09장 라이브러리

9.1 내장 함수

■ 외부 모듈과는 달리 import를 필요로 하지 않는다. 아무런 설정 없이 바로 사용할 수가 있다.

[ch09_library/ex01_internal.py]

```
#-*- coding:utf-8 -*-

# 내장함수
# abs(x)는 절대값을 돌려주는 함수
print(abs(3))
print(abs(-3))
print(abs(-1.2))


# all은 반복 가능한 자료형을 입력받아 그 값이 모두 참이면 True
print(all([1,2,-3])) # True
print(all([1,2,0]))  # False


# any는 하나라도 참이 있을 경우 True, 모두 거짓일 경우 False
print(any([1,2,0])) # True
print(any([0,'']))  # False


# chr(i)는 i에 영문자 코드(아스키 코드)값을 입력으로 받아 그
# 코드에 해당하는 문자를 출력하는 함수이다.
print(chr(97)) # 'a'
print(chr(65)) # 'A'
print(chr(44032)) # '가',  유니코드 값(44032) = '가'


# dir은 객체가 자체적으로 가지고 있는 변수나 함수를 보여준다.
print(dir([1,2,3])) # 리스트 객체
# ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
'__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']

print(dir({'1':'a'})) # 딕셔러리 객체
# ['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get',
'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']


# divmod은 몫과 나머지를 튜플 형태로 리턴하는 함수이다.
print(divmod(7,3))
# (2, 1)
print(7/3)
print(int(7/3)) # 몫
print(7%3)      # 나머지


# enumerate 함수는 순서가 있는 자료형(리스트,튜플,문자열)을 입력을 받아
# 인덱스 값을 포함하는 enumerate 객체로 리턴한다.
```

```
i = 0
for name in ['body', 'foo', 'bar']:
    i += 1  # i = i + 1
    print(i-1, name)
'''
0 body
1 foo
2 bar
'''

for i, name in enumerate(['body', 'foo', 'bar']):
    print(i, name)
'''
0 body
1 foo
2 bar
'''


# eval(expression)은 실행 가능한 문자열을 입력으로 받아
# 문자열을 실행한 결과값으로 리턴하는 함수이다.
print("1+2")       # '1+2'
print(eval("1+2")) # '3'

print("'hi'+'a'")       # 'hi'+'a'
print(eval("'hi'+'a'")) # 'hia'

print('divmod(4,3)')       # divmod(4,3)
print(eval('divmod(4,3)')) # (1, 1)


# filter 함수는 첫번째 인수로 함수이름을, 두번째 인수로 그 함수에 차례로
# 들어갈 반복 가능한 자료형을 받는다.
def positive(l):  # l = [1,-2,-3,4,-5]
    result = []  # 리스트 result = [1,4]
    for i in l:
        if i>0:
            result.append(i)
    return result

print(positive([1,-2,-3,4,-5]))
# [1, 4]


def positive1(x):
    return x > 0  # True or False

print(list(filter(positive1, [1,-2,-3,4,-5])))
# [1, 4]

print(list(filter(lambda x: x>0, [1,-2,-3,4,-5]))) # lambda는 익명함수를 만드는 식.
# [1, 4]

# [과제] Q4 filter와 lambda를 사용하여 아래 리스트에서 홀수를 모두 제거해 보자.
# [1,-2,3,-5,8,-3] -> [-2,8]
print(list(filter(lambda x: x%2==0, [1,-2,3,-5,8,-3])))
# [-2, 8]


# hex(x)는 정수값을 입력받아 16진수(hexadecimal)로 변환하여 리턴하는 함수
print(hex(234))
# 0xea
print(hex(16))
# 0x10
print(hex(3))
```

```
# 0x3


# id(object)는 객체를 입력받아 객체의 고유 주소값을 리턴하는 함수이다.
a = 3
print(id(3))
# 140709019300752
print(id(a))
# 140709019300752


# input() 사용자 입력을 받는 함수이다.
#a = input("값을 입력하세요. ")
#print(a)


# int()는 숫자 형태의 문자열이나 소수점이 있는 숫자 등을 정수 형태로
# 리턴하는 함수이다.
print(int('3'))  # 문자열('3') -> 숫자(3)
print(int(3.4))  # 실수형(3.4) -> 정수형(3)


# isinstance()
class Person:
    pass
a = Person()  # 객체를 생성하여 변수 a에 저장
print(isinstance(a, Person))
# True
b = 3
print(isinstance(b, Person))
# False


# len(s)은 입력값 s의 길이를 리턴하는 함수이다.
print(len("Python"))
# 6
print(len([1,2,3]))
# 3
print(len([1,'a']))
# 2


# list(s)는 반복 가능한 자료형 s를 입력받아 리스트로 만들어 리턴하는 함수
print(list("Python"))  # 문자형 -> 리스터형
# ['P', 'y', 't', 'h', 'o', 'n']


# map(f,iterable)은 함수(f)와 반복가능한(iterable) 자료형을 입력으로 받는다.
def two_times(numberList):  # [1,2,3,4]
    result = []  # [2,4,9,16]
    for number in numberList:
        result.append(number*2)
    return result

result = two_times([1,2,3,4])
print(result)
# [2, 4, 6, 8]


def two_times1(x):
    return x*2

print(list(map(two_times1, [1,2,3,4])))
# [2, 4, 6, 8]
```

```
print(list(map(lambda x:x*2, [1,2,3,4])))
# [2, 4, 6, 8]


print(max([1,2,3]))
# 3
print(min([1,2,3]))
# 1

# oct는 8진수로 리턴하는 함수이다.
print(oct(34))
# 0o42
print(oct(8))
# 0o10


# open
f = open("binary_file", 'wb')


# ord(c)는 문자의 아스키 코드값을 리턴하는 함수이다.
print(ord('a'))  # 97
print(chr(97))   # 'a'


# pow(x,y)는 x의 y제곱한 결과값을 리턴하는 함수이다.
print(pow(2,4))
# 16


# range([start], stop [, step])는 for문과 함께 자주 사용되는 함수이다.
print(list(range(5))) # 0~4
# [0, 1, 2, 3, 4]

print(list(range(5,10)))  # 5~9
# [5, 6, 7, 8, 9]

print(list(range(1,10,2)))  # 1~9, step=2
# [1, 3, 5, 7, 9]


# round는 숫자를 입력받아 반올림해 주는 함수이다.
print(round(4.6))
# 5
print(round(4.2))
# 4
print(round(5.678, 2))
# 5.68


# sorted(iterable) 함수는 입력값을 정렬한 후 그 결과를 리턴한다.
# sorted() vs. list.sort()
# sorted(): 정렬후에 값을 리스트로 리턴한다.
# list.sort(): 리스트를 정렬만 한다.
a = [3,1,2]
print(sorted(a))
# [1, 2, 3]
print(a.sort())
# None
print(a)
# [1, 2, 3]

print(sorted("zero"))  # 문자열('zero') -> 리스트
# ['e', 'o', 'r', 'z']
```

