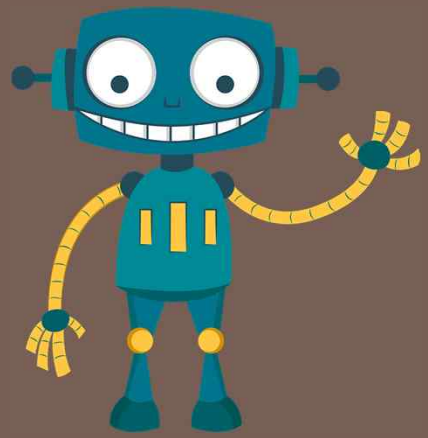
파이썬 익스프레스



11장 내장함수, 람다식, 제너레이터, 모듈

Q & A

- 언제라도 질문하세요
  1. 강의시간의 채팅창 (수신인: 모두, 수강생모두에게 답변하기에)
  2. 네이버카페 질의응답게시판

- 강사의 1번 선생님은 여러분들의 질문

앞으로 나는 할 수 있다

1. 나는 파이썬에 내장된 내장 함수들을 살펴본다.
2. 나는 람다식을 살펴본다.
3. 나는 제너레이터를 시용하여 반복 가능한 객체를 작성할 수 있다.
4. 나는 모듈의 개념을 살펴본다.
5. 나는 유용한 모듈을 사용할 수 있다.

내장함수 https://docs.python.org/3/library/functions.html

- 파이썬 인터프리터에는 항상 사용할 수 있는 많은 함수가 준비되어 있다. 이러한 함수를 내장 함수라고 한다.

내장 함수				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()

abs() 함수 (p505.py)

- abs() 함수는 숫자의 절대값을 반환하는 데 사용된다.

```
>>> i = -20
>>> abs(i)
20

>>> f = -30.92
>>> abs(f)
30.92
```

all() 함수 (p505.py)

- all() 함수는 시퀀스(리스트나 딕셔너리 등)를 받아서, 시퀀스의 모든 항목이 참이면 True를 반환한다.

```
>>> mylist = [1, 3, 4, 6]                # 모든 값이 참이다.
>>> all(mylist)
True

>>> mylist = [1, 3, 4, 0]                # 하나의 값이 거짓이다.
>>> all(mylist)
False

>>> mylist = [True, 0, False, 0]         # 하나의 값만 참이다.
>>> all(mylist)
False
```

any() 함수 (p505.py)

- any() 함수는 시퀀스 객체에 있는 한 개의 항목이라도 참인 경우 참을 반환한다.

```
>>> mylist = [0, 1, 2, 3]
>>> any(mylist)
True
>>> mylist = [0, False]
>>> any(mylist)
False
```

eval() 함수 (p506.py)

- eval() 함수는 전달된 수식을 구문 분석하고 프로그램 내에서 수식을 실행한다.

```
>>> x = 10
>>> eval('x + 1')
11


>>> exp = input("파이썬의 수식을 입력하시오: ")
파이썬의 수식을 입력하시오: 2**10
>>> eval(exp)
1024
```

sum() 함수 (p507.py)

- sum() 함수는 리스트에 존재하는 항목들을 전부 더하여 합계를 반환한다.

```
>>> sum([1, 2, 3 ])
6
>>> sum([1, 2, 3], 20)
26
```

len() 함수 (p507.py)

- len() 함수는 객체의 길이를 계산하여 반환하는 함수이다

```
>>> len("All's well that ends well. ")
27

>>> len([1, 2, 3, 4, 5])
5
```

list() 함수 (p507.py)

- 리스트를 생성하는 함수이다.

```
>>> s = 'abcdefg' # string
>>> list(s)
['a', 'b', 'c', 'd', 'e', 'f', 'g']

>>> t = (1, 2, 3, 4, 5, 6) # tuple
>>> list(t)
[1, 2, 3, 4, 5, 6]
```

map() 함수 (p507.py)

- map() 함수는 반복가능한 객체(리스트, 튜플 등)의 각 항목에 주어진 함수를 적용한 후, 결과를 반환한다

```
def square(n):
        return n*n

mylist = [1, 2, 3, 4, 5]
result = list(map(square, mylist))
print(result)
```

```
[1, 4, 9, 16, 25]
```

dir() 함수(p508.py)

- dir은 객체가 가지고 있는 변수나 함수를 보여 준다.

```
>>> dir([1, 2, 3])
['__add__',
'__class__',
'__contains__',
'__delattr__',
...
```

max(), min() 함수 (p509.py)

- max() 함수는 리스트나 튜플, 문자열에서 가장 큰 항목을 반환한다

```
>>> values = [ 1, 2, 3, 4, 5]
>>> max(values)
5
>>> min(values)
1
```

enumerate() 함수(p509.py)

- 시퀀스 객체를 입력 받아, 열거형 (enumerate) 객체를 반환한다.
  열거형 객체 : (number, value)로 구성

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']

>>> enu = enumerate(seasons)
>>> print(enu)
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]

>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
filter()
```

filter() 함수 (p509.py)

- filter() 함수는 특정 조건을 만족하는 요소만을 뽑는다.

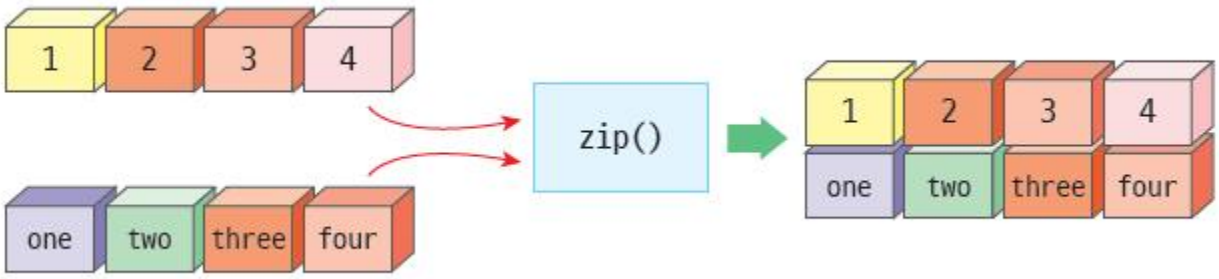
```
def myfilter(x):
        return x > 3

result = filter(myfilter, (1, 2, 3, 4, 5, 6))
print(list(result))
```

```
[4, 5, 6]
```

zip() 함수 (p510.py)

- zip() 함수는 2개의 자료형을 하나로 묶어주는 함수이다.

