PYTHON 파일 이해하기

Moon Yong Joon

OS 모듈

OS 모듈 기본 조회

OS 이름 및 environ 내의 path 관리를 조회

```
import os

print(os.name)
print(os.path.abspath('.'))
print( os.environ['PATH'])
```

```
nt
C:\Users\06411
C:\Program Files\Anaconda3\Library\bin;C:\Program F
da3\Scripts;C:\Program Files\Anaconda3\Library\bin;
C:\Program Files (x86)\Wizvera\Delfino;C:\ProgramDa
 Files\Intel\iCLS Client\;C:\Windows\system32;C:\Wi
C:\Program Files\Intel\Intel(R) Management Engine C
T;C:\Program Files (x86)\Intel\Intel(R) Management
 Components\IPT;C:\Windows\Softcamp\SDS;C:\Windows\
CC-64\bin;C:\Program Files\Mercurial;C:\Python27;C:
s (x86)\pythonxy\SciTE-3.5.1-4;C:\Program Files (x8
ipts;C:\Program Files\Anaconda3\Library\bin;C:\go\b
ercurial-3.2.3;C:\mysql-5.6.22-winx64\bin;C:\Progra
es\Java\jdk1.8.0_25\bin;
```

OS 내의 명령어 실행

OS 명령어를 실행하고 결과 받기

```
f = os.popen("dir")
print(f.read())
```

```
 C 드라이브의 볼륨: SYSTEM
 볼륨 일련 번호: 4A5F-4E72

 C:\Users\06411 디렉터리

2016-12-19  오후 01:25    <DIR>          .
2016-12-19  오후 01:25    <DIR>          ..
2015-07-13  오후 04:25    <DIR>          .android
2016-08-08  오후 01:04                 0 .clojurepyhist
```

5

Directory 조회 및 이동

OS 모듈 : 현재 directory 조회

OS 내의 디렉토리 정보를 조회하거나 디렉토리 내의 정보를 조회하는 방법

```
import os
print(os.getcwd())
print(os.name)
print(os.path.curdir)
print(os.path.dirname(os.getcwd()))
```

```
C:\Users\06411\Documents
nt
.
C:\Users\06411
```

OS 모듈 : directory 이동

OS 내의 디렉토리간 이동을 처리

```
import os
os.chdir("..")
print(os.getcwd())
os.mkdir("test1")
os.chdir("./test1")
print(os.getcwd())
os.chdir("..")
print(os.getcwd())
```

```
C:\Users\06411
C:\Users\06411\test1
C:\Users\06411
```

os.path

windows 디렉토리 작성

윈도우는 \\ 로 디렉토리를 구분해야 하며 raw string 처리시 \로 처리

```
import os
names = os.listdir('.')
print(names)

names = os.listdir(r"C:\Users\06411\Downloads")
print(names)

names = os.listdir("C:\\Users\\06411\\Downloads")
print(names)
```

```
['.android', '.clojurepyhist', '.designer', '.docker', '
 '.jupyter', '.matplotlib', '.oracle_jre_usage', '.pylin
plication Data', 'bank_deposit_20160926 (1).ipynb', 'bin
esktop', 'dict.pkl', 'Documents', 'Downloads', 'erlide_d
```

디렉토리/파일 구조분리

Os.path 내의 파일과 디렉토리 구조 분리해 보기.
Basename(파일명), dirname(디렉토리)

```
import os
path = '/Users/beazley/Data/data.csv'
# Get the last component of the path
print(os.path.split(path))
print(os.path.basename(path))
print(os.path.dirname(path))
#확장자 분리
print(os.path.splitext(path))

#server의 os가 다른 경우에는 path 생성하기를 사용해서 접근해야 함
print(os.path.join('tmp', 'data', os.path.basename(path)))
```

```
data.csv
/Users/beazley/Data
('/Users/beazley/Data', 'data.csv')
('/Users/beazley/Data/data', '.csv')
tmp\data\data.csv
```

파일이나 디렉토리 유무확인

Os.path 내의 디렉토리(dirname, isdir), size(getsize), 파일(isfile, existes), path에 대한 조정 처리

```
import os

print(os.path.dirname('C:\\Users\\06411'))
print(os.path.isdir(r'c:\python27'))

print(os.path.getsize('C:\\Users\\06411\\foo.txt'))
print(os.path.exists('C:\\Users\\06411\\foo.txt'))
print(os.path.isfile(r'c:\python27\python.exe'))

print(os.path.split(r'c:\temp\test\python\hello.exe'))
print(os.path.join(r'c:\temp', 'hello.exe'))

mixed ='C:\\\\temp\\\\public/files/index.html'
print(os.path.normpath(mixed))
```

```
C:\Users
True
7
True
True
('c:\\temp\\test\\python', 'hello.exe')
c:\temp\hello.exe
C:\temp\public\files\index.html
```

Normpath는 unix나 windows가 혼용될 경우 하나로 조정

파일명 검색

glob와 fnmach 모듈을 이용해서 파일명 검색

```
import glob
pyfiles = glob.glob('./*.txt')
print(pyfiles)

from fnmatch import fnmatch
pyfiles = [name for name in os.listdir('.') if fnmatch(name, '*.txt')]
print(pyfiles)
```

```
['.\\erlide_debug.txt', '.\\foo.txt', '.\\test.txt', '.\\untitled.txt', '.\\untitled1.txt']
['erlide_debug.txt', 'foo.txt', 'test.txt', 'untitled.txt', 'untitled1.txt']
```

SYS 모듈

파이썬 정보 조회

SYS 모듈 : 현재 파이썬 정보

현재 사용 기기의 Python 정보

```
import sys
print(sys.version)

print(sys.version_info)
```

```
2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2]
sys.version_info(major=2, minor=7, micro=6, releaselevel='final', serial=0)
```

SYS 모듈 :displayhook

sys.dispalyhook에 함수를 연결해서 sys 출력에 대한 표시 방법을 변경 할 수 있음

```
import sys
#arg 정의
v = (10,10)
# 함수정의
def add(v) :
    print(v[0]+ v[1])

#함수를 연결
sys.displayhook = add
#arg 실행하면 실제 연결된 함수 실행
v
```

```
20
```

sys 표준 입출력 처리

sys.sysin/Input 함수 사용

ide 창에서 입력을 받아 처리하기. 입력은 모두 string으로 처리됨

```
import sys
while True:
    # 3.0버전부터
    s = input('Enter something :')
    # s = raw_input('Enter something : ')
    if s == 'quit':
        break
    print('Length of the string is', len(s))
    print('string ', s)

print('Done')
```

```
Enter something :Hello World
Length of the string is 11
string  Hello World
Enter something :quit
Done
```

```
C:₩Users₩06411>python
Python 2.7.10 (default, May 23 2015, 09:40:32) [MS
n32
Type "help", "copyright", "credits" or "license" f
>>> import sys
>>> sys.stdin.readline()
qqqq
'qqqq₩n'
>>>
```

sys.sysout

ide 창에서 text/binary출력을 처리하기. 출력은 모두 string으로 처리됨

```
import sys
for i in (sys.stdin, sys.stdout, sys.stderr):
    print(i)

sys.stdout.write("Another way to do it!\n")

sys.stdout.write(b'Hello\n')
```

```
<_io.TextIOWrapper name='<stdin>' mode='r' encoding='cp949'>
<ipykernel.iostream.OutStream object at 0x0000000004DA1BE0>
<ipykernel.iostream.OutStream object at 0x0000000004DA18D0>
Another way to do it!
Hello
Hello
```

sys arguments

sys.argv 호출 방법

실행창에서 python 모듈에 argv를 주고 실행


```
!python sys_argv.py arg1 arg2
```

```
옵션 개수: 2

< 옵션 목록 >
sys.argv[0] = 'sys_argv.py'
sys.argv[1] = 'arg1'
sys.argv[2] = 'arg2'
```

sys.argv 처리 모듈

sys.argv 를 받아 처리하는 모듈 생성

```
%%writefile sys_argv.py
import sys


if len(sys.argv) is 1:
    print("arg 없이 스크립트를 실행")

print("옵션 개수: %d" % (len(sys.argv) - 1))

print("\n< 옵션 목록 >")

for i in range(len(sys.argv)):
    print("sys.argv[%d] = '%s'" % (i, sys.argv[i]))
```

```
Writing sys_argv.py
```

sys arg로 파일명 처리

sys.argv 처리 모듈

sys.argv 를 받아 파일을 읽고 출력하는 모듈 생성

```
%%writefile sys_argv.py
import sys

infile =""
if len(sys.argv) is 1:
    print("arg 없이 스크립트를 실행")

print("옵션 개수: %d" % (len(sys.argv) - 1))

print("\n< 파일처리 >")

infile = sys.argv[1]
a = open(infile, 'rt')
for i in a :
    print(i, end="")
```

```
Overwriting sys_argv.py
```

실행창에서 파이썬 모듈 실행

실행창에서 sys_argv.py를 실행하고 입력 파일에 대한 정보를 주고 실행

```
%%writefile foo.txt
A
B
C
```

```
Overwriting foo.txt
```

```
!python sys_argv.py foo.txt
```

```
옵션 개수: 1

< 파일처리 >
A
B
C
```

PICKLE 모듈

Pickle 주요 함수

Python object serialization

Pickle은 파이선 객체를 bytes 타입으로 직렬화를 처리하는 모듈

```
import pickle
import pprint

data = [ { 'a':'A', 'b':2, 'c':3.0 } ]
print('DATA:')
pprint.pprint(data)

data_string = pickle.dumps(data)
print('PICKLE:', data_string)
data_ = pickle.loads(data_string)
print('Python:', data_)
```

```
DATA:
[{'a': 'A', 'b': 2, 'c': 3.0}]
PICKLE: b'\x80\x03]q\x00}q\x01(X\x01\x00\x00\x00bq\x
G@\x08\x00\x00\x00\x00\x00\x00ua.'
Python: [{'b': 2, 'a': 'A', 'c': 3.0}]
```

Pickle: load/dump 함수

Pickle에 데이터 저장 및 로드하는 함수

로드할때도 한번씩 호출해서 저장된 순서대로 호출해서 처리

```
print(help(pickle.load))
print(help(pickle.dump))
```

```
Help on function load in module pickle:

load(file)

None
Help on function dump in module pickle:

dump(obj, file, protocol=None)

None
```

list/dict/object 처리

Pickle : 문자열 저장 및 로딩

파이썬 문자열 타입을 저장하고 다시 로딩 후에 값 비교

```
import pickle
s = "Hello world"
fileObject = open("str.pkl",'wb')
pickle.dump(s,fileObject)
fileObject.close()

fileObject = open("str.pkl",'rb')
b = pickle.load(fileObject)
print(b)
print(s == b)
```

```
Hello world
True
```

