================20191212

4장. 프로그램 입력과 출력

I. 함수

II. 사용자 입력과 출력

III. 파일 읽고 쓰기

## I. 함수

함수(function)는 입력 값을 가지고 어떤 일을 수행한 다음에 그 결과물을 내어놓은 것

함수를 사용하는 이유는 무엇일까? 

동일한 내용을 반복해서 작성할 때 한 번만 코딩해 두고 반복해서 사용 

프로그램을 함수화하면 프로그램의 흐름을 일목요연하게 볼 수 있음

=> 함수의 구조



def는 함수를 만들 때 사용하는 예약어임 

함수이름은 함수를 만드는 사람이 임의로 만들 수 있음 

매개변수는 이 함수에 입력으로 전달되는 값을 받는 변수임 

함수를 정의한 다음, 함수에서 수행할 문장들을 입력함 

return은 함수의 결과값을 돌려주는 명령어임

def add(a,b):

return a + b

함수이름은 add이고, 입력으로 2개의 값을 받으며, 결과값은 2개의 입력 값을 더한 값임

>>> def add(a,b):

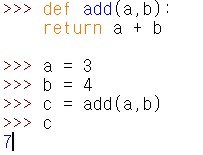
return a + b

>>> a = 3

>>> b = 4

>>> c = add(a,b)

>>> c



=>매개변수와 인수

매개변수 : 함수에 입력으로 전달된 값을 받는 변수

인수 : 함수를 호출할 때 전달하는 입력 값

def add(a,b): <------- a,b는 매개변수

return a+b

print(add(3,4)) <------- 3,4는 인수

=>입력값과 결과값에 따른 함수의 형태

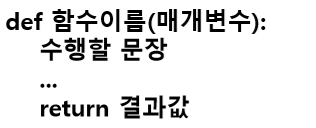
함수는 들어온 입력값을 받아 어떤 처리를 하여 적절한 결과값을 돌려줌



함수의 형태는 입력값과 결과값의 존재 유무에 따라 4가지 유형으로 나뉨

1. 일반적인 함수

입력값이 있고 결과값이 있는 함수가 일반적인 함수임

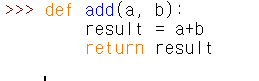


일반적인 함수의 예:

>>> def add(a, b):

result = a+b

return result



>>> a = add(3, 4)

>>> print(a)



입력값과 결과값이 있는 일반적인 함수의 사용법

**결과값을 받을 변수 = 함수이름(입력인수 1, 입력인수 2, ···)**

2. 입력값이 없는 함수

매개변수 부분을 나타내는 함수이름 뒤의 괄호 안이 비어 있는 함수의 예

입력값이 없는 함수의 사용 예:

>>> def say():

return 'Hi'



>>> a = say()

>>> print(a)



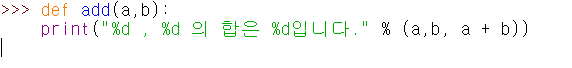
입력값이 없고 결과값만 있는 함수의 사용법

**결과값을 받을 변수 = 함수명()**

3. 결과값이 없는 함수

>>> def add(a,b):

print("%d , %d 의 합은 %d입니다." % (a,b, a + b))



>>> add(3,4)



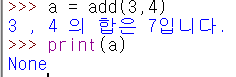
결과값이 없는 함수의 사용법

**함수이름(입력인수1, 입력인수2, ···)**

결과값이 없는지 확인하기 위한 예:

>>> a = add(3,4)

>>> print(a)



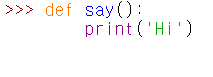
add 함수처럼 결과값이 없을 때, 리턴값으로 a 변수에 None을 돌려 줌

4. 입력값도 결과값도 없는 함수

입력값과 결과값이 없는 함수의 예:

>>> def say():

print('Hi')



>>> say()



입력값도 결과값도 없는 함수의 사용법

**함수명()**

=>매개변수 지정하여 호출하기

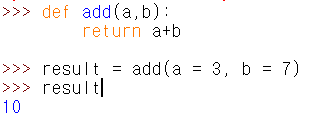
함수를 호출할 때 매개변수를 지정할 수 있음

>>> def add(a,b):

return a+b

>>> result = add(a = 3, b = 7)

>>> result



매개변수를 지정하면 순서에 상관없이 사용할 수 있다는 장점이 있음

>>> result = add(b = 7, a = 3)

>>> result



def는 변수명으로 사용불가

예약어는 변수로 사용 불가한다.

