정보교육을 위한 파이썬

정보 탐색

Version 0.0.9-d2

저자: Charles Severance
번역: 이광춘, 한정수
(xwmooc)

Copyright © 2009- Charles Severance.

Printing history:

October 2013: Major revision to Chapters 13 and 14 to switch to JSON and use OAuth. Added new chapter on Visualization.

September 2013: Published book on Amazon CreateSpace

January 2010: Published book using the University of Michigan Espresso Book machine.

December 2009: Major revision to chapters 2-10 from *Think Python: How to Think Like a Computer Scientist* and writing chapters 1 and 11-15 to produce *Python for Informatics: Exploring Information*

June 2008: Major revision, changed title to *Think Python: How to Think Like a Computer Scientist*.

August 2007: Major revision, changed title to *How to Think Like a (Python) Programmer*.

April 2002: First edition of *How to Think Like a Computer Scientist*.

This work is licensed under a Creative Common Attribution-NonCommercial-ShareAlike 3.0 Unported License. This license is available at `creativecommons.org/licenses/by-nc-sa/3.0/`. You can see what the author considers commercial and non-commercial uses of this material as well as license exemptions in the Appendix titled Copyright Detail.

The LaTeX source for the *Think Python: How to Think Like a Computer Scientist* version of this book is available from `http://www.thinkpython.com`.

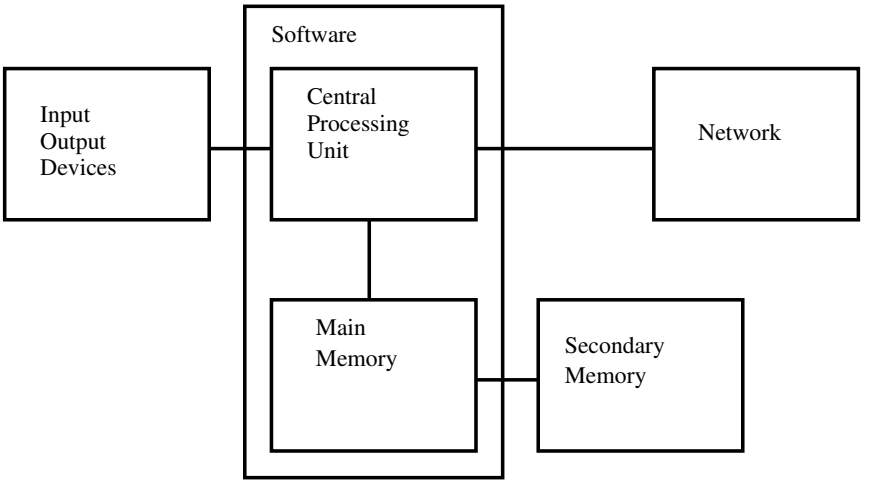
Chapter 1

파일

1.1 영속성(Persistence)

지금까지, 프로그램을 어떻게 작성하고 조건문 실행, 함수, 반복을 사용하여 **중앙처리장치(CPU, Central Processing Unit)**에 프로그래머의 의도를 커뮤니케이션하는 것을 학습했다. **주기억장치(Main Memory)**에 어떻게 자료구조를 생성하고 사용하는지를 배웠다. CPU와 주기억장치는 소프트웨어가 동작하고 실행하는 곳이고 모든 ”생각(thinking)”이 일어나는 곳이다.

하지만, 하드웨어 아키텍처를 논의했던 앞의 기억을 되살린다면, 전원이 꺼지게 되면, CPU와 주기억장치에 저장된 모든 것이 지워진다. 지금까지 작성한 프로그램은 파이썬을 배우기 위한 일시적으로 재미로 연습한 것이다.



이번 장에서는 **보조 기억장치(Secondary Memory)** 혹은 파일을 가지고 작업을 시작할 것이다. 보조 기억장치는 전원이 꺼져도 지워지지 않는다. 혹은, USB 플래쉬 드라이브를 사용한 경우에는 작성한 데이터는 프로그램으로부터 시스템에서 제거되어 다른 시스템으로 전송될 수 있다.