Pickle : list 저장 및 로딩

파이썬 list 타입을 저장하고 다시 로딩 후에 값 비교

```
import pickle

a = ['test value','test value 2','test value 3']

file_Name = "testfile"

fileObject = open(file_Name,'wb')

# this writes the object a to the
pickle.dump(a,fileObject)

# here we close the fileObject
fileObject.close()

fileObject = open(file_Name,'rb')

b = pickle.load(fileObject)
print(b)
print(a == b)
```

```
['test value', 'test value 2', 'test value 3']
True
```

Pickle : dict 저장 및 로딩

파이썬 dict를 받고 저장 후 다시 로딩

```
import pickle
d = { 'a': [1,2,3], 'b': {'a':[1,2,3]}}

fileObject = open("dict.pkl",'wb')
pickle.dump(d,fileObject)
fileObject.close()

fileObject = open("dict.pkl",'rb')
b = pickle.load(fileObject)
print(b)
print(d == b)
```

```
{'b': {'a': [1, 2, 3]}, 'a': [1, 2, 3]}
True
```

Pickle : class 저장/로딩

사용자 정의 class를 만들고 인스턴스를 생성해서 저장 후 다시 로딩

```
class Fruits: pass

banana = Fruits()

banana.color = 'yellow'
banana.value = 30
apple = Fruits()
apple.color = 'red'
apple.value = 50

import pickle

filehandler = open("Fruits.data","wb")
pickle.dump(banana,filehandler)
pickle.dump(apple,filehandler)
filehandler.close()

file = open("Fruits.data",'rb')

object_banana = pickle.load(file)
object_apple = pickle.load(file)
file.close()

print(object_banana.color, object_banana.value)
print(object_apple.color, object_apple.value)
```

```
('yellow', 30)
('red', 50)
```

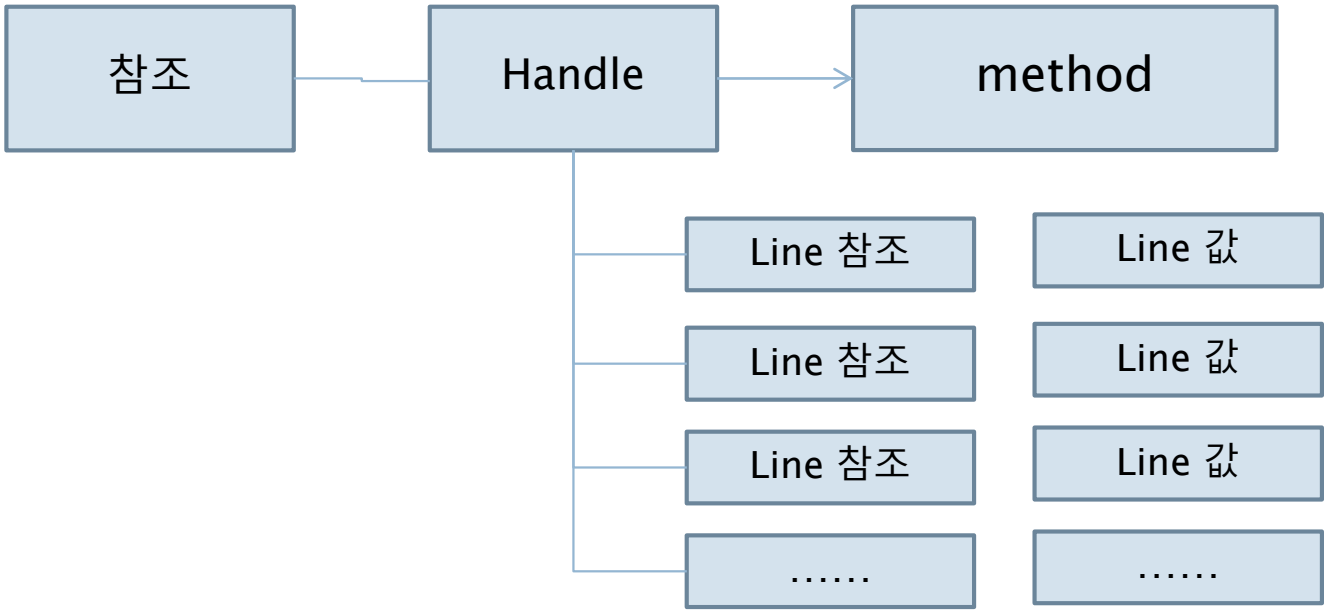
FILE 모듈

FILE 구조 이해하기

File 이해

File 은 Object

파일도 하나의 Object로 구현되어 있어 File 처리를 할 때 메소드를 이용하여 처리할 수 있도록 구성되어 있다. 파일은 라인이라는 요소들로 구성된 하나의 객체이므로 iterable 처리가 가능



File Object Variable

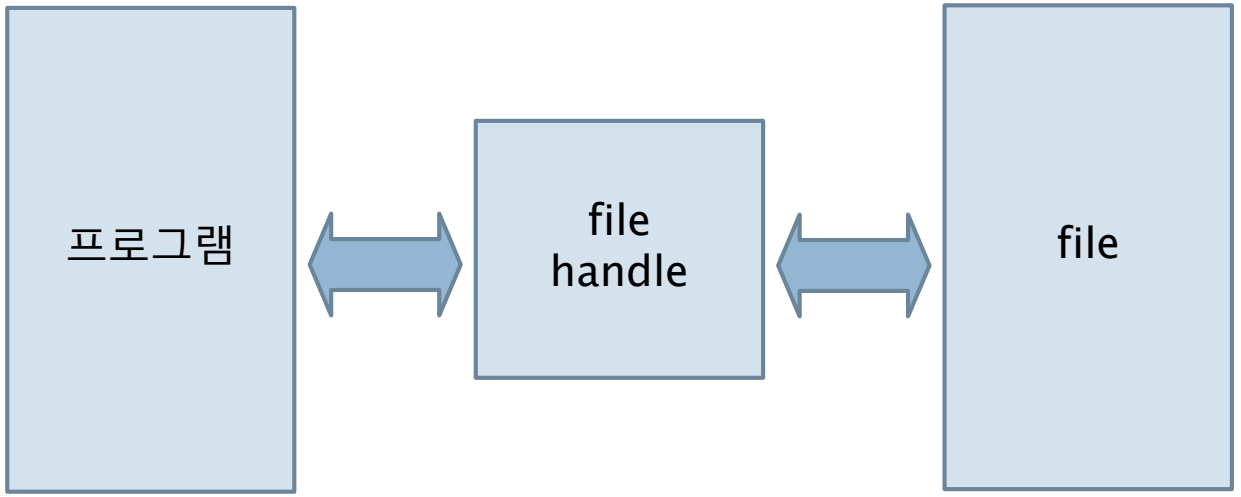
Method	설명
file.closed	bool indicating the current state of the file object.
file.encoding	The encoding that this file uses.
file.errors	The Unicode error handler used along with the encoding.
file.mode	The I/O mode for the file. If the file was created using the open() built-in function, this will be the value of the mode parameter.
file.name	If the file object was created using open(), the name of the file. Otherwise, some string that indicates the source of the file object, of the form <...>.
file.newlines	If Python was built with universal newlines enabled (the default) this read-only attribute exists, and for files opened in universal newline read mode it keeps track of the types of newlines encountered while reading the file.
file.softspace	oolean that indicates whether a space character needs to be printed before another value when using the print statement.

File 모드

파일열기모드	설명
r	읽기모드 - 파일을 읽기만 할 때 사용
r+	읽고쓰기모드 - 파일에 내용을 읽고 쓸 때 사용
a	추가모드 - 파일의 마지막에 새로운 내용을 추가 시킬 때 사용(쓰기전용)
a+	파일 끝에 추가(읽기도 가능)
w	쓰기모드 - 파일에 내용을 쓸 때 사용
w+	읽고 쓰기(기존 파일 삭제)
x	존재한 파일 없을 때만 파일 을 생성
t	텍스트모드 - 기본 텍스트
b	바이너리모드-바이너리로 처리
rb	이진 파일 읽기 전용
rb+	이진 파일 읽고 쓰기
wb+	이진 파일 읽고 쓰기(기존 파일 삭제)
ab+	이진 파일 끝에 추가(읽기도 가능)

File에서 handle object

실제 파일을 전달하는 것이 아니라 file handle를 전달해서 파일을 처리할 수 있도록 함



File 생성 및 닫기

파일을 open해서 바로 close해서 파일 생성

파일 열기 및 생성 : 파일객체 = **open**(파일이름, 파일열기모드)

파일 닫기 : 파일객체.close()

```
f = open("newfile.txt", 'w')
f.close()
```

```
%ls
```

```
1_hello_tensorflow.ipynb                  doctest.pyc
2_getting_started.ipynb                   foo.txt
3_mnist_from_scratch.ipynb                foo1.txt
C:\Users411\Downloads\line_plot_plus.pdf  image.jpg
LICENSE                                   line_plot_plus.pdf
Matplotlib_test1.ipynb                    mod.py
Mover_ch2.pyde                            mod.pyc
Untitled.ipynb                            mod_f.py
Untitled1.ipynb                           mod_f.pyc
Untitled2.ipynb                           newfile.txt
arraystore.nd                             test.txt
data.txt                                  test_1.ipynb
doctest.py                                understanding Python_20160815.ipynb
```

빈 file 생성

File은 iterable 객체

파일을 open 하면 iterable 객체인 handle을 제공

```
%%writefile test.txt
A
B
C
D
```

```
Writing test.txt
```

```
for i in open("test.txt",'rt') :
    print(i, end="")
```

```
A
B
C
D
```

```
a = open("test.txt",'rt')
print(next(a), end="")
print(next(a), end="")
print(next(a), end="")
print(next(a), end="")
print(next(a), end="")
```

```
A
B
C
D
```

```
---------------------------------------------
StopIteration                                Trac
<ipython-input-95-41ea0c415e03> in <module>()
      4 print(next(a), end="")
      5 print(next(a), end="")
----> 6 print(next(a), end="")

StopIteration:
```

