Think Python

-Korean ver.4-

㈜ 비트시스 [대표 김동칠]

동의대학교 블록체인 인력양성 사업단

부산광역시 BUSAN METROPOLITAN CITY

부산인재평생교육진흥원 BIT Busan Institute for Talent & Lifelong Education

BitSys 행복한 꿈을 공유하다

제 11 장

정규 표현식

지금까지 파일을 훑어서 패턴을 찾고, 관심있는 라인에서 다양한 비트(bits)를 뽑아냈다. `split`, `find` 같은 문자열 메쏘드를 사용하였고, 라인에서 일정 부분을 뽑아내기 위해서 리스트와 문자열 슬라이싱(slicing)을 사용했다.

검색하고 추출하는 작업은 너무 자주 있는 일이어서 파이썬은 상기와 같은 작업을 매우 우아하게 처리하는 정규 표현식(regular expressions)으로 불리는 매우 강력한 라이브러리를 제공한다. 정규 표현식을 책의 앞부분에 소개하지 않은 이유는 정규 표현식 라이브러리가 매우 강력하지만, 약간 복잡하고, 구문에 익숙해지는데 시간이 필요하기 때문이다.

정규표현식은 문자열을 검색하고 파싱하는데 그 자체가 작은 프로그래밍 언어다. 사실, 책 전체가 정규 표현식을 주제로 쓰여진 책이 몇권 있다. 이번 장에서는 정규 표현식의 기초만을 다룰 것이다. 정규 표현식의 좀더 자세한 사항은 다음을 참조하라.

http://en.wikipedia.org/wiki/Regular_expression

http://docs.python.org/library/re.html

정규 표현식 라이브러리는 사용하기 전에 프로그램을 가져오기(import)해야 한다. 정규 표현식 라이브러리의 가장 간단한 쓰임은 `search()` 검색 함수다. 다음 프로그램은 검색 함수의 사소한 사용례를 보여준다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line) :
        print line
```

파일을 열고, 각 라인을 루프로 반복해서 정규 표현식 `search()` 메쏘드를 호출하여 문자열 "From"이 포함된 라인만 출력한다. 상기 프로그램은 진정으로 강력한 정규 표현식 기능을 사용하지 않았다. 왜냐하면, `line.find()` 메쏘드를 가지고 동일한 결과를 쉽게 구현할 수 있기 때문이다.

정규 표현식의 강력한 기능은 문자열에 해당하는 라인을 좀더 정확하게 제어하기 위해서 검색 문자열에 특수문자를 추가할 때 확인될 수 있다. 매우 적은 코드를 작성할지라도, 정규 표현식에 특수 문자를 추가하는 것만으로도 정교한 일치(matching)와 추출이 가능하게 한다.

예를 들어, 탈자 기호(caret)는 라인의 "시작"과 일치하는 정규 표현식에 사용된다. 다음과 같이 "From:"으로 시작하는 라인만 일치하도록 응용프로그램을 변경할 수 있다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line) :
        print line
```

"From:" 문자열로 *시작*하는 라인만 일치할 수 있다. 여전히 매우 간단한 프로그램으로 문자열 라이브러리에서 `startswith()` 메쏘드로 동일하게 수행할 수 있다. 하지만, 무엇을 정규 표현식과 매칭하는가에 대해 특수 액션 문자를 담아 좀더 많은 제어를 할수 있게 하는 정규 표현식 개념을 소개하기에는 충분하다.

제 1 절 정규 표현식의 문자 매칭

좀더 강력한 정규 표현식을 작성할 수 있는 다른 특수문자는 많이 있다. 가장 자주 사용되는 특수 문자는 임의 문자를 매칭하는 마침표다.

다음 예제에서 정규 표현식 "F..m:"은 "From:", "Fxxm:", "F12m:", "F!@m:' 같은 임의 문자열을 매칭한다. 왜냐하면 정규 표현식 마침표 문자가 임의의 문자와 매칭되기 때문이다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^F..m:', line) :
        print line
```

정규 표현식에 "*", "+' ' 문자를 사용하여 문자가 원하는만큼 반복을 나타내는 기능과 결합되었을 때는 더욱 강력해진다. "*", "+' ' 특수 문자가 검색 문자열에 문자 하나만을 매칭하는 대신에 별표 기호인 경우 0 혹은 그 이상의 매칭, 더하기 기호인 경우 1 혹은 그 이상의 문자의 매칭을 의미한다.

다음 예제에서 반복 와일드 카드(wild card) 문자를 사용하여 매칭하는 라인을 좀더 좁힐 수 있다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:.+@', line) :
        print line
```

검색 문자열 "^From:.+@" 은 "From:" 으로 시작하고, ".+" 하나 혹은 그 이상의 문자들, 그리고 @ 기호와 매칭되는 라인을 성공적으로 찾아낸다. 그래서 다음 라인은 매칭이 될 것이다.

From: stephen.marquard @uct.ac.za

콜론(:)과 @ 기호 사이의 모든 문자들을 매칭하도록 확장하는 것으로 ".+" 와이드 카드를 간주할 수 있다.

From: .+ @

더하기와 별표 기호를 "밀어내기(pushy)" 문자로 생각하는 것이 좋다. 예를 들어, 다음 문자열은 ".+" 특수문자가 다음에 보여주듯이 밖으로 밀어내는 것처럼 문자열 마지막 @ 기호를 매칭한다.

From: stephen.marquard@uct.ac.za, csev@umich.edu, and cwen @iupui.edu

다른 특수문자를 추가함으로서 별표나 더하기 기호가 너무 "탐욕(greedy)"스럽지 않게 만들 수 있다. 와일드 카드 특수문자의 탐욕스러운 기능을 끄는 것에 대해서는 자세한 정보를 참조바란다.

제 2 절 정규 표현식 사용 데이터 추출

파이썬으로 문자열에서 데이터를 추출하려면, findall() 메쏘드를 사용해서 정규 표현식과 매칭되는 모든 부속 문자열을 추출할 수 있다. 형식에 관계없이 임의 라인에서 전자우편 주소 같은 문자열을 추출하는 예제를 사용하자. 예를 들어, 다음 각 라인에서 전자우편 주소를 뽑아내고자 한다.

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
          for <source@collab.sakaiproject.org>;
Received: (from apache@localhost)
Author: stephen.marquard@uct.ac.za
```

각각의 라인에 대해서 다르게 쪼개고, 슬라이싱하면서 라인 각각의 형식에 맞추어 코드를 작성하고는 싶지는 않다. 다음 프로그램은 findall() 메쏘드를 사용하여 전자우편 주소가 있는 라인을 찾아내고 하나 혹은 그 이상의 주소를 뽑아낸다.

