Text manipulations

Christophe@pallier.org

Sept. 2013

Contents

Download Alice in Wonderland.

(1) Write a program that prints the lines that contains the string 'Alice' (tip: you can use the find function from the module string). Then, test the same program with the strings 'Rabbit', 'rabbit', 'stone', 'office'.

```
import string
text = file('alice.txt')
for line in text:
    if string.find(line, 'Alice') != -1:
        print(line)
```

```
import string

def print_matching_lines(filename, expr):
    print "#"*30
    print("Searching " + filename + " for " + expr + ":")
    for line in file(filename):
        if string.find(line, expr) != -1:
            print(line)

print_matching_lines('alice.txt', 'Alice')
print_matching_lines('alice.txt', 'Rabbit')
print_matching_lines('alice.txt', 'rabbit')
print_matching_lines('alice.txt', 'stone')
print_matching_lines('alice.txt', 'office')
```

Get http://www.pallier.org/cours/AIP2013/text3.py

(2) Here is a program that converts the text file into a list of words, removing the punctuation marks and converting everything in lower case. Run it.

```
import string
def remove_punctuation(text):
    punct = string.punctuation + chr(10)
    return text.translate(string.maketrans(punct, " " * len(punct)))

textori = file('alice.txt').read().lower()
text = remove_punctuation(textori)
words = text.split()
print(words)
```

Now write a script that counts the number of occurences of 'Alice', 'Rabbit' or 'office' in the list of words.

```
n1, n2, n3 = 0, 0, 0
for w in words:
    if w == 'alice':
        n1 = n1 + 1
    if w == 'rabbit':
        n2 = n2 + 1
    if w == 'office':
        n3 = n3 + 1
print n1, n2, n3
```

---

(3) Read about Python's Dictionnaries http://docs.python.org/2/tutorial/datastructures.html#dictionaries and use a dictonnary to store the number of occurrences of each word in Alice in Wonderland (the keys are the words, and the values and the number of occurrences; if word= ['a', 'a', 'b']; dico={'a':2, 'b':1}).

```
dico = {}
for w in words:
    if not(dico.has_key(w)):
        dico[w] = 1
    else:
        dico[w] += 1

print(dico)

# print sorted by word frequencies
for w in sorted(dico, key=dico.get, reverse=True):
    print w, d[w]
```

Get http://www.pallier.org/cours/AIP2013/text2.py

---

(4) Use numpy and matplotlib to plot the word log(frequencies) as a function of the rank of words on the abscissae (the most frequence word being ranked #1)

You can skim through http://matplotlib.org/users/pyplot_tutorial.html.

```
# affichage des fréquences en fonction de leur rang
freqs = dico.values()

import numpy as np
import matplotlib.pyplot as plt

lf = np.sort(freqs)
lf = lf[::-1] # reverse

plt.plot(lf, 'ro')
plt.yscale('log')
plt.xscale('log')

plt.show()
```

Get http://www.pallier.org/cours/AIP2013/text3.py

Remark: The product rank X frequency is roughly constant. This 'law' was discovered by Estoup and popularized by Zipf. See http://en.wikipedia.org/wiki/Zipf%27s_law.

(5) (advanced) Plot the relationship between word length and word frequency.

---

(6) Generate random text (each letter from a-z being equiprobable, and the spacecharacter being 8 times more probable) of 1 million characters. Compute the frequencies of each 'pseudowords' and plot the rank/frequency diagram.

```
import random
letters = "abcdefghijklmnopqrstuvwxyz        "
text = "".join([ random.choice(letters) for i in range(1000000) ])
print(text)
```

```
dico = {}
for w in text.split():
    if not(dico.has_key(w)):
        dico[w] = 1
    else:
        dico[w] += 1

# affichage des fréquences en fonction de leur rang
freqs = dico.values()

import numpy as np
import matplotlib.pyplot as plt

lf = np.sort(freqs)
lf = lf[::-1] # reverse

plt.plot(lf, 'ro')
plt.yscale('log')
plt.xscale('log')

plt.show()
```

Get http://www.pallier.org/cours/AIP2013/text4.py

---

(7) (advanced) compute the table of transition frequencies between words in Alice and generate random text following this pattern.

---

(8) Read about the MU Puzzle (http://en.wikipedia.org/wiki/MU_puzzle). Write a program that generates sequences of strings based on the following production rules and the initial state 'MI'

1. `xI -> xIU`
2. `Mx -> Mxx`
3. `xIIIy -> xUy`
4. `xUUy -> xy`

(Tip: use the function string.replace)

```
import string,random

def rule1(s):
    if s[-1] == 'I':
        return s + 'U'
    else:
        return -1

def rule2(s):
    if s[0] == 'M':
        return 'M' + s[1:] + s[1:]
    else:
        return -1

def rule3(s):
    if s.find('III') != -1:
        return s.replace('III', 'U')
    else:
        return -1

def rule4(s):
    if s.find('UU') != -1:
        return s.replace('UU', '')
    else:
        return -1

s = 'MI'

n = 0
while n<10:
    r = random.randint(1,4)
    if r==1:
        news = rule1(s)
    if r==2:
        news = rule2(s)
    if r==3:
        news = rule3(s)
    if r==4:
        news = rule4(s)
    if news != -1:
        print(str(n) + ': ('+ str(r) + '): ' + s + ' -> ' + news)
        s = news
        n = n + 1
```

Get Get http://www.pallier.org/cours/AIP2013/text5.py

One way to perform pattern matching is to use regular expressions http://docs.python.org/2/howto/regex.html#regex-howto.