File 읽기 메소드

File :read

파일을 open하고 read 메소드를 호출하면 전부 str 타입으로 생성

Method	설명
file.read([*size*])	Read at most *size* bytes from the file (less if the read hits EOF before obtaining *size* bytes).

```
%%writefile test.txt
A
B
C
D
```

```
Writing test.txt
```

```
f = open("test.txt",'rt')
a = f.read(5)
print(a)
print(a[:3])
```

```
A
B
C
A
B
```

```
f = open("test.txt",'rt')
print(f.read(3))
```

```
A
B
```

```
f = open("test.txt",'rt')
print(f.read())
```

```
A
B
C
D
```

File : readline – 한라인

파일을 다시 오픈하고 파일객체.readline()를 이용하여 파일을 읽기

```
]: # 읽기모드로  파일 읽기
   f = open("newfile.txt", 'r')

   # 파일에서 한 라인 읽기
   line = f.readline()
   print(line)

   f.close()
```

```
1 번째 줄입니다.
```

File : readline :라인변경 없애기

파일을 open하고 readline 메소드로 읽고 출력시 end=“”를 넣어

```
: %%writefile test.txt
A
B
C
D
```

```
Writing test.txt
```

```
f = open("test.txt",'rt')
print(f.readline(),end="")
print(f.readline(),end="")
print(f.readline(),end="")
print(f.readline(),end="")
print(f.readline(),end="")
```

```
A
B
C
D
```

File : for문을 이용해서 처리

파일이 iterable 특성을 이용해서 for문으로 읽기

```
f = open('file_read.txt','r')
for i in f :
    print(i)

f.close()
```

```
VIVAMUS mea Lesbia, atque amemus,

rumoresque senum severiorum

omnes unius aestimemus assis!

soles occidere et redire possunt:

nobis cum semel occidit breuis lux,

nox est perpetua una dormienda.

da mi basia mille, deinde centum,

dein mille altera, dein secunda centum,

deinde usque altera mille, deinde centum.

dein, cum milia multa fecerimus,

conturbabimus illa, ne sciamus,

aut ne quis malus inuidere possit,

cum tantum sciat esse basiorum.

(GAIUS VALERIUS CATULLUS)
```

File : readlines –여러 라인

파일을 다시 오픈하고 파일객체.readlines(), 파일객체.read()를 이용하여 파일을 읽기

```
f = open("newfile.txt", 'r')
data = f.read()

print(data)

f.close()
```

```
1 번째 줄입니다.
2 번째 줄입니다.
3 번째 줄입니다.
4 번째 줄입니다.
5 번째 줄입니다.
6 번째 줄입니다.
7 번째 줄입니다.
8 번째 줄입니다.
9 번째 줄입니다.
10 번째 줄입니다.
11번째 줄입니다.
12번째 줄입니다.
13번째 줄입니다.
14번째 줄입니다.
15번째 줄입니다.
16번째 줄입니다.
17번째 줄입니다.
18번째 줄입니다.
19번째 줄입니다.
20번째 줄입니다.
```

```
f = open("newfile.txt", 'r')
for i in f.readlines():
    print(i)

f.close()
```

```
1 번째 줄입니다.

2 번째 줄입니다.

3 번째 줄입니다.

4 번째 줄입니다.

5 번째 줄입니다.

6 번째 줄입니다.

7 번째 줄입니다.

8 번째 줄입니다.

9 번째 줄입니다.

10 번째 줄입니다.

11번째 줄입니다.

12번째 줄입니다.
```

File 위치 찾기 메소드

File :tell

파일을 open하고 readline 메소드를 호출하고 있을 경우 현재 파일의 위치를 조회

Method	설명
file.tell()	현재의 파일 포인터 위치를 돌려줌.

```
fobj_in = open("file_read.txt","r")


print(fobj_in.tell())
fobj_in.readline()
print(fobj_in.tell())
fobj_in.readline()
print(fobj_in.tell())
fobj_in.close()
```

```
0
34
62
```

File : seek

파일을 open하고 readline 메소드를 처리하다 seek 메소드를 만나면 파일의 위치를 변경해서 다시 처리

Method	설명
file.seek(*offset*[, *whence*])	파일의 위치 이동.(whence 가 없으면 처음에서 offset 번째로, 1 이면 현재에서 offset번째로, 2 이면 마지막에서 offset 번째로) – seek(n) : 파일의 n번째 바이트로 이동 – seek(n, 1) : 현재 위치에서 n바이트 이동(n이 양수이면 뒤쪽으로, 음수이면 앞쪽으로 이동) – seek(n, 2) : 맨 마지막에서 n바이트 이동(n은 보통 음수)

```
fobj_in = open("file_read.txt","r")


print(fobj_in.tell())
v= fobj_in.readline()
print(v)
print(fobj_in.tell())
fobj_in.readline()
print(fobj_in.tell())

fobj_in.seek(0)
print(fobj_in.readline())
fobj_in.close()
```

```
0
VIVAMUS mea Lesbia, atque amemus,

34
62
VIVAMUS mea Lesbia, atque amemus,
```

File 쓰기 메소드

File 생성: writelines

Method	설명
file.writelines(*sequence*)	리스트 안에 있는 문자열을 연속해서 출력함.

```
aaa ='''
this is 1 line
this is 2 line
this is 3 line
this is 4 line
'''
f = open(r'C:\test.txt', 'w+')
f.writelines(aaa)
f.close()
f = open(r'C:\test.txt', 'r+').read()
print(f)
```

```
this is 1 line
this is 2 line
this is 3 line
this is 4 line
```

File 생성: write

파일을 다시 오픈하고 파일객체.write()를 이용하여 파일에 쓰고 읽을 때 printg함수에 end=“”를 사용해야 라인이 붙여서 표현

```
# Write chunks of text data
text1 = "Hello World \n"
text2 = "Dahl Moon \n"
with open('somefile.txt', 'wt') as f:
    f.write(text1)
    f.write(text2)

with open('somefile.txt', 'rt') as f:
    for line in f :
        print(line,end="")
```

```
Hello World
Dahl Moon
```

File 생성: print

파일을 다시 오픈하고 print(string, end="", file=filename)를 이용하여 파일에 쓰고 읽을 때 printg함수에 end=""를 사용해야 라인이 붙여서 표현

```
# Write chunks of text data
text1 = "Hello World \n"
text2 = "Dahl Moon \n"
with open('printfile.txt', 'wt') as f:
    print(text1,end="",file=f)
    print(text2,end="",file=f)

with open('printfile.txt', 'rt') as f:
    for line in f :
        print(line,end="")
```

```
Hello World
Dahl Moon
```

File 추가(mode=a): wirte

파일객체 = **open**(파일이름, “a”)로 세팅하여 파일객체.write(), 단 “w”모드하면 파일이 초기화됨

```
f = open("newfile.txt", 'w')
#for문을 이용하여 10라인 추가
for i in range(1, 11):
    # 파일 라인에 출력
    line = "%d 번째 줄입니다.\n" % i
    f.write(line)


f.close()
# 기존 파일에 추가모드로 세팅
f = open("newfile.txt",'a')
for i in range(11, 21):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)


f.close()

f = open("newfile.txt", 'r')
print(f.read())
```

```
1 번째 줄입니다.
2 번째 줄입니다.
3 번째 줄입니다.
4 번째 줄입니다.
5 번째 줄입니다.
6 번째 줄입니다.
7 번째 줄입니다.
8 번째 줄입니다.
9 번째 줄입니다.
10 번째 줄입니다.
11번째 줄입니다.
12번째 줄입니다.
13번째 줄입니다.
14번째 줄입니다.
15번째 줄입니다.
16번째 줄입니다.
17번째 줄입니다.
18번째 줄입니다.
19번째 줄입니다.
20번째 줄입니다.
```

File : truncate

파일을 처리시 truncate를 만나면 파일을 삭제해 버림

```
%%writefile foo.txt
this is 1 line
this is 2 line
this is 3 line
this is 4 line
```

```
Overwriting foo.txt
```

```
f = open("foo.txt","rt")
print(f.read())
f.close()
f = open("foo.txt","wt+")
line = f.readline()
f.truncate(f.tell())
f.close()
f = open("foo.txt","rt")
print(f.read())
```

```
this is 1 line
this is 2 line
this is 3 line
this is 4 line
```

파일 읽고 파일 만들기

File 오픈 후 다른 file 만들기

읽는 파일과 쓸 파일을 open 해서 line별로 읽고 라인별로 파일에 write 하기

```
fobj_in = open("file_read.txt","r")
fobj_out = open("file-write.txt","w")
i = 1
for line in fobj_in:
    print(line.rstrip())
    fobj_out.write(str(i) + ": " + line)
    i = i + 1
fobj_in.close()
fobj_out.close()
print("write count", i)
```

```
VIVAMUS mea Lesbia, atque amemus,
rumoresque senum severiorum
omnes unius aestimemus assis!
soles occidere et redire possunt:
nobis cum semel occidit breuis lux,
nox est perpetua una dormienda.
da mi basia mille, deinde centum,
dein mille altera, dein secunda centum,
deinde usque altera mille, deinde centum.
dein, cum milia multa fecerimus,
conturbabimus illa, ne sciamus,
aut ne quis malus inuidere possit,
cum tantum sciat esse basiorum.
(GAIUS VALERIUS CATULLUS)
('write count', 15)
```

File 쓰고 일부 수정

파일을 “w+” mode로 오픈하여 write한 후에 임의의 위치를 찾아 다시 write 처리

```
fh = open('file_append.txt', 'w+')
fh.write('The colour brown')

# Go to the 8th byte in the file
fh.seek(11)
print(fh.read(5))
print(fh.tell())
fh.seek(11)
fh.write('green')
fh.seek(0)
content = fh.read()
print(content)
fh.close()
```

```
brown
16
The colour green
```

File 만들기

파일을 file_read.txt로 만들기

```
%%writefile file_read.txt
VIVAMUS mea Lesbia, atque amemus,
rumoresque senum severiorum
omnes unius aestimemus assis!
soles occidere et redire possunt:
nobis cum semel occidit breuis lux,
nox est perpetua una dormienda.
da mi basia mille, deinde centum,
dein mille altera, dein secunda centum,
deinde usque altera mille, deinde centum.
dein, cum milia multa fecerimus,
conturbabimus illa, ne sciamus,
aut ne quis malus inuidere possit,
cum tantum sciat esse basiorum.
(GAIUS VALERIUS CATULLUS)
```

```
Writing file_read.txt
```

