

10. 파일

10.1 텍스트 파일 쓰기 / 읽기

▶ 파일 쓰기

텍스트 파일을 쓰는 법은 간단하다.

- ① open()내장 함수로 파일 객체를 얻는다.
- ② 얻어진 파일 객체에서 데이터를 읽고 쓴다.(read, write)
- ③ close() 함수로 객체의 사용을 종료한다.(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 languages as C
...
>>> f = open('c:\wwwjava\wwwt.txt', 'w')    # 쓰기 모드로 한다
>>> f                                         # 객체를 확인한다.
<_io.TextIOWrapper name='c:\wwwjava\wwwt.txt' mode='w' encoding='cp949'>
>>> f.write(s)                               # 문자열(str)을 파일에 기록한다.
182
>>> f.close()                               # 파일을 닫는다.
```

```
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 languages as C
```

open()내장 함수는 기본적으로 두 개의 인수를 받는다.

첫 번째는 파일 이름(문자열)이고, 두 번째는 파일을 다루고자 하는 모드이다.

파일에 쓰기 위해서는 w를 사용한다,

with 문을 이용하면 다음과 같이 더 편하게 작성할 수 있다.

with 문 앞에 있는 동안 객체 f를 이용하여 파일에 관련된 작업을 할 수 있고, with문에서 빠져나오면서 자동으로 닫히기까지 한다,

```
>>> with open('c:\wwwjava\wwwt1.txt', 'w') as f:
...     f.write('위대한 세종대왕')
...
8
```

출력된 8은 여덟 글자가 저장되었다는 의미이다,

파일을 열 때 인수 encoding을 추가하면 저장할 문자 인코딩을 지정할 수 있다.

값을 지정하지 않으면 시스템의 기본적인 문자 인코딩(윈도우인 경우는 ANSI)이 지정된다.

UTF-8형식으로 지정해 보자.

```
>>> with open('c:\wwwjava\wwwt2.txt', 'w', encoding = 'utf-8') as f:
...     f.write('위대한 세종대왕')
...
8
```

▶ 파일 읽기

open() 내장 함수에서 두 번째 인수를 r로 지정하면 읽기 모드이다.

두 번째 인수를 생략해도 읽기 모드로 동작한다,

```
>>> f = open('c:\wwjava\wt.txt')
>>> f                                     # 객체를 확인한다.
<_io.TextIOWrapper name='c:\wwjava\wt.txt' mode='r' encoding='cp949'>
>>> s = f.read()                         # 파일 전체 내용을 읽는다.
>>> 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 languages as C

>>> f.close()
>>> open('c:\wwjava\wt1.txt').read()     # 파일에 저장한 내용을 확인한다.
'위대한 세종대왕'
>>>
```

with 문을 이용하면 다음과 같이 더 깔끔한 코딩이 가능하다,.

```
>>> with open('c:\wwjava\wt.txt') as f:
...     print(f.read())
...

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 languages as C

>>>
```

기본 인코딩으로 저장되어 있지 않은 파일은 열 때 인수 encoding을 지정해야 한다,

앞의 예에서 파일 t2.txt는 UTF-8형식으로 저장하였기 때문에 읽을 때도 형식을 지정해야 한다,

```
>>> with open('c:\wwjava\wt2.txt', encoding = 'utf-8') as f:
...     print(f.read())
...
위대한 세종대왕
```


10.2 줄 단위로 파일 쓰기 / 읽기

▶ 파일 쓰기

만일 파일로 기록할 문자열을 줄 단위로 가지고 있으면 writeline()메서드를 사용할 수 있다.

이 메서드는 리스트 안에 들어 있는 문자열을 연속해서 출력해 준다.

```
>>> lines = ['first line\n', 'second line\n', 'third line\n']
>>> f = open('c:\wwjava\wt1.txt', 'w')
>>> f.writelines(lines)
```

 t1.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

first line
second line
third line

위 코드는 다음과 같이 write()함수를 사용해서 기록을 할 수도 있다.

```
>>> lines = ['first line\n', 'second line\n', 'third line\n']
>>> f = open('c:\wwjava\wt1.txt', 'w')
>>> f.write(''.join(lines))
34
```

만일 lines에 저장되어 있는 문자열들 마지막에 줄 바꾸기 코드를 \n가 없고, 여전히 이들을 줄 단위로 저장하고 싶으면 join()함수로 줄 바꾸기 코드를 삽입할 수 있다.