결과값은 무조건 하나만 리턴한다.

함수에 전달하는 값을 인수라고 한다.

함수 쪽에 있는 것은 매개변수 변수에 저장이 되서 함수 내에서 사용한다.

함수를 호출 할 때 값을 전달 하는 것을 인수라고 한다.

=> 입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?



1. 여러 개의 입력값을 받는 함수 만들기

\*args입력값 여러개 튜를로

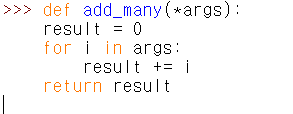
>>> def add\_many(\*args):

result = 0

for i in args:

result += i

return result



\*args처럼 매개변수 이름 앞에 \*을 붙이면 입력값을 튜플로 만들어 주기 때문에 add\_many 함수는 입력값이 몇 개이든 상관이 없음

\*args는 임의로 정한 변수명임

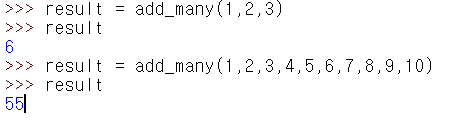
여러 개의 입력값을 받는 함수 사용 예:

>>> result = add\_many(1,2,3)

>>> result

>>> result = add\_many(1,2,3,4,5,6,7,8,9,10) )#튜플이라는 자료형

>>> result



2. 다양한 종류의 매개변수를 받는 함수 만들기

>>> def add\_mul(choice, \*args):

if choice == "add":

result = 0

for i in args:

result += i

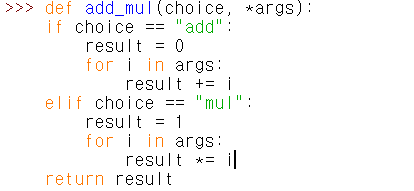
elif choice == "mul":

result = 1

for i in args:

result \*= i

return result



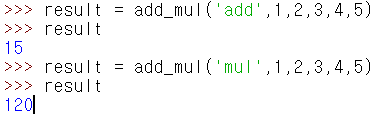
다양한 종류의 매개변수를 받는 함수 사용 예

>>> result = add\_mul('add',1,2,3,4,5)

>>> result

>>> result = add\_mul('mul',1,2,3,4,5)

>>> result



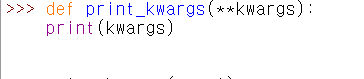
=>키워드 파라미터

키워드 파라미터를 사용할 때는 매개변수 앞에 별 두 개(\*\*)를 붙임

dic \*\*kwargs

>>> def print\_kwargs(\*\*kwargs):

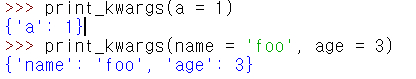
print(kwargs)



키워드 파라미터 사용 예:

>>> print\_kwargs(a = 1)

>>> print\_kwargs(name = 'foo', age = 3)



입력값이 모두 딕셔너리로 만들어져서 출력된다는 것을 확인할 수 있음 

매개변수 이름 앞에 \*\*을 붙이면 매개변수는 딕셔너리가 되고, 모든 key=value 형태의 결과값이 그 딕셔너리에 저장됨

=>함수의 결과값은 언제나 하나이다

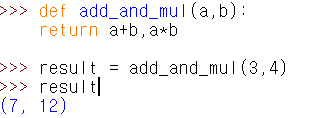
2개의 매개변수를 받아 더한 값과 곱한 값을 돌려준다

>>> def add\_and\_mul(a,b):

return a+b,a\*b

>>> result = add\_and\_mul(3,4)

