Python for Informatics

LESSON 6

Regular Expressions

- ***Regular expressions*** are a huge topic; entire books have been written to describe their use.
- From Wikipedia we have the following definition: *"A regular expression, often called a **pattern**, is an expression used to specify a set of strings required for a particular purpose."*
- ***Regular expressions*** are almost a small language that prescribes string-based pattern matching algorithms.

Regular Expressions

- For more detail on regular expressions and the history of their development, please see:

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

- As a reference for the regular expression Python module, please see:

  https://docs.python.org/2/library/re.html

Regular Expressions

- To work with ***regular expressions*** in Python, you must import the ***regular expression*** module, ***re***.
- Let us consider a scenario in which you want to read through successive text lines in a file, and you want to print only the lines that contain the string “From:”.

Regular Expressions


```
import re

fhand =
open('C:/UCSD/PythonForInformatics
/code/mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if re.search('From:', line):
        print line
```

Regular Expressions

- But, instead of…

if re.search('From:', line):

Why not use the ***find()*** method?

if line.find('From:') != -1:

- For something this simple, either approach would be fine.

Regular Expressions

- If we need more nuanced pattern matching, that's when ***regular expressions*** demonstrate their power.

- For example, consider that instead of searching for the occurance of substring ***'From:'*** anywhere in the line, we want to restrict our search to the beginning of the line.

Regular Expressions

- To match only at the beginning of a line, we use the caret character, '^'.
- Our search condition would then look like this:

```
if re.search('^From:', line):
```

Regular Expressions

- Yes, we could have used the ***startswith()*** method instead.

- But, there are additional special characters that help us define ***pattern matches***.

Regular Expressions

Characters	Effect
$	Matches the end of the string.
.	Matches any character.
*	Repeats the previous match zero or more times.
+	Repeats the previous match one or more times.
?	Matches zero or more repetitions of the preceding regular expression
*?, +?, ??	The '*', '+', and '?' qualifiers are all *greedy*; they match as much text as possible. Adding '?' after the qualifier makes it perform the match in *non-greedy* or *minimal* fashion; as *few* characters as possible will be matched.

Regular Expressions

Characters	Effect
{m}	Specifies that exactly *m* copies of the previous regular expression should be matched; fewer matches cause the entire regular expression not to match.
{m,n}	Causes the resulting regular expression to greedily match from *m* to *n* repetitions of the preceding regular expression.
{m,n}?	Causes the resulting regular expression to non-greedily match from *m* to *n* repetitions of the preceding regular expression.
'\'	Escapes special characters (permitting you to match characters like '*', '?', and so forth), or signals a special sequence.

Regular Expressions

Characters	Effect
[]	Used to indicate a set of characters.
'|'	A|B, where A and B can be arbitrary regular expressions, creates a regular expression that will match either A or B. This operation is never greedy.
(...)	Matches whatever regular expression is inside the parentheses, and indicates the start and end of a group; the contents of a group can be retrieved after a match has been performed, and can be matched later in the string with the \number special sequence.
\number	Matches the contents of the group of the same number. Groups are numbered starting from 1.

Regular Expressions

- There are many more special characters.
- For more detailed descriptions and examples, please consult the Python ***re*** module documentation.
- With our specific example, the search string ***'^From:.+@'*** specifies a match of lines beginning with ***'From:'*** , followed by one or more characters (***'.+'***) , and ending with an at-sign (***'@'***).

Regular Expressions


```
import re

fhand =
open('C:/UCSD/PythonForInformatics
/code/mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if re.search('^From:.+@', line):
        print line
```

Regular Expressions

- The ***findall()*** method allows us to extract the various substrings that match a given ***regular expression*** pattern, and returns them as a ***list***.

- Note that the sequence ***'\S'*** matches a non-whitespace character.

- To collect all of the substrings that denote email addresses, we could use the following:

  '\S+@\S+'

- The above regular expression matches one or more whitespace characters followed by an at-sign followed by one or more whitespace characters.

Regular Expressions

- Our lines of email addresses in the mbox.txt file have undesirable non-whitespace characters before and after the email addresses—we have more work to do.

- After careful encoding, we have the following ***regular expression*** search pattern:

  '[a-zA-Zo-9]\S*@\S*[a-zA-Z]'

- Note that the sequence ***'\S'*** matches a non-whitespace character.

Regular Expressions


```
import re

fhand =
open('C:/UCSD/PythonForInformatics/co
de/mbox.txt')
for line in fhand:
    line = line.rstrip()
    addr = re.findall(\
'[a-zA-Z0-9]\S*@\S*[a-zA-Z]',line)
    if len(addr) > 0:
        print addr
```

Regular Expressions

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

- This expression searches for a pattern that begins with a single lower letter, uppercase letter, or number (***[a-zA-Z0-9]***), followed by zero or more non-blank characters (***\S****), followed by an at-sign (@), followed by zero or more non-blank characters (***\S****), followed by a single lower or uppercase letter.

