6.S189 Homework 2, Optional Problems

http://web.mit.edu/6.s189/www/materials.html

Exercise OPT.1 – Pig Latin Sentences

This optional problem builds on the work you did in Exercise 5.2. Save your work for this problem in a new file, `pig_latin.py`; you may wish to reuse the code you wrote before to help with this exercise.

Converting one word to Pig Latin is okay, but it would be more useful to be able to convert whole sentences; so for this exercise, we'll use `raw_input` to ask the user for a full sentence and translate it, word by word. It's tricky for us to deal with punctuation and numbers with what we know so far, so instead, ask the user to enter only words and spaces. You can convert their input from a string to a list of strings by calling `split` on the string; also, you can use `lower` to make a string all lowercase:

```
>>> phrase = 'My namE is JohN SmIth'
>>> word_list = phrase.split()
>>> print word_list
['My', 'namE', 'is', 'JohN', 'SmIth']
>>> lowercase_phrase = phrase.lower()
>>> print lowercase_phrase
'my name is john smith'
```

Using a list of words, you can go through each word and convert it to Pig Latin.

Hint: It will make your life much easier - and your code much better - if you separate tasks into functions, e.g. have a function that converts one word to Pig Latin rather than putting it into your main program code.

More extensions: Once you have your program working, make it interactive such that it keeps translating phrases into pig latin until the user enters in the phrase `QUIT`. Or, you can add in some more complex Pig Latin rules - for example, words that start with "th", "st", "qu", "pl", or "tr" should move both of those letters to the end.

Eg, "stop" → "opstay", and "there" → "erethay"

There are many other Pig Latin rules that you can find online if you want a true converter. Finally, you could try and deal with punctuation by looking for it within a string and moving it to the end of the word (the solutions I wrote only handle commas, periods, `!`, `?`, `:` and `;` that appear at the ends of words, as they are pretty simple to handle).

Feel free to show your modified pig latin code to a staff member (although if it's close to a checkoff due date, or the lab is really busy, we may ask you to come back later), and we'll look it over with you! It's fine to show this extended version in order to get your checkoff 4.

Exercise OPT.2 – Secret Messages

OPTIONAL! This exercise is tricky! Be sure to ask the LAs for help if you need them!

The goal of this exercise is to write a cyclic cipher to encrypt messages. This type of cipher was used by Julius Caesar to communicate with his generals. It is very simple to generate but it can actually be easily broken and does not provide the security one would hope for.

The key idea behind the Caesar cipher is to replace each letter by a letter some fixed number of positions down the alphabet. For example, if we want to create a cipher shifting by 3, you will get the following mapping:

```
Plain:   ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:  DEFGHIJKLMNOPQRSTUVWXYZABC
```

To be able to generate the cipher above, we need to understand a little bit about how text is represented inside the computer. Each character has a numerical value and one of the standard encodings is ASCII (American Standard Code for Information Interchange). It is a mapping between the numerical value and the character graphic. For example, the ASCII value of 'A' is 65 and the ASCII value of 'a' is 97. To convert between the ASCII code and the character value in Python, you can use the following code:

```
letter = 'a'

# converts a letter to ascii code
ascii_code = ord(letter)

# converts ascii code to a letter
letter_res = chr(ascii_code)

print ascii_code, letter_res
```

Start small. Do not try to implement the entire program at once. Break the program into parts as follows:

1. Create a file called `cipher.py`. Start your program by asking the user for a phrase to encode and the shift value. Then begin the structure of your program by entering in this loop (we'll build on it more in a bit):

   ```
   encoded_phrase = ''

   for c in phrase:
      encoded_phrase = encoded_phrase + c
   ```

   What does this loop do? Make sure you understand what the code does *before* moving on!

2. Now modify the program above to replace all the alphabetic characters with '`x`'. For example:

   ```
   Enter sentence to encrypt: Mayday! Mayday!
   Enter shift value: 4
   The encoded phrase is:  Xxxxxx! Xxxxxx!
   ```

   We are going to apply the cipher only to the alphabetic characters and we will ignore the others.

3. Now modify your code, so that it produces the encoded string using the cyclic cipher with the shift value entered by the user. Let's see how one might do a cyclic shift. Let's say we have the sequence:

```
012345
```

If we use a shift value of 4 and just shift all the numbers, the result will be:

```
456789
```

We want the values of the numbers to remain between 0 and 5. To do this we will use the modulus operator. The expression x%y will return a number in the range 0 to y-1 inclusive, e.g. 4%6 = 4, 6%6 = 0, 7%6 =1. Thus the result of the operation will be:

```
450123
```

Hint: Note that the ASCII value of 'A' is 65 and 'a' is 97, not 0. So you will have to think how to use the modulus operator to achieve the desired result. Apply the cipher separately to the upper and lower case letters.

Here is what you program should output:

```
Enter sentence to encrypt: Mayday! Mayday!
Enter shift value: 4
The encoded phrase is:  Qechec! Qechec!
```

Feel free to show your `cipher.py` code to a staff member (although if it's close to a checkoff due date, or the lab is really busy, we may ask you to come back later), and we'll look it over with you!