**C 자료와 파이썬 자료간의 변환을 하는 struct 모듈을 공부해보죠.**

**목표 : struct 모듈에 대해서 공부해보자**

**1. pack**

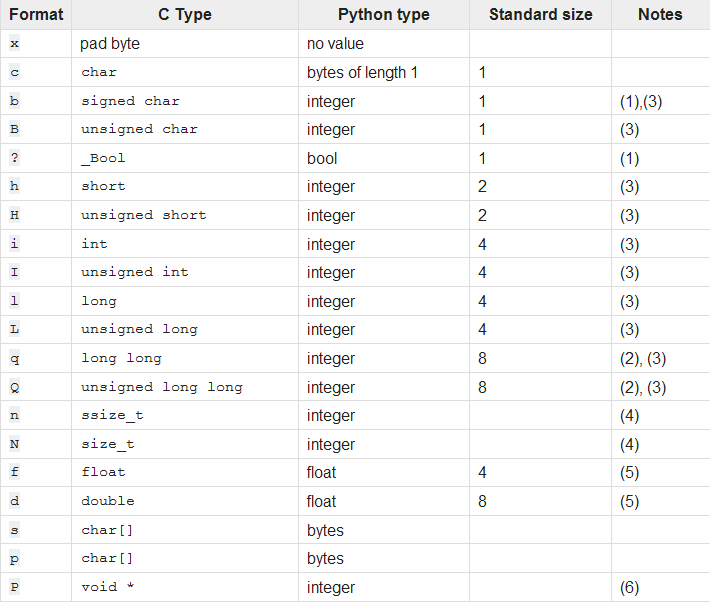
**2. unpack**

**3. 정렬 문제**

**4. 바이너리 읽어오기**

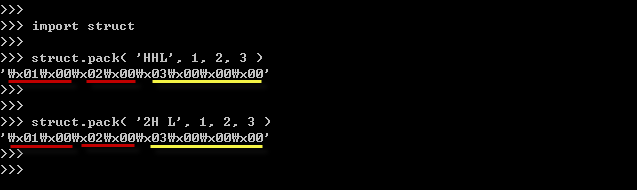
**1. pack**

**① struct 포맷 문자열**



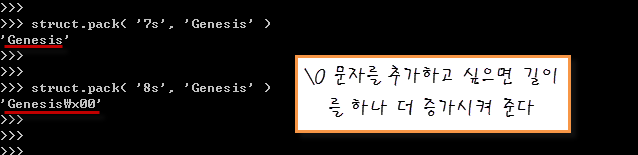
Python Documentent

**② 파이썬 자료를 C 자료로 변환**

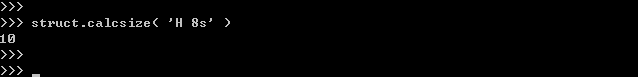


H 는 unsigned short, L 은 unsigned long 입니다. HHL 이므로 unsigned short 2개, unsigned long 1 개로 표현합니다.

HH 를 2H 로 표현 가능합니다.

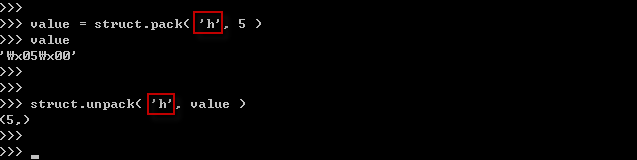


s 는 문자열입니다. 길이에 맞춰서 쓰시면 됩니다.



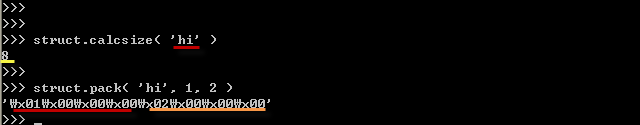
calcsize 함수는 반환 될 문자열의 길이를 알아볼 때 사용합니다. H 는 2 바이트 8s 는 8 바이트이므로 총 10 이죠.

**2. unpack**



주의할 점은 C 자료형이 short 형이므로 short 로 바꿔야한다는 점이예요. 터플로 반환되는거 잊지마세요.