Console 창과 연계 파일처리

raw_input 받고 file에 write

Console에서 입력 받을 것을 file에 저장

(3버전에서는 input으로 처리)

```
raw_input("?")

print "Opening the file..."
target = open("foo.txt", 'w')

print "Truncating the file.  Goodbye!"
target.truncate()

line1 = raw_input("line 1: ")
line2 = raw_input("line 2: ")
line3 = raw_input("line 3: ")

print "I'm going to write these to the file."

target.write(line1)
target.write("\n")
target.write(line2)
target.write("\n")
target.write(line3)
target.write("\n")

print "And finally, we close it."
target.close()
```

```
?foo.txt
Opening the file...
Truncating the file.  Goodbye!
line 1: Hello World
line 2: Neverland
line 3: Welcome
I'm going to write these to the file.
And finally, we close it.
```

처리 결과

Jupyter notebook cell 창에서 %load file명을 입력해서 실행하면 파일의 결과가 load 됨

```
# %load foo.txt
Hello World
Neverland
Welcome
```

FILE 처리하기

With문 사용하기

File 생성 및 닫기 – with 문

With문을 사용하면 file.close()를 사용하지 않아도 with문 내문에서 처리한 것이 완료되면 file이 자동으로 close 됨

```
f = open("withfile.txt", 'w')
f.write(" Hello  file World !!!!")
f.close()
f = open("withfile.txt", 'r')
print(f.read())
```

```
 Hello  file World !!!!
```

```
: with open(" withfile.txt", "w") as f:
    f.write(" Hello World !!!!")
```

```
: %ls
```

```
 withfile.txt                              doctest.pyc
1_hello_tensorflow.ipynb                   foo.txt
2_getting_started.ipynb                    foo1.txt
3_mnist_from_scratch.ipynb                 image.jpg
C:\Users411\Downloads\line_plot_plus.pdf   line_plot_plus.pdf
LICENSE                                    mod.py
```

임시 저장 구조 처리

StringIO

텍스트를 파일처리 처리하기 위해 사용

```
import io
s = io.StringIO()
s.write('Hello World\n')
print('This is a test', file=s)

# 전체 값 가져오기
print(s.getvalue())

#파일처리 read 하기
s = io.StringIO('Hello\nWorld\n')
print(s.read(4))
print(s.read())
```

```
Hello World
This is a test

Hell
o
World
```

BytesIO

binary 파일처리 처리하기 위해 사용

```
import io
s = io.BytesIO()
s.write(b'binary data')
print(s.getvalue())

#파일처리 read 하기
s = io.BytesIO(b'Hello\nWorld\n')
print(s.read(4))
print(s.read())
```

```
b'binary data'
b'Hell'
b'o\nWorld\n'
```

임시 파일 구조 사용하기

임시파일의 의미

실제 파일을 만들고 삭제해야 하지만 임시파일을 만들면 사용이 종료되면 자동으로 삭제

```
import os
import tempfile

print('Building a file name yourself:')
filename = './guess_my_name.%s.txt' % os.getpid()
temp = open(filename, 'w+b')
try:
    print('temp:', temp)
    print('temp.name:', temp.name)
finally:
    temp.close()
    # Clean up the temporary file yourself
    os.remove(filename)

print()
print('TemporaryFile:')
temp = tempfile.TemporaryFile()
try:
    print('temp:', temp)
    print('temp.name:', temp.name)
finally:
    # Automatically cleans up the file
    temp.close()
```

```
Building a file name yourself:
temp: <_io.BufferedRandom name='./guess_my_name.7688.txt'>
temp.name: ./guess_my_name.7688.txt

TemporaryFile:
temp: <tempfile._TemporaryFileWrapper object at 0x0000000000517B4A8>
temp.name: C:\Users\06411\AppData\Local\Temp\tmp77cvti22
```

임시파일을 생성해서 처리하기

tempfile 모듈을 이용해서 temp 파일로 데이터 처리하기

```
from tempfile import TemporaryFile

with TemporaryFile('w+t') as f:
    # Read/write to the file
    f.write('Hello World\n')
    f.write('Testing\n')

    # Seek back to beginning and read the data
    f.seek(0)
    data = f.read()
    print(data)
```

```
Hello World
Testing
```

임시파일에 이름 배정하기

tempfile 모듈에서 namedtemporaryfile를 이용해서 임시파일 이름을 배정

```
import tempfile

#임시파일 만들기
print(tempfile.mkstemp(suffix=".data", prefix="test", dir="C:\\Users\\06411\\AppData\\Local"))
#임시 디렉토리 가져오기
print(tempfile.gettempdir())
# 이름을 활용할 수 있는 임시파일 만들기
f = tempfile.NamedTemporaryFile(prefix='mytemp', suffix='.txt', dir=tempfile.gettempdir())
print(f.name)
```

```
(17, 'C:\\Users\\06411\\AppData\\Local\\test8lh0q1xq.data')
C:\Users\06411\AppData\Local\Temp
C:\Users\06411\AppData\Local\Temp\mytempkvntrno2.txt
```

File 존재 여부 확인

존재한 파일을 생성하지 않기

파일을 open할 때 mode를 x 로 열면 기존 존재하면 exception 처리

파일열기모드	설명
x	존재한 파일 없을 때만 파일 을 생성

```
with open('somefile.txt', 'xt') as f:
    f.write('Hello\n')
```

```
---------------------------------------------------------------
FileExistsError                           Traceback (most
<ipython-input-34-94cfc29fe00e> in <module>()
----> 1 with open('somefile.txt', 'xt') as f:
      2     f.write('Hello\n')

FileExistsError: [Errno 17] File exists: 'somefile.txt'
```

존재한 파일 점검

os 모듈의 path.exists 함수를 이용해서 점검해서 처리 가능

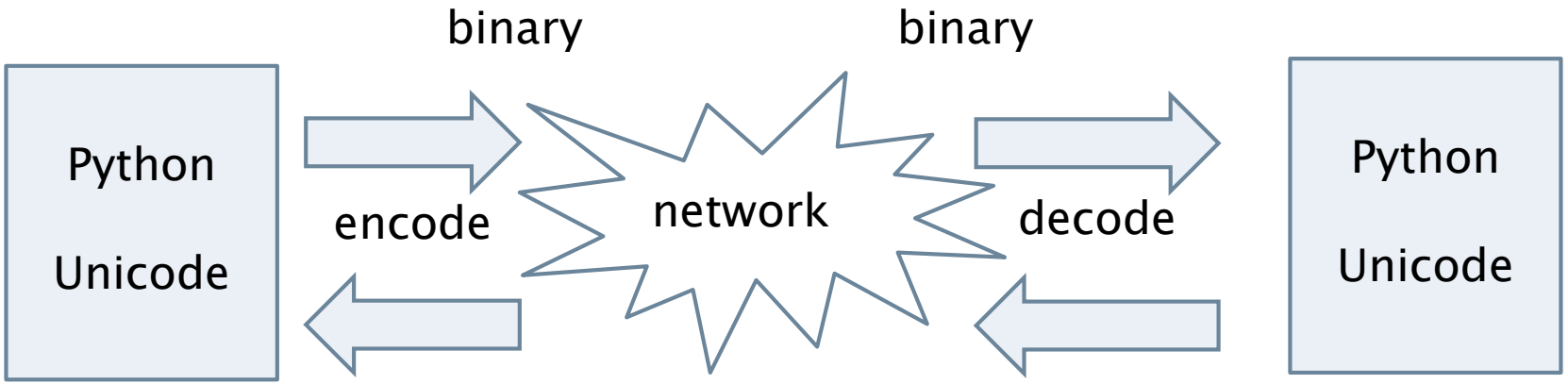
```
import os
if not os.path.exists('somefile'):
    with open('somefile', 'wt') as f:
        f.write('Hello\n')
else:
    print('File already exists!')
```

```
File already exists!
```

Binary 파일 처리

binary 변환이 필요 이유

파일이나 네트워크 연결에 저장된 바이너리 데이터를 처리하는 데 사용



인코딩 타입

한글과 영어에 대한 encoding 타입별 bytes 수

인코딩	영어	한글
EUC-KR	1 byte	2 bytes
UCS-2	2 bytes	2 bytes
UTF-8	1 byte	3 bytes
UTF-16	2 bytes	2 bytes
UTF-32	4 bytes	4 bytes

str <-> bytearray/bytes 변환

str(text).encode 메소드로 전환되지만
str.decode 메소드는 사람짐

```
# -*- coding: utf-8 -*-
'''
유니코드 --> 다른 CharacterSet : 인코딩(encode)
다른 CharacterSet --> 유니코드 : 디코딩(decode)
   decode 메소드가 str(bytestring, encoding='utf-8')
   로 대체
'''

import sys
sys.getdefaultencoding()
s= '파이썬'
print(type(s),s)
b = '파이썬'.encode()
print(type(b), b)

print(str(b, encoding="utf-8"))
```

```
<class 'str'> 파이썬
<class 'bytes'> b'\xed\x8c\x8c\xec\x9d\xb4\xec\x8d\xac'
파이썬
```

bytearray/bytes <-> str변환

byte.decode 메소드로 str로 전환이 가능

```
i = "hello world"
c = bytes(i, encoding="utf-8")
a = i.encode("utf-8")
print(a)
print(type(a))
b = a.decode("utf-8")
print(b)
print(type(b))

print(c == a)
print(a == b)
print(i == b)
```

```
b'hello world'
<class 'bytes'>
hello world
<class 'str'>
True
False
True
```

list 타입을 array 모듈로 처리

byte 파일을 생성 후에 읽기

```
import array
nums = array.array('i', [1, 2, 3, 4])
with open('data.bin','wb') as f:
    f.write(nums)


with open('data.bin', 'rb') as f:
    print(f.read())

b = array.array('i', [0, 0, 0, 0, 0, 0, 0, 0])
with open('data.bin', 'rb') as f:
    f.readinto(b)
print(b)
```

```
b'\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00\x04\x00\x00\x00'
array('i', [1, 2, 3, 4, 0, 0, 0, 0])
```

File byte 파일 추가하기

byte 파일을 생성 후에 읽기

```
# Write chunks of text data
text1 = b"Hello World \n"
text2 = b"Dahl Moon \n"
with open('printfile.bin', 'wb') as f:
    f.write(text1)
    f.write(text2)

with open('printfile.bin', 'rb') as f:
    for line in f :
        print(line)
```

```
b'Hello World \n'
b'Dahl Moon \n'
```

