CSCI 1100 — Computer Science 1 Homework 2 Functions

Overview

This homework is worth **75 points** total toward your overall homework grade (each part is 25 points), and is due Thursday, September 24, 2015 at 11:59:59 pm. There are three parts to the homework, each to be submitted separately. All parts should be submitted by the deadline or your program will be considered late.

Note on grading. Make sure you read the comments from Homework # 1. They apply to this and all future homeworks. In all parts of the homework, we will specify which functions you must provide. Make sure you write these functions, even if they can be very simple. Otherwise, you will lose points. We will write more complex functions as the semester goes on. In addition, we will also look at program structure, names of variables and functions in grading this homework. The homework submission server URL is below for your convenience:

```
https://submit.cs.rpi.edu/index.php?course=csci1100
```

Your programs for each part for this homework should be named:

```
hw2_part1.py
hw2_part2.py
hw2_part3.py
```

respectively. Each should be submitted separately.

Part 1: Geometry and rain!

This problem is a warm up of the use of functions. You will be asked to write a few simple functions. Did you know that Panama Canal runs on rain water? Every time a ship passes through a lock, the canal is filled from a lake in the middle of the Panama canal, and then this water is released to the ocean and lost forever. Basically, this man-made lake is filled with rain water. Yes, it rains a lot in Panama. We will compute the change in the depth of the lake during the dry season, i.e. the three months in the year when it rains a bit less.

We will make some simplifying assumptions: locks are rectangular prisms and the lake is a cylinder. We will ask to the user the dimensions of the rectangular prism (`length, width, height`) and the `radius` of the cylinder. You will then compute and output the following:

- Total amount of water needed to fill the rectangular prism 70 times a day for 3 months, assuming a month is 30 days (basically, about 35 ships pass through two locks every day.)

  Remember the volume of a rectangular prism is given by `length * width * height`. Write a function `volume_prism(length, width, height)` to compute this and use it in your computation.

- Then, you will find the height of a cylinder with the given `radius` needed to hold this much water. Remember, the volume of a cylinder given by `pi* radius**2 * height`.

  Write a function `find_length(radius, volume)` that finds and returns the depth of a cylinder needed to hold the computed amount of water. **You must use the math module for the value of pi**.

Here is an expected output of your program (how it will look on Wing IDE):

```
Length of rectangular prism (m) ==> 12.5
12.5
Width of rectangular prism (m) ==> 15.8
15.8
Height of rectangular prism (m) ==> 18.1
18.1
Water needed for (12.5m,15.8m,18.1m) locks is 22520925.0m^3.
Radius of cylinder (m) ==> 1150
1150
Lake with radius 1150.0m will loose 5.4m depth in three months.
```

Remember, using formatted strings will be very useful in this homework.

When you have tested your code, please submit it as `hw2_part1.py`.

Part 2: Find the hidden message!

Write a program that asks the user for a sentence written in a cipher using (`raw_input`). You need to decrypt and print the resulting sentence. Then, read a second sentence in plain English using (`raw_input`) and convert it to a cipher by encrypting it. For both sentences, you must also print the total difference in length between the cipher and clear text versions. The difference should always be printed as a positive number.

Here is an example run of the program (what you will see on Wing):

```
Enter cipher text ==> wheie stzzt saz azritzztyyys
wheie stzzt saz azritzztyyys

Deciphered as ==> why so serious
Difference in length ==> 14

Enter regular text ==> back off man I am a scientist
back off man I am a scientist

Encrypted as ==> back tzztff muq Irxrmrxr sciaz azntist
Difference in length ==> 9
```

The encryption rules are based on a pretty simple set of string replacements, they should be applied in this order exactly:

`' a' => 'rxr'`	replace any `a` after a space with `rxr`.
`'he' => 'vw'`	replace all occurrences of string `he` with `vw`
`'e' => 'az az'`	replace any remaining `e` with `az az`
`'y' => 'eie'`	replace all occurrences of string `y` with `eie`
`'u' => 'yyy'`	replace all occurrences of string `u` with `yyy`
`'an' => 'uq'`	replace all occurrences of string `an` with `uq`
`'th' => 'aige'`	replace all occurrences of string `th` with `aige`
`'o' => 'tzzt'`	replace all occurrences of string `o` with `tzzt`

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

```
>>> 'methane'.replace('e','az az')
'maz azthanaz az'
>>> 'maz azthanaz az'.replace('an','uq')
'maz azthuqaz az'
>>> 'maz azthuqaz az'.replace('th','aige')
'maz azaigeuqaz az'
```

Of course, decrypting will involve using the rules in reverse order.

Your program must use two functions:

- Write one function `encrypt(word)` that takes as an argument a string in plain English, and returns a ciphered version of it as a string.
- Write a second function `decrypt(word)` that does the reverse: takes a string in cipher and returns the plain English version of it.

Both functions will be very similar in structure, but they will use the string replacement rules in different order. You can now test whether your functions are correct by first encrypting a string, and then decrypting. The result should be identical to the original string (assuming the replacement rules are not ambiguous).

Use these functions to implement the above program. When you have tested your code, please submit it as `hw2_part2.py`.

Part 3: A simple trick (numerical functions and one if statement)!

This is a simple parlor trick attributed to Albert Einstein, but it probably dates to an earlier date!

The trick: First, write the number 1089 on a piece of paper, fold it, and put it away. Next, ask your friend to write down any three-digit number, emphasizing that the first and last digits must differ by at least two. Don't watch your friend doing the arithmetic. After your friend has written down the three-digit number, ask him to reverse it, then subtract the smaller from the larger.

Example: 321 - 123 = 198.
Tell your friend to reverse the new number.
Example: 198 becomes 891.
Next ask your friend to add the new number and its reverse together.
Example: 198 + 891 = 1089.

If all goes as planned, your friend will be amazed. The number you wrote down at the start –1089– will always be the same as the end result of this mathematical trick.

As we have just learnt how to do if statements, we will do a simple version of this program. Your program must contain one function called `reverse` that works as follows:

```
>>> reverse(123)
321
>>> reverse(264)
462
```

Make sure you write this function first and test it with many integer inputs.

Then, write a program that produces the output given below by reading an input value from the user and then carrying out the computation outlined above by calling the above function in two places where you reverse the number. Show the steps of the computation. Then, add a simple if statement at the end of your program: if the final result is 1089, you should print *"You see, I told you"*, otherwise, you should print *"Are you sure your input is valid?"*

Your output must match the following (this is how it will look on Wing):

```
Enter a value ==> 294
294

Here is the computation:
492 - 294 = 198
198 + 891 = 1089
You see, I told you.
```

Here is another run:

```
Enter a value ==> 242
242

Here is the computation:
242 - 242 = 0
0 + 0 = 0
Are you sure your input is valid?
```

When you have tested your code, please submit it as `hw2_part3.py`.

Finished and still want extra challenges to sharpen your programming skills? These are not extra credit but for you to do additional practice.

This homework has a single if statement, but you should practice with if statements for the upcoming exam. Here are some ideas.

- Write a program to take an input, encrypt first and then decrypt, then check whether the result is the same. Print "same" if they are the same and "different" otherwise.
- Try to do part 3 in such a way that if the person enters a number that is not valid (i.e. first and third digit do not differ by two), you immediately print a statement informing the user and do not compute anything. Otherwise, you carry out the above computation.

Other ideas are too complex for our current level. This should be sufficient!

Note that there are many other cases in which the total after this computation seems to be 1098. Later in semester, we can write a program to test every possible number. Not yet.