Regular Expressions

- Let us now consider another data extraction scenario: we want to process lines that begin with “X-”, and we want to extract any floating point number that appears in the line.
- To solve the problem of line selection, we construct the following ***regular expression***:

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

- In this case, because the period, “.”, is within the brackets, it is the actual period character as part of the set of matching characters, not a wildcard character.

Regular Expressions

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

- Translating the above expression into english, we are matching all lines that begin with "***X-***" followed by zero or more characters ("**.***"), followed by a colon ("**:**"), followed by a space (" "), followed by zero or more characters ("+") that are digits ("***0-9***") or a period (".").

Regular Expressions

- Now we need to address the issue of how we are going to extract only the floating point number.
- Parentheses within a regular expression are used to indicate what part of the expression is actually extracted.
- Without parentheses, whatever pattern matches is returned.
- With parentheses, when a pattern matches, only the portion specified by the parentheses is returned.

Regular Expressions


```
import re

fhand =
open('C:/UCSD/PythonForInformatics/co
de/mbox.txt')
for line in fhand:
    line = line.rstrip()
    addr = re.findall('^X-.*: ([0-9.]+)', line)
    if len(addr) > 0:
        print addr
```

Regular Expressions

- The result is a ***list*** of ***strings***, so if we wanted to use them as ***floats*** we would need to convert them to type ***float***.
- Regardless, our use of a ***regular expression*** has enabled us to perform a highly precise extraction of the data that we want.
- Consider how we might extract the revision number from our data file.

'^Details:.*rev=([0-9.]+)'

Regular Expressions

- Or how about extracting the time of day?

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

- On a final note, please remember that if you want a character to not be interpreted as a ***regular expression special character***, then all you need to do is precede that character with a ***backslash***.
- The ***backslash*** "escapes" the ***special character*** so it is not considered to be special.

Network Programming

- Up until now the input sources we have utilized include hardcoded, literal data, user input from the keyboard, and input read from data files.
- Now we are going to look at web page data that we read from the ***World Wide Web***.
- To accomplish this, we will write programs that pretend to be a web browser, and send and receive information by using the ***HyperText Transport Protocol (HTTP)***.

Network Programming

- To support the communication of two or more computers within a network, Python provides a powerful abstraction known as a ***socket***.
- A ***socket*** represents a two-way information exchange device that establishes a ***connection*** between two computers.
- ***Server*** programs can use a ***socket*** to establish ***connections*** with multiple ***clients***, however, we will be focusing only upon single ***client*** programs that have a ***connection*** with a single ***server***.

Network Programming

- If you are especially curious, you will find extensive documentation regarding the ***HTTP protocol*** at this URL:

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

- Within this document, you will find an explanation of the ***GET*** request.

Network Programming

- As an example, to request a document from a web server we could establish a connection to the www.py4inf.com server on port 80 (the standard HTTP port), and then send a GET request text string with following form:

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

- The second parameter is the web page being requested. We also need to send a blank line.
- The web server will respond with header information, a blank line, and the document content.
- ***HTTP*** uses a ***blank line*** to denote ***end-of-transmission***.

Network Programming

- As a gentle introduction to web programming, let's look at a Python HTTP version of "hello world"– i.e., a program that requests a text document from a web page and prints it to the console.
- Be sure to copy and paste this into your Python interpreter so you can see how it operates.

Network Programming


```
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()
```

Network Programming

- Whereas before you were reading the ***romeo.txt*** file as a text file on your local computer system, you are now reading the file as an ***HTTP document***!
- Our program establishes a ***connection*** through ***port 80*** with the www.py4inf.com web server.

```
mysock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
mysock.connect(('www.py4inf.com', 80))
```

Network Programming

- We send a ***GET command*** followed by a ***blank line***.

```
mysock.send('GET http://www.py4inf.com/code/romeo.txt
HTTP/1.0\n\n')
```

- We process the receipt of 512-character blocks from the socket, printing them out as we iterate within a loop.

```
while True:
  data = mysock.recv(512)
  if ( len(data) < 1 ) :  # Loop until we receive an empty string.
     break
  print data
```

Network Programming

- The output includes the headers, a blank line, and the content of the text document.
- When we are done, we are careful to close the ***socket***, since we don't need it anymore.
- This example demonstrates how to communicate with a web server, but we can also communicate with other kinds of servers—mail, FTP, etc.
- We merely need to implement the necessary protocol for the a given kind of server.