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


# tuple(iterable)은 반복가능한 자료형을 입력받아 튜플형으로 변환한다.
print(tuple("abc"))
# ('a', 'b', 'c')
print(tuple([1,2,3]))
# (1, 2, 3)


# type(object)은 입력값의 자료형이 무엇인지 알려주는 함수이다.
print(type("abc"))
# <class 'str'>
print(type([]))
# <class 'list'>
print(type(3))
# <class 'int'>
f = open("binary_file", "wb")
print(type(f))
# <class '_io.BufferedWriter'>


# zip(iterable)은 동일한 개수로 이루어진 자료형을 묶어 주는 역할을 한다.
print(list(zip([1,2,3], [4,5,6,7])))
# [(1, 4), (2, 5), (3, 6)]
print(list(zip([1,2,3],[4,5,6],[7,8,9])))
# [(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

9.2 외장 함수

- 외장 함수는 전 세계의 파이썬 사용자들이 만든 유용한 프로그램들을 모아 놓은 것이 바로 파이썬 라이브러리이다.

[ch09_library/ex02_external.py]

```
#-*- coding: utf-8 -*-

# 외장 함수
# sys 모듈은 파이썬 인터프리터가 제공하는 변수들과 함수들을 직접 제어할 수 있게 해주는 모듈이다.
# 명령 행에서 인수 전달하기 - sys.argv
import sys

if len(sys.argv) == 4:
    print(sys.argv)
    print(sys.argv[0])
    # sys.exit()   # 강제로 스크립트 종료하기
    print(sys.argv[1])
else:
    print("사용법: *.py arg1 arg2 arg3")


'''
['C:\\dev\\workspace\\python\\foundation\\ch09_library\\ex02_external.py', 'you', 'need', 'python']
C:\dev\workspace\python\foundation\ch09_library\ex02_external.py
'''

# 자신이 만든 모듈 불러와 사용하기 - sys.path
import sys
print(sys.path)
'''
```

```
['C:\\dev\\workspace\\python\\foundation\\ch09_library',
'C:\\dev\\workspace\\python\\foundation\\ch09_library',
'C:\\dev\\Anaconda3\\DLLs', 'C:\\dev\\Anaconda3\\lib',
'C:\\dev\\Anaconda3', 'C:\\dev\\Anaconda3\\lib\\site-packages',
'C:\\dev\\Anaconda3\\lib\\site-packages\\win32',
'C:\\dev\\Anaconda3\\lib\\site-packages\\win32\\lib',
'C:\\dev\\Anaconda3\\lib\\site-packages\\Pythonwin',
'C:\\dev\\Anaconda3\\python37.zip']
'''
sys.path.append("c:/doit/Mymod")
print(sys.path)

#from mod1 import add, sub
#print(add(3,4))
#print(sub(4,3))


# pickle은 객체의 형태를 그대로 유지하면서 파일에 저장하고 불러올 수 있게 하는 모듈이다.
import pickle
f = open("test.txt", 'wb')
data = {1:'python', 2:'you need'}  # 딕셔너리 객체
pickle.dump(data, f)
f.close()

# 원래 있던 딕셔너리 객체 상태 그대로 불어오기
import pickle
f = open("test.txt", 'rb')
data = pickle.load(f)
print(data)
# {1: 'python', 2: 'you need'}


# OS 모듈은 환경 변수나 디렉토리, 파일 등의 OS 자원을 제어할 수 있게 해주는 모듈이다.
# 내 시스템의 환경 변수값을 알고 싶을 때 - os.environ
import os
print(os.environ)
print(os.environ['JAVA_HOME'])
# C:\Program Files\Java\jdk1.8.0_111

# 디렉터리 위치 변경하기 - os.chdir
os.chdir("c:\windows")

