파이썬 프로그래밍

파일 입출력

한국기술교육대학교
온라인평생교육원

▣ 파일 입출력 방법

1. 파일 처리 모드의 종류

- 파일을 열어서 읽고, 쓰고, 덧붙이는 방법
  - open(filename, mode) 내장 함수로 filename 이름을 지닌 file 객체를 얻는다.
  - 얻어진 파일 객체에서 자료를 읽거나, 쓰거나, 덧붙이는 작업 수행
  - 모든 작업이 끝나면 close()를 호출하여 작업 프로세스의 자원 점유 해제

- open 내장 함수의 두번째 인자 mode 설명
  - 두번째 인자 mode 생략시에는 읽기 전용(r) 모드로 설정

Mode	간단 설명	자세한 설명
'r'	읽기 전용 (기본 모드)	파일 객체를 읽기 모드로 생성하고, 파일 포인터를 파일 처음 위치에 놓는다.
'w'	쓰기 전용	새로운 파일을 쓰기 모드로 생성하거나 해당 파일이 이미 존재하면 내용을 모두 없에면서 쓰기 모드로 생성하고, 파일 포인터를 파일 처음 위치에 놓는다.
'a'	파일 끝에 추가	이미 존재하는 파일을 쓰기 모드로 생성하거나 파일이 존재하지 않으면 새롭게 파일을 생성하면서 쓰기 모드로 생성하고, 파일 포인터를 파일의 마지막 위치에 놓는다. 그래서, 이후 작성되는 내용은 파일의 뒷 부분에 추가된다.

- filename → 첫 번째 인자에 문자열 형태로 기입
- mode → 두 번째 인자로 읽을 것인지, 쓸 것인지, 덧붙일 것인지 결정
- open 함수를 쓰면 항상 close 함수 같이 씀
- 읽기모드는 파일이 존재해야 함
- 파일포인터 → 현재 읽는 위치
- read라는 메소드를 사용 시 파일포인터 있는 위치를 읽음
- 'a'는 affend의 약자 → 이미 존재하는 내용 뒤에 쓰기

▣ 파일 입출력 방법

1. 파일 처리 모드의 종류

- 이진 파일로 저장을 위해서는 아래 모드 사용

Mode	간단 설명
'rb'	이진 파일 읽기 전용
'wb'	이진 파일 쓰기 전용
'ab'	이진 파일 끝에 추가

- rb, wb, ab → 이진 파일 형태 (0과 1)
- 이진파일이 일반 파일보다 크기가 작음

▣ 파일 입출력 방법

2. 파일 쓰기

```
import os

print os.getcwd()
```

```
/Users/yhhan/git/python-e-learning
```

- cwd → current working directory (현재 작업 디렉토리)
- os.getcwd( ) → 현재 작업중인 폴더를 알아보는 함수

```
s = """Its power: Python developers typically report
they are able to develop applications 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('t.txt', 'w')
f.write(s) # 문자열을 파일에 기록
f.close()
```

- 이중따옴표 3개 → 여러 개의 문장 입력 가능
- mode가 w → 파일이 쓰기 전용으로 열림
- open 내장함수가 반환하고 있는 객체가 file 객체
- write 메소드 → 문자열을 파일에 기록

▣ 파일 입출력 방법

3. 파일 읽기

```
f = file('t.txt') # f = open('t.txt', 'r')과 동일
s = f.read()
print s
```

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

- file( ) = open( ) → open을 더 추천
- 모드가 쓰이지 않음 → 읽기 모드
- s = f.read( ) → f라는 변수 내에 파일 내용 전체를 가져와 문자열 할당

- close()을 마지막에 호출하지 않으면 해당 file 객체가 다른 값으로 치환되거나 프로그램이 종료될 때 자동으로 close()가 불리워진다.
  - 하지만 명시적으로 close()를 호출하는 것을 권장함

- f.close( ) → 파일이 자원을 점유하고 있던 것을 해제

▣ 파일 입출력 방법

4. 라인 단위로 파일 읽기

- 총 4가지 방법 존재

 - 파일 객체의 반복자(iterator) 이용하기

  * 파일 객체의 반복자는 각 라인별로 내용을 읽어오도록 설정되어 있음

  * 파일을 라인별로 읽는 방법 중 가장 효과적임

 - readline(): 한번에 한줄씩 읽는다.

 - readlines(): 파일 전체를 라인 단위로 끊어서 리스트에 저장한다.

 - xreadlines(): readlines()과 유사하지만 파일 전체를 한꺼번에 읽지 않고, 상황별로 필요한 라인만 읽는다. 대용량의 파일을 for 문 등으로 라인 단위로 읽을 때 효율적이다.

