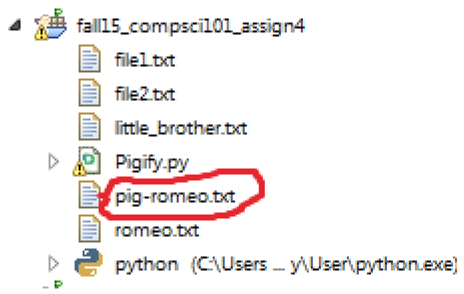
CompSci 101 Fall 2015

Assignment 4, Transform HOWTO

Getting Started

Snarf the assignment and run it. This program does several things. It reads from the file romeo.txt, it counts the number of words in romeo.txt, and it writes the file pig-romeo.txt, which is a copy of the file with extra white space removed from the file. The files to start the project are also available via the following, in addition to being accessible via snarfing.
http://www.cs.duke.edu/courses/fall15/compsci101/assign/assign4-transform/code/

After running the program the first time, you will need to right click on the PyDev project `fall15-compsci101_assign4` and **select refresh** to see the file listed in your project.



Pig-latin

These are the rules you should use to convert a word into pig-latin. We're using a hyphen to facilitate translating back from pig-latin to English. In creating pig-latin you will **not** be concerned with punctuation, so treat every character as either a vowel or not-a-vowel, and punctuation falls into the second category. ***The rules below refer to lowercase characters, but your code must work with uppercase and lowercase letters.*** In the rules below, 'A' is a vowel just as 'a' is, for example. Your code should not change the case of any letters.

1. If a word begins with 'a', 'e', 'i', 'o', or 'u', then append the string "-way" to form the pig-latin equivalent.

Word	Piglatin Version

anchor	anchor-way
oasis	oasis-way
umbrella	umbrella-way

2. If a word begins with a non-vowel (we will call this a consonant, but it could be a number, punctuation, or something else), move the prefix before the first vowel to the end with "ay" appended. Use a hyphen and treat 'y' as a vowel. If 'y' is the first letter of a word it should be considered a consonant.

word	Piglatin Version
computer	omputer-cay
yesterday	esterday-yay
strength	ength-stray
rhythm	ythm-rhay

3. Words that begin with a 'qu' should be treated as though the 'u' is a consonant.

word	Piglatin Version
quiz	iz-quay
queue	eue-quay
quay	ay-quay

A few words will not conform to these rules, but the rules should always be used. If a word contains no vowels it should be treated as though it starts with a vowel. For example "zzz" will be translated to "zzz-way".

It is possible that different words will be transformed to the same pig-latin form. For example, "it" is "it-way", but "wit" is also "it-way" using the rules above.

You'll need to write a function to turn a file into piglatin, that will be done by turning each word of the file into a piglatin equivalent. You can store the piglatin equivalent using the function **`writeFile`** in Pigify.py. This will help you unpigify a word.

In the module Pigify you should write the four functions below.

- **`def pigall(st)`**
- **`def pigword(word)`**
- **`def unpigall(st)`**
- **`def unpigword(word)`**

The **`pigall`** and **`unpigall`** functions take a string as a parameter and return a string that consists of each word in the original string either turned into piglatin or translated back from piglatin into English, by trying to undo the piglatin. Words are separated by whitespace. Here's the body of **`pigall`**, for example, which will work if you write function **`pigword`**.

```
def pigall(st):
    all = []
    for word in st.split():
        all.append(pigword(word))
    return ' '.join(all)
```

You should be sure you understand how **`pigall`** works, the purpose of **`.join`**, what the function **`pigword`** does, and by extension how **`unpigall`** and **`unpigword`** will work too.