# 디렉터리 위치 리턴받기 - os.getcwd
print(os.getcwd())
# c:\windows

# 시스템 명령어 호출하기 - os.sysm
os.system("python c:\doit\Mymod\mod1.py")


# 실행한 시스템 명령어의 결과값 리턴받기
# os.popen은 시스템 명령어를 실행시킨 결과값을 읽기 모드 형태의
# 파일 객체로 리턴한다.
f = os.popen("python c:\doit\Mymod\mod1.py")
result = f.read()
print(result)


# shutil은 파일을 복사해 주는 파이썬 모듈이다.
import shutil
shutil.copy("c:\\doit\\Mymod\\mod1.py", "c:\\doit\\Mymod\\copied.txt")


# glob은 특정 디렉토리에 있는 파일 이름 모두를 알아야 할때 사용한다.
import glob
```

```
print(glob.glob("c:\\Windows\\t*"))
# ['c:\\Windows\\TAPI', 'c:\\Windows\\Tasks', 'c:\\Windows\\Temp', 'c:\\Windows\\TextInput',
'c:\\Windows\\tracing', 'c:\\Windows\\twain_32', 'c:\\Windows\\twain_32.dll']


# tempfile은 파일을 임시로 만들어서 사용할 때 유용한 모듈이다.
import tempfile
filename = tempfile.mktemp()
print(filename)
# C:\Users\admin\AppData\Local\Temp\tmpjubwfdmk


# random은 난수를 발생시키는 모듈이다.
import random
print(random.random())
print(random.randint(1,10))  # 1~10사이의 정수중에서 난수값을 리턴

import random
data = [1,2,3,4,5]
random.shuffle(data)
print(data)
# [2, 1, 5, 3, 4]


# webbrowser는 자신의 시스템에서 사용하는 기본 웹브라우저가 자동으로
# 실행되게 하는 모듈이다.
import webbrowser
webbrowser.open("http://www.naver.com")
```

9.3 문자열 다루기

■ str은 파이썬에 기본적으로 내장돼 있는 클래스이다.

[ch09_library/ex03_str.py]

```
# str(object)는 문자열 형태로 객체를 변환하여 리턴하는 함수이다.
print(str(3))  # 3 -> '3'
print(int('3'))  # '3' -> 3
print(str('hi').upper())  # 'hi' -> 'HI'

print(dir(str))
# ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

# capitalize(): 첫 문자를 대문자로, 나머지 문자를 소문자로 바꿔준다.
print("PYTHON".capitalize())  # Python
print("python is powerful".capitalize())  # Python is powerful

# count(keyword,[start,[end]]): keyword가 몇 번이나 포함돼 있는지 알려준다.
print("python is powerful".count('p'))  # 2
print("python is powerful".count('p', 5))  # [5:]으로 슬라이싱하고 count한 결과와 동일
print("python is powerful".count('p', 0, -1))  # [0:-1]으로 슬라이싱하고 count한 결과와 동일

# encode([encoding,[errors]]): encode()를 거치면 인코딩이 있는 바이너리로 변환된다.
print('가나다'.encode('cp949'))  # b'\xb0\xa1\xb3\xaa\xb4\xd9'
```

```
print('가나다'.encode('utf-8'))  # b'\xea\xb0\x80\xeb\x82\x98\xeb\x8b\xa4'

# endswith(postfix,[start,[end]): postfix로 문자열이 끝나면 True를 반환하고 그 밖의 경우에는 False로 반환한다.
print("python is powerful".endswith('ful'))  # True
print("python is powerful".endswith('ful', 5))  # True
```

9.4 날짜 다루기

9.4.1 시간(time) 모듈

■ time 모듈의 함수는 다음과 같다.

