

함수와 모듈





함수



함수

- 사용 이유

- 코드 재사용

- 중복되는 부분을 한 번만 작성. 코드를 간결하게 만들어 줌

- 절차적 분해

- 한 번에 큰 처리 절차를 구현하는 것보다 작은 작업을 구현하는 것이 더 쉬움

함수

■ 예)

```
a = [5,5,6,7,8,3]
b = 'Life is short.'
# a, b의 길이를 구하고 싶으면?

print(len(a), len(b))
```

■ len() 이 없다면?

```
a = [5,5,6,7,8,3]
b = 'Life is short.'
# a, b의 길이를 구하고 싶으면?

length_a = 0
'''
print(length_a)

length_b = 0
'''
print(length_b)
```

함수

■ 일반적 구조

```
def 함수이름(매개변수) :  
    수행할 문장  
    ...  
    return 반환값
```

■ 함수 정의 및 호출

#함수 정의

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

#함수 호출

```
add_result = add(100, 200)  
print(add_result)
```

실습문제 0

■ len() 구현

```
def my_len(l):  
    ...  
    return length
```

```
a = [5,5,6,7,8,3]  
b = 'Life is short.'  
print(len(a), len(b))      # 내장 함수 len()  
print(my_len(a), my_len(b)) # my_len()
```

함수 - 반환 값 (Return Value)

■ 반환 값이 있는 경우

```
def say():  
    return "hello"
```

```
a = say() #a의 값: "hello"
```

■ 반환 값이 **없**는 경우

```
def say():  
    print("hello")
```

```
a = say() #a의 값: None
```

None을 반환

함수 - 매개변수 (Parameter)

■ 매개변수가 있는 경우

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

```
res1 = add(1, 2)  
res2 = add(2, 3)
```

■ 매개변수가 없는 경우

```
def add():  
    return 1 + 2
```

```
res3 = add()
```


실습문제 1

- add, sub, mul, div 함수 정의해 보기
 - add: 더하기, sub: 빼기, mul: 곱하기, div: 나누기

```
# 함수 정의
```

```
...
```

```
a = 1
```

```
b = 2
```

```
c = add(a, b)
```

```
d = sub(a, b)
```

```
e = mul(a, b)
```

```
f = div(a, b)
```

```
print(c, d, e, f)      # 3 -1 2 0
```

가변 매개변수

- 매개변수가 몇 개인지 모를 경우

```
def 함수이름 (*매개변수) :  
    수행할 문장  
    ...
```

- 예) 짝수 개수 구하기

```
def count_even(*n):  
    cnt = 0  
    for v in n:  
        if v % 2 == 0:  
            cnt += 1  
    return cnt
```

```
>>> count_even(1,2,3,4,5)
```

```
def f1(a, *b):      # 정상  
  
def f2(a, *b, c):   # 에러  
  
def f3(*a, *b):     # 에러  
  
def f4(a, b, *c):
```

지역변수, 전역변수

- 지역변수 (Local Variable)
 - : 함수 안에서 만들어져
 - 함수 안에서만 사용하는 변수

```
def add_value(a, b) :  
    add_res = a+b  
    return add_res  
  
x = 10  
y = 20  
res = add_value(x, y)  
print(add_res) # 에러 발생
```

- 전역변수 (Global Variable)
 - : 함수 밖에서 만들어져
 - 아무데서나 사용할 수 있는 변수
 - 단, 함수 안에서 전역변수 수정 불가

```
def addX(_x) :  
    return x + _x  
  
x = 10  
y = 20  
  
res = addX(y)  
print(res) # ??
```

지역변수, 전역변수

- 함수 안에서 전역변수 수정 불가, 수정하려면 global 키워드 사용

```
def addX(_x) :  
    x = x + _x # 에러 발생
```

```
x = 10  
y = 20
```

```
addX(y)  
print(x)
```

```
def addX(_x) :  
    global x # 전역변수임을 명시  
    x = x + _x
```

```
x = 10  
y = 20
```

```
addX(y)  
print(x)
```

실습문제 2

- 양의 정수 n 을 전달받아 1부터 n 까지 곱해주는 factorial 함수를 작성하시오

```
def factorial(n):  
    ...  
    ...
```

```
>>> factorial(3)  
6  
>>> factorial(5)  
120  
>>> factorial(8)  
40320  
>>> factorial(1)  
1
```

