CSCI 1100 — Computer Science 1 Homework 2
Strings

Overview

This homework is worth **75 points** toward your overall homework grade, and is due Thursday, September 18, 2014 at 11:59:59 pm. There are three parts to the homework, each to be submitted separately using the homework server. Remember to name your submissions, `hw2part1.py`, `hw2part2.py`, `hw2part3.py`. If any part is late you will be charged late days according to when the last part of the homework is submitted. The homework submission site will be available on Monday, September 15.

Although you have only learned a small fraction of the basic programming constructs, you can begin to write some interesting programs. Note however that part 3 requires material we will cover on Monday's lecture, namely if statements. The rest can be done with the material we have seen so far. Remember, we are going to be grading you on whether you follow the program format we have shown in Lecture 4 as well as correctness. Make sure your programs follow this program structure.

General Comments: Please read this first

Starting with this homework, almost all homeworks will require user input. To make sure we align your program with the correct inputs, we must note a few important things:

- Your program must read the same number of inputs as are required according to the given problem. For example, if we tell you to read a name first and an email next, that means your program must have 2 `raw_input` statements. If it does not, you will get an error like:

  ```
  EOFError:  EOF when reading a line
  ```

  This means either you are trying to read too many or too few inputs. Read the problem specification carefully.

- In all homeworks, we will use a convention from now. If we ask you to read an input, you will immediately print that input. For example, the following is the correct method:

  ```
  name = raw_input("Please enter a name==> ")
  print name
  email = raw_input("Please enter your email==> ")
  print email
  ```

- Finally, your output will look slightly different in the submission server than what you see in Wing. For the above program, the following is a possible output in Wing:

  ```
  Please enter a name==> Rensselaer
  Rensselaer
  Please enter your email==> rpiinfo@rpi.edu
  rpiinfo@rpi.edu
  ```

  But, if the same inputs are used in the submission server, you will get the following:

```
Please enter a name==> Rensselaer
Please enter your email==> rpiinfo@rpi.edu
```

And, if you forgot to add the print statements, you would actually see something like this in the submission server which will be considered incorrect output:

```
Please enter a name==> Please enter your email==>
```

The difference — and this is not really important for actually completing your homework — is that for each part of the homework, we place all the input into a file and do what's called "running from the command-line." In particular, in the above example, we are using an input a file (let's assume it is called `input.txt`) that contains two strings: `Rensselaer` and `rpiinfo@rpi.edu` in two lines.

When a program is run from a command shell, we use a command-line of the form:

```
python part1.py < input.txt
```

Unfortunately, the free version of the WingIDE that we have been using in class does not allow us to specify this input.

This is easy to do if you are on a Linux machine or a Mac where you need to just start a Terminal window. You can create a command window in Windows or download and run the linux-derived Cygwin tools. However, for now, do not worry about running your program from the command-line. If you want to learn how to do this, come and ask advice from us during office hours.

Fow now, simply make sure that you print all input that your program reads. I will not repeat this in every homework and assume that this is the way we read input from now on.

Part 1: Madlibs

In this part you will write code to construct a madlib given below:

```
Look, <proper name> ...
  I can see you're really <emotion> about this ...
  I honestly think you ought to <verb> calmly ...
  take a <adjective> <noun> and think things over ...
  I know I've made some very <adjective> decisions recently,
  but I can give you my complete <noun> that my work will be back
  to <adjective>.
```

You will ask the missing pieces of the following madlib to the user using the (`raw_input`) function we learnt in class on thursday. For each input, you will ask the specific type of word required.

You will then take all the user specified inputs, and construct the above madlib. You can use any string method we have learnt. Make sure your output looks like the above paragraph, except that the missing information is filled in with the user input. Here is an example run of the program:

```
proper name ==> Brad
Brad
emotion ==> happy
happy
verb ==> eat
eat
adjective ==> silly
silly
noun ==> mouse
mouse
adjective ==> hungry
hungry
noun ==> computer
computer
adjective ==> cute
cute
Here is your output:
Look, Brad ...
 I can see you're really happy about this ...
 I honestly think you ought to eat calmly ...
 take a silly mouse and think things over ...
 I know I've made some very hungry decisions recently,
 but I can give you my complete computer that my work will be back
 to cute.
```

Part 2: Decipher the hidden meaning

Write a program that asks the user for a string written in a cipher using (`raw_input`). Remember to print it after you read it as before. You need to decipher and print the sentence given. You must also return the total number of words in the final sentence that start with the letter `s`, `a` or `c` (including the first word).