```
>>> numbers = [1, 2, 3, 4]

>>> slist = ['one', 'two', 'three', 'four']

>>> list(zip(numbers, slist))
[(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
```



Lab: 내장 함수 예제 (p511.py)

- 우리는 파티에 적합한 양의 음식을 구입하기 위해서는 총 인원이 몇 명인지 알아야 한다.

```
>>> invitations = ["Kim", "Lee", "Park", "Choi"]
>>> persons = [1, 3, 0, 6]
>>> sum(persons)
10

파티에 한 사람이라도 오는 지를 확인해보자. 이것은 any() 함수로 가능하다.
>>> any(persons)
True

이번에는 모든 초대받은 그룹이 전부 오는 지는 확인해보자. 이것은 all() 함수로 가능하다.
>>> all(persons)
False

가장 많이 오는 그룹에는 몇 사람이나 있는지를 알아보자. max()로 가능하다.
>>> max(persons)
6
```

정렬과 탐색 (p513.py)

- 파이썬 리스트는 sort()라는 메소드를 가지고 이 메소드는 리스트를 정렬된 상태로 변경한다.
- sorted()라는 내장 함수는 반복 가능한 객체로부터 정렬된 리스트를 생성한다.

학생> sort()와 sorted()는 어떻게 다른가요?

교수> 기능은 동일한데 sorte는 함수이고 sort()는 메소드라 sort()는 객체.sort()로 호출됩니다

```
>>> sorted([4, 2, 3, 5, 1])
[1, 2, 3, 4, 5]

>>> myList = [4, 2, 3, 5, 1]
>>> myList.sort()
>>> myList
[1, 2, 3, 4, 5]
```

key 매개변수 (p513.py)

- 정렬을 하다 보면 정렬에 사용되는 키를 개발자가 변경해주어야 하는 경우가 종종 있다.
- Key 매개변수로 정렬을 하기 전에 각 요소에 대하여 호출되는 함수를 지정할 수 있음

```
>>> sorted("The health know not of their health, but only the sick".split(), key=str.lower)
['but', 'health', 'health,', 'know', 'not', 'of', 'only', 'sick', 'The', 'the', 'their']
```

교수> key매개변수 삭제한 실행결과와 비교

예제 (p513.py)

- Lamda는 11.3에서 배움
- Student의 학번으로 정렬방법

```
students = [
('홍길동', 3.9, 20160303),
('김철수', 3.0, 20160302),
('최자영', 4.3, 20160301),
]
print(sorted(students, key=lambda student: student[2]))
```

```
[('최자영', 4.3, 20160301), ('김철수', 3.0, 20160302), ('홍길동', 3.9, 20160303)]
```

오름차순 정렬과 내림차순 정렬 (reverse 매개변수)

```
>>> help (sorted)
```

```
>>> sorted(students, key=lambda student: student.number, reverse=True)
[('홍길동', 3.9, 20160303), ('김철수', 3.0, 20160302), ('최자영', 4.3, 20160301)]
```

Lab: 키를 이용한 정렬 예제 (contacts.py)

- 주소록을 작성한다고 하자. 간단하게 사람들의 이름과 나이만 저장하고자 한다. 사람을 Person 이라는 클래스로 나타낸다. Person 클래스는 다음과 같은 인스턴스 변수를 가진다.
  - name – 이름을 나타낸다.(문자열)
  - age – 나이를 나타내낸다.(정수형)
- 나이순으로 정렬하여 보여주는 프로그램을 작성하자. 정렬의 기준의 사람의 나이이다.

```
[<이름: 홍길동, 나이: 20>, <이름: 김철수, 나이: 35>, <이름: 최자영, 나이: 38>]
```

Lab: 키를 이용한 정렬 예제

```
class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __repr__(self):
        return "<이름: %s, 나이: %s>" % (self.name, self.age)

def keyAge(person):
    return person.age

people = [Person("홍길동", 20), Person("김철수", 35), Person("최자영", 38)]
print(sorted(people, key = keyAge))
```

```
[<이름: 홍길동, 나이: 20>, <이름: 김철수, 나이: 35>, <이름: 최자영, 나이: 38>]
```

람다식

- 람다식는 이름은 없고 몸체만 있는 함수이다.
- 람다식은 **lambda** 키워드로 만들어진다.
- 람다식은 딱 한 번 사용하는 함수를 만드는데 사용된다.
  이름이 없어서 여러 번 call할 수 없음

Syntax: 람다식 정의

형식 lambda 매개 변수들: 수식

예 lambda x, y: x+y;

매개 변수 함수의 몸체

Syntax: 함수 정의

형식
```
def 함수이름(매개변수1, 매개변수2, ...) :
    명령문1
    명령문2
```

예
```
def get_area(radius) :
    area = 3.14*radius**2
    return area
```

함수 이름 매개 변수 함수 헤더 함수 몸체 return 문장은 함수를 종료시키고 결과를 반환한다.

예제 (lambda와 def의 비교)

- 람다식는 이름은 없고 몸체만 있는 함수이다.
- 람다식은 return 이 필요없음

```
f = lambda x, y: x+y;

print( "정수의 합 : ", f( 10, 20 ))
print( "정수의 합 : ", f( 20, 20 ))
```

```
정수의 합 : 30
정수의 합 : 40
```

```
def get_sum(x, y):
        return x+y

print( "정수의 합 : ", get_sum( 10, 20 ))
print( "정수의 합 : ", get_sum( 20, 20 ))
```

map() 함수와 람다식 (map(p517).py)

학생> 매개변수로 함수도 되네요

교수> 매개변수로 변수, 객체 (8장, television2.py)
매개변수로 함수 (https://brownbears.tistory.com/107)

```
list_a = [ 1, 2, 3, 4, 5 ]
f = lambda x : 2*x
result = map(f, list_a)
print(list(result))
```

```
[2, 4, 6, 8, 10]
```

- 매개변수로 함수 사용 예제

```
def send3(func, a):
    print(func(a))
f = lambda x : 2*x
send3(f,2)
#4
send3(lambda x : 2*x,3)
#6
```

filter() 함수와 람다식

