10장 함수의 모듈

10.1 고급함수 사용하기

10.1.1 지역변수와 전역변수

지역변수와 전역변수 차이 (`객관식으로 나올 수 있음`)

지역변수	전역변수
- 코드 블록 내부에서만 사용되며, 그 안에서만 유효 - 코드블록이 끝나면 소멸	- 모든 곳에서 사용가능 - 프로그램 종료시까지 남아있음

함수 내에서 전역변수 사용

```
a = 3
def f():
    global a
    a += 3
f()
print(a) #6
```

함수를 리스트에 할당해 사용하기

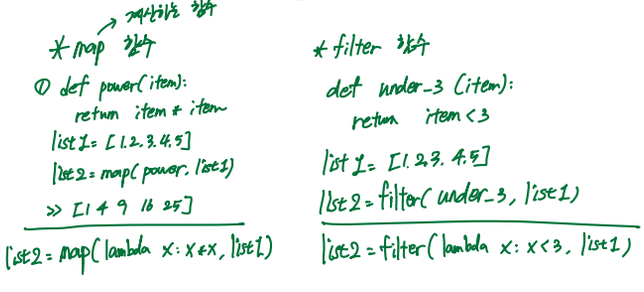
```
def plus(a,b):
    return a+b
def minus(a,b):
    return a-b

whole = [plus, minus]

print(whole[0](1,3)) # 4
print(whole[1](4,2)) # 2
```

함수를 매개변수로 전달하기 1 : map()함수

map(함수, 리스트)



```
def power(item):
    # 리스트끼리 곱하기는 불가능..
    return item*item

list1 = [1,2,3,4,5]
list2 = map(power,list1)
print(list2) # <map object at 0x000001AC08D67310>
print(list(list2)) # [1,4,9,16,25]
```

map 함수를 사용해서 `map(power, list1)`을 실행하면 list1의 값들 하나하나가 제곱되어 나온 리스트를 얻을 수 있다.

```
def under_3(item):
    return item < 3
list1 = [1,2,3,4,5]
list2 = map(under_3, list1)
print(list(list2))
'''
[True, True, False, False, False]
```

map함수를 사용해서 `map(under_3, list1)`를 실행하면 bool 형태로 값이 나옴을 알 수 있다.

함수를 매개변수로 전달하기 2 : filter()함수

filter(함수, 리스트)

```
def power(item):
    return item*2

list1 = [1,2,3,4,5]
list2 = filter(power, list1)
print(list(list2))
'''
[1,2,3,4,5]
'''
```

filter를 사용해서 `filter(power, list)`를 실행하면 값이 곱해지지 않은 채로 반환됨을 알 수 있다.

```
def under_3(item):
    return item < 3

list1 = [1,2,3,4,5]
list2 = filter(under_3, list1)
print(list(list2))
'''
[1,2]
'''
```

filter를 사용해서 `filter(under_3, list1)`을 실행하면 3보다 작은 1,2만 리스트로 출력된다. (False인 값들이 걸려졌다.)

차이점을 비교해보자면 다음과 같다.

map	filter
기존 리스트 또는 튜플을 인자로 전달받아 하나의 iterable한 객체를 생성하여 반환해준다.	filter는 걸려주는 역할을 한다 map함수처럼 함수와 매개변수를 인자로 전달받는다는 것은 공통점이지만, 거럴내는 것이 목적이기 때문에 `power`함수를 사용했을 땐 아무런 변화가 없으며, `under_3`함수를 사용할 땐 3보다 큰 3,4,5가 걸려졌음을 알 수 있다.

함수를 매개변수로 전달하기 3: lambda

이름없는 함수

def를 사용해야 할 정도로 복잡하지 않은 경우 한줄로 함수를 생성할 수 있다

함수를 매개변수로 전달할 때 유용하게 사용가능

```
# map함수를 사용하는 경우
def power(item):
    return item*item
list1 = [1,2,3,4,5]
list2 = map(power, list1)
print(list(list2)) # [1,4,9,16,25]
```

```
# lambda를 사용하는 경우
power = lambda x: x*x
list1 = [1,2,3,4,5]
list2 = map(power,list1)
print(list(list2)) # [1,4,9,16,25]
```

```
# lambda를 사용하는 경우
list1 = [1,2,3,4,5]
list2 = map(lambda x:x**2, list1)
print(list(list2))# [1,4,9,16,25]
```

10.2 모듈 이용하기

Python의 모듈

함수나 변수를 모아놓은 파일

다른 파이썬 프로그램에서 불러와 사용할 수 있도록 만들어진 파이썬 파일

각각의 소스파일(.py)들을 모듈이라고 부른다.