```
>>> lines = ['first line', 'second line', 'third line']
>>> f = open('c:\wwjava\wt1.txt', 'w')
>>> f.write('\n'.join(lines))
33
```

올바로 기록되었는지 파일을 읽어 보자. 앞의 세가지 예는 모두 같은 파일을 만든다,

```
>>> f = open('c:\wwjava\wt1.txt')
>>> print(f.read())
first line
second line
third line
>>>
```

▶ 파일 읽기

텍스트 파일을 전체로 읽지 않고 줄 단위로 읽어 처리할 수 있다.

- 파일 객체의 반복자 이용하기
- readline() 파일을 한 번에 한 줄씩 읽는다.
- readlines() 파일 전체를 줄 단위로 끊어서 리스트에 저장한다,

① for문에 파일 객체에 반복자를 직접 사용할 수 있다.

```
>>> with open('c:\wwjava\wt.txt') as f:
...     for line in f:
...         print(line, end = '')
...

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 languages as C
>>>
```

일반적인 크기의 파일을 줄 단위로 읽어서 처리하기 위해서 readline()나 readlines()메서드를 사용한다, 한 줄씩 읽는 readlimne()메서드는 주로 while 문과 함께 사용한다,

```
>>> f = open('c:\wwjava\wt.txt')
>>> line = f.readline()
>>> while line:
...     print(line, end = '')     # line이 ''이면 파일의 끝을 의미한다.
...     line = f.readline()
...

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 languages as C
```


줄 전체를 읽어서 리스트에 저장하려면 readlines()메서드를 사용한다,

```
>>> f = open('c:\wwjava\wt1.txt')
>>> for line in f.readlines():
...     print(line, end = '')
...
first line
```

10.3 파일에서 원하는 만큼의 문자 읽기

파일을 전체나 줄 단위로 읽지 않고 원하는 바이트만큼씩 읽을 수 있다.

read() 메서드에 인수로 원하는 바이트를 지정한다,

```
>>> f = open('c:\wwjava\wt1.txt')
>>> f.read(10)
'\nIts power'
>>> f.read(10)
': Python '
```

10.4 이진 파일 쓰기 / 읽기

▶ 파일 쓰기

이진 파일을 작성하려면 열기 모드에 b를 추가한다. 이진 파일에 출력되는 자료형은 바이트이어야 한다, 문자열은 허용되지 않는다.

```
>>> f = open('c:\wwjava\wt1.bin', 'wb')           # 이진 모드로 연다.
>>> f.write('abcd')                               # 문자열은 허용되지 않는다
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: a bytes-like object is required, not 'str'
>>> 'abcd'.encode()                             # 문자열 ----> 바이트로 형변환
b'abcd'
>>> f.write('abcde'.encode())                     # 바이트는 쓸 수 있다.
5
>>> f.close()
```

▶ 파일 읽기

이진 파일을 읽으려면 rb모드로 연다. 이진 파일은 바이트로 읽는다.

```
>>> f = open('c:\wwjava\wt1.bin', 'rb')
>>> b = f.read(5)                                # 바이트로 읽는다.
>>> b
b'abcde'
>>> b.decode()                                    # 바이트 ----> 문자열로 형변환
'abcde'
>>> f.readline()                                  # 이와 같은 메서드도 당연히 가능하다. 바이트로 읽는다.
b' '
>>>
```

10.5 파일 처리 모드

파일을 열 때 사용하는 파일 처리모드에는 r과 w 이외에도 다양하다.

파일 처리 모드	설명
r	읽기 전용이다
w	쓰기 전용이다.
a	파일 끝에 추가(쓰기 전용)한다.

r+	읽고 쓰기를 한다.
w+	읽고 쓰기(기존 파일 삭제)를 한다.
a+	파일 끝에 추가(읽기도 가능)한다.
rb	이진 파일 읽기 전용
wb	이진 파일 쓰기 전용
ab	이진 파일 끝에 추가(쓰기 전용)한다.
rb+	이진 파일 읽고 쓰기를 한다.
wb+	이진 파일 읽기 쓰기(기존 파일 삭제)를 한다.
ab+	이진 파일 끝에 추가 (읽기도 가능)한다.

텍스트 파일에서 추가하는 간단한 예제