실습문제 3

- n 과 a 를 전달받아 n^a 를 계산하는 거듭제곱 함수를 작성하시오

```
def pow(n, a):  
    ...  
    ...
```

```
>>> pow(5, 3)  
125  
>>> pow(4, 2)  
16  
>>> pow(2, 4)  
16
```

디폴트 매개변수

- 매개변수의 기본값을 지정해줄 수 있다
 - 함수 호출 시, 해당 매개변수에 대응되는 값을 주지 않으면 정의할 때 지정한 기본값을 가짐

```
def printName(first, second='Kim'): #second 변수 기본값 'Kim'
    print ('My name is', first, second + '.')

#함수 호출
printName('Gildong', 'Hong')
printName('Gildong') #second 변수에 대응되는 값을 주지 않았으므로
                    #정의할 때 지정한 'Kim'을 가짐
```

디폴트 매개변수

- 앞 매개변수가 디폴트 값을 가지면, 뒤에 오는 매개변수는 반드시 디폴트 값을 가져야 함

```
def printName(first, second='Kim'): # 정상  
  
def printName(first='Kim', second): # 에러 발생  
  
def printName(first, second, third='Kim'): # 정상  
  
def printName(first, second='Kim', third='M') # 정상
```


키워드 매개변수

- 함수호출 시, 해당 매개변수 이름을 명시적으로 지정해서 전달

```
def calc(x, y, z) :  
    return x+y+z
```

```
result = calc(y=20, x=10, z=30) # 매개변수 순서 상관 없음  
print(result)
```

```
def calc(x, y=0, z=0) :  
    return x+y+z
```

```
>>> calc(y=20, x=10) # 정상  
>>> calc(10, y=30, z=20) # 정상  
>>> calc(10, 30, y=20) # 에러 발생
```

반환 값이 여러 개인 함수

- 튜플이나 리스트를 이용해 여러 개의 값을 반환할 수 있음

```
# 함수 정의
```

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

```
# 함수 호출
```

```
add_res, mul_res = add_and_mul(3, 4)  
print(add_res)  
print(mul_res)
```

실습문제 4

- 초를 전달받아 시간, 분, 초를 반환하는 함수를 작성하시오

```
def hour_min_sec(second):  
    ...  
    ...  
  
hour, min, sec = hour_min_sec(57894)  
print("%d시간 %d분 %d초" % (hour, min, sec))
```

16시간 4분 54초

매개변수 종류

- Mutable 타입의 매개변수
 - Call by Reference 방식과 동일

```
def test(a):  
    a[1] = 20
```

```
t = [1, 2, 3]
```

```
test(t)  
print(t)
```

```
def test(a):  
    a['b'] = 3
```

```
t = {'c':1}
```

```
test(t)  
print(t)
```

- 리스트, 딕셔너리, 집합

매개변수 종류

- Immutable 타입의 매개변수
 - Call by Value 방식과 동일

```
def test(a):  
    a += (4, 5, 6)
```

```
t = (1, 2, 3)
```

```
test(t)  
print(t)
```

```
def test(a):  
    a += 1
```

```
t = 1
```

```
test(t)  
print(t)
```

- 숫자, 문자열, 튜플 등

실습문제 5

- 숫자로 구성된 리스트를 전달받아 최대값, 최소값을 반환하고, 두 수를 리스트에서 제거하는 함수를 작성하시오

```
def get_min_max(l):  
    ...  
    ...  
  
l = [3, 5, 9, 1, 2]  
min_val, max_val = get_min_max(l)  
  
print(min_val)  
print(max_val)  
print(l)
```

```
1  
9  
[3, 5, 2]
```



외장 함수

RANDOM, TIME, SYS, OS 등



난수(Random Number) 생성

- 난수는 random 모듈을 이용하여 생성

```
>>> import random
```

- 리스트에서 랜덤하게 선택

```
>>> random.choice('abcdefg')
```

- a이상 b이하인 임의의 정수 뽑기 (a, b: int 타입)

```
>>> random.randint(a, b)    # a~b 정수 값 중 하나
```

```
>>> random.randrange(a, b+1) # randint(a, b)와 동일한 기능
```


난수(Random Number) 생성

- 리스트 내용 랜덤으로 섞기

```
>>> Ls = [1,2,3,4,5]
```

```
>>> random.shuffle(Ls)
```

```
>>> print(Ls)
```