함수	내용
time.time()	1970년 1월 1일 자정 이후로 누적된 초를 local 단위로 반환한다.
time.sleep(secs)	현재 동작 중인 프로세스를 주어진 초만큼 정지시킨다.
time.gmtime([secs])	입력된 초를 변환해, UTC 기준의 struct_time 시퀀스 객체로 반환한다.
time.localtime([sec])	입력된 초를 변환해 지방표준시 기준의 struct_time 시퀀스 객체를 반환한다.
time.asctime([t])	struct_time 시퀀스 객체를 인자로 받아 'Sun Mar 15 18:49:28 2009'와 같은 형태로 반환한다.
time.mktime(t)	지방표준시인 struct_time 시퀀스 객체를 인자로 받아 time()과 같은 누적된 초를 반환한다.

[ch09_library/ex04_time.py]

```
# time 모듈은 시간과 관련되어 있다.
# time.time()은 UTC를 이용하여 현재 시간을 실수 형태로 리턴한다.
# UTC(Universal Time Code)는 1970년 1월1일 0시0분0초를 기준으로 지난시간을 초 단위로 리턴한다
import time

print(time.time())
print(time.gmtime())    #UTC 기준의 현재 시간
print(time.localtime()) #시스템 기준의 현재 시간
t = time.gmtime(1234567890)  #타임스탬프를 struct_time 시퀀스 객체로 변환
print(t)
print(t.tm_mon)
print(t.tm_hour)
print(time.asctime(t))
'''
time.struct_time(tm_year=2019, tm_mon=11, tm_mday=29, tm_hour=3, tm_min=44, tm_sec=7, tm_wday=4, tm_yday=333,
tm_isdst=0)
time.struct_time(tm_year=2019, tm_mon=11, tm_mday=29, tm_hour=12, tm_min=44, tm_sec=7, tm_wday=4, tm_yday=333,
tm_isdst=0)
time.struct_time(tm_year=2009, tm_mon=2, tm_mday=13, tm_hour=23, tm_min=31, tm_sec=30, tm_wday=4, tm_yday=44,
tm_isdst=0)
2
23
Fri Feb 13 23:31:30 2009
'''

# time.sleep 함수는 일정한 시간 간격을 줄 수 있다.
import time
for i in range(10):
    print(i)
    time.sleep(1)
```

9.4.2 날짜시간(datetime) 모듈

■ 날짜, 시간 연산을 위해 파이썬에서는 날짜시간 모듈에 관련 클래스들이 이미 구현돼 있다.

클래스	내용
class datetime.date	일반적으로 사용되는 그레고리안 달력(Gregorian Calendar)의 년,월,일을 나타낸다.
class datetime.time	시간을 시,분,초,마이크로 초,시간대(Time zone)로 나타낸다.
class datetime.datetime	date 클래스와 time 클래스의 조합으로 년,월,일,시,초,마이크로 초,시간대 정보를 나타낸다.
class datetime.timedelta	두 날짜 혹은 시간 사이의 기간을 표현한다.

[ch09_library/ex04_datetime.py]

```
import datetime
import time

print(datetime.date(2009,5,5))  # 2009-05-05
print(datetime.date.today())  # 2019-11-29
d = datetime.date.today()
print(d.year)  # 2019
print(d.month) # 11
print(d.day)   # 29

from datetime import time
print(time(7))  # 07:00:00
print(time(8,14,20,3000))  # 08:14:20.003000


# calendar는 파이썬에서 달력을 볼 수 있게 해주는 모듈이다.
import calendar
print(calendar.calendar(2015))
'''
                                  2015

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

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

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

      October                 November                December
Mo Tu We Th Fr Sa Su    Mo Tu We Th Fr Sa Su    Mo Tu We Th Fr Sa Su
          1  2  3  4                       1        1  2  3  4  5  6
 5  6  7  8  9 10 11     2  3  4  5  6  7  8     7  8  9 10 11 12 13
12 13 14 15 16 17 18     9 10 11 12 13 14 15    14 15 16 17 18 19 20
19 20 21 22 23 24 25    16 17 18 19 20 21 22    21 22 23 24 25 26 27
```

```
26 27 28 29 30 31         23 24 25 26 27 28 29      28 29 30 31
                          30
'''
```

9.5 숫자 다루기

9.5.1 math 모듈

- math 모듈은 연산을 위한 합계, 최대값과 같은 내장 함수와 수치 연산, 지수/로그 연산, 삼각함수 연산에 필요한 함수와 관련 상수를 모아 놓은 수학 모듈이다.

[ch09_library/ex05_math.py]

```
l = list(range(0,10))
print(sum(l))
print(sum(l,100))   #누적 합계가 100부터 시작
print(max(l))
print(min(l))
print(pow(2,10))    #2의 10 제곱
print(divmod(10,7)) #몫과 나머지 반환
print(round(152.394))   #소수 첫째 자리에서 반올림
print(round(152.394,1)) #소수 둘째 자리에서 반올림
```

9.5.2 random 모듈

- 파이썬에서는 임의의 정수, 실수를 생성하거나, 시퀀스 객체 중 임의의 값을 선택하는 등의 연산을 위해 랜덤 모듈을 제공한다.