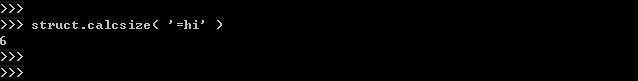
**3. 정렬 문제**



계산상 6 바이트가 나와야하는데 8 이나왔습니다. h 가 int 로 인식해버렸군요.

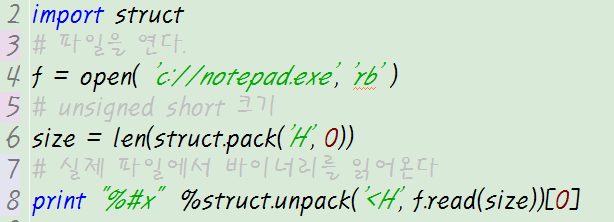
|  |  |  |
| --- | --- | --- |
| 문자 | 바이트 순서 | 크기와 정렬 |
| @ | 시스템에 따름 | 시스템에 따름(기본 값) |
| = | 시스템에 따름 | 없음 |
| < | 리틀 엔디안 | 없음 |
| > | 빅 엔디안 | 없음 |
| ! | 네트워크(빅 엔디안) | 없음 |

@ 또는 문자를 쓰지 않을 경우 기본값으로 시스템에 따라요. 일정한 규칙에 따라 패딩을 넣어서 계산합니다. 문자를 넣으면 크기와 정렬을 계산하지 않습니다.



**4. 바이너리 읽어오기**

실행 파일의 바이너리를 한번 읽어와봅시다. 파일 맨 앞에 있는 MZ 를 읽어봅시다.



﻿

**다음 시간에는 프로세스에 관련된 모듈들을 공부해보죠**.

**[출처]** [[파이썬 기초] struct](http://blog.naver.com/shw20319/20180617590)|**작성자** [GenesisShin](http://blog.naver.com/shw20319)

[파이썬 파일 입출력 + 이진(binary) 형 파일 입출력](http://pracon.tistory.com/130)

2012.10.06 01:36

[binary](http://pracon.tistory.com/tag/binary), [File](http://pracon.tistory.com/tag/File), [fileIO](http://pracon.tistory.com/tag/fileIO), [pickle](http://pracon.tistory.com/tag/pickle), [Python](http://pracon.tistory.com/tag/Python), [파일입출력](http://pracon.tistory.com/tag/%ED%8C%8C%EC%9D%BC%EC%9E%85%EC%B6%9C%EB%A0%A5)

 출처 : <http://mail.python.org/pipermail/tutor/2000-July/001883.html>

python 에서 wf.write() method 는 string 만 가능하다.

그러므로 다양한 size 로 file 을 작성하기 위해서는 struct module 을 사용해야 한다.

(<http://blog.empas.com/i5on9i/29642398>)

|  |
| --- |
| int형을 이진파일에 저장 |
| #/usr/bin/env python    import struct  def main(argv=None):  # write a file  binfile = open('bin.dat', 'wb')  for num in range(50):  data = struct.pack('i', num) # pack 'num' with a integer size  binfile.write(data)  # read the file  binfile = open('bin.dat', 'rb')  intsize = struct.calcsize('i')  while 1:  data = binfile.read(intsize)  if data == '':  break  num = struct.unpack('i', data)  print num  if \_\_name\_\_ == '\_\_main\_\_' :  main() |

struct reference

<http://docs.python.org/py3k/library/struct.html>

1. 파일쓰기  
  - open(file) 내장 함수로 파일 객체를 얻음  
  - 얻어진 파일 객체서 자료를 읽고 씀  
  - close로 객체 사용을 종료(생략 가능)

>>> s = '''  
Its power: Python developers typically report  
they are able to develop application in a half  
to a tenth the amount of time it takes them to do  
the same work in such langueges as C.  
'''  
>>> f = file('t.txt', 'w')  
>>> f.write(s)  
>>> f.close()

2. 파일 읽기

>>> f = file('t.txt') # 두 번째 인수 생략 시, 읽기 모드로 동작  
>>> s = f.read()  
>>> print s

Its power: Python developers typically report  
they are able to develop application in a half  
to a tenth the amount of time it takes them to do  
the same work in such langueges as C.

3. 라인 단위로 파일 읽기  
  - 파일 객체의 반복자(Iterator) 이용하기(가장 효과적인 방법)

>>> f = open('t.txt')  
>>> for line in f:  
 print line,

  - readline : 한 번에 한 줄씩 읽는다.