```
list_a = [1, 2, 3, 4, 5, 6]
result = filter(lambda x : x % 2 == 0, list_a)
print(list(result))
```

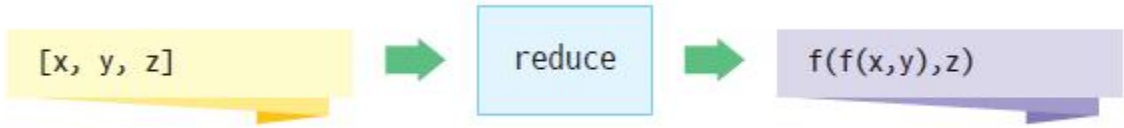
```
[2, 4, 6]
```

```
data = [(3, 100), (1, 200), (7, 300), (6, 400)]
sorted(data, key=lambda item: item[0])
print(data)
```

```
[(3, 100), (1, 200), (7, 300), (6, 400)]
```

reduce() 함수와 람다식

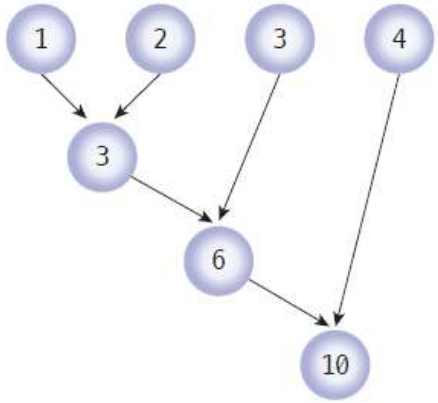
- reduce(func, seq) 함수는 func() 함수를 시퀀스 seq에 연속적으로 적용하여 단일 값을 반환한다.



예제 (reduce(p517).py)

```
import functools
result = functools.reduce(lambda x,y: x+y, [1, 2, 3, 4])
print(result)
```

```
10
```



Lab: 람다식으로 온도 변환하기 def방식 (console 또는 00shim.py)

- 람다식과 map() 함수를 사용하여서 화씨 온도를 섭씨 온도로 변환하는 명령문을 작성해보자. 람다식을 사용하지 않는 버전은 다음과 같다.

```
def celsius(T):
    return (5.0/9.0)*(T-32.0)

f_temp = [0, 10, 20, 30, 40, 50]

c_temp = map(celsius, f_temp)
print(list(c_temp))
```

```
[-17.77777777777778, -12.22222222222223, -6.666666666666667, -
1.1111111111111112, 4.444444444444445, 10.0]
```

Sol: 람다식으로 온도 변환하기 lambda방식 (lambda1.py)

```
##
#          이 프로그램은 람다식을 이용하여 온도를 변환한다.
#

f_temp = [0, 10, 20, 30, 40, 50]
c_temp = map(lambda x: (5.0/9.0)*(x-32.0), f_temp)
print(list(c_temp))
```

```
[-17.77777777777778, -12.222222222222223, -6.666666666666667, -
1.1111111111111112, 4.444444444444445, 10.0]
```

Lab: 람다식으로 데이터 처리하기 (lambda2.py)

- 아래의 데이터를 처리하여서 2개의 튜플이 저장된 리스트를 반환하는 프로그램을 작성하라

주문 번호	물품	수량	품목 당 가격
1	"재킷"	5	120000
2	"셔츠"	6	24000
3	"바지"	3	50000
4	"코트"	6	300000

```
[('1', 600000), ('2', 144000), ('3', 150000), ('4', 1800000)]
```

Sol:

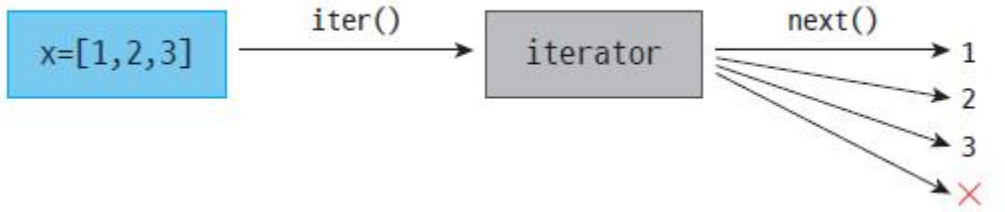
```
##
#          이 프로그램은 람다식을 이용하여 주문을 처리한다.
#
orders = [ ["1", "재킷", 5, 120000],
           ["2", "셔츠", 6, 24000],
           ["3", "바지", 3, 50000],
           ["4", "코트", 6, 300000] ]

result = list(map(lambda x: (x[0], x[2] * x[3]), orders))
print(result)
```

이터레이터

- 파이썬에서는 for 루프와 함께 사용할 수 있는 여러 종류의 객체가 있으며 이들 객체는 **이터러블 객체 (iterable object)**이라고 불린다.
- for 변수 in iterable object :
  문장1
  문장2
- for 변수 in range(start=0, stop, step=1) :
  문장1
  문장2

객체가 이터러블 객체가 되려면

- __iter__ ()은 이터러블 객체 자신을 반환한다.
- __next__ ()은 다음 반복을 위한 값을 반환한다. 만약 더 이상의 값이 없으면 StopIteration 예외를 발생하면 된다.



예제 (iterator1.py) : 이터레이터 클래스 만드는 방법 __iter__()와 __next__() 2가지 구현

```
class MyCounter(object):
    # 생성자 메소드를 정의한다.
    def __init__(self, low, high):
        self.current = low
        self.high = high

    # 이터레이터 객체로서 자신을 반환한다.
    def __iter__(self):
        return self

    def __next__(self):
        # current가 high 보다 크면 StopIteration 예외를 발생한다.
        # current가 high 보다 작으면 다음 값을 반환한다.
        if self.current > self.high:
            raise StopIteration
        else:
            self.current += 1
            return self.current - 1
```

예제

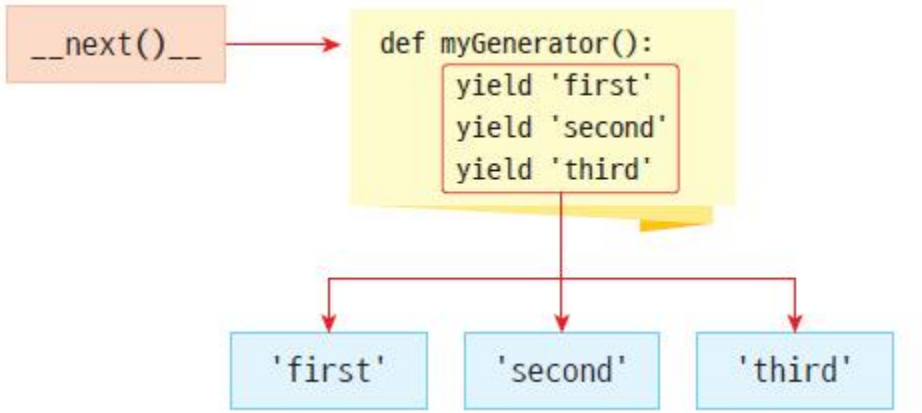
```
c = MyCounter(1, 10)
for i in c:
        print(i, end=' ')
```