한글/영어 변환

문자열은 문자단위로 길이를 산정하나
bytearray/bytes는 byte 단위로 문자를 산정

```
i = "hello world"
print(type(i), len(i))
a = i.encode("utf-8")
print(type(a), len(a))
b = bytearray(i, encoding="utf-8")
print(type(b), len(b))

ih = "가나다라마"
print(type(i), len(i))
ah = ih.encode("utf-8")
print(ah)
print(type(ah), len(ah))
bh = bytearray(ih, encoding="utf-8")
print(bh)
print(type(bh), len(bh))
```

```
<class 'str'> 11
<class 'bytes'> 11
<class 'bytearray'> 11
<class 'str'> 11
b'\xea\xb0\x80\xeb\x82\x98\xeb\x8b\xa4\xeb\x9d\xbc\xeb\xa7\x88'
<class 'bytes'> 15
bytearray(b'\xea\xb0\x80\xeb\x82\x98\xeb\x8b\xa4\xeb\x9d\xbc\xeb\xa7\x88')
<class 'bytearray'> 15
```

UNICODEDATA 모듈

Unicode 한글문자

unicodedata :한글 letter

한글에 대한 문자의 기본 위치

```
#한글 letter 3131..318E
print(unicodedata.name("\u3141"))
print(unicodedata.lookup(unicodedata.name("\u3141")))

print(unicodedata.name("ㅁ"))
print(unicodedata.lookup(unicodedata.name("ㅁ")))
```

```
HANGUL LETTER MIEUM
ㅁ
HANGUL LETTER MIEUM
ㅁ
```

유니코드 정보
http://www.unicode.org/Public/8.0.0/ucd/

unicode normalization form

unicode normalization form 으로 equivalence있는 문자를 표시

NFD *Normalization Form Canonical Decomposition*	Characters are decomposed by canonical equivalence , and multiple combining characters are arranged in a specific order.
NFC *Normalization Form Canonical Composition*	Characters are decomposed and then recomposed by canonical equivalence.
NFKD *Normalization Form Compatibility Decomposition*	Characters are decomposed by compatibility, and mu ltiple combining characters are arranged in a specific order.
NFKC *Normalization Form Compatibility Composition*	Characters are decomposed by compatibility, then re composed by canonical equivalence.

unicodedata : normalization

unicode equivalence을 표시

```
import unicodedata
print(unicodedata.name("\u20A9"))
print(unicodedata.lookup(unicodedata.name("\u20A9")))

print(unicodedata.name("₩"))
print(unicodedata.lookup(unicodedata.name("₩")))

print(unicodedata.normalize('NFD', '가'))
print(unicodedata.normalize('NFC', '가'))

print(unicodedata.normalize('NFKC', '가'))
print(unicodedata.normalize('NFKD', '가'))
```

```
WON SIGN
₩
WON SIGN
₩
가
가
가
가
```

Unicode 한글 자모/음절

unicodedata:한글 자모/음절 구성

한글에 대한 자모(1100~11C2)와 음절(AC00 ~ D788)의 기본 위치

```
# ================================================

# Hangul_Syllable_Type=Leading_Jamo

1100..115F    ; L # Lo   [96] HANGUL CHOSEONG KIYEOK..HANGUL CHOSEONG FILLER
A960..A97C    ; L # Lo   [29] HANGUL CHOSEONG TIKEUT-MIEUM..HANGUL CHOSEONG SSANGYEORINHIEUH

# Total code points: 125

# ================================================

# Hangul_Syllable_Type=Vowel_Jamo

1160..11A7    ; V # Lo   [72] HANGUL JUNGSEONG FILLER..HANGUL JUNGSEONG O-YAE
D7B0..D7C6    ; V # Lo   [23] HANGUL JUNGSEONG O-YEO..HANGUL JUNGSEONG ARAEA-E

# Total code points: 95

# ================================================

# Hangul_Syllable_Type=Trailing_Jamo

11A8..11FF    ; T # Lo   [88] HANGUL JONGSEONG KIYEOK..HANGUL JONGSEONG SSANGNIEUN
D7CB..D7FB    ; T # Lo   [49] HANGUL JONGSEONG NIEUN-RIEUL..HANGUL JONGSEONG PHIEUPH-THIEUTH

# Total code points: 137
```

unicodedata :한글자모/음절

한글에 대한 자모와 음절의 기본 위치

```
import unicodedata
#한글 자모 1100 ~ 11C2
print(unicodedata.name("\u1100"))
print(unicodedata.lookup(unicodedata.name("ㄱ")))
print(unicodedata.name("\u11C2"))
print(unicodedata.lookup(unicodedata.name("\u11C2")))

#한글 음절 AC00 ~ D788
print(unicodedata.name("\uAC00"))
print(unicodedata.lookup(unicodedata.name("\uAC00")))

print(unicodedata.name("\uD788"))
print(unicodedata.lookup(unicodedata.name("\uD788")))
```

```
HANGUL CHOSEONG KIYEOK
ㄱ
HANGUL JONGSEONG HIEUH
ㅎ
HANGUL SYLLABLE GA
가
HANGUL SYLLABLE HI
히
```

BINASCII / BINHEX 모듈

uuencode 변환

binascii :uuencode

UUCP 메일 시스템을 통한 전송을 위해 바이너리 데이터를 인코딩하기 위해 유닉스 프로그램 uuencode에서 시작된 바이너리 – 텍스트 인코딩 형식

```
import binascii
s = "abc"
sb = s.encode("ascii")
print('a', hex(ord('a')))
print('b', hex(ord('b')))
print('c', hex(ord('c')))

# bytes를 uuencode
sauu = binascii.b2a_uu(sb)
print(sauu)

# uuencode를 bytes
suu = binascii.a2b_uu(sauu)
print(suu)
```

```
a 0x61
b 0x62
c 0x63
b'#86)C\n'
b'abc'
```

hex 변환

binascii : hexlify/unhexlify

hexlify/unhexlify로 바이트를 바이너리 hex로 변환

```
import binascii
s = "abc"
sb = s.encode("utf-8")
print('a', hex(ord('a')))
print('b', hex(ord('b')))
print('c', hex(ord('c')))

print("bytes : ",sb)
sbin = binascii.hexlify(sb)
print("binary : ",sbin)
print("bytes : ",binascii.unhexlify(sbin))
```

```
a 0x61
b 0x62
c 0x63
bytes :  b'abc'
binary :  b'616263'
bytes :  b'abc'
```

binascii : hex

b2a_hex/a2b_hex로 바이트를 바이너리 hex로 변환

```
import binascii
s = "abc"
sb = s.encode("ascii")
print('a', hex(ord('a')))
print('b', hex(ord('b')))
print('c', hex(ord('c')))
# bytes -> hex
bah = binascii.b2a_hex(sb)
print(bah)

# hex --> bytes
bab = binascii.a2b_hex(bah)
print(bab)
```

```
a 0x61
b 0x62
c 0x63
b'616263'
b'abc'
```

binhex 모듈 : hex

파일을 받아서 파일로 변환 값을 처리하는 binhex/hexbin함수

```
import binhex
s = "abc"
sb = s.encode("ascii")
print('a', hex(ord('a')))
print('b', hex(ord('b')))
print('c', hex(ord('c')))

fw = open("fw.bin","rb")
print(fw.read())
fw.close()

binhex.binhex("fw.bin","fr.bin")

f = open("fr.bin","rb")
print(f.read())
f.close()

binhex.hexbin("fr.bin", "fr.hex")
f = open("fr.hex","rb")
print(f.read())
f.close()
```

```
a 0x61
b 0x62
c 0x63
b'abc'
b'(This file must be converted with BinHex 4.0)\r\r:"QCh,Q*TEJ"849K82j!%!*!&!`#3""6kB@*MRGB!!!:\n'
b'abc'
```

base64 변환

binascii : base64

b2a_base64/a2b_base64로 바이트를 바이너리 hex로 변환

```
import binascii
s = "abc"
sb = s.encode("ascii")
print('a', hex(ord('a')))
print('b', hex(ord('b')))
print('c', hex(ord('c')))
# bytes -> base64
bab = binascii.b2a_base64(sb)
print(bab)

# base64 -> bytes
bab = binascii.a2b_base64(bab)
print(bab)
```

```
a 0x61
b 0x62
c 0x63
b'YWJj\n'
b'abc'
```

BASE64 모듈

Binary 파일 처리(base64)

base64 색인

2**6 기준으로 base64색인표

Base64 색인표

값	문자	값	문자	값	문자	값	문자
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

base64 색인 예시

바이너리 데이터를 2**6 단위로 바이트 해석

Text content	M								a								n							
ASCII	77								97								110							
Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Index	19								22								5							
Base64-Encoded	T								W								F							

```
import base64

print(ord('M'))
print(bin(ord('M')))
print(ord('a'))
print(bin(ord('a')))
print(ord('n'))
print(bin(ord('n')))
print(base64.b64encode(b"Man"))
```

```
77
0b1001101
97
0b1100001
110
0b1101110
b'TWFu'
```

Base64: encode/decode

바이트 문자열 및 바이트 배열과 같은 바이트 지향 데이터에서만 사용

```
import base64
text = b"hello world"
print(text)

encoded_text = base64.encodebytes(text)
print(encoded_text)

decoded_text = base64.decodebytes(encoded_text)
print(decoded_text)

print(text == decoded_text)
```

```
b'hello world'
b'aGVsbG8gd29ybGQ=\n'
b'hello world'
True
```

Base64: b64encode

바이너리 데이터를 ASCII 문자로 인코딩하고 다시 바이너리 데이터로 디코딩하는 기능을 제공

```
s= 'complex string: ñáéíóúÑ'
#bytes로 encode
i = bytes(s, "utf-8")
#bytes를 binary bytes로 변환
a = base64.b64encode(i)

# binary bytes를 bytes로 변환
d = base64.b64decode(a)

# binary bytes를 str로 변환
b = base64.b64decode(a).decode("utf-8", "ignore")

print(s)
print(i)
print(type(a),a)
print(type(d),d)
print(type(b),b)
```

```
complex string: ñáéíóúÑ
b'complex string: \xc3\xb1\xc3\xa1\xc3\xa9\xc3\xad\xc3\xb3\xc3\xba\xc3\x91'
<class 'bytes'> b'Y29tcGxleCBzdHJpbmc6IMOxw6HDqcOtw7PDusOR'
<class 'bytes'> b'complex string: \xc3\xb1\xc3\xa1\xc3\xa9\xc3\xad\xc3\xb3\xc3\xba\xc3\x91'
<class 'str'> complex string: ñáéíóúÑ
```