- 파일 객체의 반복자 사용

```
f = open('t.txt')
i = 1
for line in f:
    print i, ":", line,
    i += 1
f.close()
```

```
1 : Its power: Python developers typically report
2 : they are able to develop applications in a half
3 : to a tenth the amount of time it takes them to do
4 : the same work in such languages as C.
```

- line 변수에는 첫 번째 라인 내용이 들어옴

▣ 파일 입출력 방법

4. 라인 단위로 파일 읽기

- readline() 사용

```
f = open('t.txt')
line = f.readline()
i = 1
while line:
    print i, ":", line,
    line = f.readline()
    i += 1
f.close()
```

```
1 : Its power: Python developers typically report
2 : they are able to develop applications in a half
3 : to a tenth the amount of time it takes them to do
4 : the same work in such languages as C.
```

- readline( ) → 현재의 파일포인터에서 개행 문자까지 읽음 = 한 라인

▣ 파일 입출력 방법

4. 라인 단위로 파일 읽기

• readlines() 사용
- 각 라인을 모두 읽어서 메모리에 리스트로 저장함

```
f = open('t.txt')
print f.readlines()
print

f.seek(0)
i = 1
for line in f.readlines():
    print i, ":", line,
    i += 1
f.close()
```

```
['Its power: Python developers typically report  \ n', 'they are able to develop applications in a half \ n', 'to a tenth the amount of time it takes them to do \ n', 'the same work in such languages as C.']

1 : Its power: Python developers typically report
2 : they are able to develop applications in a half
3 : to a tenth the amount of time it takes them to do
4 : the same work in such languages as C.
```

- readlines( ) → 리스트를 반환
- 리스트 안 원소 → 개행문자까지 포함하는 한 라인
- readlines는 메모리는 비효율적으로 사용
- seek(0) → 파일포인터를 처음으로 돌려줌
- f.readlines( ) 하면 파일포인터가 젤 마지막에 위치하게 됨

▣ 파일 입출력 방법

4. 라인 단위로 파일 읽기

- xreadlines() 사용

```
f = open('t.txt')
print f.xreadlines()
print

f.seek(0)
i = 1
for line in f.xreadlines():
    print i, ":", line,
    i += 1
f.close()
```

```
<open file 't.txt', mode 'r' at 0x10ddbfa50>
1 : 0123456789abcdef
```

- 메모리의 비효율적 사용을 개선할 수 있는 함수 = xreadlines( )
- xreadlines( ) → 리스트가 반환되지 않고 파일 객체 내용이 찍힘
- 상황별 → 코딩을 통해 노하우를 알아둬야 함
- 첫 번째 상황 → for-in 구문
- xreadlines( ) → 전체 X, 라인별 → 효과적인 메모리 사용

▣ 파일 입출력 방법

5. 라인 단위로 파일 쓰기

- writelines(): 리스트 안에 있는 각 문자열을 연속해서 파일로 출력한다.

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

f = open('t1.txt')
f.seek(0)
print f.read()
f.close()
```

```
first line
second line
third line
```

- writelines(lines) → 각각의 문자열을 라인 단위로 출력
- read( ) → 전체 내용을 모두 읽음
- 각각의 라인을 입력할 때 ' \ n'은 필히 사용
- ' \ n'을 넣지 않으면 개행이 되지 않음

▣ 파일 입출력 방법

5. 라인 단위로 파일 쓰기

- write() 이용하여 여러 문자열을 각 라인별로 파일로 출력하는 방법

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

f = open('t1.txt')
f.seek(0)
print f.read()
f.close()
```

```
first line
second line
third line
```

- ' \ n'.join(lines) → 라인에 존재하는 원소를 ' \ n'로 이어줌

▣ 파일 입출력 방법

5. 라인 단위로 파일 쓰기

- 텍스트 파일 t.txt의 단어(공백으로 분리된 문자열) 수를 출력하는 방법

```
f = open('t.txt')
s = f.read()
n = len(s.split())
print n
f.close()
```

```
35
```

- split( ) → 공백문자를 기준으로 문자를 잘라 list 화
- len(s.split( )) → 리스트 안에 존재하는 원소의 개수

▣ 파일 입출력 방법

6. 기존 파일에 내용 추가

```
f = open('removeme.txt', 'w') # 파일의 생성
f.write('first line\n')
f.write('second line\n')
f.close()
```

```
f = open('removeme.txt', 'a') # 파일 추가 모드로 오픈
f.write('third line\n')
f.close()

f = open('removeme.txt')      # 파일 읽기
print f.read()
```

```
first line
second line
third line
```