>>> f = open('t.txt')  
>>> line = f.readline()  
>>> while line:  
 print line, # line 자체에 \n이 포함되어 있어 콤마(,)를 사용  
 line = f.readline()

  - readlines : 파일 전체를 라인 단위로 끊어서 리스트에 저장한다.

>>> f = open('t.txt')  
>>> for line in f.readlines():  
 print line,

  - xreadlines : readlines와 유사하지만 파일 전체를 한꺼번에 읽지는 않고, 필요할 때만 읽어서 공급한다. 큰 파일을 for 문으로 라인 단위로 읽을 때 편리하다.

>>> f = open('t.txt')  
>>> for line in f.xreadlines():  
 print line,

4. 라인 단위로 파일 쓰기

>>> lines = ['first line\n', 'second line\n', 'third line\n']  
>>> f = open('t1.txt', 'w')  
>>> f.writelines(lines)  
>>>  
>>> lines = ['first line', 'second line', 'third line']  
>>> f = open('t1.txt', 'w')  
>>> f.write('\n'.join(lines))

※ 단어의 수 구하기

>>> n = len(open('t.txt').read().split())  
>>> print n  
35

※ 라인의 수 구하기

>>> len(open('t.txt').readlines())  
5  
>>> open('t.txt').read().count('\n')  
5

※ 문자의 수 구하기

>>> f = open('t.txt')  
>>> len(f.read()) # 줄바꾸기 : '\012'  
182  
>>> os.path.getsize('t.txt') # 줄바꾸기 : '\015\012'  
187L

5. 파일에서 원하는 만큼의 문자 읽기

>>> f = open('t.txt')  
>>> f.read(10) # 10바이트 만큼만 읽기  
'\nIts power'  
>>> f.read(10)  
': Python d'

6. 파일 처리 모드  
  - 'r' : 읽기 전용  
  - 'w' : 쓰기 전용  
  - 'a' : 파일 끝에 추가(쓰기 전용)  
  - 'r+' : 읽고 쓰기  
  - 'w+' : 읽고 쓰기(기존 파일 삭제)  
  - 'a+' : 파일 끝에 추가(읽기도 가능)  
  - 'rb' : 이진 파일 읽기 전용  
  - 'wb' : 이진 파일 쓰기 전용  
  - 'ab' : 이진 파일 끝에 추가(쓰기 전용)  
  - 'rb+' : 이진 파일 읽고 쓰기  
  - 'wb+' : 이진 파일 읽고 쓰기(기존 파일 삭제)  
  - 'ab+' : 이진 파일 끝에 추가(읽기도 가능)  
※ 플랫폼에 의존하지 않는 코드를 작성하려 한다면 이진 파일을 다룰 때 b 플래그를 사용하는 것이 좋다.  
  
7. 임의 접근 파일  
  - 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)  
  - tell() : 현재의 파일 포인터 위치를 돌려줌

>>> fn = 't.txt'  
>>> f = open(fn, 'w+')  
>>> s = '0123456789abcdef'  
>>> f.write(s)  
>>> f.seek(5)  
>>> print f.tell() # 현재 위치를 돌려줌  
5  
>>> print f.read(1) # 1바이트 읽기  
5  
>>> f.seek(-3, 2) # 끝부터 앞으로 3번째 지점  
>>> print f.tell()  
13  
>>> print f.read(1)  
d