하나 또는 여러 개의 모듈을 모아놓은 것을 패키지라고 한다.

모듈의 분류

- 표준 모듈 : 파이썬에 기본적으로 내장되어 있는 모듈
- 외부 모듈 : 다른 프로그래머 혹은 업체가 만들어서 공개한 모듈
- 사용자 생성 모듈 : 프로그래머가 직접 작성한 모듈

사용자 생성 모듈 만들기

```
%%writefile cal.py
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
```

```
import cal
print(cal.add(1,3)) #4
```

모듈 사용하기

cal 모듈에서 add함수를 사용하는 방법

```
# 1 import
import cal
print(cal.add(1,3)) # 4
```

```
# 2 from ~ import ~
from cal import add
print(add(1,3)) # 4
```

```
# 3 as
import cal as c
print(c.add(1,3)) #4
```

메인모듈과 하위모듈

모듈	__ name __변수
main 모듈 (파이썬에서 제일 먼저 실행되는 파일)	__ main __
하위 모듈 (test_module.py), import되어서 실행되는 파일	test_module

```
%%writefile test_module.py
# 하위모듈
print("모듈의 __name__ 출력하기")
print(__name__) # test_module
```

```
import test_module

print("메인의 __name__ 출력하기")
print(__name__)
'''
모듈의 __name__출력하기
test_module
메인의 __name__출력하기
__main__
'''
```

하위모듈

우선 test_module을 import하였으므로, test_module의 __ name __이 출력된다. (test_module)

그 다음, main 모듈의 __ name __ 이 출력된다. (__ main __)

메인모듈

__ name __ 변수를 메인 모듈인지 아닌지를 확인하기 위한 수단으로 사용하기도 함.

우선 하위 모듈을 선언

```
%%writefile call.py
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
print("add(1,3):", add(1,3))
print("sub(1,3):", sub(1,3))
'''
writing call.py
'''
```

```
import call as c
print(c.add(1,2))
print(c.sub(1,2))
'''
add(1,3):4
sub(1,3):-2
3
-1
'''
```

대표적 표준 모듈 : random 모듈

```
import random
random.random()
# 0.0과 1.0 사이의 랜덤한 float
```

random.random()
0과 1 사이의 실수 값

```
random.uniform(0,2)
# 0.0과 1.0사이의 랜덤한 float 반환
```

uniform
random.~~random~~(a,b)
a와 b 사이의 실수 값

```
random.randrange(5)
# 0과 5사이의 랜덤한 int 반환
```

random.randrange(a)
0과 a 사이의 정수 값

```
random.randrange(2,5)
# 2와 5사이의 랜덤한 int 반환
```

random.randrange(a,b)
a와 b 사이의 정수값

```
list1 = [3,2,4,1,5]
random.choice(list1)
# list1에서 임의로 한개 choice
```

list1 = [1,5,4,3,5]
random.choice(list1)
랜덤으로 list1에서 한개 추출

```
list1 = [1,2,3,4,5,6,7]
random.shuffle(list1)
list1
#list1을 섞는다.
```

list1 = [1,2,3,4,5]
random.shuffle(list1)
섞기

```
list1 = [1,2,3,45,1,3,65,]
random.sample(list1, 5)
# list1에서 5개를 뽑는다.
```

list1 = [1,2,3,4,5]
random.sample(list1, k=3)
list1에서 랜덤으로 3개 뽑기

10.3 패키지 이해하기

python의 패키지

모듈(.py들)이 폴더에 정리되어 계층적으로 모여있는 것

예를 통해 알아본다.

```
%%writefile testing/first.py
var_a = "a입니다."
```

```
%%writefile testing/second.py
var_b = "b입니다."
```

```
%%%writefile testing/third.py
var_c = "c입니다."
```

이렇게 하면 testing이라는 폴더에 first.py, second.py, third.py라는 모듈 3개가 만들어졌다.


```
from testing import first as a
print(a.var_a)
'''
a입니다
'''
```

```
%%writefile testing/__init__.py

__all__ = ["first","second"]
```

__ all __ 리스트 변수를 설정한다.

from 패키지 import *할 때 포함될 모듈의 목록을 선언한다. third모듈은 없음.

```
import testing
from testing import *
```

```
print(first.var_a)
print(second.var_b)
print(third.var_c)

'''
a입니다.
b입니다.
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [9], in <cell line: 6>()
      4 print(first.var_a)
      5 print(second.var_b)
----> 6 print(third.var_c)

NameError: name 'third' is not defined
'''
```