- 리스트 중 랜덤으로 여러 개 뽑기

```
>>> random.sample([1, 2, 3, 4, 5, 6], 3)
```

실습문제 6

- 로또 추첨 프로그램
 - 1~45 숫자 중 6개를 임의로 선택 후 출력
 - 단, 중복된 번호를 가질 수 없음

**** 로또 추첨을 시작합니다. ****

추첨된 로또 번호 ==> 1 3 18 32 38 44 ————— 자동으로 추출된 번호 6개

Time 모듈

- import **time**

- 1970년 1월 1일 0시 이후로 누적된 시간을 초단위로 반환

```
>>> time.time()
```

- 현재 한국 시각을 반환

```
>>> time.localtime( time.time() )
```

- 시각을 원하는 형식대로 출력

```
>>> time.strftime('%Y-%m-%d', time.localtime(time.time()))
```

날짜/시간 포맷코드- %Y:연, %m:월, %d:일

실습문제 8

1. 전달받은 문장에서 각 알파벳이 몇 개씩 나오는지 저장하는 딕셔너리를 만든 후, 반환하는 함수 `letter_dict()`를 작성하시오.
2. `letter_dict()` 함수를 통해 생성된 딕셔너리에서 가장 많이 등장한 알파벳을 찾아 반환하는 함수를 작성하시오. (여러 개일 경우 아무 값이나 하나만 반환)

```
def letter_dict(str):  
    ...  
    ...  
  
def max_letter(dict):  
    ...  
    ...  
  
a = letter_dict('red apple')  
print(a)  
print(max_letter(a))
```

```
{'r':1, 'e':2, 'd':1, ' ':1,  
'a':1, 'p':2, 'l':1}
```

e

실습문제 9

1. `letter_dict()` 함수에 'yellow banana' 를 전달하여 새로운 딕셔너리 `b` 를 생성한다.
2. 두 딕셔너리 `a`, `b`를 전달받아 각 딕셔너리에 등장한 값을 모두 더한 새로운 딕셔너리 `c`를 반환하는 함수 `comb_dict()`를 작성하시오.

```
def comb_dict(dict1, dict2):  
    ...  
    ...  
  
b = letter_dict('yellow banana')  
c = comb_dict(a, b)  
  
print(c)  
print(max_letter(c))
```

```
{'r':1, 'e':3, 'd':1, ' ':2,  
'a':4, 'p':2, 'l':3, 'y':1,  
'o':1, 'w':1, 'b':1, 'n':2}
```

a

과제 1 - prime.py

- 소수(Prime Number) 판단하는 함수 정의하기
 - 소수이면, True 반환
 - 소수가 아니면, False 반환

```
def check_prime(num):  
    # TO BE IMPLEMENTED  
    ...  
  
def main():  
    a = 13  
    b = 15  
  
    if check_prime(a):  
        print(str(a)+'는 소수입니다.')  
    else:  
        print(str(a)+'는 소수가 아닙니다.')  
  
    if check_prime(b):  
        print(str(b)+'는 소수입니다.')  
    else:  
        print(str(b)+'는 소수가 아닙니다.')
```

과제 2 – add_comma.py

- 천 단위마다 쉼표(,) 추가하는 함수 정의하기
- 1000000(int)이면, '1,000,000'(str) 반환

```
def add_comma(val):  
    # TO BE IMPLEMENTED  
    ...  
  
def main():  
    comma_added_1234 = add_comma(1234)  
    comma_added_12345678 = add_comma(12345678)  
    comma_added_12 = add_comma(12)  
  
    print(comma_added_1234)      # '1,234'  
    print(comma_added_12345678) # '12,345,678'  
    print(comma_added_12)       # '12'
```

과제 3 - tok.py

- 전달받은 단어열의 n-gram을 구해주는 tokenize() 함수 정의하기

```
def tokenize(trg, N=1):  
    # TO BE IMPLEMENTED  
    ...  
  
def main():  
    a="There was a farmer who had a dog ."  
    print(tokenize(a))  
    print(tokenize(a, 2))
```

```
['There', 'was', 'a', 'farmer', 'who', 'had', 'a', 'dog', '.']  
['There was', 'was a', 'a farmer', 'farmer who', 'who had', 'had a', 'a dog', 'dog .']
```