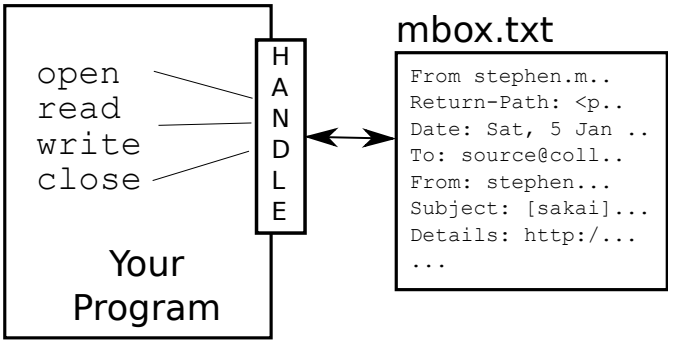
우선 텍스트 편집기로 작성한 텍스트 파일을 읽고 쓰는 것에 초점을 맞출 것이다. 나중에 데이터베이스 소프트웨어로 읽고 쓰도록 설계된 바이너리 파일인 데이터베이스와 어떻게 작업하는지를 보게 될 것이다.

1.2 파일 열기

하드 디스크의 파일을 읽거나 쓸려고 할 때, 파일을 **열여야** 한다. 파일을 여는 것은 운영체제와 커뮤니케이션하는데 운영체제는 각 파일의 데이터가 어디에 저장되었는지를 알고 있다. 여러분이 파일을 열 때, 운영체제는 파일이 존재하는 확인하고 이름으로 파일을 찾을 수 있게 요청한다. 이번 예제에서, 파이썬을 시작한 동일한 폴더에 저장된 `mbox.txt` 파일을 열 것이다. `www.py4inf.com/code/mbox.txt` 에서 파일은 다운로드할 수 있다.

```
>>> fhand = open('mbox.txt')
>>> print fhand
<open file 'mbox.txt', mode 'r' at 0x1005088b0>
```

`open`이 성공하면, 운영체제는 **파일 핸들러(file handle)**을 반환한다. **파일 핸들러(file handle)**은 파일에 담겨있는 실제 데이터가 아니고, 대신에 데이터를 읽을 수 있도록 "핸들(handle)"을 사용할 수 있도록 한다. 요청한 파일이 존재하고, 파일을 읽을 수 있는 적절한 권한이 있다면 이제 핸들이 여러분에게 주어졌다.



파일이 존재하지 않는다면, `open`은 트레이스백(traceback) 오류로 파일 열기를 실패하고, 파일에 존재하는 핸들도 얻지 못한다.

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'stuff.txt'
```

나중에 `try`와 `except`를 가지고, 존재하지 않는 파일을 열려고 하는 상황을 좀더 우아하게 처리할 것이다.

1.3 텍스트 파일과 라인

파이썬 문자열이 일련의 문자의 열로 생각하는 것과 마찬가지로 텍스트 파일은 일련의 라인으로 생각할 수 있다. 예를 들어, 다음은 오픈 소스 프로젝트 개발팀에서 다양한 참여자들의 전자우편 활동을 기록한 텍스트 파일 샘플이다.

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
...
```

상호 의사소통한 전자우편 전체 파일은 `www.py4inf.com/code/mbox.txt` 에서 가능하고, 간략한 버젼의 파일은 `www.py4inf.com/code/mbox.txt`에서 얻을 수 있다. 이들 파일은 여러개의 전자우편 메시지를 담고 있는 파일로 표준 표맷으로 되어 있다. "From "으로 시작하는 라인은 메시지 본문을 구별하고 "From: "으로 시작하는 줄은 본문 메시지의 일부다. 더 자세한 정보는 `en.wikipedia.org/wiki/Mbox`에서 찾을 수 있다.