>>> result



결과값은 a+b와 a\*b 2개인데 받아들이는 변수는 result 하나만 쓰였으니 오류가 발생하지 않을까? 의문을 가지나 오류는 발생하지 않음

그 이유는 함수의 결과값은 2개가 아니라 언제나 1개라는 데 있음

add\_and\_mul 함수의 결과값 a+b와 a\*b는 튜플값 하나인 (a+b, a\*b)로 돌려줌

따라서 result 변수는 다음과 같은 값을 갖게 됨

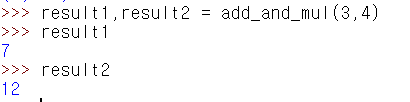
결과값으로 (7, 12)라는 튜플 값을 갖게 되는 것임

하나의 튜플 값을 2개의 결과값처럼 받고 싶다면 다음과 같이 함수를 호출하면 됨

>>> result1,result2 = add\_and\_mul(3,4)

>>> result1

>>> result2



아래와 같이 return문을 2번 사용하면, 2개의 결과값을 돌려주지 않을까?

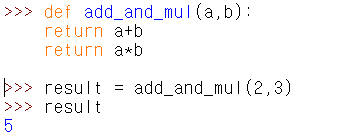
>>> def add\_and\_mul(a,b):

return a+b

return a\*b

>>> result = add\_and\_mul(2,3)

>>> result



두 번째 return문인 return a\*b는 실행되지 않음

함수는 return문을 만나는 순간 결과값을 돌려준 다음 함수를 빠져나가게 됨

=> return의 또 다른 쓰임새

특별한 상황일 때 함수를 빠져나가고자 싶다면, return을 단독으로 써서 함수를 즉시 빠져 나갈 수 있음

>>> def say\_nick(nick):

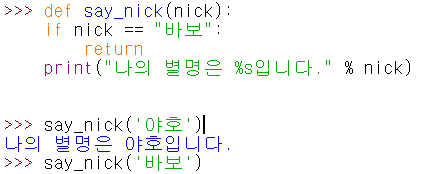
if nick == "바보":

return

print("나의 별명은 %s입니다." % nick)

>>> say\_nick('야호')

>>> say\_nick('바보')



return으로 함수를 빠져나가는 방법은 실제 프로그래밍에서 자주 사용됨

=>매개변수에 초기값 미리 설정하기

>>> def say\_myself(name,old, man = True):

print("나의 이름은 %s 입니다." % name)

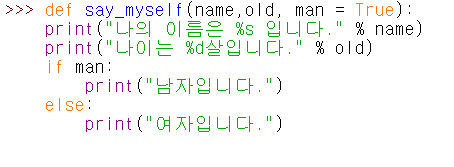
print("나이는 %d살입니다." % old)

if man:

print("남자입니다.")

else:

print("여자입니다.")



위 함수는 매개변수가 name, old, man=True 이렇게 3개임

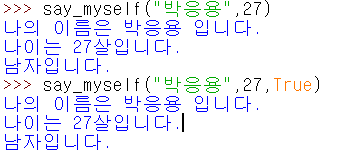
man=True처럼 매개변수에 미리 값을 넣어주는 것이 함수의 매개변수 초기값을 설정하는 방법임

함수의 매개변수에 들어갈 값이 항상 변하는 것이 아닐 경우에는 함수의 초기값 을 미리 설정해 두면 유용함

매개변수에 초기값 설정하는 사용 예:

>>> say\_myself("박응용",27)

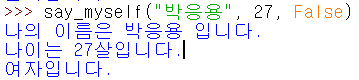
>>> say\_myself("박응용",27,True)



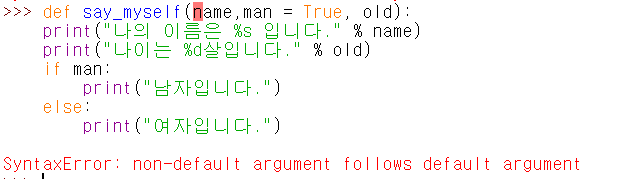
man이라는 변수에는 입력값을 주지 않았지만 초기값인 True를 갖게 됨

위의 예에서 함수를 사용한 2가지 방법은 모두 동일한 결과를 출력함

>>> say\_myself("박응용", 27, False)



==> 함수의 매개변수에 초기값 설정할 때 주의사항



오류 메시지는 초기값을 설정해 놓은 매개변수 뒤에 초기값을 설정해 놓지 않은 매개변수는 사용할 수 없다는 뜻임

초기화시키고 싶은 매개변수는 항상 뒤쪽에 놓아야 함

=> 함수 안에서 선언한 변수의 효력 범위

함수 안에서 사용할 변수의 이름을 함수 밖에서도 동일하게 사용한다면 어떻게 될까?