```
import re
s = 'Hello from csev@umich.edu to cwen@iupui.edu about the meeting @2PM'
lst = re.findall('\S+@\S+', s)
print lst
```

findall() 메쏘드는 두번째 인자 문자열을 찾아서 전자우편 주소처럼 보이는 모든 문자열을 리스트로 반환한다. 공백이 아닌 문자 (\S)와 매칭되는 두 문자 순서(sequence)를 사용한다.

프로그램의 출력은 다음과 같다.

```
['csev@umich.edu', 'cwen@iupui.edu']
```

정규 표현식을 해석하면, 적어도 하나의 공백이 아닌 문자, @과 적어도 하나 이상의 공백이 아닌 문자를 가진 부속 문자열을 찾는다. 또한, "\S+" 특수 문자는 가능한 많이 공백이 아닌 문자를 매칭한다. (정규 표현식에서 "탐욕(greedy)" 매칭이라고 부른다.)

정규 표현식은 두번 매칭(csev@umich.edu, cwen@iupui.edu)하지만, 문자열 "@2PM"은 매칭을 하지 않는다. 왜냐하면, @ 기호 앞에 공백이 아닌 문자가 하나도 없기 때문이다. 프로그램의 정규 표현식을 사용해서 파일에 모든 라인을 읽고 다음과 같이 전자우편 주소처럼 보이는 모든 문자열을 출력한다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('\S+@\S+', line)
    if len(x) > 0 :
        print x
```

각 라인을 읽어 들이고, 정규 표현식과 매칭되는 모든 부속 문자열을 추출한다. findall() 메쏘드는 리스트를 반환하기 때문에, 전자우편 처럼 보이는 부속 문자열을 적어도 하나 찾아서 출력하기 위해서 반환 리스트 요소 숫자가 0 보다 큰지 여부를 간단히 확인한다.

mbox.txt 파일에 프로그램을 실행하면, 다음 출력을 얻는다.

```
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['<postmaster@collab.sakaiproject.org>']
['<200801032122.m03LMFo4005148@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
```

전자우편 주소 몇몇은 "<", ";" 같은 잘못된 문자가 앞과 뒤에 붙어있다. 문자나 숫자로 시작하고 끝나는 문자열 부분만 관심있다고 하자.

그러기 위해서, 정규 표현식의 또 다른 기능을 사용한다. 매칭하려는 다중 허용 문자 집합을 표기하기 위해서 꺾쇠 괄호를 사용한다. 그런 의미에서 "\S"은 공백이 아닌 문자 집합을 매칭하게 한다. 이제 매칭하려는 문자에 관해서 좀더 명확해졌다.

여기 새로운 정규 표현식이 있다.

```
[a-zA-Z0-9]\S*@\S*[a-zA-Z]
```

약간 복잡해졌다. 왜 정규 표현식이 자신만의 언어인가에 대해서 이해할 수 있다. 이 정규 표현식을 해석하면, 0 혹은 그 이상의 공백이 아닌 문자("\S*")로 하나의 소문자, 대문자 혹은 숫자("[a-zA-Z0-9]")를 가지며, @ 다음에 0 혹은 그 이상의 공백이 아닌 문자("\S*")로 하나의 소문자, 대문자 혹은 숫자("[a-zA-Z0-9]")로 된 부속 문자열을 찾는다. 0 혹은 그 이상의 공백이 아닌 문자를 나타내기 위해서 "+"에서 "*"으로 바꿨다. 왜냐하면 "[a-zA-Z0-9]" 자체가 이미

하나의 공백이 아닌 문자이기 때문이다. “*”, “+”는 단일 문자에 별표, 더하기 기호 왼편에 즉시 적용됨을 기억하세요.

프로그램에 정규 표현식을 사용하면, 데이터가 훨씬 깔끔해진다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('[a-zA-Z0-9]\S*@\S*[a-zA-Z]', line)
    if len(x) > 0 :
        print x

...
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['postmaster@collab.sakaiproject.org']
['200801032122.m03LMFo4005148@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
```

“source@collab.sakaiproject.org” 라인에서 문자열 끝에 “>;” 문자를 정규 표현식으로 제거한 것을 주목하세요. 정규 표현식 끝에 “[a-zA-Z]”을 추가하여서 정규 표현식 파서가 찾는 임의 문자열은 문자로만 끝나야 되기 때문이다. 그래서, “sakaiproject.org>;”에서 “>”을 봤을 때, ”g”가 마지막 맞는 매칭이 되고, 거기서 마지막 매칭을 마치고 중단한다.

프로그램의 출력은 리스트의 단일 요소를 가진 문자열로 파이썬 리스트이다.

제 3 절 검색과 추출 조합하기

다음과 같은 “X-” 문자열로 시작하는 라인의 숫자를 찾고자 한다면,

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
```

임의의 라인에서 임의 부동 소수점 숫자가 아니라 상기 구문을 가진 라인에서만 숫자를 추출하고자 한다.

라인을 선택하기 위해서 다음과 같이 정규 표현식을 구성한다.

```
^X-.*: [0-9.]+
```

정규 표현식을 해석하면, ' ' ^ ' ' 에서 “X-”으로 시작하고, “.*”에서 0 혹은 그이상의 문자를 가지며, 콜론(”:”)이 나오고 나서 공백을 만족하는 라인을 찾는다. 공백 뒤에 “[0-9.]+”에서 숫자 (0-9) 혹은 점을 가진 하나 혹은 그 이상의 문자가 있여야 한다. 꺽쇠 기호 사이에 마침표는 실제 마침표만 매칭함을 주목하기 바란다. (즉, 꺾쇠 기호 사이는 와일드 카드 문자가 아니다.)

관심을 가지고 있는 특정한 라인과 매우 정확하게 매칭이되는 매우 빠듯한 정규 표현식으로 다음과 같다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^X\S*: [0-9.]+', line) :
        print line
```

프로그램을 실행하면, 잘 걸러져서 찾고자 하는 라인만 볼 수 있다.

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000
```

하지만, 이제 `rsplit` **사용해서 숫자를 뽑아내는 문제를 해결해야 한다.** `rsplit` **을 사용하는 것이 간단해 보이지만, 동시에 라인을 검색하고 파싱하기 위해서 정규 표현식의 또 다른 기능을 사용할 수 있다.**

괄호는 정규 표현식의 또 다른 특수 문자다. 정규 표현식에 괄호를 추가한다면, 문자열이 매칭될 때, 무시된다. 하지만, `findall()` **을 사용할 때, 매칭할 전체 정규 표현식을 원할지라도, 정규 표현식을 매칭하는 부속 문자열의 부분만을 뽑아낸다는 것을 괄호가 표시한다.**

그래서, 프로그램을 다음과 같이 수정한다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^X\S*: ([0-9.]+)', line)
    if len(x) > 0 :
        print x
```

`search()` **을 호출하는 대신에, 매칭 문자열의 부동 소수점 숫자만 뽑아내는** `findall()` **에 원하는 부동 소수점 숫자를 표현하는 정규 표현식 부분에 괄호를 추가한다.**

프로그램의 출력은 다음과 같다.