8. 파일 객체 속성들  
  1) 기본 파일 메쏘드  
    - file.close() : 파일을 닫는다. 더 이상 입ㆍ출력할 수 없게 된다.  
    - file.read([size]) : 원하는 바이트 수만큼 파일에서 읽어 온다. 인수를 지정하지 않으면 전체 파일을 읽어 온다.  
    - file.readline([size]) : 라인 하나를 읽어 들인다. size가 지정되면 읽을 수 있는 최대 바이트 수가 된다.  
    - file.readlines() : 전체 라인을 readline()을 이용하여 읽어 들인 라인을 리스트에 넣어서 리턴한다.  
    - file.write(str) : 문자열 str을 파일에 쓴다.  
    - file.writelines(list) : 문자열 리스트를 파일에 쓴다. 줄바꾸기가 자동으로 삽입되지는 않는다.  
    - file.seek(offset[, whence]) : whence의 기본 값은 0이다. 0이면 시작 기준, 1이면 현재 위치 기준, 2이면 끝 기준에서 offset만큼 떨어진 위치에 파일 포인터를 위치시킨다.  
    - file.tell() : 파일의 현재 위치를 리턴한다.  
  2) 기타의 파일 메쏘드  
    - file.flush() : 버퍼가 다 채워지지 않았어도 내부 버퍼의 내용을 파일에 보낸다.  
    - file.fileno() : file 객체의 파일 기술자(File Descriptor)(정수)를 리턴한다.  
    - file.isatty() : 만일 file 객체가 tty와 같은 장치이면 1이면 0을 리턴  
    - file.truncate([size]) : 파일 크기를 지정된 크기로 잘라 버림. 인수를 주지 않으면 현재 위치에서 자른다.  
  3) 파일 객체 속성  
    - file.closed : file이 close 되었으면 1 아니면 0  
    - file.mode : 파일이 오픈된 모드  
    - file.name : open()할 때 사용된 파일 이름  
    - file.softspace : 1이면 print문을 사용할 때 값 출력 사이에 자동적으로 스페이스가 출력됨. 0이면 스페이스가 자동으로 삽입되지 않음  
  
9. 파일 입ㆍ출력 예제

## 지정한 파일의 특정 문자열을 다른 문자열로 변환  
# @file repalce.py  
  
import sys  # argv를 위해서 사용  
import re   # 정규식 처리 모듈, subn을 위해서 사용

def replace(fName, srcStr, desStr):  
    f = open(fName)  
    txt = f.read()  
    txt = re.subn(srcStr, desStr, txt)[0]  
    return txt

if \_\_name\_\_ == '\_\_main\_\_':  
    if len(sys.argv) != 4:  
        print '''Usage : replace fileName srcStr desStr'''  
        sys.exit()  
    print replace(sys.argv[1], sys.argv[2], sys.argv[3])

10. 표준 입ㆍ출력 방향 전환  
  1) 표준 출력을 파일로 저장하기  
    - 출력 방향 전환하기

import sys

f = open('t.txt', 'w')  
stdout = sys.stdout # 표준 출력 파일 저장해 두기  
sys.stdout = f  # 파일 객체로 변경  
print 'Sample output'  
print 'Good'  
print 'Good'  
f.close()  
sys.stdout = stdout # 필요하면 복구

    - print 문을 직접 이용하기

>>> import sys  
>>> print >> sys.stderr, 'Warning: action filed not supplied'  
Warning: action filed not supplied  
>>> f = open('t.txt', 'w')  
>>> print >> f, 'spam string'  
>>> f.close()

  2) 표준 출력을 문자열로 저장하기

import sys  
import StringIO

stdout = sys.stdout # 표준 출력 파일 저장해 두기  
sys.stdout = f = StringIO.StringIO()    # 출력 파일 방향 전환

print 'Sample output'  
print 'Good'  
print 'Good'

sys.stdout = stdout # 표준 출력 복구  
s = f.getvalue()    # 내부 문자열 가져오기

print 'Done-----'  
print s

  3) 문자열을 파일 객체처럼 읽어 내기

try:  
    import cStringIO    # 빠른 처리를 원한다면 C 버전 모듈인 cStringIO 사용  
    StringIO = cStringIO  
except:  
    import StringIO

s = '''  
Python is a cool little language.  
It is wall designed, compact, easy to learn and fun to program in.  
'''

f = StringIO.StringIO(s)    # 문자열 객체에서 파일 객체 얻어내기  
print f.read().upper()  # 대문자로 변환

11. 지속 모듈 : 프로그램이 종료되고 나서도 존재하게 하고, 그 후에 다시 그 데이터를 프로그램에서 사용  
  1) 종류  
    - DBM 관련 모듈  
    - pickle 모듈  
    - marshal 모듈  
    - shelve 모듈  
  2) DBM 파일 관련 모듈 사용하기  
    - anydbm, dbm, gdbm, dbhash, dumbdbm 등  
    - anydbm 모듈 : 사용 가능한 DBM 호환 가능 최적의 모듈을 찾아 준다.  
    - 키에 의한 참조(인덱싱)로 파일에서 자료를 읽어 오고, 인덱싱으로 치환하는 것으로 파일에 자료를 저장  
    - 키와 값은 반드시 문자열이어야 한다.