[ch09_library/ex05_random.py]

```
# 임의의 실수 생성
import random
print(random.random())  # 함수를 호출할 때마다 다른 값이 반환된다.
print(random.random())
print(random.uniform(3,4))  # 입력받은 두 값 사이의 float 값을 임의로 생성한다.

# 임의의 정수 생성
# random.randrange(20)은 0~20 사이의 수 중 중복을 허용한 임의의 정수를 생성한다.
r = [random.randrange(20) for i in range(10)]
print(r)
#[3, 11, 18, 13, 3, 18, 5, 1, 0, 2]

# random.sample(은 중복을 허용하지 않는다.
r = random.sample(range(20), 10)
print(r)
# [10, 12, 1, 8, 7, 11, 14, 9, 5, 15]

# 0~20 사이의 수 중 3의 배수만 출력한다.
r = [random.randrange(0, 20, 3) for i in range(5)]
print(r)
# [12, 0, 12, 6, 18]

# 시퀀스 객체 관련 연산
l = list(range(10))
print(l)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
c = [random.choice(l) for i in range(3)]  # 중복 선택 가능
print(c)
# [6, 6, 9]
c = random.sample(l,3)  # 중복을 허용하지 않는다.
print(c)

print(l)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
random.shuffle(l)  # 시퀀스 객체를 임의로 섞는다.
print(l)
# [9, 6, 5, 0, 2, 4, 3, 8, 1, 7]

# 원본 객체를 변경하지 않고 원본 객체가 임의로 섞인 또 다른 객체를 얻는다.
l = list(range(10))
s = random.sample(l, 10)
print(s)
print(l)
```

9.6 파일시스템 다루기

9.6.1 CSV 파일

(1) 기본 파이썬

■ 기본 파이썬으로 CSV 파일을 읽고 처리하고 작성하는 방법에 대해 알아본다.

[ch09_library/ex06_csv_simple_parsing_and_write.ipynb]

```
#!/usr/bin/env python3
# sys 모듈을 임포트한다. 파이썬에 기본으로 내장되어 있는 sys 모듈은
# 명령 줄에서 추가적으로 입력된 인수를 스크립트로 넘겨준다.
import sys

input_file = "supplier_data.csv"
output_file = "output_file.csv"

# input_file을 filereader라는 파일 객체로 열어주는 with문이다.
# open() 함수에서 'r'은 읽기 모드를 할당한다.
with open(input_file, 'r', newline='') as filereader:
          with open(output_file, 'w', newline='') as filewriter:
                    header = filereader.readline()
                    # filereader 객체의 readline() 함수를 사용하여 입력 파일의 첫 번째 행(헤더 행)을
                    # 문자열로 읽고 이를 header라는 변수에 할당한다.

                    header = header.strip()
                    # strip() 함수를 사용하여 header에 있는 문자열의 양끝에서 공백,
                    # 탭 및 개행문자(\n) 등을 제거한 뒤, header에 다시 할당한다.

                    header_list = header.split(',')
                    print(header_list)
                    filewriter.write(','.join(map(str,header_list))+'\n')
                    # header_list의 각 원소에 str() 함수를 적용하여 각 원소를 문자열로 변환한다.
                    # join() 함수는 header_list의 각 값 사이에 쉼표를 삽입하고 리스트를 문자열로 변환한다.

                    for row in filereader:
                              row = row.strip()
                              row_list = row.split(',')
                              print(row_list)
                              filewriter.write(','.join(map(str,row_list))+'\n')
```

(2) 팬더스

- 리스트, 딕셔너리, 튜플과 마찬가지로 데이터프레임(Data Frame)도 데이터를 저장하는 한 방식이다.

[ch09_library/ex06_csv_pandas.ipynb]

```
#!/usr/bin/env python3
import sys
import pandas as pd

input_file = "supplier_data.csv"
output_file = "output_file.csv"

# data_frame = pd.read_csv(input_file)
data_frame = pd.read_csv(input_file, error_bad_lines=False)
# error_bad_lines=False 옵션은 문제되는 줄을 생략한다.
print(type(data_frame))

print(data_frame)
data_frame.to_csv(output_file)
```

[꿀팁] 더러운 데이터

- 값이 종종 누락되어 있거나 입력한 사람의 실수로 데이터가 잘못 입력되었거나 센서의 오작동에 의해서 값이 잘못 기록되어 있는 데이터를 의미한다.
- 어떤 경우에 사람들은 의도적으로 잘못된 데이터를 기록하기도 한다. 그러한 의도된 오류 자체가 기록 방법의 하나이기 때문이다.
- 대부분의 분석 상황에서 이러한 더러운(dirty) 데이터를 깨끗하게 처리해야 한다. 이러한 데이터 정제 작업은 데이터 분석가의 직무 중에서 가장 많은 시간과 노력이 드는 작업이다.