함수내에 사용하는 것은 지역변수이다.

a = 1 #함수 밖의 변수 a

def vartest(a): #vartest 함수 선언

a += 1

vartest(a) #vartest 함수의 입력값으로 a를 줌

print(a) # a의 값 출력 1



아래의 것은 오류입니다.

#vartest\_error.py

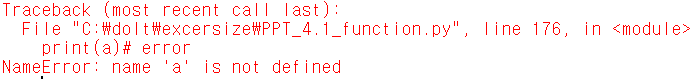
del a

def vartest(a):

a += 1

vartest(3)

print(a)# error



함수 안에서 선언된 변수는 함수 안에서만 사용될 뿐 함수 밖에서는 사용되지 않음

=>함수 안에서 함수 밖의 변수를 변경하는 방법

#vartest\_return.py

a = 1

def vartest(a):

a += 1

return a

a = vartest(a)

print(a)#2



#vartest\_global.py

#global 명령어 이용하기

a = 1

def vartest():

global a

a += 1

vartest()

print(a)#2



=>lambda

lambda(람다)는 함수를 생성할 때 사용하는 예약어로 def와 동일한 역할을 함 

함수를 한 줄로 간결하게 만들 때 사용함 

def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰임

lambda 매개변수1, 매개변수2, … : 매개변수를 사용한 표현식

#lambda

add = lambda a,b:a+b

result = add(3,4)

print(result)

mul = lambda a,b:a\*b

result = mul(3,4)

print(result)

sub = lambda a,b:a-b

result = sub(3,4)

print(result)

div = lambda a,b:a/b

result = div(3,4)

print(result)

return문이 없다. 자동으로 받는다.

lambda는 함수를 생성할 때 사용하는 예약어로, def와 동일한 역할을 합니다.

이는 def를 사용해야 할 정도로 복잡하지 않거나 def를

사용할 수 없는 곳(런타임 또는 익명으로 생성해야 하는 경우)에 주로 쓰입니다.

#주어진 자연수가 홀수인지 짝수인지 판별해 주는 함수

def is\_odd(number):

if number%2 != 0:#홀수

return True

else:

return False

print(is\_odd(1))

print(is\_odd(2))

is\_odd = lambda number:'홀수' if number % 2 != 0 else '짝수'

print(is\_odd(1))

print(is\_odd(2))

#입력으로 들어오는 모든 수의 평균값을 계산해 주는 함수를 작성해보자.

def avg\_numbers(\*args):

result = 0

for i in args:

result += i

return result/len(args)

print(avg\_numbers(1,2))

print(avg\_numbers(1,2,3,4,5))

print(avg\_numbers(1,2))

print(avg\_numbers(1,2,3,4,5))

#다음은 두 개의 숫자를 입력받아 더하여 돌려주는 프로그램이다.

input1 = int(input("첫번째 숫자를 입력하세요."))

input2 = int(input("두번째 숫자를 입력하세요."))

total = input1+input2

print("두개의 합은 %s 입니다." % total)

#4.

print("you" "need" "python")

print("you"+"need"+"python")

print("you","need","python")#결과가 다르다.

print("".join(["you","need","python"]))

================20191213

#파일 생성하기

f = open("새파일.txt", 'w')

f.close()