파일을 라인으로 쪼개기 위해서, **새줄(newline)** 문자로 불리는 "줄의 끝(end of the line)"을 표시하는 특수 문자가 있다.

파이썬에서, 문자열 상수 역슬래쉬-n(`\n`)으로 **새줄(newline)** 문자를 표현한다. 두 문자처럼 보이지만, 사실은 단일 문자다. 인터프리터에 "stuff"을 입력한 변수를 보면, 문자열에 `\n`가 있다. 하지만, `print`문을 사용하여 문자열을 출력하면, 문자열이 새줄 문자에 의해서 두줄로 나누어지는 것을 볼 수 있다.

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print stuff
Hello
World!
>>> stuff = 'X\nY'
>>> print stuff
X
Y
>>> len(stuff)
3
```

문자열 `'X\nY'`의 길이는 *3* 문자다. 왜냐하면 새줄(newline) 문자는 한 문자이기 때문이다.

그래서, 파일의 라인을 볼 때, 라인의 끝을 표시하는 새줄(newline)로 불리는 눈에 보이지 않는 특수 문자가 각 줄의 끝에 있다고 상상할 필요가 있다.

그래서, 새줄(newline) 문자는 파일의 문자를 라인으로 분리한다.

1.4 파일 읽어오기

파일 핸들러(file handle)가 파일의 자료를 담고 있지 않지만, 파일의 각 라인을 읽고 라인수를 세기 위해서 `for` 루프를 사용하여 쉽게 구축할 수 있다.

```
fhand = open('mbox.txt')
count = 0
for line in fhand:
    count = count + 1
print 'Line Count:', count

python open.py
Line Count: 132045
```

파일 핸들을 for문 열에 사용할 수 있다. for문 단순하게 파일의 라인 수를 세고 총 라인수를 출력한다. for문을 대략 일반어로 풀어 말하면, "파일 핸들로 표현되는 파일의 각 라인마다, count변수에 1을 다하세요"

open 함수가 전체 파일을 바로 읽지 않는 이유는 파일이 수 기가 바이트 파일 크기를 가질 수도 있기 때문이다. open 문은 파일 크기에 관계없이 파일을 여는데 동일한 시간이 걸린다. for문이 파일로부터 자료를 읽어오는 역할을 한다.

파일을 이 같은 방식의 for문을 사용해서 읽어올 때, 파이썬은 새줄(newline) 문자를 사용해서 파일의 자료를 라인으로 쪼갠다. 파이썬은 새줄(newline)문자 구분되는 각 라인 단위로 읽어어고, for 루프가 매번 반복할 때마다 마지막 문자로 새줄(newline)을 line 변수에 추가한다.

for 문은 데이터를 한번에 한줄씩 읽어오기 때문에 데이터를 저장하기 위해서 주기억장치의 저장공간 부족없이 매우 큰 파일을 효과적으로 읽어서 라인을 셀 수 있다.

만약 주기억장치의 크기에 비해서 상대적으로 작은 크기의 파일이라는 것을 안다면, 전체 파일을 파일 핸들로 read 메쏘드를 사용해서 하나의 문자열로 읽을 수 있다.

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print len(inp)
94626
>>> print inp[:20]
From stephen.marquar
```

이 예제에서,mbox-short.txt 전체 파일 내용(94,626 문자)이 변수 inp에 바로 읽혀졌다. inp에 저장된 문자열 자료의 첫 20 문자를 출력하기 위해서 문자열 쪼개기를 사용했다.

파일이 이런 방식으로 읽혀질 때, 모든 라인과 새줄(newline)문자를 포함한 모든 문자는 변수 inp에 할당된 큰 문자열이다. 파일 데이터가 컴퓨터의 주기억장치에 안정적으로 감당해 낼 수 있다면, 이런 형식의 open 함수가 사용될 수 있다는 것을 기억하라.