```
1 2 3 4 5 6 7 8 9 10
```

제너레이터

- 제너레이터**(generators)**는 키워드 yield(생산하다)를 사용하여서 함수로부터 반복가능한 객체를 생성하는 하나의 방법이다.
- 반복가능한 객체 만드는 2가지
  1. iterator class
  2. generator function

```
def myGenerator():
    yield 'first'
    yield 'second'
    yield 'third'
```



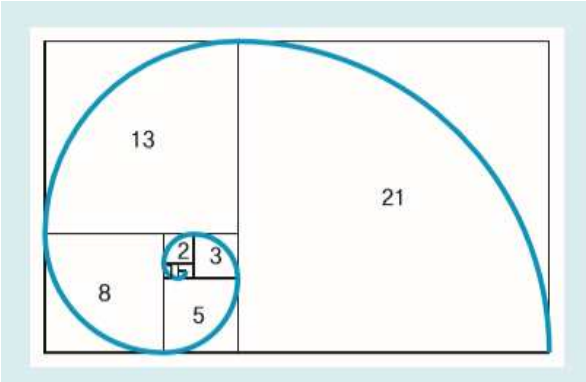
예제 (00shim.py)

```
def myGenerator():
    yield 'first'
    yield 'second'
    yield 'third'

for word in myGenerator():
    print(word)
```

```
first
second
third
```

Lab: 피보나치 이터레이터 (00shim.py)

- 피보나치 수열이란 앞의 두 수의 합이 바로 뒤의 수가 되는 수열을 의미한다. 피보나치 수열의 수들을 생성하는 이터레이터 클래스를 정의해보자.



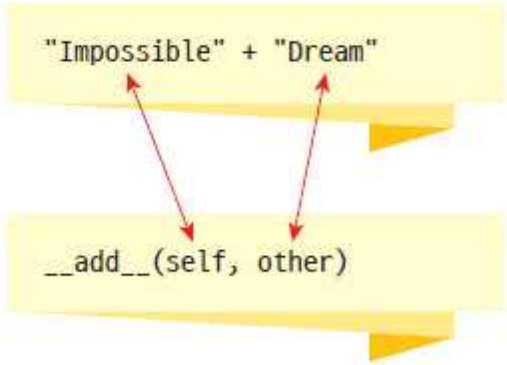

```
1 1 2 3 5 8 13 21 34
```

Solution

```
class FibIterator:
    def __init__(self, a=1, b=0, maxValue=50):

        self.a = a
        self.b = b
        self.maxValue = maxValue
    def __iter__(self):
        return self

    def __next__(self):
        n = self.a + self.b
        if n > self.maxValue:
            raise StopIteration()
        self.a = self.b
        self.b = n
        return n

for i in FibIterator():
    print(i, end=" ")
```

연산자 오버로딩

- 연산자를 메소드로 정의하는 것을 **연산자 오버로딩(operator overloading)**이라고 한다.



오버로딩할 수 있는 연산자

연산자	수식예	내부적인 함수 호출
덧셈	$x + y$	x.__add__(y)
뺄셈	$x - y$	x.__sub__(y)
곱셈	$x * y$	x.__mul__(y)
지수	$x ** y$	x.__pow__(y)
나눗셈(실수)	x / y	x.__truediv__(y)
나눗셈(정수)	$x // y$	x.__floordiv__(y)
나머지	$x \% y$	x.__mod__(y)
작음	$x < y$	x.__lt__(y)
작거나 같음	$x <= y$	x.__le__(y)
같음	$x == y$	x.__eq__(y)
같지 않음	$x != y$	x.__ne__(y)
큼	$x > y$	x.__gt__(y)
크거나 같음	$x >= y$	x.__ge__(y)

예제

```
>>> s1="Impossible "
>>> s2="Dream"
>>> s3 = s1.__add__(s2)
>>> s3
'Impossible Dream'
>>>
```

```
>>> class Point:
    def __init__(self,x = 0,y = 0):
        self.x = x
        self.y = y

>>> p1 = Point(1, 2)
>>> p2 = Point(3, 4)
>>> p1 + p2
...
TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

예제 (point_change.py)

```
>>> class Point:
    def __init__(self,x = 0,y = 0):
        self.x = x
        self.y = y

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)

    def __str__(self):
        return 'Point('+str(self.x)+', '+str(self.y)+')'

>>> p1 = Point(1, 2)
>>> p2 = Point(3, 4)
>>> print(p1+p2)
(4,6)
```

Lab: Book 클래스 (operator_over.py)

- 책을 나타내는 Book 클래스를 작성하고 Book 클래스 내부에 __gt__() 함수를 정의하여서 Book 클래스의 객체들을 서로 비교할 수 있게 하라. __gt__() 함수는 책의 페이지들을 비교하여 반환한다.

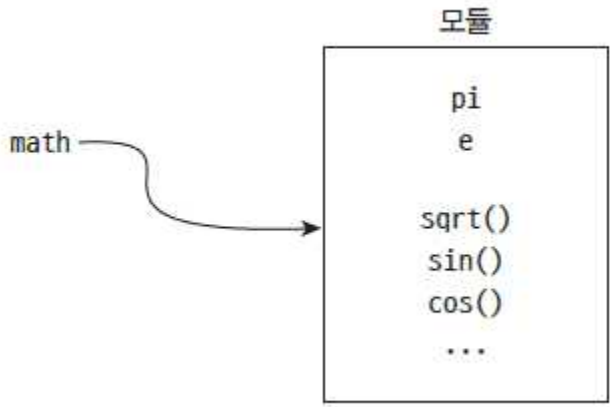
```
>>> book1 = Book('Magic of Python', 600)
>>> book2 = Book('Master of Python', 700)
>>> print(book1>book2)
False
```