- 새로운 파일 추가 시 'a' 모드 사용
- 현재 존재하는 파일내용의 맨 뒤에 내용 삽입됨

▣ 파일 입출력 방법

7. 파일 내 임의 위치로 접근

- 파일 포인터 (pointer)
  - 파일 내에서 현재 위치를 가리키고 있음
- 파일 접근 방법
  - 순차 접근 (기본 방식): 파일을 앞에서 부터 순차적으로 읽고 쓰는 방식
  - 임의 접근: 파일 내 임의 위치에서 읽고 쓰는 방식
    - * 임의 접근을 위한 file 객체 포인터 (pointer) 관련 메소드
      - * seek(n): 파일의 가장 첫번째 위치에서 n번째 바이트로 포인터 이동
      - * tell(): 파일 내 현재 포인터 위치를 반환

- 'r', 'w' → 첫 번째 위치 가리킴, 'a' → 맨 마지막 위치 가리킴

```
name = 't.txt'
f = open(name, 'w+')   # 읽고 쓰기로 오픈, 단, 파일이 이미 존재한다면
                                기존 파일은 없어지고 다시 생성된다.
s = '0123456789abcdef'
f.write(s)

f.seek(5)      # 시작부터 5바이트 포인터 이동
print f.tell()  # 현재 위치 돌려줌
print f.read(1) # 1문자 읽기
print f.tell()
print
```

```
5
5
6
```

- 'w+' → 파일을 쓰는데, 읽기도 가능함
- f.seek(5) → 문자열의 5번째 인덱스까지 파일 포인터 이동
- f.tell( ) → 현재 위치를 돌려줌
- f.read(1) → 현재 위치에 있는 문자 하나를 읽음
- 문자 하나를 읽었기 때문에 파일포인터 하나 이동

파이썬 프로그래밍

파일 입출력

한국기술교육대학교
온라인평생교육원

▣ 표준 출력 방향 전환

1. 표준 출력을 파일로 저장하기

- sys 모듈의 표준 입출력 관련 객체
  - sys.stdout: 표준 출력
  - sys.stderr: 표준 에러 출력
  - sys.stdin: 표준 입력
- 예를 들어, sys.stdout을 파일 객체로 변환하면 모든 표준 출력(print 출력)은 해당 파일로 저장된다.

```
import sys

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

- stdout 변수는 모니터(표준 출력) 화면에 해당하는 레퍼런스 값 가짐
- print → 표준 출력 (stdout 쪽으로 내용 출력)
- print → 표준 출력 (stdout 쪽으로 내용 출력)
- w → 쓰기 모드
- sys.stdout에서 stdout 변수는 sys 내에 존재하는 변수
- sys.stdout이 화면이 아닌 파일쪽(t.text) 출력이 됨
- print는 이제 sys.stdout로 내용 보내서 출력
- 파일쪽으로 출력이 되었던 부분을 다시 모니터로 복원

▣ 표준 출력 방향 전환

1. 표준 출력을 파일로 저장하기

```
f = open('t.txt')
print f.read()
```

```
Sample output
Good
Good
```

- print를 직접 이용하여 출력을 다른 객체로 전환하기

```
print >> sys.stderr, "Warning, action field not supplied"
```

```
Warning, action field not supplied
```

- >> (부등호 2개) → 표준 출력이 아닌 다른 쪽으로 출력
- stderr → 표준 에러
- stdout, stderr → 이클립스 내에서는 console로 내용 출력
- stderr → 빨간색으로 내용 출력함
- stderr → 표준 에러 쪽에 reference를 가지고 있는 객체

▣ 표준 출력 방향 전환

1. 표준 출력을 파일로 저장하기

- 동일 방법으로 표준 출력(print)을 파일 객체로 전환

```
f = open('t.txt', 'w')
print >> f, 'spam string'
f.close()

f = open('t.txt')
print f.read()
f.close()
```

```
spam string
```

- >> : 리다이랙트 기호

▣ 표준 출력 방향 전환

2. StringIO 모듈 사용하기

- StringIO 모듈의 StringIO 클래스 객체
  - 파일 객체처럼 입출력 가능한 문자열 객체
  - StringIO에 지원되는 메소드는 파일 객체가 지원하는 메소드와 거의 동일하다.
  - getvalue() 메소드
    * 현재까지 담아 놓은 전체 내용을 반환한다.

- 문자열을 자기가 받고 내보낼 수 있는 파일객체와 비슷한 객체