주기억장치가 감당해 낼 수 없는 매우 큰 크기의 파일이라면, for나 while 루프를 사용해서 파일을 쪼개서 읽는 프로그램을 작성해야 한다.

1.5 파일을 통한 검색

파일의 데이터를 검색할 때, 파일을 읽고, 대부분의 줄은 넘어가고, 특정한 조건을 만족하는 라인만 처리하는 것이 흔한 패턴이다. 간단한 검색 메카니즘을 구현하기 위해서 파일을 읽어 들이는 패턴과 **문자열(methods)**를 조합하여 사용한다.

예를 들어, 파일을 읽고, “From:”으로 시작하는 라인만 출력하고자 한다면, 원하는 접두사로 시작하는 라인만을 선택하기 위해서 **startswith** 문자열 메쏘드를 사용한다.

```
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith('From:') :
        print line
```

이 프로그램이 실행될 때, 다음 출력값을 얻는다.

```
From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu
...
```

”From:”으로만 시작하는 라인만 출력하기 때문에 출력값은 훌륭해 보인다. 하지만, 추가적으로 왜 빈 라인이 보이는 걸까? 왜냐하면 눈에 보이지 않는 **새줄(newline)** 문자 때문이다. 각 라인이 새줄(newline)으로 끝나서 `print`문은 새줄(newline)을 포함하는 변수 **line**의 문자열을 출력한다. 그리고 나서 **print**문이 추가로 새줄(newline)을 추가해서 결국 우리가 보기에는 두 줄 효과가 나타난다.

마지막 문자를 제외한 라인을 출력하기 위해서 라인 쪼개기(slicing)를 할수 있지만, 좀더 간단한 접근법은 다음과 같이 문자열 오른쪽 끝에서부터 공백을 벗겨내는 **rstrip 메쏘드**를 사용하는 것인다.

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:') :
        print line
```

프로그램을 실행하면, 다음 출력값을 얻는다.

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
...
```

파일 처리 프로그램이 점점 더 복잡해짐에 따라 검색 루프(search loop)를 continue를 사용해서 구조화할 필요가 있다. 검색 루프의 기본 생각은 "흥미로운" 라인을 집중적으로 찾고, "흥미롭지 않은" 라인은 효과적으로 건너뛰는 것이다. 그리고 나서 흥미로운 라인을 찾게되면, 그 라인에서 무슨 연산을 수행하는 것이다.

흥미롭지 않은 라인은 건떠 뛰는 패턴을 따르는 루프를 다음과 같이 구성할 수 있다.

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    # Skip 'uninteresting lines'
    if not line.startswith('From:') :
        continue
    # Process our 'interesting' line
    print line
```

프로그램의 출력값은 동일하다. 흥미롭지 않는 라인은 "From:"으로 시작하는 라인이어서 continue문을 사용해서 건너뛴다. "흥미로운" 라인 (즉, "From:"으로 시작하는 라인)에 대해서는 연산처리를 수행한다.

find 문자열 메쏘드를 사용해서 검색문자열이 어는 라인에 있는지를 찾아주는 텍스트 편집기의 검색기능을 흉내낼 수 있다. find 메쏘드는 다른 문자열 내부에 찾는 문자열이 있는지 찾고, 문자열의 위치를 반환하거나, 만약 문자열이 없다면 -1을 반환하기 때문에, "@uct.ac.za"(남아프리카 케이프 타운 대학으로부터 왔다) 문자열을 포함하는 라인을 보이기 위해 다음과 같이 루프를 작성한다.

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.find('@uct.ac.za') == -1 :
        continue
    print line
```

출력결과는 다음과 같다.

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f
From: david.horwitz@uct.ac.za
Author: david.horwitz@uct.ac.za
...
```

1.6 사용자가 파일명을 고르게 만들기

다른 파일을 처리할 때마다 파이썬 코드를 편집하기를 원치 않는다. 매번 프로그램이 실행될 때마다, 파일명을 사용자가 입력하도록 요청하는 것이 좀더 유

용할 것이다. 그래서 파이썬 코드를 바꾸지 않고, 다른 파일에 대해서도 동일한 프로그램을 사용할 수 있다.

아래와 같이 raw_input을 사용해서 사용자로부터 파일명을 읽어서 수행하는 것이 간단하다.