Solution

```
##
#       이 프로그램은 연산자 중복을 사용하여 Book 객체의 len() 연산을 구현한다.
#
class Book:
        title = ''
        pages = 0

        def __init__(self, title='', pages=0):

Book 객체끼리의 + 연산을 정의한다.
                self.title = title
                self.pages = pages

        def __str__(self):
                return self.title

        def __gt__(self, other):
                return self.pages > other.pages
```

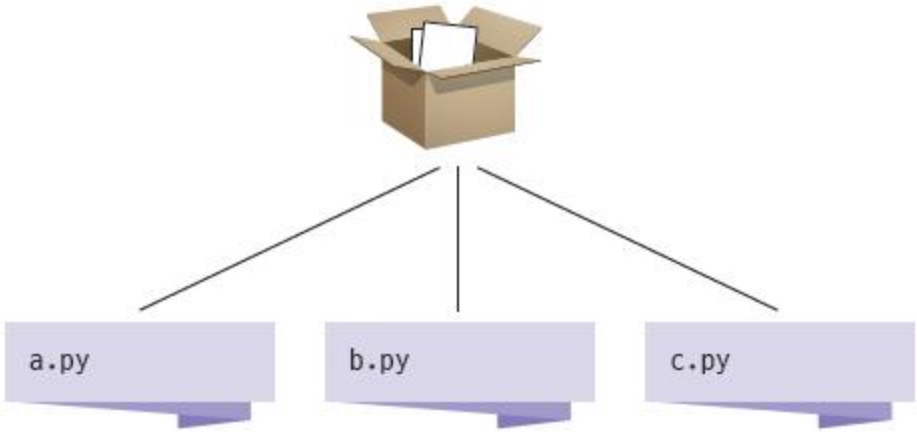
모듈

- 파이썬에서 **모듈(module)**이란 함수나 변수 또는 클래스 들을 모아 놓은 파일이다.



모듈

- 파이썬 프로그램이 길어지면, 유지 보수를 쉽게 하기 위해 여러 개의 파일로 분할 할 수 있다. 또한 파일을 사용하면 한번 작성한 함수를 복사하지 않고 여러 프로그램에서 사용할 수있다.



모듈 작성하기

fibo(p529).py

```
# 피보나치 수열 모듈

def fib(n):           # 피보나치 수열 출력
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):          # 피보나치 수열을 리스트로 반환
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

모듈 사용하기

```
>>> import fibo
학생> 교수님 에러나요~
교수> fibo(p529).py를 fibo.py로 복사

>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

>>> fibo.__name__
'fibo'
```

from 모듈 import 함수

- 만약 fibo.fib()와 같이 함수를 사용할 때마다 모듈의 이름을 적어주는 것이 귀찮다면 다음과 같이 "from 모듈 import 함수" 문장을 사용하여도 된다.

```
>>> from fibo import fib
>>> fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

>>> from fibo import *
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

모듈의 별칭

```
import mymodule as lib
...
lib.func(100)
```

모듈 실행하기

```
C> python fibo.py <arguments>
```

- 모듈 실행할때 if __name__ == "__main__": 가 true가 됨

```
...
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

```
C> python fibo.py 50
1 1 2 3 5 8 13 21 34
```

- 모듈 import할때 if __name__ == "__main__": 가 false가 됨
- 모듈을 독립적으로 테스트할때 사용

모듈 탐색 경로

1) 입력 스크립트가 있는 디렉토리(파일이 지정되지 않으면 현재 디렉토리)
2) PYTHONPATH 환경 변수
3) 설치에 의존하는 디폴트값

>>> sys.path

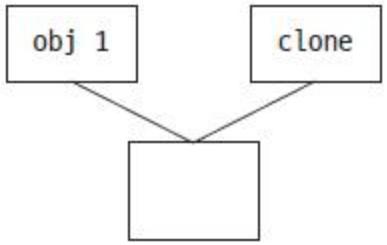
```
Out[10]:
['C:\\SPB_Data',
 'C:\\Anaconda3\\python37.zip',
 'C:\\Anaconda3\\DLLs',
 'C:\\Anaconda3\\lib',
 'C:\\Anaconda3',
 '',
 'C:\\Anaconda3\\lib\\site-packages',
 'C:\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\SPB_Data\\.ipython']

In [11]:
```

유용한 모듈

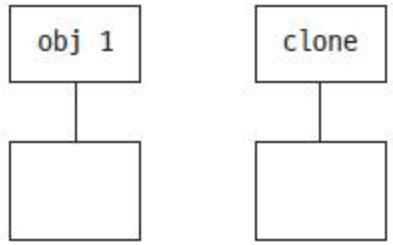
- 프로그래밍에서 중요한 하나의 원칙은 이전에 개발된 코드를 적극적으로 재활용하자는 것

copy 모듈

- src : 더블클릭>오른쪽버튼>Go to definition
- doc : https://docs.python.org/3/library/index.html
- 얇은 복사(shallow copy) - 객체의 참조값만 복사되고 객체 자체는 복사되지 않는다.
- 깊은 복사(deep copy) - 객체까지 복사된다.



얕은 복사

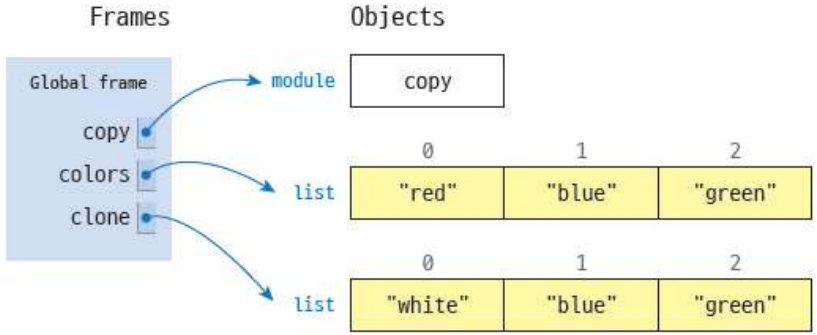


깊은 복사

copy 모듈 (deepcopy.py)

```
import copy
colors = ["red", "blue", "green"]
clone = copy.deepcopy(colors)

clone[0] = "white"
print(colors)
print(clone)
```

```
['red', 'blue', 'green']
['white', 'blue', 'green']
```



copy 모듈 (deepcopy_change.py)

- 얇은 복사(shallow copy) - 객체의 참조값만 복사되고 객체 자체는 복사되지 않는다.
- 깊은 복사(deep copy) - 객체까지 복사된다.

