```
# 1. return 사용하기
a = 1
def vartest(a):
    a = a + 1
    return a
a = vartest(a)
print(a)

# result
2
```

→ vartest(a) 함수를 거치며 a의 값이 a+1이 된다. 따라서 결과값은 2가 된다.

```
# 2. global 명령어 사용하기
a = 1
def vartest():
    global a
    a = a + 1
vartest()
print(a)

# result
2
```

→ global a 문장은 함수 안에서 함수 밖의 a 변수를 직접 사용하겠다는 뜻이다.

→ 코딩을 할 때 global 명령어는 사용하지 않는 것이 좋다. 함수는 독립적으로 존재하는 것이 좋기 때문이다. 외부 변수에 종속적인 함수는 그다지 좋은 함수가 아니다.

- lambda

→ 함수 생성 예약어로 def와 동일한 역할

→ 함수를 한 줄로 간결하게 만들 때 사용

→ def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰인다.

→ lambda 매개변수1, 매개변수2, …. : 매개변수를 사용한 표현식

```
# 예제 : 다음 두 함수는 하는 일이 완전히 동일하다.

# lambda 예약어 사용
add = lambda a, b: a+b
result = add(3,4)
print(result)

# def 예약어 사용
def add(a+b):
    return a+b
result = add(3,4)
print(result)
```