>>> import anydbm  
>>> f = anydbm.open('music', 'c') # 'c'(create)는 파일이 없으면 생성, 있으면 읽기로 오픈  
>>> f['flute'] = 'wood wind' # 인덱싱으로 치환  
>>> f['violin'] = 'string'  
>>> f['piano'] = 'keyboard'  
>>> f.keys() # keys() 메쏘드  
['flute', 'violin', 'piano']  
>>> f.values()  
['wood wind', 'string', 'keyboard']  
>>> f.items()  
[('flute', 'wood wind'), ('violin', 'string'), ('piano', 'keyboard')]  
>>> len(f) # 레코드 수  
3  
>>> 'oboe' in f # 멤버십 테스트  
False  
>>> f['flute'] # 인덱싱으로 값 읽어 오기  
'wood wind'  
>>> f['violin']  
'string'  
>>> f.close() # 파일 닫기  
>>> ========================== RESTART ==========================  
>>> import anydbm  
>>> f = anydbm.open('music', 'c') # 'music' 파일 열기  
>>> f.keys() # 키 목록 얻기  
['flute', 'violin', 'piano']  
>>> for k in f: # 전체 내용 출력  
 print k, f[k]

flute wood wind  
violin string  
piano keyboard  
>>> f['oboe'] = 'wood wind' # 내용 추가  
>>> f['piano'] = 'keyboard instrument' # 내용 변경  
>>> del f['violin'] # 내용 삭제  
>>> for k in f: # 다시 전체 내용 출력  
 print k, f[k]

flute wood wind  
piano keyboard instrument  
oboe wood wind

  3) 피클링  
    - 모든 객체 저장  
    - 재귀적 관계도 모두 처리  
    - import pickle(또는 cPickle) : 모듈 import  
      pickle.dump(출력할 객체, 파일 객체) : 파일로 객체 출력  
      object = pickle.load(파일 객체) : 파일에서 객체를 읽어들임  
      s = pickle.dumps(출력할 객체) : 문자열로 객체 출력  
      object = pickle.loads(s) : 문자열에서 객체를 읽어들임  
    - 장점 : 디버깅을 쉽게하고 문제가 생겼을 때 일반 텍스트 에디터로 복구를 쉽게할 수 있다.  
    - 단점 : 파일 크기도 크고 또한 처리 속도도 느리다.  
    ※ cPickle 모듈은 pickle 모듈과 같은 인터페이스를 가지고 있지만 약 1000배 빨리 수행되도록 설계되었다.

## pickle sample  
  
# @file pickleSample1.py  
try:  
    import cPickle  
    pickle = cPickle  
except:  
    import pickle

phone = {'tom': 4358382, 'jack': 9465215, 'jim': 6851325, 'Joseph': 6584321}  
List = ['string', 1234, 0.2345]  
Tuple = (phone, List)   # 리스트, 튜플, 사전의 복합 객체

f = open('t2.txt', 'w') # 파일 객체를 얻는다.

pickle.dump(Tuple, f)   # 파일로 복합 객체 출력(pickling)  
f.close()

f = open('t2.txt')

x, y = pickle.load(f)   # 파일에서 읽어 오기(unpickling)  
print x # x는 사전  
print y # y는 리스트

# @file pickleSample2.py  
try:  
    import cPickle  
    pickle = cPickle  
except:  
    import pickle

class Simple:   # 가장 단순한 클래스를 정의  
    pass

s = Simple()    # 인스턴스 객체 생성  
s.count = 10    # 인스턴스 이름 공간에 변수 생성

f = open('t3.txt', 'w') # 인스턴스 저장  
pickle.dump(s, f, 1)    # bin=1 : 이진 모드로 저장  
f.close()

f = open('t3.txt')  
t = pickle.load(f)  # 인스턴스 가져오기

print t.count

원본글 주소  : <http://creaplz.tistory.com/57>