Name: Date:

Lesson 7: Frequency fun

split and join

We’re going to go back to strings and lists now. Remember how a string can be decomposed to a list of characters:

```
>>> s=”hello”
>>> for c in s:
...   print c
h
e
l
l
o
```

We can also apply the built-in function `list` to a string to literally create a list of characters:

```
>>> s=”hello”
>>> l=list(s)
[‘h’,’e’,’l’,’l’,’o’]
```

To convert from a list of characters back to a string, you need to call `join()` on an empty string `‘’`. The `join()` **method** on a string takes a list and joins together everything element in that list using the join string (an empty string in this case). The following code shows you how this works:

```
>>> s=”hello”
>>> l=list(s)
[‘h’,’e’,’l’,’l’,’o’]
>>> ‘’.join(l)
‘hello’
>>> ‘_’.join(l)
‘h_e_l_l_o’
```

Another important method on a string is `split()`. The `split()` method takes a string and splits it on a particular character to return a list of smaller strings. If you call it without any parameters it will split on blank space `' '`. That means `split()` returns a list of all the words in a sentence string.

Here are some examples of **`split`** and **join** in action:

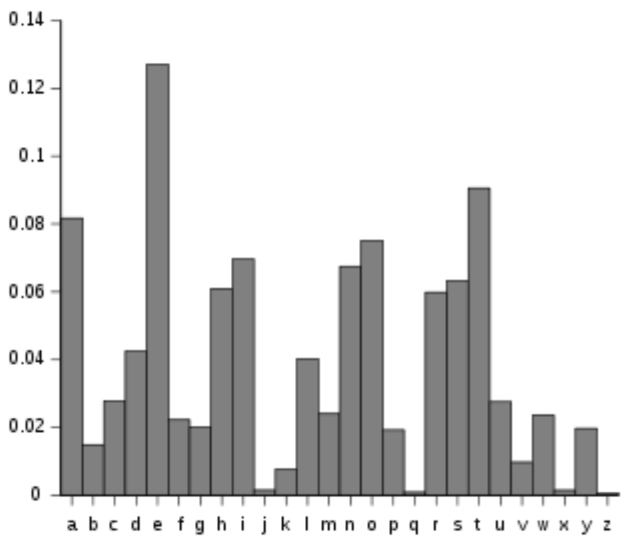
```
>>> s="hello everyone this is a sentence that has a number of
words in it"
>>> s.split()
['hello', 'everyone', 'this', 'is', 'a', 'sentence', 'that',
'has', 'a', 'number', 'of', 'words', 'in', 'it']
>>> l=s.split()
>>> ''.join(l)
'helloeveryonethisisasentencethathasanumberofwordsinit'
>>> '_'.join(l)
'hello_everyone_this_is_a_sentence_that_has_a_number_of_words_in_
it'
>>> ' '.join(l)
'hello everyone this is a sentence that has a number of words in
it'
```

Splitting and joining strings is a very common way of breaking down sentences and paragraphs into a list of words. When you do a Google query, your sentence is split into separate words and each one is used as a **key** into a massive distributed dictionary split across thousands of computers. The process of breaking text up into individual words using `split()` is called **indexing**.

TRY IT OUT #1: Try splitting some sentence strings into lists of words. Now join them back up into sentences. Try making some sentences with interesting joins between the words.

Frequency Analysis

We've seen that `split()` is a very handy tool to help us break apart any kind of text to understand all the words it is made of and how many times they occur. And `list()` can be used to break down text into individual characters. This technique when applied to a secret code is called **frequency analysis**. Any simple form of substitution cipher like Caesar can be broken using it. Frequency analysis involves counting the frequency of each letter in a sufficiently large piece of **cipher text** and comparing it with the distribution of characters in the cipher. Even if strange characters are used as in the case with the Pigpen cipher, they can still be analysed for their frequency.