```
>>> f = open('c:\wwjava\removement.txt', 'w')
>>> f.write('first line\n')
11
>>> f.write('second line\n')
12
>>> f.close()
>>> f = open('c:\wwjava\removement.txt', 'a') # 파일 추가 모드로 연다.
>>> f.write('third line\n')
11
>>> f.close()
>>> f = open('c:\wwjava\removement.txt') # 읽기
>>> print(f.read())
first line
second line
third line
```

10.6 임의 접근 파일

지금까지 설명한 파일 접근 방식을 순차적 접근이라고 한다. 왜냐하면 파일을 앞에서부터 순차적으로 읽기 때문이다.

파일에서 임의의 위치에 있는 내용에 접근하는 모드가 필요할 때가 종종있다.

- seek(n) 파일의 n번째 바이트로 이동한다,
- seek(n, os.SEEK_CUR) 현재 위치에서 n바이트 이동한다. n이 양수이면 뒤쪽으로, 음수이면 앞쪽으로 이동한다. 이진 파일에서만 가능하다.
- seek(n, os.SEEK_END) 맨 마지막에서 n바이트 이동한다. n은 보통 음수이다.
- tell() 현재의 파일 포인터 위치를 돌려준다. 이진 파일에서만 가능하다.

```
>>> f = open('c:\wwjava\wt.txt', 'wb+') # 읽고 쓰기(기존 파일 삭제)
>>> s = b'0123456789abcdef' # 이진 파일이므로 바이트 자료형으로 출력한다.
>>> f.write(s)
16
>>> f.seek(5) # 시작부터 5번째 위치로 이동
5
>>> f.tell() # 위치 확인
5
>>> f.read(1) # 1바이트 읽기
b'5'
>>> import os
>>> f.seek(2, os.SEEK_CUR) # 현재 위치에서 2바이트 더 이동, 8번 위치
8
>>> f.seek(-3, os.SEEK_END) # 마지막에서 -3 바이트 이동, 13번 위치
13
>>> f.read(1)
b'd'
```

10.7 파일 객체의 메서드와 속성

메서드	실행
file.close()	파일을 닫는다. 더 이상 입출력을 할 수 없게 된다.
file.read([size])	원하는 바이트 수만큼 파일에서 읽어 온다. 인수를 지정하지 않으면 전체 파일을 읽어 온다.
file.readline()	줄 하나를 읽어온다. 크기를 지정하면 읽을 수 있는 최대 바이트 수가 된다.
file.readlines()	전체 줄을 readline()메서드를 사용하며 읽어들이는 줄을 리스트에 넣어서 반환한다.
file.write(str)	문자열 str을 파일에 쓴다.
file.writelines(list)	문자열 리스트를 파일에 쓴다. 줄 바꾸기가 자동으로 삽입되지는 않는다.
file.seek(offset [,whence])	인수 whence의 기본값은 0이다. 0이면 시작기준, 1이면 현재 위치 기준, 2이면 끝 기준에서 인수 offset만큼 떨어진 위치에 파일 포인터를 위치시킨다.
file.tell()	파일의 현재 위치를 반환한다.

앞서 설명한 메서드를 외에도 파일 객체가 제공하는 메서드는 많다.

파일 객체의 기타 메서드	
메서드	설명
file.flush()	버퍼가 다 채워지지 않아도 내부 버퍼의 내용을 파일에 보낸다.
file.fileno()	파일 객체의 파일 기술자(File Descriptor)(정수)를 반환한다.
file.isatty()	파일 객체가 tty와 같은 장치이면 1, 아니면 0을 반환한다.
file.truncate([size])	파일 크기를 지정한 크기로 잘라버린다. 인수를 지정하지 않으면 현재 위치에서 자른다.

파일 객체가 가진 속성들은 다음과 같다.

파일 객체의 속성들	
file.closed	파일 객체가 닫혔으면 1, 아니면 0을 반환한다.
file.mode	파일이 열기 모드이다.
file.name	파일을 열 때 사용한 파일 이름이다.

10.8 예제 : 파일 입출력

예제) 특정한 문자열을 다른 문자열로 변환하는 예

```
>>> import re                                # 정규식
>>> def replace(fname, src, dst):
    with open(fname) as f:
        txt = f.read()
        txt = re.sub(src, dst, txt)         # txt에서 src를 dst로 변경한다.
    return txt
```

sample.txt 파일을 만든다.

Do you work in C, C++, or Java on large projects?

We'll show how Python will give you results quicker and more reliably.

Are you already scripting in a language other than Python?

Whatever programing you're doing now. Python can improve it.