9.6.2 엑셀 파일

(1) 엑셀 통합 문서 내부 살펴보기

- xlrd와 xlwt 패키지를 설치하여 사용한다.
- 엑셀 통합 문서는 CSV 파일과 달리 여러 개의 워크시트를 포함한다

```
// 엑셀 파일 처리를 위한 라이브러리를 설치한다.
(workspace-pycharm) D:\dev\workspace\workspace-pycharm\analysis>pip install --upgrade xlrd
(workspace-pycharm) D:\dev\workspace\workspace-pycharm\analysis>pip install --upgrade xlwt
(workspace-pycharm) D:\dev\workspace\workspace-pycharm\analysis>pip install --upgrade openpyxl
```

[ch09_library/ex06_excel1.ipynb]

```
#!/usr/bin/env python3

# xlrd 모듈에서 open_workbook() 함수를 임포트하여 엑셀 파일을 읽고 파싱하는데
# 사용할 수 있게 한다.
from xlrd import open_workbook
```

```
input_file = "sales_2013.xlsx"

# open_workbook() 함수를 사용하여 엑셀 입력 파일을 workbook이라는 이름의 객체로 연다.
workbook = open_workbook(input_file)

# workbook에 있는 워크시트 수를 출력한다.
print('Number of worksheets:', workbook.nsheets)

for worksheet in workbook.sheets():
          print("Worksheet name:", worksheet.name, "\tRows:", \
                              worksheet.nrows, "\tColumns:", worksheet.ncols)
```

(2) 단일 워크시트 처리

- 팬더스에는 엑셀 파일을 읽고 쓰는데 특화되어 있는 명령어들이 있다.
- 다음은 엑셀 파일 파싱에 팬더스를 사용하는 예제이다.

[ch09_library/ex06_excel2.ipynb]

```
#!/usr/bin/env python3
import pandas as pd

input_file = "sales_2013.xlsx"
output_file = "ex05_excel2.xlsx"

data_frame = pd.read_excel(input_file, sheet_name='january_2013')

writer = pd.ExcelWriter(output_file)
data_frame.to_excel(writer, sheet_name='jan_13_output', index=False)
writer.save()
```

(3) 특정 집합의 값을 포함하는 행의 필터링

- 기본 파이썬을 사용하여 Purchase Date 열의 값이 특정 집합(01/24/2013 및 01/31/2013)에 포함되는 행을 필터링한다.

[ch09_library/ex06_excel3.ipynb]

```
#!/usr/bin/env python3
import pandas as pd

input_file = "sales_2013.xlsx"
output_file = "ex05_excel3.xlsx"

data_frame = pd.read_excel(input_file, 'january_2013', index_col=None)

important_dates = ['01/24/2013', '01/31/2013']
data_frame_value_in_set = data_frame[data_frame['Purchase Date'].isin(important_dates)]

writer = pd.ExcelWriter(output_file)
data_frame_value_in_set.to_excel(writer, sheet_name='jan_13_ouput', index=False)
writer.save()
```

~~9.6.3 JSON 파일~~

- JSON(JavaScript Object Notation)은 웹브라우저와 다른 애플리케이션이 HTTP 요청으로 데이터를 보낼 때 널리 사용하는 표준 파일 형식 중 하나이다.
- JSON 은 CSV 같은 표 형식의 텍스트보다 좀더 유연한 데이터 형식이다.
- 아래는 JSON 데이터의 예이다.

```
[{"a": 1, "b": 2, "c": 3},
 {"a": 4, "b": 5, "c": 6},
 {"a": 7, "b": 8, "c": 9}]
```

[ch09_library/ex06_json.ipynb]

```
#-*- coding:utf-8 -*-

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

obj = """
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 30, "pets": ["Zeus", "Zuko"]},
              {"name": "Katie", "age": 38,
               "pets": ["Sixes", "Stache", "Cisco"]}]
}
"""


# json.loads는 JSON 문자열을 파이썬 형태(딕셔너리)로 변환한다.
import json
result = json.loads(obj)
print(result)
# {'name': 'Wes', 'places_lived': ['United States', 'Spain', 'Germany'], 'pet': None, 'siblings': [{'name':
'Scott', 'age': 30, 'pets': ['Zeus', 'Zuko']}, {'name': 'Katie', 'age': 38, 'pets': ['Sixes', 'Stache',
'Cisco']}]}


siblings = pd.DataFrame(result['siblings'], columns=['name', 'age'])
print(siblings)
'''
0  Scott   30
1  Katie   38
'''

# json.dumps는 파이썬 객체를 JSON 형태로 변환한다.
asjson = json.dumps(result)



# 별다른 옵션이 주어지지 않았을 경우 pandas.read_json은 JSON 배열에 담긴 각 객체를
# 테이블의 로우로 간주한다.
data = pd.read_json('example.json')
print(data)
'''
   a  b  c
0  1  2  3
1  4  5  6
```

```
2  7  8  9
'''


# pandas의 데이터를 JSON으로 저장하는 한 가지 방법은 Series나 DataFrame의 to_json 함수를 이용하는 것이다.
print(data.to_json())  # Series
print(data.to_json(orient='records'))  # DataFrame
'''
{"a":{"0":1,"1":4,"2":7},"b":{"0":2,"1":5,"2":8},"c":{"0":3,"1":6,"2":9}}
[{"a":1,"b":2,"c":3},{"a":4,"b":5,"c":6},{"a":7,"b":8,"c":9}]
'''
```