Here is an example run of the program:

```
Please enter a sentence ==> why stwt saz azritwtyyys
why stwt saz azritwtyyys
Deciphered as ==>  why so serious
Number of words that start with s: 2
Number of words that start with a: 0
Number of words that start with c: 0
```

Your program must use two functions:

- Write one function that takes as an argument a string to decipher, and returns a string that is the deciphered version of it.
- Write a second function that takes a string and a letter as an argument and returns the total number of words that start with the given letter.

  You can assume there is no punctuation and all the letters are lower case. (**Hint: words are separated by spaces. If there is no space in the beginning of the sentence, you can always add one, right?**)

String function to use

To solve this part, you need a string function called `replace`. That is the only function you need, but we will see others in class on monday. You are free to use any function you desire. The replace function is called as follows:

```
>>> name = "Rensselaer"
>>> name.replace("e", "^_^")
'R^_^nss^_^la^_^r'
>>> name
'Rensselaer'
>>> "abracadabra".replace("ab","z")
'zracadzra'
```

Basically, `mystring.replace(old,new)` replaces all instances of substring `old` with the string `new`, and returns a new string. You can see that it does not change the actual string it operates on.

Cipher rules

The sentence to be deciphered is constructed by using a replacement cipher with the following rules:

`'he' => 'bb'`	replace all occurrences of string `he` with `bb`
`'e' => 'az az'`	valid only for any `e` that is not part of `he`, replace `e` with `az az`
`'an' => 'he'`	replace all occurrences of string `an` with `he`
`'th' => 'xx'`	replace all occurrences of string `th` with `xx`
`'u' => 'yyy'`	replace all occurrences of string `u` with `yyy`
`'o' => 'twt'`	replace all occurrences of string `o` with `twt`
`' a' => 'rxr'`	for any `a` that is not part of `an`, note the space before `a`.

For example cipher for `methane` is `maz azxxheaz az`. Here is how we get this:

```
>>> 'methane'.replace('th','xx')
'mexxane'
>>> 'mexxane'.replace('e','az az')
'maz azxxanaz az'
>>> 'maz azxxanaz az'.replace('an','he')
'maz azxxheaz az'
```

The trick here is that we can do the enciphering (and deciphering) by continuously replacing a part of the string with a new one. But, the replacement must be done in a logical order. For example, in the above example, if we replaced `an` with `he` first, then we cannot use the rule of replacing `e` with `ax ax` because the newly added `e` was not part of the original word and should not be changed.

Part 3: Playing with Strings and Numbers

Write a program that takes as input two numbers, `height` and `width` of a rectangle. It then prints a rectangle with `height` lines and `width` characters in each line. The rectangle should contain information regarding its dimensions and area inside (centered vertically but not horizontally). If any part of the information (including two stars on each end and a space before and after the line) does not fit in the rectangle, then print the complete information after the rectangle.

Here are two example runs of the program:

```
Height==> 6
6
Width==> 15
15
***************
*             *
* h: 6, w: 15 *
* area: 90    *
*             *
***************
```

```
Height==> 6
6
Width==> 10
10
**********
*        *
*        *
*        *
*        *
**********
h: 6, w: 10
area: 60
```

To simplify this problem, you will make some assumptions:

- Assume the user enters a correct integer for height and width.
- Assume the height is an even number greater than or equal to 4.

This part will require you to use a lot of the constructs we have learnt so far. This is a good example of something that looks hard at first, but when you break it down, it is not so difficult. You do not need loops, and you should try hard not to use loops. You are not required to use functions. However, you will need an if statement for a special case explained below. Leave that part as the last thing to work on. We will see if statements on monday. Here are a few pointers for you to get started:

- Read and print values. That should be easy to do.
- Remember that `raw_input` reads values as a string, but you need to convert them to integer to compute the area. You will need these integers throughout your program.
- You can create two separate strings for the information to be displayed. One will have the height and width information, and the other will have the area. Create and print these two strings. Don't worry about printing them inside the rectangle yet.
- Make sure you can create a rectangle without anything in it of the correct dimensions. Create this rectangle as a string with new lines in it. Basically, it is a special instance of what you have done in Lab 2. Once you are done, print it out.
- Now, you have to figure out how to place the two information strings inside your rectangle. First step is to figure out how to print them with the appropriate number of spaces and a star symbol at the two ends. When you are done, now figure out how to place it in the middle of it.
- Finally, the final step is figuring out when one of the message lines is too long to fit in it, and have a different rectangle string for this case. If you have come this far, you will find this is not that hard.