```
['0.8475']
['0.0000']
['0.6178']
['0.0000']
['0.6961']
['0.0000']
..
```

숫자가 여전히 리스트에 있어서 문자열에서 부동 소수점으로 변환할 필요가 있지만, 흥미로운 정보를 찾아 뽑아내기 위해서 정규 표현식의 강력한 힘을 사용했다.

이 기술을 활용한 또 다른 예제로, 파일을 살펴보면, 폼(form)을 가진 라인이 많다.

```
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

상기 언급한 동일한 기법을 사용하여 모든 변경 번호(라인의 끝에 정수 숫자)를 추출하고자 한다만, 다음과 같이 프로그램을 작성할 수 있다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^Details:.*rev=([0-9.]+)', line)
    if len(x) > 0:
        print x
```

작성한 정규 표현식을 해석하면, "Details:"로 시작하는 ".*"에 임의의 문자들로, "rev="을 포함하고 나서, 하나 혹은 그 이상의 숫자를 가진 라인을 찾는다. 전체 정규 표현식을 만족하는 라인을 찾고자 하지만, 라인 끝에 정수만을 추출하기 위해서 "[0-9]+"을 괄호로 감쌌다.

프로그램을 실행하면, 다음 출력을 얻는다.

```
['39772']
['39771']
['39770']
['39769']
...
```

"[0-9]+"은 "탐욕(greedy)"스러워서, 숫자를 추출하기 전에 가능한 큰 문자열 숫자를 만들려고 한다는 것을 기억하라. 이런 "탐욕(greedy)"스러운 행동으로 인해서 왜 각 숫자로 모두 5자리 숫자를 얻은 이유가 된다. 정규 표현식 라이브러리는 양방향으로 파일 처음이나 끝에 숫자가 아닌 것을 마주칠 때까지 뻗어 나간다.

이제 정규 표현식을 사용해서 각 전자우편 메시지의 요일에 관심이 있었던 책 앞의 연습 프로그램을 다시 작성한다. 다음 형식의 라인을 찾는다.

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

그리고 나서, 각 라인의 요일의 시간을 추출하고자 한다. 앞에서 `split`를 두번 호출하여 작업을 수행했다. 첫번째는 라인을 단어로 쪼개고, 다섯번째 단어를 뽑아내서, 관심있는 두 문자를 뽑아내기 위해서 콜론 문자에서 다시 쪼갰다.

작동을 할지 모르지만, 실질적으로 정말 부서지기 쉬운 코드로 라인이 잘 짜여져 있다고 가정하에 가능하다. 잘못된 형식의 라인이 나타날 때도 결코 망가지지 않는 프로그램을 담보하기 위해서 충분한 오류 검사기능을 추가하거나 커다란 try/except 블록을 넣으면, 참 읽기 힘든 10-15 라인 코드로 커질 것이다.

다음 정규 표현식으로 훨씬 간결하게 작성할 수 있다.

```
^From .* [0-9][0-9]:
```

상기 정규 표현식을 해석하면, 공백을 포함한 "From "으로 시작해서, ".*"에 임의 갯수의 문자, 그리고 공백, 두 개의 숫자 "[0-9][0-9]" 뒤에 콜론(:) 문자를 가진 라인을 찾는다. 일종의 찾고 있는 라인에 대한 정의다.

findall()을 사용해서 단지 시간만 뽑아내기 위해서, 두 숫자에 괄호를 다음과 같이 추가한다.

```
^From .* ([0-9][0-9]):
```

작업 결과는 다음과 같이 프로그램에 반영한다.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^From .* ([0-9][0-9]):', line)
    if len(x) > 0 : print x
```

프로그램을 실행하면, 다음 출력 결과가 나온다.

```
['09']
['18']
['16']
['15']
...
```

제 4 절 이스케이프(Escape) 문자

라인의 처음과 끝을 매칭하거나, 와일드 카드를 명세하기 위해서 정규 표현식의 특수 문자를 사용했기 때문에, 정규 표현식에 사용된 문자가 "정상(normal)"적인 문자임을 표기할 방법이 필요하고 달러 기호와 탈자 기호(^) 같은 실제 문자를 매칭하고자 한다.

역슬래쉬(')를 가진 문자를 앞에 덮붙여서 문자를 단순히 매칭하고자 한다고 나타낼 수 있다. 예를 들어, 다음 정규표현식으로 금액을 찾을 수 있다.

```
import re
x = 'We just received $10.00 for cookies.'
y = re.findall('\$[0-9.]+',x)
```

역슬래쉬 달러 기호를 앞에 덮붙여서, 실제로 "라인 끝(end of line)" 매칭 대신에 입력 문자열의 달러 기호와 매칭한다. 정규 표현식 나머지 부분은 하나 혹은 그 이상의 숫자 혹은 소수점 문자를 매칭한다. *주목:* 꺾쇠 괄호 내부에 문자는 "특수 문자"가 아니다. 그래서 "[0-9.]"은 실제 숫자 혹은 점을 의미한다. 꺾쇠 괄호 외부에 점은 "와일드 카드(wild-card)" 문자이고 임의의 문자와 매칭한다. 꺾쇠 괄호 내부에서 점은 점일 뿐이다.

제 5 절 요약

지금까지 정규 표현식의 표면을 긁은 정도지만, 정규 표현식 언어에 대해서 조금 학습했다. 정규 표현식은 특수 문자로 구성된 검색 문자열로 "매칭(matching)"

정의하고 매칭된 문자열로부터 추출된 결과물을 정규 표현식 시스템과 프로그래머가 의도한 바를 의사소통하는 것이다. 다음에 특수 문자 및 문자 시퀀스의 일부가 있다.

^
라인의 처음을 매칭.

$
라인의 끝을 매칭.

.
임의의 문자를 매칭(와일드 카드)

\s
공백 문자를 매칭.

\S
공백이 아닌 문자를 매칭.(\s 의 반대).

*
바로 앞선 문자에 적용되고 0 혹은 그 이상의 앞선 문자와 매칭을 표기함.

*?
바로 앞선 문자에 적용되고 0 혹은 그 이상의 앞선 문자와 매칭을 "탐욕적이지 않은(non-greedy) 방식"으로 표기함.