The program you submit should have the main block below (part of the Pigify.py module you'll snarf) that you will have to add some code to:

```
if __name__ == '__main__':
     # start with reading in data file
     words = readFile("romeo.txt")
     print "read",len(words),"words"
     result = ' '.join(words)
     # convert to piglatin and write to file
     pigstr = pigall(result)
     writeFile(pigstr.split(),"pig-romeo.txt")
     print "PIGIFIED romeo.txt"
     print pigstr[0:100]

     # ****** replace comments with code ******
     # read in pigified file
     # ADD CODE HERE
     # unpigify file that was read
     # ADD CODE HERE
     # write to file "unpig-romeo.txt"
     # ADD CODE HERE
     print "UNPIGIFIED romeo.txt"
     # ADD CODE HERE
```

Caesar Cipher Encryption

You'll need to be able to encode (or decode) using a Caesar cipher. Details on encrypting and decrypting are given below. You should start by creating a new Python module Caesar.py -- create a main block like the one that came in Pigify

```
if __name__ == '__main__':
```

You should copy the functions **`readFile`** and **`writeFile`** from the Pigify.py module. When you're done creating the functions below you should include in main code code as defined below.

We include basic instructions and hints here, but you should absolutely read more about Caesar ciphers with some online reading including:

- https://en.wikipedia.org/wiki/Caesar_cipher
- https://inventwithpython.com/chapter14.html

For full credit you must use the approach outlined here, in this document, which uses the string method .find, and does not use the concept of adding numbers to characters and the chr and ord functions.

To convert a letter character to its rotated Caesar-equivalent create a string of letters in order of the alphabet, and then a version of this string that represents a shifted-version. We show an example below with a shift of four, but it generalizes to any shift. You can create the shifted version automatically with the slicing operator as shown.

Using a shift of 4 you'll get the strings below. Note that **`alph[:4] == "ABCD"`** and **`alph[4:] == "EFGHIJKLMNOPQRSTUVWXYZ"`**

```
alph   = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
caesar = "EFGHIJKLMNOPQRSTUVWXYZABCD"
```

To encode the letter 'S', you would be able to use **`alph.find('S')`** which returns the value 18, then using **`caesar[18]`** you see that the encoding of 'S' with a shift of 4 is 'W'. Using this approach will help you write code to translate any alphabetic character.

- Find the index of the character in alph using find, call this value **`findex`**
- The encoding is **`caesar[findex]`**

There are other ways to do the CAESAR cipher using the functions chr, ord and the % operator, but the approach suggested by using indexing and the strings above is much easier to get working. You must use the approach above with .find and [ ] for full credit.

You can determine if a string is a alphabetic character using the boolean method **`.isalpha()`** --- for example **`'a'.isalpha()`** evaluates to True, **`'7'.isalpha()`** evaluates to False, and **`','.isalpha()`** evaluates to False. I***n your program you should only transform alphabetic characters, not others.***

To encrypt all the words in a string, write a function encrypt:

```
def encrypt(str, shift)
```

where str is a string, shift is an index in range 0-25. The function **`encrypt`** should return a string that represents the words of str, but each word encrypted using a Caesar cipher. Here's the code, it assumes you have a helper function **`shiftword`** that does the work of encrypting one word. This code breaks the string into a list of words, encrypts each word, and re-assembles the encrypted words into a string.

```
def encrypt(str, shift):
    all = []
    for word in str.split():
        all.append(shiftword(word,shift))
    return ' '.join(all)
```

Summary: you'll write code to encrypt a string using a Caesear cipher. This will be in the module Caesar.py, you'll include at least these functions.

- **`encrypt` (started)**
- **`shiftword`**

Caesar Cipher Decryption

You'll need to write code to decrypt a file that's been encrypted with a Caesar cipher. You'll do this using two different methods. The first method will require you, an intelligent person, to determine the shift needed for decrypting. You'll do this by trying each possible shift from 0-25, and looking at the output. For the second method, you'll implement a completely automated approach to finding the shift to decrypt text that was encrypted using a Caesar cipher.

Eyeball Decryption

You'll be able to find what the shift-value is for decrypting a Casesar-encrypted string by simply eye-balling the results of trying each possible shift. For example: Suppose you were trying to decrypt this string:

> ***"Bxvncrvnb rc'b njbh cx lxdwc oaxv 1-10, kdc wxc jufjhb"***

Using each shift-value from 1-25 generates the output below (a shift of zero results in no change to the string being encrypted). Can you find the original string that was encrypted?

```
1 Cywodswoc sd'c okci dy myexd pbyw 1-10, led xyd kvgkic
2 Dzxpetxpd te'd pldj ez nzfye qczx 1-10, mfe yze lwhljd
3 Eayqfuyqe uf'e qmek fa oagzf rday 1-10, ngf zaf mximke
4 Fbzrgvzrf vg'f rnfl gb pbhag sebz 1-10, ohg abg nyjnlf
5 Gcashwasg wh'g sogm hc qcibh tfca 1-10, pih bch ozkomg
6 Hdbtixbth xi'h tphn id rdjci ugdb 1-10, qji cdi palpnh
7 Iecujycui yj'i uqio je sekdj vhec 1-10, rkj dej qbmqoi
8 Jfdvkzdvj zk'j vrjp kf tflek wifd 1-10, slk efk rcnrpj
9 Kgewlaewk al'k wskq lg ugmfl xjge 1-10, tml fgl sdosqk
10 Lhfxmbfxl bm'l xtlr mh vhngm ykhf 1-10, unm ghm teptrl
11 Migyncgym cn'm yums ni wiohn zlig 1-10, von hin ufqusm
12 Njhzodhzn do'n zvnt oj xjpio amjh 1-10, wpo ijo vgrvtn
13 Okiapeiao ep'o awou pk ykqjp bnki 1-10, xqp jkp whswuo
14 Pljbqfjbp fq'p bxpv ql zlrkq colj 1-10, yrq klq xitxvp
15 Qmkcrgkcq gr'q cyqw rm amslr dpmk 1-10, zsr lmr yjuywq
16 Rnldshldr hs'r dzrx sn bntms eqnl 1-10, ats mns zkvzxr
17 Sometimes it's easy to count from 1-10, but not always
18 Tpnfujnft ju't fbtz up dpvou gspn 1-10, cvu opu bmxbzt
19 Uqogvkogu kv'u gcua vq eqwpv htqo 1-10, dwv pqv cnycau
20 Vrphwlphv lw'v hdvb wr frxqw iurp 1-10, exw qrw dozdbv
21 Wsqixmqiw mx'w iewc xs gsyrx jvsq 1-10, fyx rsx epaecw
22 Xtrjynrjx ny'x jfxd yt htzsy kwtr 1-10, gzy sty fqbfdx
23 Yuskzosky oz'y kgye zu iuatz lxus 1-10, haz tuz grcgey
24 Zvtlaptlz pa'z lhzf av jvbua myvt 1-10, iba uva hsdhfz
25 Awumbquma qb'a miag bw kwcvb nzwu 1-10, jcb vwb iteiga
```

You can determine that a Caesar shift of 17 was used by ***examining the output from trying each shift from 1-25***. You can see that the only output that looks like English is the one whose shift is labeled 17. You'll do this by writing a function that takes a string (encrypted) as a parameter and tries all possible shifts from 0-25, printing the results (you should only print he first 80 characters of the result using slicing --- that's enough to print and eyeball/examine). We'll add the zero shift in case the string passed in wasn't encrypted.

```
    def eyeball(encrypted)
```

This function should ***NOT*** return a value, but should print 26 rows, labeled with the value of the shift being applied, an int from 0-25 inclusive, and the first 80 characters of the string that results from applying a Caesar-cipher shift to the string parameter. You'll then use your own cryptological judgment in determining the original message, that is in decrypting the encrypted text. You'll find the original text from the files *file1.txt* and *file2.txt* that you snarfed for this assignment

The original string, which you can see is found with a Caesar shift of 17 in the list above, can be determined by eyeballing all shifts, then running encrypt with the identified shift.

Chi-Squared Decryption

To automate decryption you'll use a statistical measure called a chi-squared test. For full details you can read online, some resources are linked here, you may find these useful for more details than what's given here, but enough details to implement the automated decryption are given in this document.

- http://practicalcryptography.com/cryptanalysis/text-characterisation/chi-squared-statistic/
- http://mitchellkember.com/blog/post/caesar-cipher/
- http://www.cs.trincoll.edu/~crypto/historical/caesar.html

The equation below is from the first resource above. The math may be unfamiliar, so we'll explain it below the formula.

$$\chi^2(C,E) = \sum_{i=A}^{i=Z} \frac{(C_i - E_i)^2}{E_i}$$

The idea with a chi-squared value is to measure how "close" a decrypted text is to what would be expected with a real English text.