#r 읽기 모드 - 파일을 읽기만 할 때 사용

#w 쓰기 모드 - 파일에 내용을 쓸 때 사용

#a 추가 모드 - 파일의 마지막에 새로운 내용을 추가할 때 사용

f = open("C:\doit\새파일.txt", 'w')

f.close()

f = open("새파일.txt",'w')

for i in range(1,11):

data = "%d번째 줄입니다.\n" % i

f.write(data)

f.close()

f = open("새파일.txt","r")

line = f.readline()

print(line)

f.close()

f = open("새파일.txt","r")

while True:

line = f.readline()

if not line:break

print(line)

f.close()

#사용자 입력을 받아서 출력하는 경우와 파일을 읽어서 출력하는 예제를 비교

f = open("새파일1.txt","w")

while 1:

data = input()

if not data:break

print(data)

f.close()

#read.py

# readlines() 함수 이용하기

f = open("새파일.txt",'r')

lines = f.readlines()

for line in lines:

print(line)

f.close()

#파일 읽고 쓰기

f= open("새파일.txt",'r')

data = f.read()

print(data)

f.close()

#adddata

f= open("새파일.txt",'a')

for i in range(11,20):

data = "%d번때 줄입니다.\n" % i

f.write(data)

f.close()

f= open("foo.txt",'w')

f.write("Life is too short,you need pyhon"\_

f.close()

with open("foo.txt",'w') as f:

f.write("Life is too sort, you need python")

sys 모듈로 매개변수 주기

명령 프롬프트 명령어 [인수1 인수2 …]

#sys1.py

import sys

args = sys.argv[1:]

for i in args:

print(i)

C:\doit>python sys1.py aaa bbb ccc aaa bbb ccc

#7

#파일명 하드코딩하는 방법

f= open('test.txt','r')

body = f.read()

print(body)

f.close()

body = body.replace("java","python")

f = open('test1.txt','w')

f.write(body)

f.close()

#6

user\_input = input("저장할 내용을 입력하세요")

f= open('test.text','a')

f.write(usesr\_input)

f.write("\n")

f.close()

#readline02

#파일명 입력받는 방법#readline02.py

fname = input("파일명을 입력하세요.")

f= open("c:\doIt\%s" % fname,'r')

body = f.read()

print(body)

f.close()

#파일명 매개변수로 입력받는 방법 readline03.py

#sys 명령프롬프트

import sys

def fileOpen(i):

f= open("c:\doIt\%s" % i,'r')

body = f.read()

print(body)

f.close()

args = sys.argv[1:]

for i in args:

f= open("c:\doIt\%s" % i,'r')

body = f.read()

print(body)

f.close()

import re

operList = ('+','-','\*','/')

def cal(oper, var1, var2):

#for oper1 in operList:

# if oper == operList[0]:

# return var1 + var2

# elif oper == operList[1]:

# return var1 - var2

# elif oper == operList[2]:

# return var1 \* var2

# elif oper == operList[3]:

if oper == "+":

return var1 + var2

elif oper == "-":

return var1 - var2

elif oper == "\*":

return var1 \* var2

elif oper == "/":

return var1 / var2

#한줄로 리턴

return eval(str(var1)+oper+str(var2))

#에러메시지

def errMes(var):

print('숫자만 입력 가능합니다.'.format(var))

#계산 구분 입력 안할 경우 종료

#% 시 0으로 할경우 종료

while True:

#1.계산 구분 판단

oper2= ""

while True:

oper= input("계산 구분을 입력하세요")

if oper not in operList:

print("정확한 계산 구분을 입력하여주세요.")

continue

else:

break

#숫자입력

var1 = 0

while True:

try:

var1= int(input("첫 번째 수를 입력하세요. : "))

except ValueError : # 에러 종류

errMes(var1)

continue

break

var2 = 0

while True:

try:

var2= int(input("두 번째 수를 입력하세요. : "))

if oper=="/" and var2== 0:

print("두번째 수를 확인하여 주십시요. : ")

continue

else:

break

except ValueError : # 에러 종류

errMes(var2)

continue

break

result= cal(oper, var1, var2);

print("계산기 : %d %s %d = %d" % (var1,oper,var2,result))

result = input("enter 눌릴 겨우 종료됩니다.")

if result == '':

print("계산기 종료합니다.")

break

else:

print("계산기 진행됩니다.")

a.isnumeric()