Base64로 한글

한글을 처리하려면 일단 bytes타입으로 전환하고 변환후 다시 문자열에서 decoding 처리해야 함

```
import base64
h = '한글'
s = h.encode('utf-8')
# Encode as Base64
a = base64.b64encode(s)
print(a)

# Decode from Base64
print(base64.b64decode(a))
print(str(base64.b64decode(a), encoding='utf-8'))
```

```
b'7ZWc6riA'
b'\xed\x95\x9c\xea\xb8\x80'
한글
```

```
import base64
s = "가을"
print(s)
#bytes로 변환
se = s.encode("utf-8")
# b64로 변환
sebe = base64.b64encode(se)
# 다시 bytes로 변환
sebd = base64.b64decode(sebe)
#unicode로 변환
print(sebd.decode("utf-8"))
```

```
가을
가을
```

Base16/32 변환

Base16 변환

2**4 단위씩 데이터 변환

base16

```
s = b'hello'
print(len(s))
print(ord('h'))
print(bin(ord('h')))
h = base64.b16encode(s)
print(h)
print(len(h))

print(base64.b16decode(h))
```

```
5
104
0b1101000
b'68656C6C6F'
10
b'hello'
```

Base32 변환

2**5단위씩 데이터 변환

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
0	A	9	J	18	S	27	3
1	B	10	K	19	T	28	4
2	C	11	L	20	U	29	5
3	D	12	M	21	V	30	6
4	E	13	N	22	W	31	7
5	F	14	O	23	X		
6	G	15	P	24	Y		
7	H	16	Q	25	Z		
8	I	17	R	26	2	*pad*	=

```
s = b'hello'
print(len(s))
print(ord('h'))
print(bin(ord('h')))
h = base64.b32encode(s)
print(h)
print(len(h))

print(base64.b32decode(h))
```

```
5
104
0b1101000
b'NBSWY3DP'
8
b'hello'
```

STRUCT 모듈

STRUCT

Byte order/ format

struct 모듈

이 모듈은 Python 바이트 객체로 표현 된 Python 값과 C 구조체 사이의 변환을 수행

압축 된 바이너리 데이터로 바이트 해석

Endian 처리 이해

Byte Order, Size, Alignment

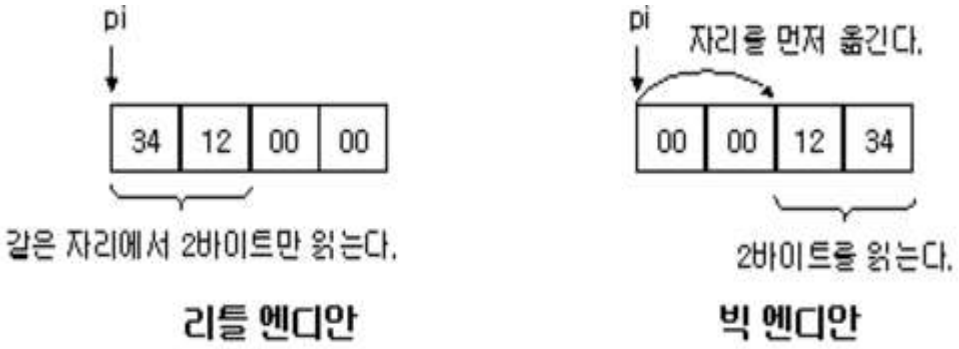
데이터를 패킹 및 언 패킹 할 때 예상되는 레이아웃을 지정하는 데 사용되는 메커니즘

Character	Byte order	Size	Alignment
@	native	native	native
=	native	standard	none
<	little-endian	standard	none
>	big-endian	standard	none
!	network (= big-endian)	standard	none

엔디언(Endianness)

엔디언(Endianness)은 컴퓨터의 메모리와 같은 1차원의 공간에 여러 개의 연속된 대상을 배열하는 방법을 뜻하며, 바이트를 배열하는 방법을 특히 바이트 순서(Byte order)

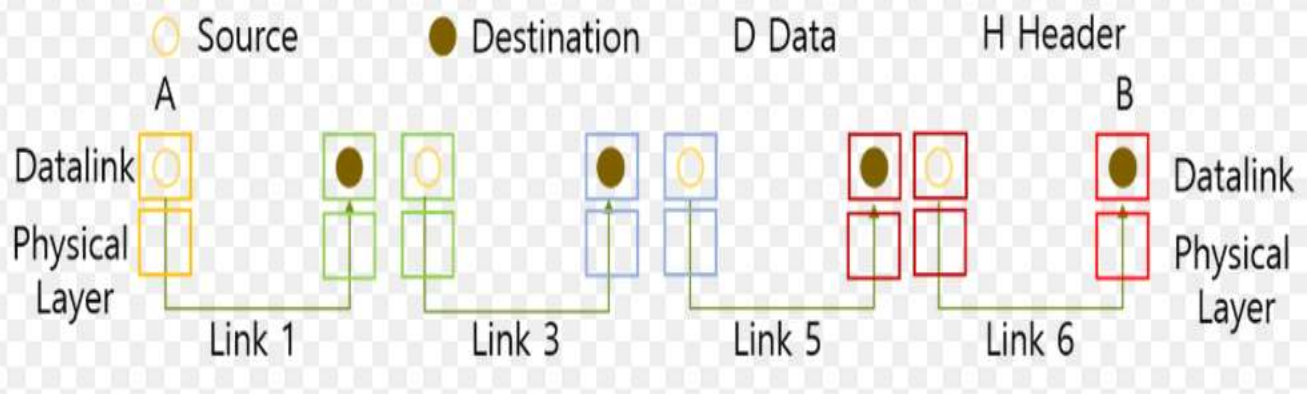
- 큰 단위가 앞에 나오는 **빅 엔디언**(Big-endian)
- 작은 단위가 앞에 나오는 **리틀 엔디언**(Little-endian)
- 두 경우에 속하지 않거나 둘을 모두 지원하는 것을 **미들 엔디언**(Middle-endian)



엔디언은 언제 필요한가?

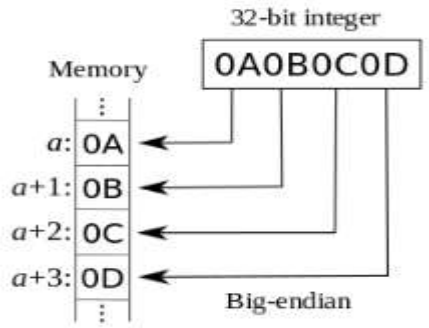
Endian 에 의한 byte order 은 해당 시스템의 CPU 내부 처리, 그러나 네트워크 프로그래밍은 이기종 간의 통신을 염두에 Endian 에 신경을 써주지 않으면 전혀 엉뚱한 결과를 가지고 오게 된다.

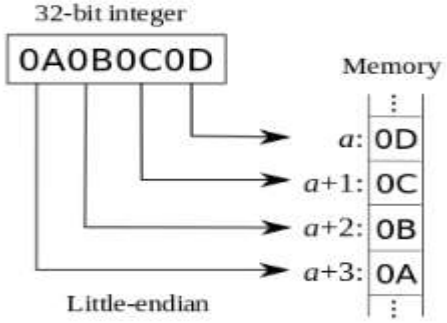
네트워크 처리시 이기종간 연결



struct pack/unpack 처리

sturct 모듈로 숫자(파이썬 타입)를 메모리 bytes(c-타입) 체계로 변환






```
from struct import *

#little endian
print(pack('h', 1))
print(pack('l', 1))
print(unpack('h', pack('h', 1)))
print(unpack('l', pack('l', 1)))
#big endian
print(pack('>h', 1))
print(pack('>l', 1))
print(unpack('>h', pack('>h', 1)))
print(unpack('>l', pack('>l', 1)))
```

```
b'\x01\x00'
b'\x01\x00\x00\x00'
(1,)
(1,)
b'\x00\x01'
b'\x00\x00\x00\x01'
(1,)
(1,)
```

struct 숫자와 문자 처리

숫자와 문자 처리시 파라미터가 상이. 문자처리 할 경우 반드시 b'*' 등 bytes 처리 기준을 넣어줘야 함

종류	0x1234의 표현	0x12345678의 표현
빅 엔디언	12 34	12 34 56 78
리틀 엔디언	34 12	78 56 34 12
미들 엔디언	-	34 12 78 56 또는 56 78 12 34

```
from struct import *
# 숫자처리
print(pack('i', 500))
print(pack('>i', 500))

# 문자 처리는 b'*' format을 추가
print(pack('ch', b'*', 32767))
print(pack('>ch', b'*', 32767))
```

```
b'\xf4\x01\x00\x00'
b'\x00\x00\x01\xf4'
b'*\x00\xff\x7f'
b'*\x7f\xff'
```

Formatting 처리

Format character 1

데이터를 패킹 및 언 패킹 할 때 예상되는 레이아웃을 지정하는 데 사용되는 메커니즘

Format	C Type	Python type	Standard size
x	pad byte	no value	
c	char	bytes of length 1	1
b	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I	unsigned int	integer	4
l	long	integer	4
L	unsigned long	integer	4

Format character 2

데이터를 패킹 및 언 패킹 할 때 예상되는 레이아웃을 지정하는 데 사용되는 메커니즘

Format	C Type	Python type	Standard size
q	long long	integer	8
Q	unsigned long long	integer	8
n	ssize_t	integer	
N	size_t	integer	
f	float	float	4
d	double	float	8
s	char[]	bytes	
p	char[]	bytes	
P(대문자)	void *	integer	

struct 변환 size 처리

변환 타입에 대한 총 길이를 확인

```
from struct import *
print(pack('hhl', 1, 2, 3))

print(unpack('hhl', b'\x01\x00\x02\x00\x03\x00\x00\x00'))

print(calcsize('hhl'))
```

```
b'\x01\x00\x02\x00\x03\x00\x00\x00'
(1, 2, 3)
8
```

struct 변환 : format size 차이

변환 타입에 따라 총 길이가 차이 발생

함수 이용

```
import struct

print(struct.pack('ci', b'*', 0x12131415))

print(struct.pack('ic', 0x12131415, b'*'))

print(struct.calcsize('ci'))

print(struct.calcsize('ic'))
```

```
b'*\x00\x00\x00\x15\x14\x13\x12'
b'\x15\x14\x13\x12*'
8
5
```