```

print('hello', end = "", file = f) # 출력을 메모리 스트림 f에 한다.
print('world', file = f)
print(f.getvalue())

```

```

>>> import io
>>> with io.StringIO() as f:
...     print('hello', end = '', file = f)
...     print('world', file = f)
...     print(f.getvalue())
...
helloworld

```

▶ 문자열을 파일 객체처럼 읽기

주어진 문자열이 있고, 이것을 파일 객체를 이용해서 파일처럼 읽어야 할 필요가 있을 때 String()클래스를 사용한다.

```

>>> s = """
Python is a cool little language.
It is well designed. compact, easy to learn and fun to program in.
Python strongly encourages the programmer to program in an QQ-way
but does not require it
In my opinion. it is one of the best languages to use when learning QQ-programming.
The implementation of QQ in Python is clean and simple, while being incredibly powerful.
The basic Python execution environment is also the most interactive of the five
discussed here.
which can be very useful (especially when debugging code)
"""

```

```

>>> import io
>>> with io.StringIO(s) as f:      # 문자열을 파일 유사 객체로 변환한다,
...     print(f.read(6))          # 파일처럼 문자열을 읽는다.
...     f.seek(10)
...     print(f.read(20))

```

```

>>> s = """
... Python is a cool little language.
... It is well designed. compact, easy to learn and fun to program in.
... Python strongly encourages the programmer to program in an QQ-way
... but does not require it
... In my opinion. it is one of the best languages to use when learning QQ-programming.
... The implementation of QQ in Python is clean and simple, while being incredibly powerful.
... The basic Python execution environment is also the most interactive of the five
... discussed here.
... which can be very useful (especially when debugging code)
...
>>> import io
>>> with io.StringIO(s) as f:      # 문자열을 파일 유사 객체로 변환한다.
...     print(f.read(6))          # 파일처럼 문자열을 읽는다.
...     f.seek(10)
...     print(f.read(20))
...
Pytho
10
a cool little langu

```


10.10 지속 모듈

지속성이란 특정 프로그램이 만든 데이터가 프로그램이 종료되고 나서도 존재하고, 나중에 다시 데이터를 프로그램에서 사용하는 것이다.

▶ DBM 관련 모듈

데이터를 DBM형식으로 파일을 기록한다. 시스템에 따라서 관련 모듈이 다양하게 제공된다.

모듈 dbm은 dbm.gnu나 dbm.ndbm과 같은 DBM데이터베이스에 대한 일반적인 인터페이스이다.

이와 같은 모듈이 설치되어 있지 않으면 dbm.dumb이 사용된다. 사전 자료형을 사용하는 방법과 동일한 인터페이스를 제공한다.

키와 값은 모두 문자열이어야 한다.

dbm 모듈은 시스템에서 사용할 수 있는 DBM호환 가능한 최적의 모듈을 찾아 준다.

사용하는 방법은 사전 자료형과 거의 유사하다. 키에 의한 참조(인덱싱)로 파일에서 데이터를 읽어 오고, 인덱싱으로 치환하는 것으로 파일에 데이터를 저장한다.

키와 값은 반드시 문자열이어야 한다.

```
>>> import dbm
>>> db = dbm.open('c:\\java\\cache','c')          # 'c' : 없으면 만들고 있으면 연다.
>>> db
<dbm.dumb._Database object at 0x034311F0>
>>> db[b'hello'] = b'there'
>>> db['name'] = 'gslee'
>>> db['name']                                     # 바이트로 저장되어 있다.
b'gslee'
>>> db['hello']
b'there'
>>> db['counter'] = 10                             # 문자열이나 바이트가 아니면 안된다.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\\Program Files (x86)\\Python36-32\\lib\\dbm\\dumb.py", line 199, in __setitem__
    raise TypeError("values must be bytes or strings")
TypeError: values must be bytes or strings
>>> db.keys()                                     # 사전과 유사한 인터페이스를 찾는다.
[b'hello', b'name']
>>> list(db.values())
[b'there', b'gslee']
>>> db.items()
[(b'hello', b'there'), (b'name', b'gslee')]
>>> db.get('age','?')
'?'
>>> len(db)
2
```

```
>>> 'name' in db                                # 멤버검사
True
>>> db.close()                                  # 파일 닫기
```

만든 파일은 다른 프로그램에 의해서 활용될 수 있다.
다음은 만들어진 DBM파일을 읽어서 처리하는 예이다.