+
바로 앞선 문자에 적용되고 1 혹은 그 이상의 앞선 문자와 매칭을 표기함.

+?
바로 앞선 문자에 적용되고 1 혹은 그 이상의 앞선 문자와 매칭을 "탐욕적이지 않은(non-greedy) 방식"으로 표기함.

[aeiou]
명세된 집합 문자에 존재하는 단일 문자와 매칭. 다른 문자는 안되고, "a", "e", "i", "o", "u" 문자만 매칭되는 예제.

[a-z0-9]
음수 기호로 문자 범위를 명세할 수 있다. 소문자이거나 숫자인 단일 문자만 매칭되는 예제.

[^A-Za-z]
집합 표기의 첫문자가 ^인 경우, 로직을 거꾸로 적용한다. 대문자나 혹은 소문자가 아닌 임의 단일 문자만 매칭하는 예제.

( )
괄호가 정규표현식에 추가될 때, 매칭을 무시한다. 하지만 `findall()`을 사용할 때 전체 문자열보다 매칭된 문자열의 상세한 부속 문자열을 추출할 수 있게 한다.

\b
빈 문자열을 매칭하지만, 단어의 시작과 끝에만 사용된다.

\B
빈 문자열을 매칭하지만, 단어의 시작과 끝이 아닌 곳에 사용된다.

\d
임의 숫자와 매칭하여 [0-9] 집합에 상응함.

\D
임의 숫자가 아닌 문자와 매칭하여 [^0-9] 집합에 상응함.

제 6 절 유닉스 사용자를 위한 보너스

정규 표현식을 사용하여 파일을 검색 기능은 1960년대 이래로 유닉스 운영 시스템에 내장되어 여러가지 형태로 거의 모든 프로그래밍 언어에서 이용가능하다.

사실, `search()` **예제에서와 거의 동일한 기능을 하는 grep (Generalized Regular Expression Parser)으로 불리는 유닉스 내장 명령어 프로그램이 있다. 그래서, 맥킨토시나 리눅스 운영 시스템을 가지고 있다면, 명령어 창에서 다음 명령어를 시도할 수 있다.**

```
$ grep '^From:' mbox-short.txt
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
```

`grep`**을 사용하여,** `mbox-short.txt` **파일 내부에 "From:" 문자열로 시작하는 라인을 보여준다.** `grep` **명령어를 가지고 약간 실험을 하고** `grep`**에 대한 문서를 읽는다면, 파이썬에서 지원하는 정규 표현식과** `grep`**에서 지원되는 정규 표현식과 차이를 발견할 것이다. 예를 들어,** `grep` **공백이 아닌 문자 "\S"을 지원하지 않는다. 그래서 약간 더 복잡한 집합 표기 "[^]"을 사용해야 한다. "[^]"은 간단히 정리하면, 공복을 제외한 임의의 문자와 매칭한다.**

제 7 절 디버깅

만약 특정 메쏘드의 정확한 이름을 기억해 내기 위해서 빠르게 생각나게 하는 것이 필요하다면 도움이 많이 될 수 있는 간단하고 초보적인 내장 문서가 파이썬에 포함되어 있다. 내장 문서 도움말은 인터랙티브 모드의 파이썬 인터프리터에서 볼 수 있다.

`help()`**를 사용하여 인터랙티브 도움을 받을 수 있다.**

```
>>> help()

Welcome to Python 2.6!  This is the online help utility.

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/tutorial/.
```

```
Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",
"keywords", or "topics".  Each module also comes with a one-line summary
of what it does; to list the modules whose summaries contain a given word
such as "spam", type "modules spam".

help> modules
```

특정한 모듈을 사용하고자 한다면, `dir()` **명령어를 사용하여 다음과 같이 모듈의 메쏘드를 찾을 수 있다.**

```
>>> import re
>>> dir(re)
[.. 'compile', 'copy_reg', 'error', 'escape', 'findall',
'finditer', 'match', 'purge', 'search', 'split', 'sre_compile',
'sre_parse', 'sub', 'subn', 'sys', 'template']
```

또한, `dir()` **명령어를 사용하여 특정 메쏘드에 대한 짧은 문서 도움말을 얻을 수 있다.**

```
>>> help (re.search)
Help on function search in module re:

search(pattern, string, flags=0)
    Scan through string looking for a match to the pattern, returning
    a match object, or None if no match was found.
>>>
```

내장 문서는 광범위하지 않아서, 급하거나, 웹 브라우저나 검색엔진에 접근할 수 없을 때 도움이 될 수 있다.

제 8 절 용어정의

부서지기 쉬운 코드(brittle code): 입력 데이터가 특정한 형식일 경우에만 작동하는 코드. 하지만 올바른 형식에서 약간이도 벗아나게 되면 깨지기 쉽니다. 쉽게 부서지기 때문에 ”부서지기 쉬운 코드(brittle code)”라고 부른다.

욕심쟁이 매칭(greedy matching): 정규 표현식의 “+”, “*” 문자는 가능한 큰 문자열을 매칭하기 위해서 밖으로 확장하는 개념.

grep: 정규 표현식에 매칭되는 파일을 탐색하여 라인을 찾는데 대부분의 유닉스 시스템에서 사용가능한 명령어. "Generalized Regular Expression Parser"의 약자.

정규 표현식(regular expression): 좀더 복잡한 검색 문자열을 표현하는 언어. 정규 표현식은 특수 문자를 포함해서 검색 라인의 처음 혹은 끝만 매칭하거나 많은 비슷한 것을 매칭한다.

와일드 카드(wild card): 임의 문자를 매칭하는 특수 문자. 정규 표현식에서 와이드 카드 문자는 마침표 문자다.

제 9 절 연습 문제

Exercise 11.1 유닉스의 `grep` 명령어를 모사하는 간단한 프로그램을 작성하세요. 사용자가 정규 표현식을 입력하고 정규 표현식에 매칭되는 라인수를 셈하는 프로그램입니다.

```
$ python grep.py
Enter a regular expression: ^Author
mbox.txt had 1798 lines that matched ^Author

$ python grep.py
Enter a regular expression: ^X-
mbox.txt had 14368 lines that matched ^X-

$ python grep.py
Enter a regular expression: java$
mbox.txt had 4218 lines that matched java$
```

Exercise 11.2 다음 형식의 라인만을 찾는 프로그램을 작성하세요.

```
New Revision: 39772
```

그리고, 정규 표현식과 `findall()` 메쏘드를 사용하여 각 라인으로부터 숫자를 추출하세요. 숫자들의 평균을 구하고 출력하세요.

```
Enter file:mbox.txt
38549.7949721

Enter file:mbox-short.txt
39756.9259259
```

제 12 장

네트워크 프로그램

지금까지 책의 많은 예제는 파일을 읽고 파일의 정보를 찾는데 집중했지만, 다양한 많은 정보의 원천이 인터넷에 있다.