메소드 이용

```
# 문자와 숫자는 우측
a = struct.Struct('ci')

print(a.pack( b'*', 0x12131415))

# 숫자와 문자는 좌측
b = struct.Struct('ic')

print(b.pack(0x12131415, b'*'))
```

```
b'*\x00\x00\x00\x15\x14\x13\x12'
b'\x15\x14\x13\x12*'
```

struct 변환 :pad 발생

big endian 처리시 pad 발생

```
print(struct.pack('llh0l', 1, 2, 3))
print(struct.calcsize('llh0l'))
```

```
b'\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00'
12
```

```
print(struct.pack('>llh0l', 1, 2, 3))
print(struct.calcsize('>llh0l'))
```

```
b'\x00\x00\x00\x01\x00\x00\x00\x02\x00\x03'
10
```

format : H/i

연속되는 숫자들에 대한 H(2bytes), i(4bytes) unpack 처리

```
import struct
#little endian
hi, lo = struct.unpack("<HH", b"\x05\x00\xC0\x00")
print(hi,lo)

#big endian
hi, lo = struct.unpack(">HH", b"\x00\x05\x00\xC0")
print(hi,lo)
```

```
5 192
5 192
```

```
import struct
#little endian
hi, lo = struct.unpack("<ii", b"\x05\xC0\x00\x00\x05\xC0\x00\x00")
print(hi,lo)

#big endian
hi, lo = struct.unpack(">ii", b"\x00\x00\xC0\x05\x00\x00\xC0\x05")
print(hi,lo)
```

```
49157 49157
49157 49157
```

format : s

연속되는 문자들에 대한 s(bytes)에 대해 unpack 처리

```
import struct

# ynpact return 결과는 튜플로 처리 됨
print(struct.unpack('sss', b'abc'))
print(struct.unpack('3s', b'abc'))

#bytes-> str로 전환
print(struct.unpack('3s', b'abc')[0].decode("utf-8"))

i = "가나다"
bs = i.encode("utf-8")
print(struct.unpack('%ds' % (len(bs)), bs)[0].decode("utf-8"))
```

```
(b'a', b'b', b'c')
(b'abc',)
abc
가나다
```

format : b/B

연속되는 문자들에 대한 b/B를
integer(python 타입)에 대해 pack/unpack 처리

```
import struct
a = b'Hello'
print(a)
print(a[0], chr(a[0]))

print(list(b'Hello'))


print(type(struct.pack(b'BBBBB', 72, 101, 108, 108, 111)))
print(struct.pack(b'bbbbb', 72, 101, 108, 108, 111))
pa = struct.pack('5s', b"Hello")
upa = struct.unpack('sssss',pa)
print(upa)
print(type(upa[0]))
```

```
b'Hello'
72 H
[72, 101, 108, 108, 111]
<class 'bytes'>
b'Hello'
(b'H', b'e', b'l', b'l', b'o')
<class 'bytes'>
```

PYTHON SOCKET MODULE

Moon Yong Joon

SOCKET
기본

Socket

Socket 이란

Socket이란 양방향 통신채널(endpoint)이고,
Sockets은 프로세스간, 머신들간 등의 통신을 지원

Term	Description
domain	Tranport 메커니즘에 사용하는 Proctocol Family AF_INET, PF_INET, PF_UNIX, PF_X25 등
type	2개의 endpoint 사이의 커뮤니케이션 타입. - SOCK_STREAM : connection-oriented protocols(TCP) - SOCK_DGRAM : connectionless protocols.(UDP)
protocol	a domain and type 내의 다양한 protocol를 의미. 기본값 0
hostname	실제 서버 네임 및 주소(DNS, IP)
port	각 서버가 서비스를 처리하기 위한 주소

Socket 종류

- SOCKET STREAM : SOCK_STREAM

TCP 트랜스포트 계층 프로토콜을 사용하여 통신하는 소켓

연결-지향 형태를 지원

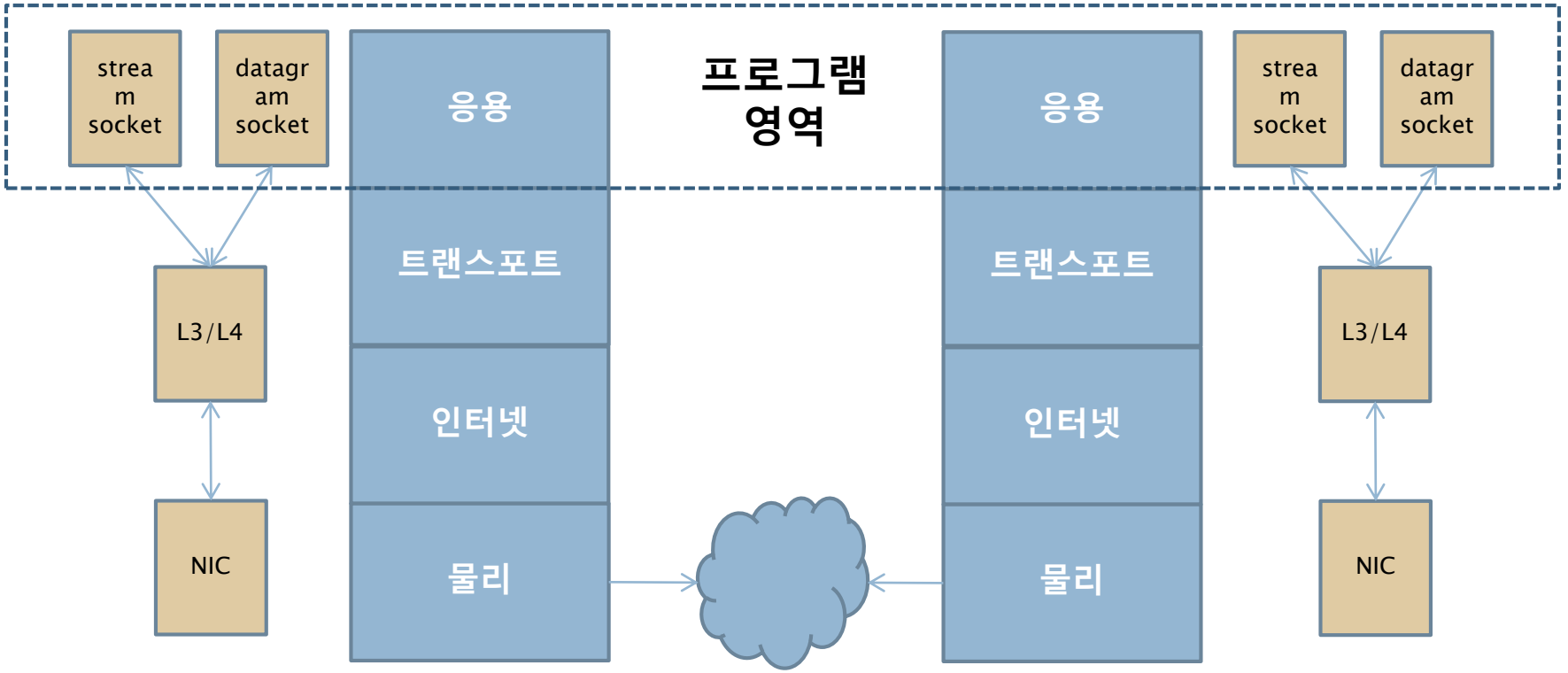
- SOCKET DGRAM : SOCK_DGRAM

UDP 트랜스포트 계층 프로토콜을 사용하여 통신하는 소켓

신뢰적이지 못한 데이터그램 형태를 지원

Socket 구조

Socket 프로그램 구조



Socket 생성

Socket 객체를 생성하기 위해서는 도메인, 타입, 프로토콜을 파라미터로 받아서 생성

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

domain ➔ socket_family: AF_UNIX or AF_INET
type ➔socket_type: SOCK_STREAM or SOCK_DGRAM.
protocol: default는 0.

AF_INET : IP version 4 or IPv4
SOCK_STREAM : TCP protocol

Client Socket 연결

클라이언트에서 서버 연결

```
s.connect((TCP_IP, TCP_PORT))
```

Method	Description
s.connect()	TCP server 연결하기 위해 파라미터로 서버의 IP 주소와 Port를 넘김

Server Socket 연결

서버 내의 socket 활성화 및 클라이언트 연결

Method	Description
s.bind()	TCP 서버 연결 s.bind((TCP_IP, TCP_PORT))
s.listen()	클라이언트에서 이벤트 요청을 위한 TCP listener 시작 s.listen(1)
s.accept()	TCP client 연결, 클라이언트 연결이 올 때까지 대기(blocking). conn, addr = s.accept()

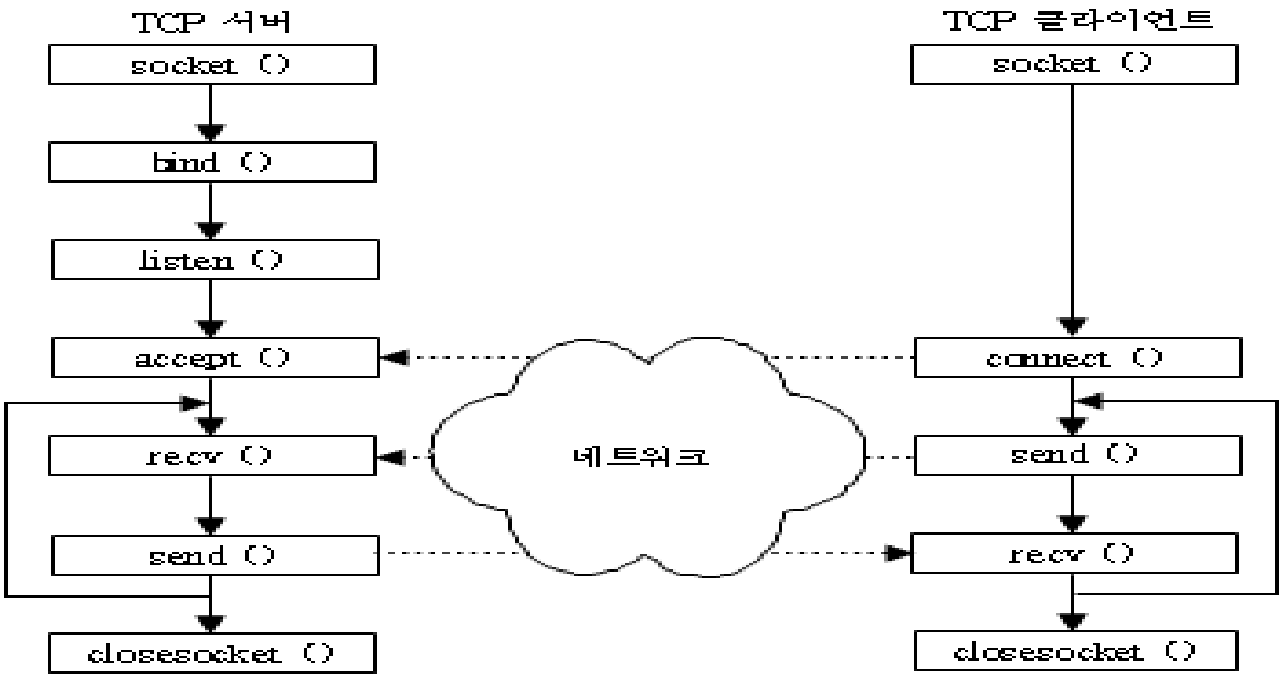
Socket간 메시지 송수신

클라이언트와 서버간 TCP/UDP 메시지 송수신

Method	Description
s.recv()	수신 TCP message : data = s.recv(BUFFER_SIZE)
s.send()	송신 TCP message : s.send(MESSAGE)
s.sendall()	송신 TCP message : s.sendall(MESSAGE)
s.recvfrom()	수신 UDP message
s.sendto()	송신 UDP message
s.close()	socket 클로징