```
fname = raw_input('Enter the file name: ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print 'There were', count, 'subject lines in', fname
```

사용자로부터 파일명을 일고 변수명으로 fname에 저장하고, 그 파일을 연다. 이제 다른 파일에 대해서 반복적으로 프로그램을 실행할 수 있다.

```
python search6.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python search6.py
Enter the file name: mbox-short.txt
There were 27 subject lines in mbox-short.txt
```

다음 절을 엿보기 전에, 상기 프로그램을 살펴보고 자신에게 다음을 질문해 보세요. "여기서 어디가 잘못될 수 있는가?" 혹은 "우리의 친절한 사용자가 멋진 작은 프로그램을 트레이스백(traceback)을 남기고 바로 끝날 수 있게 만들어 사용자의 눈에는 뭐 좋지 않은 프로그램이라는 인상을 남기기 위해서 무엇을 할 수 있을까?

1.7 try, except, open 사용하기

제가 여러분에게 엿보지 말라고 말씀드렸습니다. 이번이 마지막 기회입니다. 우리의 사용자가 파일명이 아닌 뭔가 다른 것을 입력하면 어떻습니까?

```
python search6.py
Enter the file name: missing.txt
Traceback (most recent call last):
  File "search6.py", line 2, in <module>
    fhand = open(fname)
IOError: [Errno 2] No such file or directory: 'missing.txt'

python search6.py
Enter the file name: na na boo boo
Traceback (most recent call last):
  File "search6.py", line 2, in <module>
    fhand = open(fname)
IOError: [Errno 2] No such file or directory: 'na na boo boo'
```

웃지마시구요, 사용자는 결국 여러분이 작성한 프로그램을 망가뜨리기 위해 고의든 악의를 가지든 가능한 모든 수단을 강구할 것입니다. 사실, 소프트웨어 개

발팀의 중요한 부문은 **품질 보증(Quality Assurance, QA)**이라는 조직이다. 품질보증 조직은 프로그래머가 만든 소프트웨어를 망가뜨리기 위해 가능한 말도 안 되는 것을 합니다.

품질보증 조직은 소프트웨어를 제품으로 구매하거나 작성한 프로그램에 대해 월급을 지급하는 사용자에게 프로그램이 전달되기 전까지 프로그램의 오류를 발견하는 것에 책임이 있다. 그래서 품질보증 조직은 프로그래머의 최고의 친구다.

프로그램의 오류를 찾았기 때문에, `try`/`except` 구조를 사용해서 오류를 우아하게 고쳐봅시다. `open` 호출이 잘못될 수 있다고 가정하고, `open` 호출이 실패할 때 다음과 같이 복구 코드를 추가한다.

```
fname = raw_input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print 'File cannot be opened:', fname
    exit()

count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print 'There were', count, 'subject lines in', fname
```

`exit`함수는 프로그램을 끝낸다. 결코 돌아오지 않는 함수를 호출한 것이다. 이제 사용자 혹은 품질 보증 조직에서 옳지 않거나 어처구니 없는 파일명을 입력했을 때, “catch”로 잡아서 우아하게 복구한다.