```
import StringIO

f = StringIO.StringIO()
f.write("abc")
f.seek(0)
s = f.read()
print s
print

s2 = f.getvalue()
print s2
```

```
abc

abc
```

- 하드 디스크에 존재하는 파일 X → 메모리에 존재하는 특정 영역
- StringIO 객체가 지니고 있는 영역
- abc를 썼으니까 파일포인터의 위치는 맨 뒤
- seek(0) → 파일포인터 위치를 앞으로 당김
- getvalue() → 현재 지니고 있는 전체 내용 반환

▣ 표준 출력 방향 전환

2. StringIO 모듈 사용하기

- 표준 출력으로 문자열 객체에 내용 작성하기

```
import sys
import StringIO

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

print type(f)
print 'Sample output'
print 'Good'
print 'Good'

sys.stdout = stdout
```

- 기존 표준출력 → f (파일이 아니라 StringIO 객체)

▣ 표준 출력 방향 전환

2. StringIO 모듈 사용하기

```
s = f.getvalue()

print 'Done-------'
print s
```

```
Done-------
<type 'instance'>
Sample output
Good
Good
```

- getvalue() → 현재 지니고 있는 전체 내용 반환

파이썬 프로그래밍

파일 입출력

한국기술교육대학교
온라인평생교육원

▣ 파일로의 지속 모듈

1~2. 지속성과 지속성 기능을 지원하는 모듈

- 지속성(Persistence)
  - 프로그램 내에 생성된 각종 객체들을 해당 프로그램 종료 이후에도 존재하게 만들고, 그것들을 동일한 또는 다른 프로그램에서 사용하는 기능
- 지속성 기능을 지원하는 모듈
  - DBM 관련 모듈
    * anydbm, dbm, gdbm, dbhash, dumbdbm
    * anydbm: 시스템에서 사용가능한 모듈 중 가장 최적의 모듈을 반환함
      - 기본적으로 dumbdbm을 반환한다
      - 사전 자료형을 사용하는 것과 동일한 방법으로 사용
  - pickle 모듈
    * 파이썬의 객체를 저장하는 일반화된 지속성 모듈
    * 파이썬의 기본 객체뿐만 아니라 사용자 정의의 복잡한 객체도 저장 가능
    * 기본적으로 텍스트 모드로 저장하지만 이진 모드로도 저장 가능

- 파이썬에서는 모든 것이 객체인데 메모리에 존재
- 메모리에 존재하기 때문에 프로그램 종료 시 삭제됨
- 지속성 관련된 모듈 활용 시 객체 파일 저장 가능
- 얼음과 같이 객체를 얼려 언제든지 사용 가능 = '지속성'
- DBM 관련 모듈 → 파이썬이 제공하는 내장자료형 저장에 최적화
- pickle 모듈 → 파이썬 내장자료형 + 사용자가 정의한 객체 저장 가능
- DBM 모듈보다는 pickle 모듈이 좀 더 일반적21w3ㄷ

▣ 파일로의 지속 모듈

1~2. 지속성과 지속성 기능을 지원하는 모듈

- 피클링(pickling) 모듈 사용하기

```
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('pickle.txt', 'w') # 파일 객체를 얻는다.

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

f = open('pickle.txt')

x,y = pickle.load(f) # 파일에서 읽어오기. 튜플의 내용을 x, y에 받는다.
print x # 사전
print y # 리스트
```

```
{'jim': 6851325, 'Joseph': 6584321, 'jack': 9465215, 'tom': 4358382}
['string', 1234, 0.2345]
```

- dump(Tuple, f) → 저장하고자 하는 Tuple을 f에 넣어 얼림
- dump() 함수의 반대 → load() 함수

▣ 파일로의 지속 모듈

1~2. 지속성과 지속성 기능을 지원하는 모듈

```
import pickle

class Simple: # 가장 단순한 클래스를 정의
    pass

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

f = open('pickle2.txt', 'w')
pickle.dump(s, f) # 인스턴스 저장
f.close()

f = open('pickle2.txt')
t = pickle.load(f) # 인스턴스 가져오기
print t.count
```

```
10
```

- class 정의 방법 → 'class' 키워드 사용
- 콜론(:)을 썼으므로 그 아래 몸체를 적겠다는 것
- pass → 몸체 적지 않고 끝내겠다는 의미
- s = Simple() → Simple 클래스가 가지고 있는 생성자 호출
- s. count = 10 → 인스턴스 내 새로운 변수를 정의
- pickle.dump(저장하고자 하는 객체, 저장하는 위치)
- pickle.load(저장했던 위치) → 저장했던 객체 불러옴