Here's a way to understand the formula. First, there are 26 counts that you'll need to use the chi-squared formula. These are the counts of how many times 'A', 'B', … 'Z' occurs in the string being decrypted. So you'll need to write code, a helper function to determine these counts. Treat upper and lowercase letters as the same in establishing counts. ***Be sure to write a helper function for this, the function should have a string parameter and should return a list of the number of times each character 'A' - 'Z' occurs, a list of 26 integer values. Treat uppercase letters the same as the corresponding lowercase letter.*** The helper function should take a string as a parameter and return a list of counts such that **`list[0]`** is count of number of a's in string parameter, **`list[1]`** is number of b's, ... and **`list[25]`** is number of z's. Treat upper and lower case letters the same in counting. Call this function **`alphaFreq`**, in writing it you may want to use the statement: **`dex = ord(ch) - ord('a')`** to assign to dex the value 0 for 'a', 1 for 'b', … 24 for 'y', and 25 for 'z'.

You'll find the square of the difference between the actual count of each letter in the encrypted text and the expected count for each letter. The expected count is based on what happens in English, so we're assuming the message that was encrypted is in English. In the formula, this is the value $(E_i - C_i)^2$ in the equation for chi-squared. The subscript means that this square of differences should be done for 26 values of i, from 'A' to 'Z'. The value of C_i is what your program calculates -- the counts of each character in the encrypted text. The expected values are calculated from the probabilities/percentages below which you can read about in the articles linked above.

```
freqs = [8.04, 1.48, 3.34, 3.82, 12.49, 2.40, 1.87, 5.05,  7.57,
         0.16, 0.54, 4.07, 2.51, 7.23, 7.64, 2.14, 0.12, 6.28,
         6.51, 9.28, 2.73, 1.05, 1.68, 0.23, 1.66, 0.09]
```

These are the 26 frequencies *expressed as percentages* of the characters 'A'-'Z' that appear in books scanned by Google, there are more than 3 trillion characters counted to create the values in the list **`freqs`** shown. These values are percentages, but the value needed for each E_i in the chi-squared formula is a count. ***To get a count, divide the percentage by 100, and multiply it by the number of alphabetic characters in the encrypted string.*** To be concrete, suppose that there are 2500 alphabetic characters in the encrypted string. The number of expected A's is calculated by taking **`freqs[0]`**, dividing by 100 to get a probability, then multiplying by 2500

8.04/100*2500 = 201

So the expected number of A's is 201, similarly the expected number of Z's is 2.25 which is 0.09/100*2500. You'll need these expected counts for each character, so you'll need the number of alphabetic characters in the encrypted string (the 2500 in the example above). You'll find the square of the difference between the actual count of A's and the expected count, and divide this by the expected count. That's the $(E_i-C_i)^2/E_i$ in the formula -- sum up 26 of these and that's the value of chi-square for an encrypted text. If you find this chi-squared value for all 26 shifts 0-25, the smallest of these is the shift you'll use for decrypting. So you should write a function

```
def chisquare(encrypted):
```

That takes an encrypted string as a parameter and returns the shift needed to decrypt the string. The shift is the smallest chi-squared value from calculating chi-square from each of the possible

26 shifts from 0-25, and using the minimal of these to return the shift that generates the minimal value. You can test your function on file1.txt and file2.txt.

In writing chisquare you'll need at least one helper function as described above, you may find it useful to write several helper functions. ***Be sure to write comments for each one you write.***

You'll add at least the functions below, likely more

- **`chisquare`**
- **`alphaFreq`**

You'll also want to be sure you've tested the decrypting functions and write about them in the README you submit.

Caesar.py main

Your Caesar.py program should include code in main to do the following, each bullet below should include a print statement that identifies the output to someone running the program, viewing the results in the console. For example before decrypting file1.txt by eyeballing it you might want to print a line that says "Eyeballing from file1:" and then follow that by the results of calling the appropriate function.

- read in romeo.txt , encrypt it with a Caesar cipher of 19, and write the result to a new file named crypt-romeo.txt. Print the first 80 characters of the encrypted file to the console as well.
- read in the file1.txt and print the 26 lines resulting from eyeballing it.
- read the file file2.txt, call the function chisquare with the file and print the result returned by the function. Then print the result of decrypting with the returned value.