```
python search7.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python search7.py
Enter the file name: na na boo boo
File cannot be opened: na na boo boo
```

`open` 호출을 보호하는 것은 파이썬 프로그램을 작성할 때 `try`, `except`의 적절한 사용 예제가 된다. 파이썬 방식(”Python way”)으로 무언가를 작성할 때, ”파이썬스럽다(Pythonic)”이라는 용어를 사용한다. 상기 파일을 여는 것은 파이썬스러운 방식의 좋은 예가 된다고 말한다.

파이썬에 좀더 자신감이 붙게되면, 다른 파이썬 프로그래머와 동일한 문제에 대해서 두 가지 동치하는 해답을 가지고 어는 것이 좀더 ”파이썬스러운지”에 대한 현답을 찾는데 관여하게 된다.

”좀더 파이썬스럽게” 되는 이유는 프로그램밍이 엔지니어링적인 면과 예술적인 면을 동시에 가지고 있기 때문이다. 항상 무언가를 단지 작동하는 것에만 관심이 있지 않고, 프로그램으로 작성한 해결책이 좀더 우아하고, 다른 동료에 의해서 우아한 것으로 인정되기를 또한 원합니다.

1.8 파일에 쓰기

파일에 쓰기 위해서는 두 번째 매개 변수로 'w' 모드로 파일을 열어야 합니다.

```
>>> fout = open('output.txt', 'w')
>>> print fout
<open file 'output.txt', mode 'w' at 0xb7eb2410>
```

파일이 이미 존재한다면, 쓰기 모드에서 여는 것은 과거 데이터를 모두 지워버리고, 파일이 깨끗한 다시 시작되니 주의가 필요합니다. 만약 파일이 존재하지 않는다면, 새로운 파일이 생성됩니다.

파일 핸들 개체의 write 메쏘드는 데이터를 파일에 저장합니다.

```
>>> line1 = 'This here's the wattle,\n'
>>> fout.write(line1)
```

다시, 파일 개체는 어디에 마지막 포인터가 있는지 위치를 추적해서, 만약 write 메쏘드를 다시 호출하게 되면, 새로운 데이터를 파일 끝에 추가합니다.

라인을 끝내고 싶을 때 명시적으로 새줄(newline) 문자를 삽입해서 파일에 쓰도록 라인 끝을 관리하도록 꼭 확인해야 합니다.

print문은 자동적으로 새줄(newline)을 추가하지만, write 메쏘는 자동적으로 새줄(newline)을 추가하지는 않습니다.

```
>>> line2 = 'the emblem of our land.\n'
>>> fout.write(line2)
```

파일에 쓰기가 끝났을 때, 파일을 필히 닫아야 합니다. 파일을 닫는 것은 데이터의 마지막 비트까지 디스크에 물리적으로 쓰여져서, 전원이 나가더라도 자료가 유실되지 않도록 하는 역할을 합니다.

```
>>> fout.close()
```

파일 읽기로 연 파일을 닫을 수 있지만, 몇개의 파일을 열어논다면 약간 단정치 못한 것으로 끝날 수 있습니다. 왜냐하면 프로그램이 종료될 때 열린 모든 파일을 닫도록 파이썬이 자동으로 확인합니다. 파일에 쓰기를 할 때는, 파일을 명시적으로 닫아서 다른 일들이 발생할 여지를 없애야 합니다.

1.9 디버깅

파일을 읽고 쓸 때, 공백으로 문제에 종종 봉착합니다. 이런 종류의 오류는 공백, 탭, 새줄(newline)이 눈에 보이지 않기 때문에 디버그하기가 쉽지 않습니다.

```
>>> s = '1 2\t 3\n 4'
>>> print s
1 2  3
 4
```

내장함수 repr이 도움이 될 수 있습니다. 인수로 임의의 개체를 잡아 개체의 문자열 표현으로 반환합니다. 문자열의 공백문자는 역슬래쉬 열로 나타냅니다.