```
>>> id(colors)
>>> id(clone)
>>> id(shallow)
```

random 모듈 (console 또는 00shim.py)

```
>>> import random
>>> print(random.randint(1, 6))
6
>>> print(random.randint(1, 6))
3

>>> import random
>>> print(random.random()*100)
81.1618515880431

>>> myList = [ "red", "green", "blue" ]
>>> random.choice(myList)
'blue'
```

random 모듈

```
>>> myList = [ [x] for x in range(10) ]
>>> random.shuffle(myList)
>>> myList
[[3], [2], [7], [9], [8], [1], [4], [6], [0], [5]]

>>> for i in range(3):
        print(random.randrange(0, 101, 3))

81
21
57
```

sys 모듈

- sys 모듈은 파이썬 인터프리터에 대한 다양한 정보를 제공하는 모듈이다

```
>>> import sys
>>> sys.prefix # 파이썬이 설치된 경로
'C:\\Users\\chun\\AppData\\Local\\Programs\\Python\\Python35-32'

>>> sys.executable
'C:\\Users\\chun\\AppData\\Local\\Programs\\Python\\Python35-
32\\pythonw.exe'
```

time 모듈

```
>>> import time
>>> time.time() # 1970년 1월1일 자정이후 지금까지 흘러온 시간을 초단위로 출력
1461203464.6591916
```

예제 (time.py)

```
import time
def fib(n):    # 피보나치 수열 출력
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

start = time.time()
fib(1000)
end = time.time()
print(end-start)    # 코드 수행 시간 측정
```

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
0.03500199317932129
```

calendar 모듈

```
import calendar

cal = calendar.month(2016, 8)
print(cal)
```

```
    August 2016
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

keyword 모듈 (keyword.py)

```
import keyword

name = input("변수 이름을 입력하시오: ")

if keyword.iskeyword(name):
    print (name, "은 예약어임.")
    print ("아래는 키워드의 전체 리스트임: ")
    print (keyword.kwlist)
else:
    print (name, "은 사용할 수 있는 변수이름임.")
```

```
변수 이름을 입력하시오: for
for 은 예약어임.
아래는 키워드의 전체 리스트임.
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Lab: 동전 던지지 게임 (coin_toss.py)

- 하나의 예제로 동전 던지기 게임을 파이썬으로 작성해보자. random 모듈을 사용한다.

```
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) yes
tail
동전 던지기를 계속하시겠습니까?( yes, no) yes
tail
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) no
```

Sol:

```
import random
myList = [ "head", "tail" ]

while (True):
        response = input("동전 던지기를 계속하시겠습니까?( yes, no) ");
        if response == "yes":
                coin = random.choice(myList)
                print (coin)
        else :
                break
```

모듈이 없다고 하면? 모듈 추가 설치
사례)16장 Keras_mnist.py 실행시

- 교수님~ 에러 나요

```
  File "C:/idec/sources/chap16/keras_mnist.py", line 2, in <module>
    import tensorflow as tf

ModuleNotFoundError: No module named 'tensorflow'
```

- <Python 심화> IDEC-VOD에 모듈 추가 설치 자세한 설명 16장 slide61~64

1개의 파일을 여러 개의 파일로 나누는 사례

- 13장 breakout.py 1개의 파일을 여러 개의 파일(모듈)로 나누어 봄 (HW13장-2)