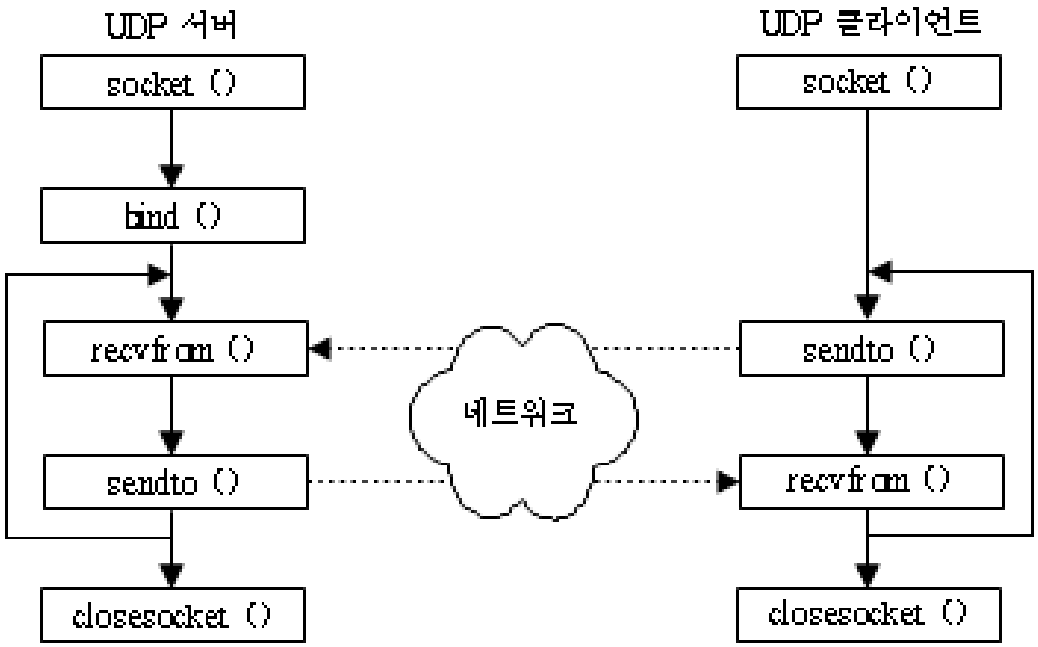
TCP : SOCK_STREAM

TCP 즉 연결-지향 (Connection-oriented)에 대한 클라이언트와 서버간 메시지 송수신



UDP : SOCK_DGRAM

UDP 즉 비연결(Connectionless)에 대한 클라이언트와 서버간 메시지 송수신



Blocking & Non-Blocking

Socket 처리는 기본 Blocking 처리

- 어떤 일이 일어나기를 기다리면서 멍하니 있는 상태
- 기본값 : socket.setblocking(1) ➔ socket.settimeout(None)

Non-blocking 처리 : flag인자가 0

socket.setblocking(0) ➔ socket.settimeout(0.0)

Blocking 처리 : flag 인자가 1

socket.setblocking(1) ➔ socket.settimeout(100)

Socket 서버 정보 검색

Socket 함수

서버에 대한 host 이름이나 ip 주소 검색

Method	Description
socket.gethostbyname(obj)	DNS로 IP 주소 가져오기
socket.gethostname()	내부 및 외부 서버 내의 DNS 나 서버 네임 가져오기
socket.getservbyport(obj,'tcp')	특정 port가 처리하는 서비스 네임 가져오기

Hostname/ipaddress 검색(1)

파이썬 함수는 인자와 결과값에 대한 타입정보를 3.5버전 이상부터 hint로 추가되어 결과값에 대한 확인을 별도의 함수로 작성하여 확인

```
%%writefile return_check.py
# return_check.py
type_str = ['str','int','float','list','dict','function','object']

def return_type(obj) :

    type_check = obj.__class__.__name__
    if type_check in type_str :
        return True, type_check
    else :
        return False, type_check
```

```
Writing return_check.py
```

Hostname/ipaddress 검색(2)

Hostname을 가지고 ip address 검색하는 함수 정의

```
%%writefile socket_test.py

# socket_test.py
import socket
import return_check as ret


def get_ipaddress(obj) :
    '''
    get ip address
    '''
    print(" host name :", obj)

    ip_addr = socket.gethostbyname(obj)

    print(ret.return_type(ip_addr))
    print(" ip address :", ip_addr)

# 자신의 PC hostname 가져오기
obj = socket.gethostname()
print(ret.return_type(obj))
get_ipaddress(obj)

# localhost
obj = 'localhost'
get_ipaddress(obj)

# python org
obj = 'www.python.org'
get_ipaddress(obj)
```

```
Overwriting socket_test.py
```

Hostname/ipaddress 검색(3)

자신의 서버 및 remote 검색하기

```
!python socket_test.py
```

```
(True, 'str')
 host name : localhost
(True, 'str')
 ip address : 127.0.0.1
 host name : www.python.org
(True, 'str')
 ip address : 151.101.88.223
```

외부ip 호출하여 client연결

구글을 검색해서 클라이언트 서버 생성

```
import socket # for socket
import sys

try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print("Socket successfully created")
except socket.error as err:
    print("socket creation failed with error %s" %(err))

# default port for socket
port = 80

try:
    host_ip = socket.gethostbyname('www.google.com')
except socket.gaierror:
    # this means could not resolve the host
    print("there was an error resolving the host")
    sys.exit()

# connecting to the server
s.connect((host_ip,port))

print("the socket has successfully connected to google on port == %s" %(host_ip))
```

```
Socket successfully created
the socket has successfully connected to google on port == 216.58.199.4
```

Port Protocol 정보 조회

세부 서비스 프로토콜

어플리케이션 프로토콜 및 파이선 모듈

Protocol	Common function	Port No	Python module
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib

Port별 서비스 검색

TCP 내의 port별 프로토콜 서비스를 검색

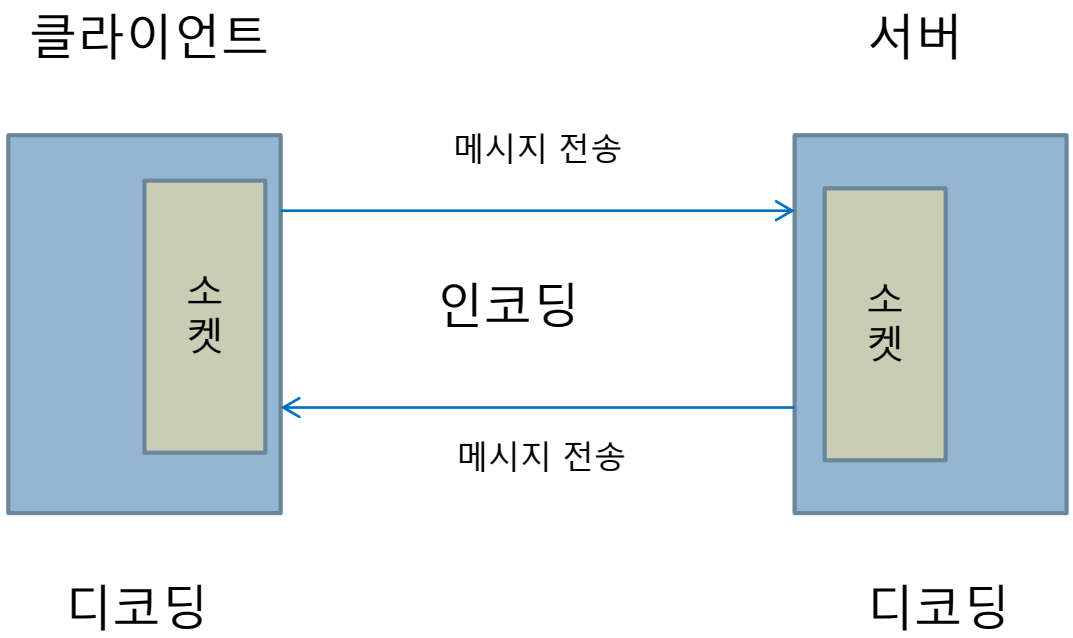
```
import socket
def get_service(obj) :
    service =socket.getservbyport(obj,'tcp')
    print(" port : " + str(obj) + " service name : " + service)

print(' get port ')
get_service(80)
get_service(53)
get_service(25)
```

```
get port
port : 80 service name : http
port : 53 service name : domain
port : 25 service name : smtp
```

Socket 생성 기초

Echo 통신처리 흐름

클라이언트에서 서버로 전송하면 그대로 전달하는 통신을



서버 생성

jupyter notebook 에서 서버를 만들고 실행

```
%%writefile soket_test2.py
import socket

HOST = ''                    # Symbolic name meaning all available interfaces
                # Symbolic name meaning all available interfaces
PORT = 50009                # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            data1 = data.decode("utf-8")
            data2 = data1.encode("utf-8")
            conn.sendall(data2)
```

```
Overwriting soket_test2.py
```

```
!python soket_test2.py
```

클라이언트 생성

다른 jupyter notebook에서 클라이언트 모듈을 작성해서 실행시키도 메시지를 전송하면 리턴값이 처리됨

```
# Echo client program
import socket

HOST = 'localhost'                # The remote host
PORT = 50009              # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

while 1 :

    data = input('> ')       #데이터를 입력창에서 받음
    if not data: break
    s.send(bytearray(data,encoding='utf-8'))
    data = s.recv(1024)          # 서버로부터 데이터 수신
    data1 = data.decode("utf-8")
    if not data: reak
    print('Received', repr(data1))
s.close()
```

```
> hello world
Received 'hello world'

>
```