```
>>> print repr(s)
'1 2\t 3\n 4'
```

디버깅에 도움이 될 수 있습니다.

여러분이 봉착하는 또 다른 문제는 다른 시스템은 라인의 끝을 표기하기 위해서 다른 문자를 사용합니다. 어떤 시스템은 \n으로 새줄(newline)을 표기하고, 다른 시스템은 \r으로 반환 문자(return character)를 사용합니다. 둘다 사용하는 시스템도 있습니다. 파일을 다른 시스템으로 옮긴다면, 이러한 불일치가 문제를 야기합니다.

대부분의 시스템에는 A 포맷에서 B 포멧으로 변환하는 응용프로그램이 있습니다. wikipedia.org/wiki/Newline 에서 응용프로그램을 찾을 수 있고, 좀더 많은 것을 읽을 수 있다. 물론, 여러분이 직접 프로그램을 작성할 수도 있다.

1.10 용어정의

잡기(catch): try와 except 문을 사용해서 프로그램이 끝나는 예외 상황을 방지하는 것.

새줄(newline): 라인의 끝을 표기 위한 파일이나 문자열에 사용되는 특수 문자.

파이썬스러운(Pythonic): 파이썬에서 우아하게 작동하는 기술. "try와 catch를 사용하는 것은 파일이 없는 경우를 복구하는 파이썬스러운 방식이다."

품질 보증(Quality Assurance, QA): 소프트웨어 제품의 전반적인 품질을 보증하는데 집중하는 사람이나 조직. 품질 보증은 소프트웨어 제품을 시험하고, 제품이 시장에 출시되기 전에 문제를 확인하는데 관여한다.

텍스트 파일(text file): 하드디스크 같은 영구 저장소에 저장된 일련의 문자 집합.

1.11 연습문제

Exercise 1.1 파일을 읽고 한줄씩 파일의 내용을 모두 대문자로 출력하는 프로그램을 작성하세요. 프로그램을 싱행하면 다음과 같이 보일 것입니다.

```
python shout.py
Enter a file name: mbox-short.txt
FROM STEPHEN.MARQUARD@UCT.AC.ZA SAT JAN  5 09:14:16 2008
RETURN-PATH: <POSTMASTER@COLLAB.SAKAIPROJECT.ORG>
RECEIVED: FROM MURDER (MAIL.UMICH.EDU [141.211.14.90])
 BY FRANKENSTEIN.MAIL.UMICH.EDU (CYRUS V2.3.8) WITH LMTPA;
 SAT, 05 JAN 2008 09:14:16 -0500
```

You can download the file from www.py4inf.com/code/mbox-short.txt

Exercise 1.2 파일명을 입력받아, 파일을 일고, 다음 형식의 라인을 찾는 프로그램을 작성하세요.

```
X-DSPAM-Confidence: 0.8475
```

"X-DSPAM-Confidence:"로 시작하는 라인을 만나게 되면, 부동 소수점 숫자를 뽑아내기 위해 해당 라인을 별도로 보관하세요. 라인의 수를 세고, 라인으로부터 스팸 신뢰값의 총계를 계산하세요. 파일의 끝에 도달할 했을 때, 평균 스팸 신뢰도를 출력핫에ㅛ.

```
Enter the file name: mbox.txt
Average spam confidence: 0.894128046745

Enter the file name: mbox-short.txt
Average spam confidence: 0.750718518519
```

mbox.txt과 mbox-short.txt 파일에 프로그램을 시험하세요.

Exercise 1.3 때때로, 프로그래머가 지루하거나, 약간의 재미를 목적으로, 프로그램에 무해한 **부활절 달걀(Easter Egg)**를 넣습니다. (en.wikipedia.org/wiki/Easter_egg_(media)) 사용자가 파일명을 입력하는 프로그램을 변형시켜, 'na na boo boo'로 파일명을 정확하게 입력했을 때, 재미난 메시지를 출려하는 프로그램을 작성하세요. 파일이 존재하거나, 존재하지 않는 모든 파일에 대해서 정상적으로 작동해야 합니다. 여기 프로그램을 실행한 견본이 있습니다.

```
python egg.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python egg.py
Enter the file name: missing.tyxt
File cannot be opened: missing.tyxt

python egg.py
Enter the file name: na na boo boo
NA NA BOO BOO TO YOU - You have been punk'd!
```

프로그램에 부활절 달걀을 넣도록 격려하지는 않습니다. 단지 연습입니다.