정보교육을 위한 파이썬

정보 탐색

Version 0.0.9-d2

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

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*.

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

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 몇권있다. 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 프로그램에 가져오기(import)하여야 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, 매 라인을 루프를 반복해서 정규 표현식 `search()` 메쏘드를 호출하여 문자열 "From"이 포함된 라인만 출력한다. 상기 프로그램은 정규 표현식의 진정 강력한 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 확인할 수 있다. 매우 적은 코드를 작성해도 정규 표현식에 특수 문자를 추가하는 것 만으로도 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 일치하는 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, 무엇이 정규 표현식을 매칭하는가에 대해서 좀더 많은 제어를 할 수 있게 하는 특수 액션 문자를 담고 있는 정규 표현식의 개념을 소개하기에는 충분하다.

1.1 정규 표현식의 문자 매칭

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

다음 예제에서 정규 표현식 "F..m:"은 "From:", "Fxxm:", "F12m:", "F!@m:' 같은 임의의 문자열을 매칭한다. 왜냐하면 정규 표현식의 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 때, 더욱 강력해진다. "*", "+' ' 특수 문자는 검색 문자열에 하나의 문자만을 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, ".+" @ 기호로 되는 하나 혹은 그 이상의 문자들의 라인을 성공적으로 매칭한다. 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 것 처럼 문자령릐 마지막 @ 기호를 매칭한다.

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

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

1.2 정규 표현식 사용 데이터 추출

파이썬에서 문자열에서 데이터를 추출하려면, findall() 메쏘드를 사용해서 정규 표현식과 매칭되는 모든 부속 문자열을 추출할 수 있다. 형식에 관계없이 임의의 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 라인의 각각의 형식에 맞는 코드를 작성하고는 싶지 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 인수 문자열을 찾아서 전자우편 주소처럼 보이는 모든 문자열을 리스트로 반환한다. 공백이 아닌 문자 (\S)와 매칭되는 두 문자 스퀀스를 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 큰가를 만을 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 수조 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 임의의 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.

1.3 검색과 추출 조합하기

다음과 같은 "X-" 문자열로 시작하는 라인의 숫자를 찾고자 한다면,

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

임의의 라인에서 임의의 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.

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 합니다. `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 추가할 때, 문자열이 매칭될 때, 무시된다. 하지만, `findall()`을 사용할 때, 매칭할 전체 정규 표현식을 원할지라도, 정규 표현식을 매칭하는 부속 문자열의 부분만을 뽑아내다는 것을 괄호가 표시한다.

그래서, 프로그램에 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, 라인의 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 행동이 왜 각 숫자로 모든 5자리 숫자를 얻은 이유다. 정규 표현식 라이브러리는 양방향으로 파일 처음이나 끝에 숫자가 아닌 것을 마주칠 때 까지 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 라인의 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 라인의 정의다.

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

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

작업 결과는 다음과 같이 프로그램에 나타난다.

```
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']
...
```

1.4 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 기호화 매칭한다. 정규 표현식의 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.

1.5 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 앞선문자와 매칭을 표기함.

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

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

+?
바로 앞선 문자에 적용되고 1 혹은 그 이상의 앞선문자와 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 임의의 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
임의의 숫자와 매칭하여 [0-9] 집합에 상응함.

\D
임의의 숫자가 아닌 문자와 매칭하여 [^0-9] 집합에 상응함.

1.6 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
```

This tells `grep` to show you lines that start with the string "From:" in the file `mbox-short.txt`. If you experiment with the `grep` command a bit and read the documentation for `grep`, you will find some subtle differences between the regular expression support in Python and the regular expression support in `grep`. As an example, `grep` does not support the non-blank character "\S" so you will need to use the slightly more complex set notation "[^]"- which simply means - match a character that is anything other than a space.

1.7 Debugging

Python has some simple and rudimentary built-in documentation that can be quite helpful if you need a quick refresher to trigger your memory about the exact name of a particular method. This documentation can be viewed in the Python interpreter in interactive mode.

You can bring up an interactive help system using `help()`.

```
>>> 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
```

If you know what module you want to use, you can use the `dir()` command to find the methods in the module as follows:

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

You can also get a small amount of documentation on a particular method using the dir command.

```
>>> 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.
>>>
```

The built in documentation is not very extensive, but it can be helpful when you are in a hurry or don't have access to a web browser or search engine.

1.8 Glossary

brittle code: Code that works when the input data is in a particular format but prone to breakage if there is some deviation from the correct format. We call this "brittle code" because it is easily broken.

greedy matching: The notion that the "+" and "*" characters in a regular expression expand outward to match the largest possible string.

grep: A command available in most Unix systems that searches through text files looking for lines that match regular expressions. The command name stands for "Generalized Regular Expression Parser".

regular expression: A language for expressing more complex search strings. A regular expression may contain special characters that indicate that a search only matches at the beginning or end of a line or many other similar capabilities.

wild card: A special character that matches any character. In regular expressions the wild card character is the period character.

1.9 Exercises

Exercise 1.1 Write a simple program to simulate the operation of the `grep` command on Unix. Ask the user to enter a regular expression and count the number of lines that matched the regular expression:

```
$ 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 1.2 Write a program to look for lines of the form

```
New Revision: 39772
```

And extract the number from each of the lines using a regular expression and the `findall()` method. Compute the average of the numbers and print out the average.

```
Enter file:mbox.txt
38549.7949721

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