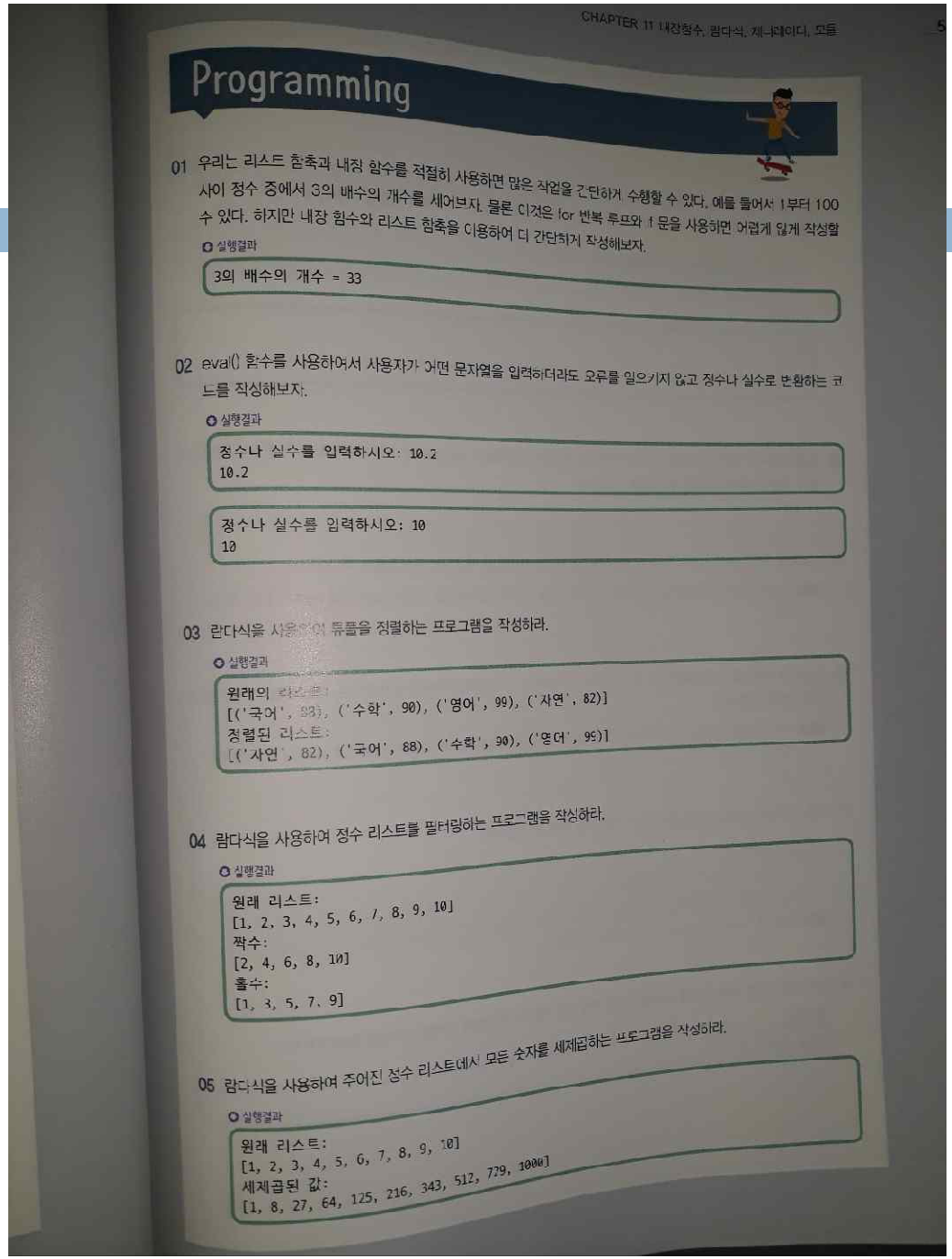
이제 나는 할 수 있다

1. 나는 파이썬에 내장된 내장 함수들을 살펴본다.
2. 나는 람다식을 살펴본다.
3. 나는 제너레이터를 시용하여 반복 가능한 객체를 작성할 수 있다.
4. 나는 모듈의 개념을 살펴본다.
5. 나는 유용한 모듈을 사용할 수 있다.

HW11장-1,2

- HW11장-1,2 게시판에 업로드

프로그래밍문제_1



Programming

01 우리는 리스트 함축과 내장 함수를 적절히 사용하면 많은 작업을 간단하게 수행할 수 있다. 예를 들어서 1부터 100 사이 정수 중에서 3의 배수의 개수를 세어보자. 물론 이것은 for 반복 루프와 if 문을 사용하면 어렵게 않게 작성할 수 있다. 하지만 내장 함수와 리스트 함축을 이용하여 더 간단하게 작성해보자.

실행결과

```
3의 배수의 개수 = 33
```

02 eval() 함수를 사용하여서 사용자가 어떤 문자열을 입력하더라도 오류를 일으키지 않고 정수나 실수로 변환하는 코드를 작성해보자.

실행결과

```
정수나 실수를 입력하시오: 10.2
10.2
```

```
정수나 실수를 입력하시오: 10
10
```

03 람다식을 사용하여 튜플을 정렬하는 프로그램을 작성하라.

실행결과

```
원래의 리스트:
[('국어', 93), ('수학', 90), ('영어', 99), ('사연', 82)]
정렬된 리스트:
[('사연', 82), ('국어', 88), ('수학', 90), ('영어', 95)]
```

04 람다식을 사용하여 정수 리스트를 필터링하는 프로그램을 작성하라.

실행결과

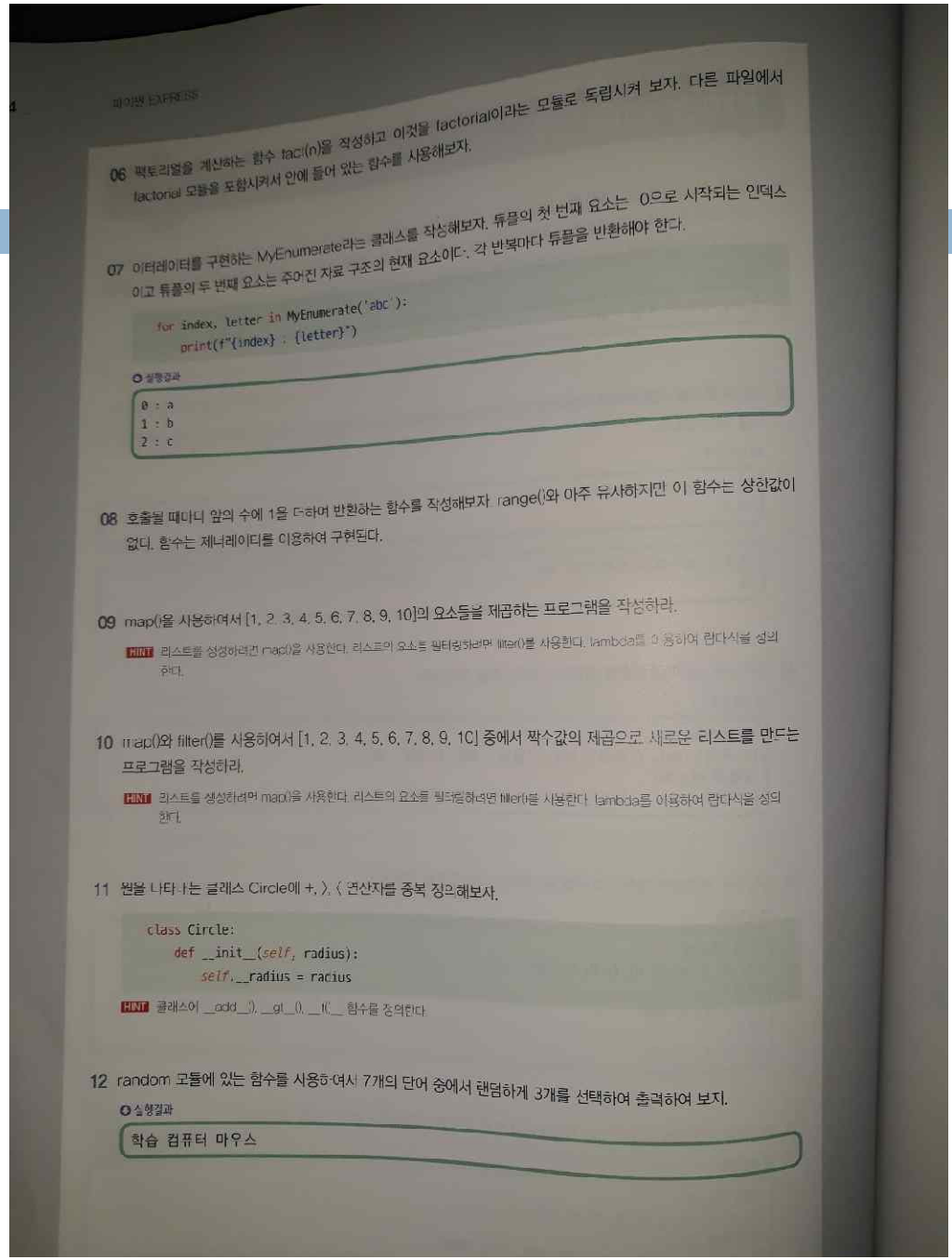
```
원래 리스트:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
짝수:
[2, 4, 6, 8, 10]
홀수:
[1, 3, 5, 7, 9]
```

05 람다식을 사용하여 주어진 정수 리스트에서 모든 숫자를 세제곱하는 프로그램을 작성하라.