```
>>> f = dbm.open('c:\\java\\cache', 'c')
>>> f.keys()
[b'hello', b'name']
>>> f.items()
[(b'hello', b'there'), (b'name', b'gslee')]
>>> f['oboe'] = 'wood wind'                      # 내용 추가
>>> f['name'] = 'joelene'                         # 내용 변경
>>> del f['oboe']                                 # 내용 삭제
>>> f.close()
>>>
```

▶ pickle 모듈

파이썬의 객체를 피클링하여 파일에 저장하는 일반화된 지속성 모듈이다.

파이썬 기본 객체뿐 아니라 복잡하게 얽혀 있는 객체들의 관계까지도 저장할 수 있다.

재귀적인 관계도 모두 처리한다.

사용자 정의 클래스와 인스턴스 객체도 처리한다. 기본적으로 텍스트 모드로 저장하지만 이진 모드로도 저장할 수 있다.

• 피클링

객체를 파일로 출력하려면 pickle.dump를 이용한다.

```
import pickle
```

```
pickle.dump(출력할 객체, 파일 객체)
```

객체를 파일에서 읽어들이려면 pickle.load를 사용한다,

```
object = pickle.load(파일 객체)
```

예제)

```
>>> import pickle
>>> with open('c:\\java\\test.pickle', 'wb') as f:
...     phone = {'tom' : 4358382, 'jack' : 9465215, 'jim' : 6851325, 'joseph' : 6584321}
...     L = ['string', 1234, 0.2345]
...     T = (phone, L)                                # 리스트, 튜플, 사전의 복잡 객체
...     pickle.dump(T, f)                             # 복합 객체 출력
...     pickle.dump(L, f)                             # L 한번 더 출력
... 
```

```

>>> with open('c:\\java\\test.pickle', 'rb') as f:
...     x, y = pickle.load(f)                # 언피클링
...     L2 = pickle.load(f)                  # 한번 더
...
>>>
>>> print(x)                                # 값 확인
{'tom': 4358382, 'jack': 9465215, 'jim': 6851325, 'joseph': 6584321}
>>> print(y)
['string', 1234, 0.2345]
>>> print(L2)
['string', 1234, 0.2345]
>>>

>>> import pickle
>>> with open('c:\\java\\test.pickle', 'wb') as f:
...     phone = {'tom' : 4358382, 'jack' : 9465215, 'jim' : 6851325, 'joseph' : 6584321}
...     L = ['string', 1234, 0.2345]
...     T = (phone, L)                        # 리스트, 튜플, 사전의 복잡 객체
...     pickle.dump(T, f)                    # 복합 객체 출력
...     pickle.dump(L, f)                    # L 한번 더 출력
...
>>> with open('c:\\java\\test.pickle', 'rb') as f:
...     x, y = pickle.load(f)                # 언피클링
...     L2 = pickle.load(f)                  # 한번 더
...
>>>
>>> print(x)                                # 값 확인
{'tom': 4358382, 'jack': 9465215, 'jim': 6851325, 'joseph': 6584321}
>>> print(y)
['string', 1234, 0.2345]
>>> print(L2)
['string', 1234, 0.2345]
>>>

```

pickle 모듈이 갖는 장점은 사용자가 정의한 임의의 클래스와 인스턴스 객체도 저장할 수 있다는 것이다. 저장하는 시점의 모든 멤버 변수의 값이 그대로 유지되므로 언제든지 다시 불러서 지속적으로 실행할 수 있다. 예를 들어, 일정한 분량의 작업을 수행하다가 멈추고 현재 상태를 pickle모듈을 이용하여 저장한 후 나중에 다시 읽어들여 지속적으로 실행할 수 있다.

사용자 클래스의 인스턴스 객체를 저장하고 읽는 예를 살펴보자.

```

>>> import pickle
>>> class Simple:                            # 가장 단순한 클래스를 정의한다.
...     pass
...
>>> s = Simple()                             # 인스턴스 객체를 생성한다,
>>> s.count = 10                             # 이 인스턴스의 이름 공간에 변수를 만든다,
>>>
>>> with open('c:\\java\\t3.pickle', 'wb') as f:
...     pickle.dump(s, f)
...

```