Network Programming

- With a little ingenuity, we can generalize our code to allow the user to provide a web page address as input.

```
import socket

url = raw_input('Enter: ')
words = url.split('/')
host = words[2]

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((host, 80))
mysock.send('GET '+url+' HTTP/1.0\n\n')

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

mysock.close()
```

Network Programming

- Since we all know that a picture is worth a thousand words, wouldn't it be worthwhile to retreive an image file from a web server?

```
import socket
import time


# Establish a socket connection and send a GET request.
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')
```

Network Programming


```
# Process the jpg file received in 5120 character chunks
# (chunks may be < 5120 depending on network traffic).
count = 0
picture = "";
while True:
    data = mysock.recv(5120)
    if ( len(data) < 1 ) : break
    time.sleep(0.25)  # Delay for pacing ourselves.
    count = count + len(data)
    print len(data),count
    picture = picture + data

mysock.close()
```

Network Programming


```
# 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()
```

Network Programming

- If the server fills up the socket's buffer before we have a chance to start processing it, the server will pause in its transmission.
- Similarly, if we finish processing the current buffer before the server has had a chance to send another one, our program will pause in its reception.
- This approach to buffer management is known as ***flow control***.

Network Programming

- We've seen how we can request and receive a document by using an ***HTTP*** based ***socket***, but there's an easier way provided by the ***urllib*** library.

```
import urllib

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

Network Programming

- Note that by using ***urllib*** we avoid having to write all of that socket code because ***urllib*** handles those details for us.
- ***urllib*** allows us to treat a web ***URL*** as if it were a regular ***file***.
- The following slide shows a program that uses ***urllib*** to retrieve the ***romeo.txt*** web document and determine the frequency of each word.

Network Programming


```
import urllib  # This is much easier!


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

Network Programming

- ***Web scraping*** is the process of retrieving web pages, analyzing their contents, and then extracting data from them based on that analysis.
- ***urllib*** is an extremely useful resource for ***web scraping***.
- One very common technique is to search a web page for ***links*** to other web pages.

Network Programming

- Here's an HTML snippet from a Vanity Fair web article regarding the surprising connection between Monty Python and The Game of Thrones:

```
content="http://media.vanityfair.com/photos/53305e5d5502796f770000d6/master/pass/benioff-weiss-tout.jpg" />
    <meta name="twitter:image:src"
content="http://media.vanityfair.com/photos/53305e5d5502796f770000d6/master/pass/benioff-weiss-tout.jpg?mbid=social_retweet">
```

Network Programming

- You can find the entire web page at this URL:

http://www.vanityfair.com/hollywood/2014/03/game-of-thrones-benioff-weiss-interview

- Now, consider a ***regular expression*** that helps us identify a ***web link***.

href = 'http://.+?'

Network Programming

- Let's write a program that uses the ***regular expression*** along with ***urllib***.

```
# Search for lines that start with From and
# have an at sign
import urllib
import re
# Test this with http://www.py4inf.com/book.htm
url = raw_input('Enter - ')
html = urllib.urlopen(url).read()
links = re.findall('href="(http://.*?)"', html)
for link in links:
    print link
```

Network Programming

- While the ***regular expression*** approach works, it is also quite fragile.
- The web is replete with broken links, and valid links can adopt a wide variety of formats.

Network Programming

- There are a number of Python libraries that help us parse HTML pages and extract data from them.
- Each one of them has advantages and disadvantages.
- By getting to know them, you'll be able to choose the best one that suits your current needs.

Network Programming

- We are going to look at the ***BeautifulSoup*** library.
- The ***BeautifulSoup*** library can be downloaded and installed from:

  http://www.crummy.com/software/

- Alternatively, instead of "installing" ***BeautifulSoup***, you can just put the BeautifulSoup.py file in the same directory as your program.

Network Programming

- The approach that ***BeautifulSoup*** takes is to be very forgiving with respect to the HTML format.
- Even if the HTML we are parsing is highly flawed, there's a good chance that ***BeautifulSoup*** will still allow us to parse it and extract the data that we need.
- The following slide demonstrates the code that will extract the ***href*** attributes from the ***anchor*** (***a***) tags.

Network Programming


```
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)
```

Network Programming

- Our program accepts a web page URL from the user, opens it, reads the page data, and passes it to the ***BeautifulSoup*** parser.
- The ***BeautifulSoup*** parser returns all of the ***anchor tags***.
- Our program prints the ***href attribute*** of each ***anchor tag***.

Network Programming

- So far we've seen how the ***urllib*** library can help us deal with sources of text data.
- ***urllib*** can also be used to read and write binary data such as image and sound files.

```
import urllib

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

Network Programming

- While this approach will work fine as long as your file is small enough to fit into available memory all at one time, it would be better if we could buffer our processing of the file so we don't have such as risk of running out of memory.

```
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()
```

Network Programming

- This last version of the program code has been optimized for usability and efficiency.

```
import os
import urllib

print 'Please enter a URL like http://www.py4inf.com/cover.jpg'
urlstr = raw_input().strip()
img = urllib.urlopen(urlstr)

# Get the last "word"
words = urlstr.split('/')
fname = words[-1]
```

Network Programming


```
# Don't overwrite the file
if os.path.exists(fname) :
  if raw_input('Replace '+fname+' (Y/n)?') != 'Y' :
    print 'Data not copied'
    exit()
  print 'Replacing',fname

fhand = open(fname, '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 to',fname
fhand.close()
```