이번 장에서는 웹브라우져로 가장하고 HTTP 프로토콜(HyperText Transport Protocol,HTTP)을 사용하여 웹페이지를 검색할 것이다. 웹페이지 데이터를 읽고 파싱할 것이다.

제 1 절 하이퍼 텍스트 전송 프로토콜(HyperText Transport Protocol - HTTP)

웹에 동력을 공급하는 네트워크 프로토콜은 실제로 매우 단순하다. 파이썬에는 소켓 (sockets)이라고 불리는 내장 지원 모듈이 있다. 파이썬 프로그램에서 소켓 모듈을 통해서 네트워크 연결을 하고, 데이터 검색을 매우 용이하게 한다.

소켓(socket)은 단일 소켓으로 두 프로그램 사이에 양방향 연결을 제공한다는 점을 제외하고 파일과 매우 유사하다. 동일한 소켓에 읽거나 쓸 수 있다. 소켓에 무언가를 쓰게 되면, 소켓의 다른 끝에 있는 응용프로그램에 전송된다. 소켓으로부터 읽게 되면, 다른 응용 프로그램이 전송한 데이터를 받게 된다.

하지만, 소켓의 다른쪽 끝에 프로그램이 어떠한 데이터도 전송하지 않았는데 소켓을 읽으려고 하면, 단지 앉아서 기다리기만 한다. 만약 어떠한 것도 보내지 않고 양쪽 소켓 끝의 프로그램 모두 기다리기만 한다면, 모두 매우 오랜 시간동안 기다리게 될 것이다.

인터넷으로 통신하는 프로그램의 중요한 부분은 특정 종류의 프로토콜을 공유하는 것이다. 프로토콜(protocol)은 정교한 규칙의 집합으로 누가 메시지를 먼저 보내고, 메세지로 무엇을 하며, 메시지에 대한 응답은 무엇이고, 다음에 누가 메세지를 보내고 등등을 포함한다. 이런 관점에서 소켓 끝의 두 응용프로그램이 함께 춤을 추고 있으니, 다른 사람 발을 밟지 않도록 확인해야 한다.

네트워크 프로토콜을 기술하는 문서가 많이 있다. 하이퍼텍스트 전송 프로토콜(HyperText Transport Protocol)은 다음 문서에 기술되어 있다.

http://www.w3.org/Protocols/rfc2616/rfc2616.txt

매우 상세한 176 페이지나 되는 장문의 복잡한 문서다. 흥미롭다면 시간을 가지고 읽어보기 바란다. RFC2616에 36 페이지를 읽어보면, GET 요청(request)에 대한 구문을 발견하게 된다. 꼼꼼히 읽게 되면, 웹서버에 문서를 요청하기 하기 위해서, 80 포트로 `www.py4inf.com` 서버에 연결을 하고 나서 다음 양식 한 라인을 전송한다.

```
GET http://www.py4inf.com/code/romeo.txt HTTP/1.0
```

두번째 매개변수는 요청하는 웹페이지가 된다. 그리고 또한 빈 라인도 전송한다. 웹서버는 문서에 대한 헤더 정보와 빈 라인 그리고 문서 본문으로 응답한다.

제 2 절 세상에서 가장 간단한 웹 브라우져(Web Browser)

아마도 HTTP 프로토콜이 어떻게 작동하는지 알아보는 가장 간단한 방법은 매우 간단한 파이썬 프로그램을 작성하는 것이다. 웹서버에 접속하고 HTTP 프로토콜 규칙에 따라 문서를 요청하고 서버가 다시 보내주는 결과를 보여주는 것이다.

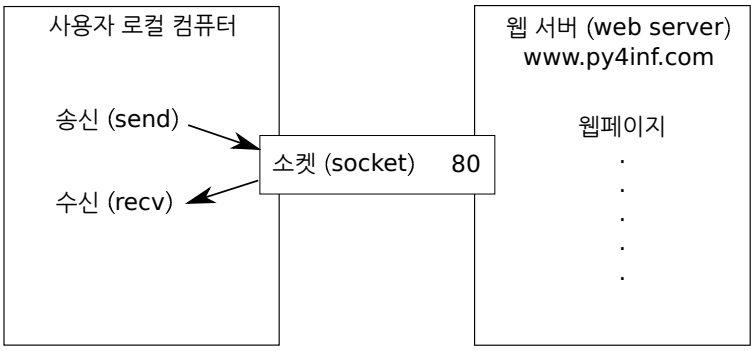
```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('www.py4inf.com', 80))
mysock.send('GET http://www.py4inf.com/code/romeo.txt HTTP/1.0\n\n')

while True:
    data = mysock.recv(512)
    if ( len(data) < 1 ) :
        break
    print data

mysock.close()
```

처음에 프로그램은 `www.py4inf.com` 서버에 80 포트로 연결한다. "웹 브라우져" 역할로 작성된 프로그램이 하기 때문에 HTTP 프로토콜은 GET 명령어를 공백 라인과 함께 보낸다.



공백 라인을 보내자 마자, 512 문자 덩어리의 데이터를 소켓에서 받아 더 이상 읽을 데이터가 없을 때까지(즉, recv()이 빈 문자열을 반환한다.) 데이터를 출력하는 루프를 작성한다.

프로그램 실행결과 다음을 얻을 수 있다.

```
HTTP/1.1 200 OK
Date: Sun, 14 Mar 2010 23:52:41 GMT
Server: Apache
Last-Modified: Tue, 29 Dec 2009 01:31:22 GMT
ETag: "143c1b33-a7-4b395bea"
Accept-Ranges: bytes
Content-Length: 167
Connection: close
Content-Type: text/plain

But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

출력결과는 웹서버가 문서를 기술하기 위해서 보내는 헤더(header)로 시작한다. 예를 들어, `Content-Type` 헤더는 문서가 일반 텍스트 문서(`text/plain`)임을 표기한다.

서버가 헤더를 보낸 후에, 빈 라인을 추가해서 헤더 끝임을 표기하고 나서 실제 파일`romeo.txt`을 보낸다.

이 예제를 통해서 소켓을 통해서 저수준(low-level) 네트워크 연결을 어떻게 하는지 확인할 수 있다. 소켓을 사용해서 웹서버, 메일 서버 혹은 다른 종류의 서버와 통신할 수 있다. 필요한 것은 프로토콜을 기술하는 문서를 찾고 프로토콜에 따라 데이터를 주고 받는 코드를 작성하는 것이다.

하지만, 가장 흔히 사용하는 프로토콜은 HTTP (즉, 웹) 프로토콜이기 때문에, 파이썬에는 HTTP 프로토컬을 지원하기 위해 특별히 설계된 라이브러리가 있다. 이것을 통해서 웹상에서 데이터나 문서를 검색을 쉽게 할 수 있다.

제 3 절 HTTP를 통해서 이미지 가져오기

상기 예제에서는 파일에 새줄(newline)이 있는 일반 텍스트 파일을 가져왔다. 그리고 나서, 프로그램을 실행해서 데이터를 단순히 화면에 복사했다. HTTP를 사용하여 이미지를 가져오도록 비슷하게 프로그램을 작성할 수 있다. 프로그램 실행 시에 화면에 데이터를 복사하는 대신에, 데이터를 문자열로 누적하고, 다음과 같이 헤더를 잘라내고 나서 파일에 이미지 데이터를 저장한다.