~~9.6.4 XML 파일~~

(1) XML 문서 생성하기

[ch09_library/ex06_xml_create.py]

```
from xml.etree.ElementTree import Element, SubElement, dump

note = Element("note")
note.attrib["date"] = "20120104"

to = Element("to")
to.text = "Tove"
note.append(to)

SubElement(note, "from").text = "Jani"
SubElement(note, "heading").text = "Reminder"
SubElement(note, "body").text = "Don't forget me this weekend"


# index 함수는 정렬된 형태의 xml으로 보여준다.
def indent(elem, level=0):
    i = "\n" + level*" "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + " "
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
        for elem in elem:
            indent(elem, level+1)
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
    else:
        if level and (not elem.tail or not elem.tail.strip()):
            elem.tail = i

indent(note)
dump(note)
'''
<note date="20120104">
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend</body>
</note>
'''


# 파일에 쓰기
```

```
from xml.etree.ElementTree import ElementTree
ElementTree(note).write("note.xml")
```

(2) XML 문서 파싱하기

```
[ch09_library/ex06_xml_parsing.py]

from xml.etree.ElementTree import parse

tree = parse("note.xml")
note = tree.getroot()

# 애트리뷰트 값 읽기
print(note.get("date"))
print(note.get("foo", "default"))
print(note.keys())
print(note.items())
'''
20120104
default
['date']
[('date', '20120104')]
'''

# xml 태그 접근하기
from_tag = note.find("from")
from_tags = note.findall("from")
from_text = note.findtext("from")
```

[과제] XML 처리

■ ElementTree를 이용하여 다음 XML 문서를 작성하고 파일에 저장해 보자.

```
<blog date="20151231">
 <subject>Why python?</subject>
 <author>Eric</author>
 <content>Life is too short. You need Python!</content>
</blog>
```

[정답]

```
from xml.etree.ElementTree import Element, SubElement, ElementTree, dump

blog = Element("blog")
blog.attrib["date"] = "20151231"

SubElement(blog, "subject").text = "Why python?"
SubElement(blog, "author").text = "Eric"
SubElement(blog, "content").text = "Life is too short. You need Python!"

# index 함수는 정렬된 형태의 xml으로 보여준다.
def indent(elem, level=0):
    i = "\n" + level*" "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + " "
```

```
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
        for elem in elem:
            indent(elem, level+1)
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
    else:
        if level and (not elem.tail or not elem.tail.strip()):
            elem.tail = i

indent(blog)
dump(blog)
'''
<blog date="20151231">
 <subject>Why python?</subject>
 <author>Eric</author>
 <content>Life is too short. You need Python!</content>
</blog>
'''

ElementTree(blog).write("blog.xml")
```

9.7 데이터베이스 다루기

9.7.1 SQLite3

- SQLite3는 디스크 기반의 가벼운 데이터베이스 라이브러리이다.
- SQL문 수행: 기본적으로 Cursor, execute() 메서드는 SQL문을 입력받아 수행한다.
- 레코드 조회: execute() 메서드를 이용해 SELECT문을 수행하면, Cursor 객체를 조회된 Row 객체에 대한 이터레이터처럼 사용할 수 있다.
- 트랜잭션 처리: Connection.commit()을 호출하면 수행한 SQL 구문을 데이터베이스에 반영할 수 있다.

[ch09_library/ex07_sqlite3.py]

```
#-*- coding: utf-8 -*-

# sqlite3 연결
import sqlite3

con = sqlite3.connect("test.db")
#con = sqlite3.connect(":memory:")  # 메모리상에 db 파일을 만든다.


# sql문 수행
cur = con.cursor()
cur.execute("create table phonebook(name text, phonenum text);")
cur.execute("insert into phonebook values('test2', '010-1234-5678');")


# 레코드 조회
cur.execute("select * from phonebook;")
print(cur.fetchall())


# 트랜잭션 처리
cur.execute("insert into phonebook values('derick', '010-1234-5678');")
con.commit()
```

```
cur.execute("select * from phonebook;")
print(cur.fetchall())
```