실행결과

```
원래 리스트:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
세제곱된 값:
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

프로그래밍문제-2



파이썬 EXPRESS

06 팩토리얼을 계산하는 함수 fact(n)을 작성하고 이것을 factorial이라는 모듈로 독립시켜 보자. 다른 파일에서 factorial 모듈을 포함시켜서 안에 들어 있는 함수를 사용해보자.

07 이터레이터를 구현하는 MyEnumerate라는 클래스를 작성해보자. 튜플의 첫 번째 요소는 0으로 시작되는 인덱스이고 튜플의 두 번째 요소는 주어진 자료 구조의 현재 요소이다. 각 반복마다 튜플을 반환해야 한다.

```
for index, letter in MyEnumerate('abc'):
    print(f"{index} : {letter}")
```

실행결과

```
0 : a
1 : b
2 : c
```

08 호출될 때마다 앞의 수에 1을 더하여 반환하는 함수를 작성해보자. range()와 아주 유사하지만 이 함수는 상한값이 없다. 함수는 제너레이터를 이용하여 구현된다.

09 map()을 사용하여서 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]의 요소들을 제곱하는 프로그램을 작성하라.

HINT 리스트를 생성하려면 map()을 사용한다. 리스트의 요소를 필터링하려면 filter()를 사용한다. lambda를 이용하여 람다식을 생성한다.

10 map()와 filter()를 사용하여서 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 중에서 짝수값의 제곱으로 새로운 리스트를 만드는 프로그램을 작성하라.

HINT 리스트를 생성하려면 map()을 사용한다. 리스트의 요소를 필터링하려면 filter()를 사용한다. lambda를 이용하여 람다식을 생성한다.

11 원을 나타내는 클래스 Circle에 +, >, < 연산자를 중복 정의해보자.

```
class Circle:
    def __init__(self, radius):
        self.__radius = radius
```

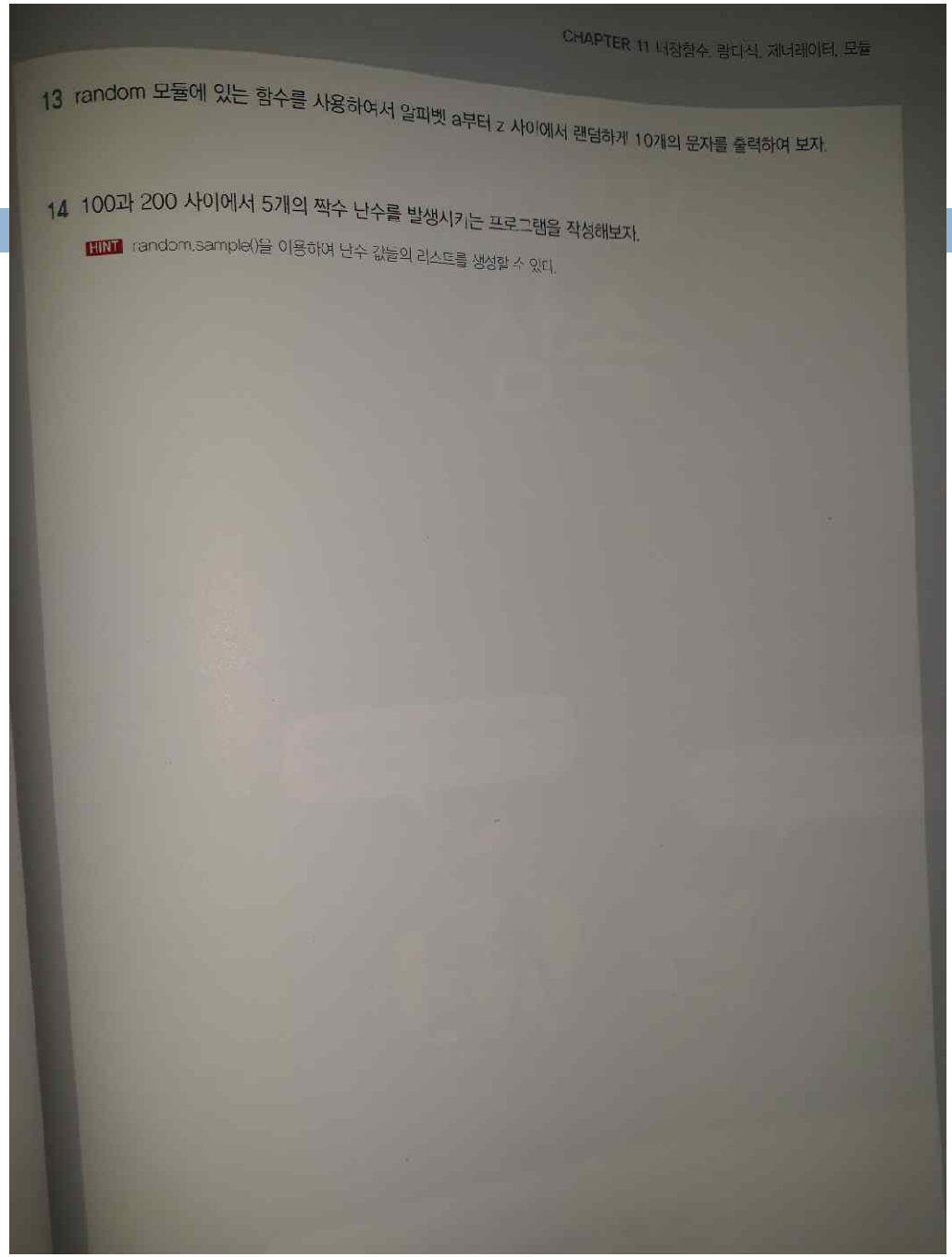
HINT 클래스에 __add__(), __gt__(), __lt__() 함수를 정의한다.

12 random 모듈에 있는 함수를 사용하여서 7개의 단어 중에서 랜덤하게 3개를 선택하여 출력하여 보자.

실행결과

```
학습 컴퓨터 마우스
```

프로그래밍문제-3



CHAPTER 11 내장함수, 람다식, 제너레이터, 모듈

13 random 모듈에 있는 함수를 사용하여서 알파벳 a부터 z 사이에서 랜덤하게 10개의 문자를 출력하여 보자.

14 100과 200 사이에서 5개의 짝수 난수를 발생시키는 프로그램을 작성해보자.

HINT random.sample()을 이용하여 난수 값들의 리스트를 생성할 수 있다.