```
import socket
import time

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('www.py4inf.com', 80))
mysock.send('GET http://www.py4inf.com/cover.jpg HTTP/1.0\n\n')


count = 0
picture = "";
```

```
while True:
    data = mysock.recv(5120)
    if ( len(data) < 1 ) : break
    # time.sleep(0.25)
    count = count + len(data)
    print len(data),count
    picture = picture + data

mysock.close()

# Look for the end of the header (2 CRLF)
pos = picture.find("\r\n\r\n");
print 'Header length',pos
print picture[:pos]

# Skip past the header and save the picture data
picture = picture[pos+4:]
fhand = open("stuff.jpg","wb")
fhand.write(picture);
fhand.close()
```

프로그램을 실행하면, 다음과 같은 출력을 생성한다.

```
$ python urljpeg.py
2920 2920
1460 4380
1460 5840
1460 7300
...
1460 62780
1460 64240
2920 67160
1460 68620
1681 70301
Header length 240
HTTP/1.1 200 OK
Date: Sat, 02 Nov 2013 02:15:07 GMT
Server: Apache
Last-Modified: Sat, 02 Nov 2013 02:01:26 GMT
ETag: "19c141-111a9-4ea280f8354b8"
Accept-Ranges: bytes
Content-Length: 70057
Connection: close
Content-Type: image/jpeg
```

상기 url에 대해서, `Content-Type` **헤더가 문서 본문이 이미지(**`image/jpeg`**)를 나타내는 것을 볼 수 있다. 프로그램이 완료되면, 이미지 뷰어로** `stuff.jpg` **파일을 열어서 이미지 데이터를 볼 수 있다.**

프로그램을 실행하면, `recv()` **메쏘드를 호출할 때 마다 5120 문자는 전달받지 못하는 것을 볼 수 있다.** `recv()` **호출하는 순간마다 웹서버에서 네트워크로 전송되는 가능한 많은 문자를 받을 뿐이다. 매번 5120 문자까지 요청하지만, 1460 혹은 2920 문자만 전송받는다.**

결과값은 네트워크 속도에 따라 달라질 수 있다. recv() 메쏘드 마지막 호출에는 스트림 마지막인 1681 바이트만 받았고, recv() 다음 호출에는 0 길이 문자열을 전송받아서, 서버가 소켓 마지막에 close() 메쏘드를 호출하고 더이상의 데이터가 없다는 신호를 준다.

주석 처리한 time.sleep()을 풀어줌으로써 recv() 연속 호출을 늦출 수 있다. 이런 방식으로 매번 호출 후에 0.25초 기다리게 한다. 그래서, 사용자가 recv() 메쏘드를 호출하기 전에 서버가 먼저 도착할 수 있어서 더 많은 데이터를 보낼 수가 있다. 정지 시간을 넣어서 프로그램을 다시 실행하면 다음과 같다.

```
$ python urljpeg.py
1460 1460
5120 6580
5120 11700
...
5120 62900
5120 68020
2281 70301
Header length 240
HTTP/1.1 200 OK
Date: Sat, 02 Nov 2013 02:22:04 GMT
Server: Apache
Last-Modified: Sat, 02 Nov 2013 02:01:26 GMT
ETag: "19c141-111a9-4ea280f8354b8"
Accept-Ranges: bytes
Content-Length: 70057
Connection: close
Content-Type: image/jpeg
```

recv() 메쏘드 호출의 처음과 마지막을 제외하고, 매번 새로운 데이터를 요청할 때마다 이제 5120 문자가 전송된다.

서버 send() 요청과 응용프로그램 recv() 요청 사이에 버퍼가 있다. 프로그램에 지연을 넣어 실행하게 될 때, 어느 지점엔가 서버가 소켓 버퍼를 채우고 응용프로그램이 버퍼를 비울 때까지 잠시 멈춰야 된다. 송신하는 응용프로그램 혹은 수신하는 응용프로그램을 멈추게 하는 행위를 "흐름 제어(flow control)"이라고 한다.

제 4 절 urllib 사용하여 웹페이지 가져오기

수작업으로 소켓 라이브러리를 사용하여 HTTP로 데이터를 주고 받을 수 있지만, urllib 라이브러리를 사용하여 파이썬에서 동일한 작업을 수행하는 좀더 간편한 방식이 있다.

urllib을 사용하여 파일처럼 웹페이지를 다룰 수가 있다. 단순하게 어느 웹페이지를 가져올 것인지만 지정하면 urllib 라이브러리가 모든 HTTP 프로토콜과 헤더관련 사항을 처리해 준다.

웹에서 romeo.txt 파일을 읽도록 urllib를 사용하여 작성한 상응 코드는 다음과 같다.

```
import urllib

fhand = urllib.urlopen('http://www.py4inf.com/code/romeo.txt')
for line in fhand:
   print line.strip()
```

urllib.urlopen을 사용하여 웹페이지를 열게 되면, 파일처럼 다룰 수 있고 for 루프를 사용하여 데이터를 읽을 수 있다.

프로그램을 실행하면, 파일 내용 출력결과만을 볼 수 있다. 헤더정보는 여전히 전송되었지만, urllib 코드가 헤더를 받아 내부적으로 처리하고, 사용자에게는 단지 데이터만 반환한다.

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

예제로, romeo.txt 데이터를 가져와서 파일의 각 단어 빈도를 계산하는 프로그램을 다음과 같이 작성할 수 있다.