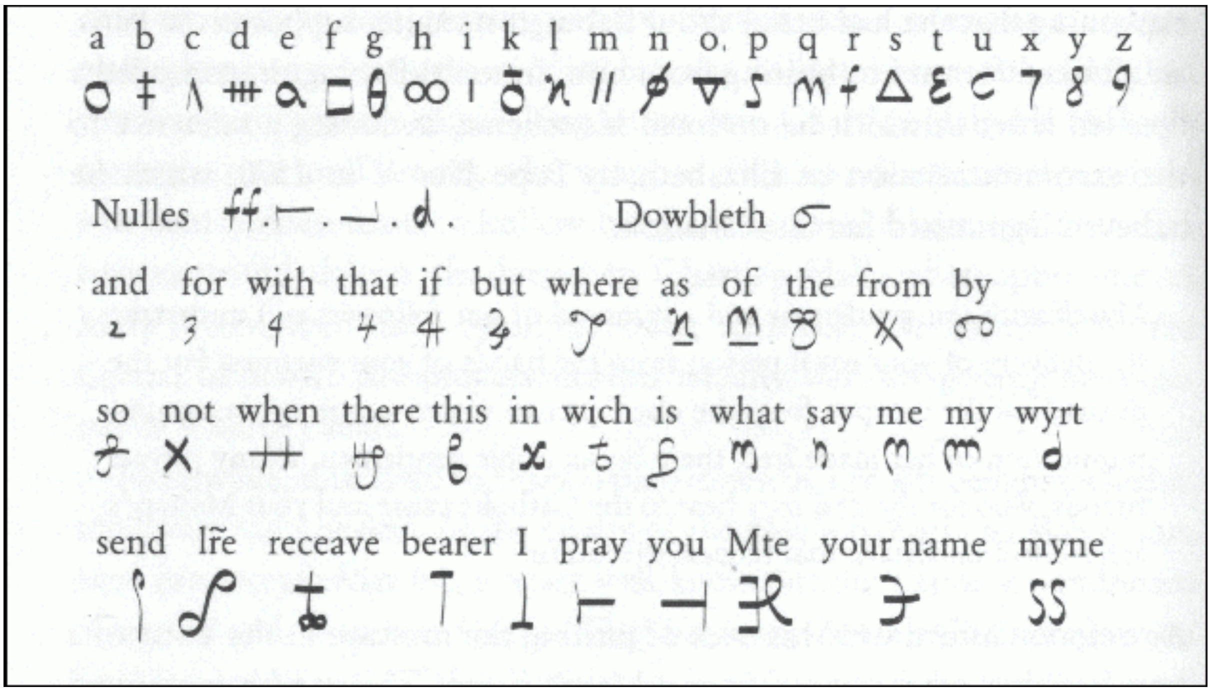


Typical distribution of letters in English language text

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

TRY IT OUT #2: Compare these frequency counts with the Scrabble values of these letters ☺

Often in history, whenever a new "unbreakable" cipher has been developed, it has eventually been broken. In the 16th century, Mary Queen of Scots used a more complex variant of a substitution cipher called a **nomenclator** to exchange secret messages with conspirators who wanted to assassinate Queen Elizabeth I. In a nomenclator, in addition to the codes for each letter, certain common words are also encoded with shortcut characters. All to no avail for the Mary Queen of Scots cipher was decoded by Thomas Phelippes, a master codebreaker. Once the plot was unveiled, it provided Elizabeth with the reason to execute Mary. Here is a breakdown of the code:


a b c d e f g h i k l m n o p q r s t u x y z
Nulles
Dowbleth
and for with that if but where as of the from by
so not when there this in wich is what say me my wyrt
send lre receave bearer I pray you Mte your name myne

TRY IT OUT #3: Can you think of how you would write some code to count the frequency of characters in a piece of text using `list()`?

To count the frequency of characters in a piece of text you need to iterate every character and if it doesn't already exist, add it as a key to a dictionary and set its initial value to 1. If the key does exist, then add 1 to the value. At the end you should have a dictionary with all the letters in and the number of times they occur in the text.

Sorting counted frequencies

Once you have a dictionary of counts of individual characters it is possible to sort the data to provide a list. In order to do that you need to convert each key-value pair in the dictionary into a new data type called a **tuple**. A tuple is a fixed list of often two values that you cannot append or delete from. Crucially a tuple can be **sorted** according to its first value. So we need to switch the letter and its frequency around in each tuple in the list and then sort and return the list. The code to do this is listed below. It's important that you study and understand how this works:

```
>>> frequencies={‘e’:20,‘x’:2,‘t’:12,‘p’:5,‘r’:11,‘q’:1}
>>> ls=[]
>>> for k,v in f.items():
...     ls.append((v,k))
...
>>> ls
[(20, 'e'), (1, 'q'), (5, 'p'), (11, 'r'), (12, 't'), (2, 'x')]
>>> sorted(ls)
[(1, 'q'), (2, 'x'), (5, 'p'), (11, 'r'), (12, 't'), (20, 'e')]
>>> sorted(ls,reverse=True)
[(20, 'e'), (12, 't'), (11, 'r'), (5, 'p'), (2, 'x'), (1, 'q')]
>>> a=sorted(ls,reverse=True)
>>> a
[(20, 'e'), (12, 't'), (11, 'r'), (5, 'p'), (2, 'x'), (1, 'q')]
```

Bringing it all together ….

At this point, you have the main elements you need to build a function that will return a sorted list of counts of characters in a piece of text! That's what this week's Code Quest will focus on. As with other recent lessons, the full solution will be presented on the last page for you to look at.

SECRET CODE QUEST!

Write a Python program that counts the frequency of all characters in a string and returns a list of the most common N letters. You will need to use `list()` to help you construct a dictionary whose keys are each of the characters in the text and whose values are the frequency with which those characters appear. You will then need to convert each key-value to a tuple, sort the resulting list and return it

```
Input:
I wandered lonely as a cloud, o'er hills and dales when all at
once I saw a crowd, a host of golden daffodils
Top 1: [(' ', 22)]
Top 2: [(' ', 22), ('a', 11)]
Top 10: [(' ', 22), ('a', 11), ('l', 10), ('o', 9), ('d', 9),
('e', 8), ('s', 6), ('n', 6), ('w', 4), ('r', 3)]
```

```
from collections import defaultdict

def topLettersByCount(text,N=10):
    """
    Algorithm:
    1. split input text
    2. Create dictionary with characters as keys and their counts
    Each time you hit a new character, you create a new key
    with value to 1.  Each repeat increments the value counter.
    3. Sort on the values (frequencies) to get the ordering
    """
    letters=list(text)
    dico=defaultdict(int)   # sets initial value in dictionary to 0
    for letter in letters:
        dico[letter]+=1
    # Now sort on the counts
    a=[]
    for key,value in dico.iteritems():
        a.append((value,key))
    a.sort(reverse=True)
    tops=[(x[1],x[0]) for x in a[:N]]
    return tops

if __name__=='__main__':
    text="I wandered lonely as a cloud, o'er hills and dales when all
at once I saw a crowd, a host of golden daffodils"
    print "Input:\n%s" % text
    print "Top 1: %s" % topLettersByCount(text,1)
    print "Top 2: %s" % topLettersByCount(text,2)
    print "Top 10: %s" % topLettersByCount(text,10)
```