~~9.7.2 Oracle~~

[ch09_library/ex07_oracle.py]

```
#-*- coding: utf-8 -*-

# oracle 연결
# cx_Oracle 설치: c:\> pip install cx_Oracle --upgrade
import cx_Oracle

connection = cx_Oracle.connect('SCOTT/TIGER@XE')
cursor = connection.cursor()
querystring = "SELECT * FROM EMP"
cursor.execute(querystring)

print(cursor.fetchall())

cursor.close()
connection.close()
```

~~9.7.3 MySQL~~

[ch09_library/ex07_mysql.py]

```
#-*- coding: utf-8 -*-

# MySQL 연결
# PyMySQL 설치:  c:\> pip install PyMySQL --upgrade
import pymysql

conn = pymysql.connect(host='localhost', user='zerock', password='zerock',
                       db='book_ex', charset='utf8')

curs = conn.cursor()

sql = "select * from tbl_board"
curs.execute(sql)

rows = curs.fetchall()
print(rows)

conn.close()
```

9.8 운영체제 다루기

9.8.1 threading 모듈

- 작업(task)은 처리되어야 할 일로 여러 작업들이 묶어 한 스레드로 구성된다.
- 스레드(thread)란 하나의 프로세스 내에서 진행되는 하나의 실행 단위를 뜻하며, 하나의 프로세스에서 여러 실행 단위가 실행되는 것을 멀티스레드라고 한다.

■ 파이썬에서는 threading이라는 멀티스레드 기능을 지원하는 모듈을 제공한다.

[ch09_library/ex08_thread.py]

```
#-*- coding:utf-8 -*-

# 다음 예제 코드는 threading의 Thread와 Lock 객체를 이용해 각각 버그를 수정하는 시뮬레이션
# 프로그램이다. 전역 변수로 총 버그 개수(10)를 정의해 놓고, 생성된 3개의 developer 스레드가
# 각각 0.1초에 하나씩 버그를 해결해 나가게 된다. 버그 개수가 0이 되어 모든 스레드가 종료되면
# 각자 몇 개씩 버그를 해결했는지 출력한다.

# 파이썬에서는 threading이라는 멀티스레드 기능을 지원하는 모듈을 제공한다.
from threading import Thread, Lock
import time

count = 10
# 2개 이상의 스레드가 동시에 수행되는 경우, 스레드 간에 프로세스 공간의 메모리를 공유한다.
# 이때 공유된 메모리에 서로 영향을 받지 않게 하기 위해 동기화 객체(Lock)를 사용한다.
# Lock 객체는 locked와 unlocked의 2가지 상태를 가지며, acquire()와 release()의 두 가지 함수만을
# 제공한다.
lock = Lock()

# 스레드를 사용하려면 일반적으로 threading.Thread를 상속받은 클래스 객체를 생성해 사용한다.
class developer(Thread):
    # 생성자를 재정의하는 경우, 반드시 Thread.__init__()을 수행해야 한다.
    def __init__(self, name):
        Thread.__init__(self)
        self.name = name
        self.fixed = 0
    # Thread.run()은 스레드의 주요 동작을 정의한다.
    def run(self):
        global count
        while 1:
            lock.acquire()  # locked 상태로 바뀐다.
            if count > 0:
                count -= 1
                lock.release()  # unlocked 상태로 바뀐다.
                self.fixed += 1
                time.sleep(0.1)
            else:
                lock.release()
                break

dev_list = []
for name in ['Shin', 'Woo', "Choi"]:
    dev = developer(name)
    dev_list.append(dev)
    # Thread.start()는 스레드를 시작한다.
    dev.start()

for dev in dev_list:
    # Thread.join()는 스레드가 종료되기를 기다린다.
    dev.join()
    print(dev.name, 'fixed', dev.fixed)

'''
Shin fixed 4
Woo fixed 3
Choi fixed 3
'''
```

[ch09_library/ex08_multiprocessing.py]

```
#-*- coding:utf-8 -*-

# 다음 예제 코드는 여러 개의 프로세스를 다루는 방법도 threading 모듈과 거의 비슷한데, 이번에는
# multiprocessing 모듈을 사용한다.
# threading이 multiprocessing으로 바뀌고, 전체 버그 개수를 관리하던 전역 변수가 Value라는
# 공유 객체로 바뀐 점말고는 끄게 바뀐 점이 없다.

from multiprocessing import Process, Lock, Value
import time

def run(name, l, c):
    print(name, 'process is created.')
    fixed = 0
    while 1:
        l.acquire()
        if c.value > 0:
            c.value -= 1
            l.release()
            fixed += 1
            time.sleep(0.1)
        else:
            l.release()
            print(name, 'fixed', fixed, 'defects')
            break

if __name__=='__main__':
    lock = Lock()
    count = Value('i', 10)
    
    dev_list = []
    for name in ['Shin', 'Woo', 'Choi']:
        dev = Process(target=run, args=(name, lock, count))
        dev_list.append(dev)
        dev.start()
    
    for dev in dev_list:
        dev.join()
    print('All processs are finished.')
```