과제 4 - mean_and_var.py

- 여러 벡터를 전달받아, 평균(Mean)과 분산 (Variance)을 구하는 함수 정의하기

```
def mean_and_var(*val):  
    # TO BE IMPLEMENTED  
    ...  
  
def main():  
    v1=(0, 1)  
    v2=(0.5, 0.5)  
    v3=(1, 0)  
    m, var = mean_and_var(v1, v2, v3)  
    print('평균: ', m)  
    print('분산: ', var)
```

```
평균:  [0.5, 0.5]  
분산:  [0.16666666666666666, 0.16666666666666666]
```

부록

- python OOO.py 를 실행할 때 main() 호출 방법
 - 내장변수 `__name__`이 `'__main__'`이라는 문자열을 갖는 것을 이용

```
...
```

```
if __name__ == '__main__' : # 파일 제일 아래에 작성  
    main()
```

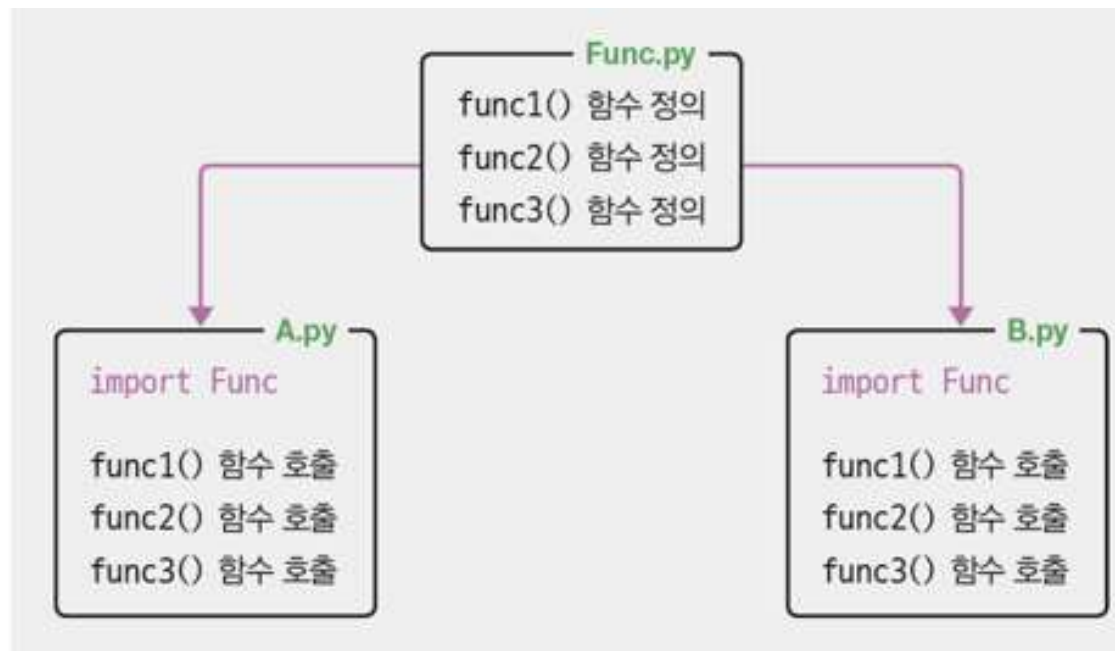


모듈



모듈

- 많이 사용하는 함수를 만들어 놓고, 프로그램에서 해당 함수 사용 시, import하여 사용하면 편리함



모듈

■ 모듈의 작성과 사용

- 모듈로 사용할 파일과 호출하는 파일은 모두 같은 폴더에 저장

```
#FILE: test.py
import arth
```

```
numbers = input('Enter two integers: ')
```

```
a, b = numbers.split()
a = int(a)
b = int(b)
```

모듈이름.함수이름() 형식으로 사용

```
print ("%d + %d = %2d" % (a, b, arth.add(a, b)))
print ("%d - %d = %2d" % (a, b, arth.sub(a, b)))
print ("%d * %d = %2d" % (a, b, arth.mul(a, b)))
print ("%d / %d = %.1f" % (a, b, arth.div(a, b)))
```

```
#FILE: arth.py

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

def sub(a, b):
    return a - b

def mul(a, b):
    return a * b

def div(a, b):
    return a / b
```

모듈이름 생략하고 함수이름만으로 사용하고 싶다면 **from 모듈명 import ***