```
import urllib

counts = dict()
fhand = urllib.urlopen('http://www.py4inf.com/code/romeo.txt')
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1
print counts
```

다시 한번, 웹페이지를 열게 되면, 로컬 파일처럼 웹페이지를 읽을 수 있다.

제 5 절 HTML 파싱과 웹 스크래핑

파이썬 urllib 활용하는 일반적인 사례는 웹 스크래핑(scraping)이다. 웹 스크래핑은 웹브라우저를 가장한 프로그램을 작성하는 것이다. 웹페이지를 가져와서, 패턴을 찾아 페이지 내부의 데이터를 꼼꼼히 조사한다. 예로, 구글같은 검색엔진은 웹 페이지의 소스를 조사해서 다른 페이지로 가는 링크를 추출하고, 그 해당 페이지를 가져와서 링크 추출하는 작업을 반복한다. 이러한 기법으로 구글은 웹상의 거의 모든 페이지를 거미(spiders)줄처럼 연결한다.

구글은 또한 발견한 웹페이지에서 특정 페이지로 연결되는 링크 빈도를 사용하여 얼마나 중요한 페이지인지를 측정하고 검색결과에 페이지가 얼마나 높은 순위로 노출되어야 하는지 평가한다.

제 6 절 정규 표현식 사용 HTML 파싱하기

HTML을 파싱하는 간단한 방식은 정규 표현식을 사용하여 특정한 패턴과 매칭되는 부속 문자열을 반복적으로 찾아 추출하는 것이다.

여기 간단한 웹페이지가 있다.

```
<h1>The First Page</h1>
<p>
If you like, you can switch to the
<a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>.
</p>
```

모양 좋은 정규표현식을 구성해서 다음과 같이 상기 웹페이지에서 링크를 매칭하고 추출할 수 있다.

```
href="http://.+?"
```

작성된 정규 표현식은 "href="http://"로 시작하고, 하나 이상의 문자를 ".+?" 가지고 큰 따옴표를 가진 문자열을 찾는다. ".+?"에 물음표가 갖는 의미는 매칭이 "욕심쟁이(greedy)" 방식보다 "비욕심쟁이(non-greedy)" 방식으로 수행됨을 나타낸다. 비욕심쟁이(non-greedy) 매칭방식은 가능한 *가장 적게* 매칭되는 문자열을 찾는 방식이고, 욕심 방식은 가능한 *가장 크게* 매칭되는 문자열을 찾는 방식이다.

추출하고자 하는 문자열이 매칭된 문자열의 어느 부분인지를 표기하기 위해서 정규 표현식에 괄호를 추가하여 다음과 같이 프로그램을 작성한다.

```
import urllib
import re

url = raw_input('Enter - ')
html = urllib.urlopen(url).read()
links = re.findall('href="(http://.*?)"', html)
for link in links:
    print link
```

`findall` 정규 표현식 메쏘드는 정규 표현식과 매칭되는 모든 문자열 리스트를 추출하여 큰 따옴표 사이에 링크 텍스트만을 반환한다.

프로그램을 실행하면, 다음 출력을 얻게된다.

```
python urlregex.py
Enter - http://www.dr-chuck.com/page1.htm
http://www.dr-chuck.com/page2.htm

python urlregex.py
Enter - http://www.py4inf.com/book.htm
http://www.greenteapress.com/thinkpython/thinkpython.html
http://allendowney.com/
http://www.py4inf.com/code
http://www.lib.umich.edu/espresso-book-machine
http://www.py4inf.com/py4inf-slides.zip
```

정규 표현식은 HTML이 예측가능하고 잘 구성된 경우에 멋지게 작동한다. 하지만, "망가진" HTML 페이지가 많아서, 정규 표현식만을 사용하는 솔류션은 유효한 링크를 놓치거나 잘못된 데이터만 찾고 끝날 수 있다.

이 문제는 강건한 HTML 파싱 라이브러리를 사용해서 해결될 수 있다.

제 7 절 BeautifulSoup 사용한 HTML 파싱

HTML을 파싱하여 페이지에서 데이터를 추출할 수 있는 파이썬 라이브러리는 많이 있다. 라이브러리 각각은 강점과 약점이 있어서 사용자 필요에 따라 취사선택한다.

예로, 간단하게 HTML 입력을 파싱하여 BeautifulSoup 라이브러리를 사용하여 링크를 추출할 것이다. 다음 웹사이트에서 BeautifulSoup 코드를 다운로드 받아 설치할 수 있다.

www.crummy.com

BeautifulSoup 라이브러리를 다운로드 받아 "설치"하거나 `BeautifulSoup.py` **파일을 응용프로그램과 동일한 폴더에 저장한다.**

HTML이 XML 처럼 보이고 몇몇 페이지는 XML로 되도록 꼼꼼하게 구축되었지만, 일반적으로 대부분의 HTML이 깨져서 XML 파서가 HTML 전체 페이지를 잘못 구성된 것으로 간주하고 받아들이지 않는다. BeautifulSoup 라이브러리는 결점 많은 HTML 페이지에 내성이 있어서 사용자가 필요로하는 데이터를 쉽게 추출할 수 있게 한다.

`urllib`**를 사용하여 페이지를 읽어들이고,** `BeautifulSoup`**를 사용해서 앵커 태그 (**`a`**)로부터** `href` **속성을 추출한다.**

```
import urllib
from BeautifulSoup import *

url = raw_input('Enter - ')
html = urllib.urlopen(url).read()
soup = BeautifulSoup(html)

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
   print tag.get('href', None)
```

프로그램이 웹 주소를 입력받고, 웹페이지를 열고, 데이터를 읽어서 BeautifulSoup 파서에 전달하고, 그리고 나서 모든 앵커 태그를 불러와서 각 태그별로 `href` **속성을 출력한다.**

프로그램을 실행하면, 아래와 같다.

```
python urllinks.py
Enter - http://www.dr-chuck.com/page1.htm
http://www.dr-chuck.com/page2.htm

python urllinks.py
Enter - http://www.py4inf.com/book.htm
http://www.greenteapress.com/thinkpython/thinkpython.html
http://allendowney.com/
http://www.si502.com/
http://www.lib.umich.edu/espresso-book-machine
http://www.py4inf.com/code
http://www.pythonlearn.com/
```

BeautifulSoup을 사용하여 다음과 같이 각 태그별로 다양한 부분을 뽑아낼 수 있다.

```
import urllib
from BeautifulSoup import *

url = raw_input('Enter - ')
html = urllib.urlopen(url).read()
soup = BeautifulSoup(html)

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
   # Look at the parts of a tag
   print 'TAG:',tag
   print 'URL:',tag.get('href', None)
   print 'Content:',tag.contents[0]
   print 'Attrs:',tag.attrs
```

상기 프로그램은 다음을 출력합니다.

```
python urllink2.py
Enter - http://www.dr-chuck.com/page1.htm
TAG: <a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>
URL: http://www.dr-chuck.com/page2.htm
Content: [u'\nSecond Page']
Attrs: [(u'href', u'http://www.dr-chuck.com/page2.htm')]
```

HTML을 파싱하는데 BeautifulSoup이 가진 강력한 기능을 예제로 보여줬다. 좀더 자세한 사항은 `www.crummy.com` 에서 문서와 예제를 살펴보세요.

제 8 절 urllib을 사용하여 바이너리 파일 읽기

이미지나 비디오 같은 텍스트가 아닌 (혹은 바이너리) 파일을 가져올 때가 종종 있다. 일반적으로 이런 파일 데이터를 출력하는 것은 유용하지 않다. 하지만, `urllib`을 사용하여, 하드 디스크 로컬 파일에 URL 사본을 쉽게 만들 수 있다.

이 패턴은 URL을 열고, `read`를 사용해서 문서 전체 내용을 다운로드하여 문자열 변수(`img`)에 다운로드하고, 그리고 나서 다음과 같이 정보를 로컬 파일에 쓴다.

```
img = urllib.urlopen('http://www.py4inf.com/cover.jpg').read()
fhand = open('cover.jpg', 'w')
fhand.write(img)
fhand.close()
```

작성된 프로그램은 네트워크로 모든 데이터를 한번에 읽어서 컴퓨터 주기억장치 `img` 변수에 저장하고, `cover.jpg` 파일을 열어 디스크에 데이터를 쓴다. 이 방식은 파일 크기가 사용자 컴퓨터의 메모리 크기보다 작다면 정상적으로 작동한다.

하지만, 오디오 혹은 비디오 파일 대용량이면, 상기 프로그램은 멈추거나 사용자 컴퓨터 메모리가 부족할 때 극단적으로 느려질 수 있다. 메모리 부족을 회피하기 위해서, 데이터를 블럭 혹은 버퍼로 가져와서, 다음 블럭을 가져오기 전에 디스크에 각각의 블럭을 쓴다. 이런 방식으로 사용자가 가진 모든 메모리를 사용하지 않고 어떠한 크기 파일도 읽어올 수 있다.

```
import urllib

img = urllib.urlopen('http://www.py4inf.com/cover.jpg')
fhand = open('cover.jpg', 'w')
size = 0
while True:
    info = img.read(100000)
    if len(info) < 1 : break
    size = size + len(info)
    fhand.write(info)

print size,'characters copied.'
fhand.close()
```

상기 예제에서, 한번에 100,000 문자만 읽어 오고, 웹에서 다음 100,000 문자를 가져오기 전에 `cover.jpg` 파일에 읽어온 문자를 쓴다.

프로그램 실행 결과는 다음과 같다.

```
python curl2.py
568248 characters copied.
```

UNIX 혹은 매킨토시 컴퓨터를 가지고 있다면, 다음과 같이 상기 동작을 수행하는 명령어가 운영체제 자체에 내장되어 있다.

```
curl -O http://www.py4inf.com/cover.jpg
```

`curl`은 "copy URL"의 단축 명령어로 두 예제는 `curl` 명령어와 비슷한 기능을 구현해서, **`www.py4inf.com/code`** 사이트에 `curl1.py`, `curl2.py` 이름으로 올라가 있다. 동일한 작업을 좀더 효과적으로 수행하는 `curl3.py` 샘플 프로그램도 있어서 실무적으로 작성하는 프로그램에 이런한 패턴을 이용하여 구현할 수 잇다.

제 9 절 용어정의

BeautifulSoup: 파이썬 라이브러리로 HTML 문서를 파싱하고 브라우저가 일반적으로 생략하는 HTML의 불완전한 부분을 보정하여 HTML 문서에서 데이터를 추출한다. **`www.crummy.com`** 사이트에서 BeautifulSoup 코드를 다운로드 받을 수 있다.

포트(port): 서버에 소켓 연결을 만들 때, 사용자가 무슨 응용프로그램을 연결하는지 나타내는 숫자. 예로, 웹 트래픽은 통상 80 포트, 전자우편은 25 포트를 사용한다.

스크래핑(scraping): 프로그램이 웹브라우저를 가장하여 웹페이지를 가져와서 웹 페이지의 내용을 검색한다. 종종 프로그램이 한 페이지의 링크를 따라 다른 페이지를 찾게 된다. 그래서, 웹페이지 네트워크 혹은 소셜 네트워크 전체를 훑을 수 있다.

소켓(socket): 두 응용프로그램 사이 네트워크 연결. 두 응용프로그램은 양 방향으로 데이터를 주고 받는다.

스파이더(spider): 검색 색인을 구축하기 위해서 한 웹페이지를 검색하고, 그 웹페이지에 링크된 모든 페이지 검색을 반복하여 인터넷에 있는 거의 모든 웹페이지를 가져오기 위해서 사용되는 검색엔진 행동.

제 10 절 연습문제

Exercise 12.1 소켓 프로그램 `socket1.py`을 변경하여 임의 웹페이지를 읽을 수 있도록 URL을 사용자가 입력하도록 바꾸세요. `split('/')`을 사용하여 URL을 컴포턴트로 쪼개서 소켓 `connect` 호출에 대해 호스트 명을 추출할 수 있다. 사용자가 적절하지 못한 형식 혹은 존재하지 않는 URL을 입력하는 경우를 처리할 있도록 `try`, `except`를 사용하여 오류 검사기능을 추가하세요.

Exercise 12.2 소켓 프로그램을 변경하여 전송받은 문자를 계수(count)하고 3000 문자를 출력한 후에 그이상 텍스트 출력을 멈추게 하세요. 프로그램은 전체 문서를 가져와야 하고, 전체 문자를 계수(count)하고, 문서 마지막에 문자 계수(count)결과를 출력해야 합니다.

Exercise 12.3 `urllib`을 사용하여 이전 예제를 반복하세요. (1) 사용자가 입력한 URL에서 문서 가져오기 (2) 3000 문자까지 화면에 보여주기 (3) 문서의 전체 문자 계수(count)하기. 이 연습문제에서 헤더에 대해서는 걱정하지 말고, 단지 문서 본문에서 첫 3000 문자만 화면에 출력하세요.

Exercise 12.4 `urllinks.py` 프로그램을 변경하여 가져온 HTML 문서에서 문단 (p) 태그를 추출하고 프로그램의 출력물로 문단을 계수(count)하고 화면에 출력하세요. 문단 텍스트를 화면에 출력하지 말고 단지 숫자만 셉니다. 작성한 프로그램을 작은 웹페이지 뿐만 아니라 조금 큰 웹 페이지에도 테스트해 보세요.

Exercise 12.5 (고급) 소켓 프로그램을 변경하여 헤더와 빈 라인 다음에 데이터만 보여지게 하세요. `recv`는 라인이 아니라 문자(새줄(newline)과 모든 문자)를 전송받는다는 것을 기억하세요.