CSCI 1100 — Computer Science 1
Fall 2019
Homework 2: Strings and Functions

Overview

This homework, worth **100 points** total toward your overall homework grade, is due Thursday, October 3, 2019 at 11:59:59 pm. Three parts should each 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 `Submission Guidelines` document. It applies to this and all future homework assignments and will be of increasing importance. In all parts of the homework, we will specify which functions you must provide. Make sure you write these functions, even if they are 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 (see Lecture 4), and at the names of variables and functions in grading this homework.

You are not allowed to use any loops anywhere in this assignment.

Fair Warning About Excess Collaboration

For all homework assignments this semester we will be using software that compares **all** submitted programs, looking for inappropriate similarities. This handles a wide variety of differences between programs, so that if you either (a) took someone else's program, modified it (or not), and submitted it as your own, (b) wrote a single program with one or more colleagues and submitted modified versions separately as your own work, or (c) submitted (perhaps slightly modified) software submitted in a previous year as your software, this software will mark these submissions as very similar. All of (a), (b), and (c) are beyond what is acceptable in this course — they are violations of the academic integrity policy. Furthermore, this type of copying will prevent you from learning how to solve problems and will hurt you in the long run. The more you write your own code, the more you learn.

Please read the **Collaboration Policy** document for acceptable levels of collaboration and how you can protect yourself. The document was/will be distributed during class and can also be found on the Course Materials page on **Submitty**. Penalties for excess collaboration can be as high as:

- 0 on the homework, and
- an additional overall 5% reduction on the semester grade.

Penalized students will also be prevented from dropping the course. More severe violations, such as stealing someone else's code, will lead to an automatic F in the course. A student caught in a second academic integrity violation will receive an automatic F.

By submitting your homework you are asserting that you both (a) understand the academic integrity policy and (b) have not violated it.

Finally, please note that this policy is in place for the small percentage of problems that will arise in this course. Students who follow the strategies outlined above and use common sense in doing so will not have any trouble with academic integrity.

Part 1: A Penny for a Gum Ball Mickey (40 pts)

Thanks for the Gum Ball!

We are going to conduct an experiment in selling gum balls, but we are going to make a few assumptions. Assume that you are selling gum balls from a vending machine. The machine is a cube, and the gum balls are spheres. You check the machine once a week. The goal is to size the machine so that it is completely full at the start of the week, you never run out of gum balls before you get back to check the machine, and you do not have too many gum balls left at the end of the week to go stale. We will make the assumption that all the gum balls line up so that the number of gum balls along any dimension of the cube is simply the side length divided by the diameter of the spheres. So if the side length is 9.0 and the gum balls have a radius of 0.5 exactly 9 gum balls would fit along each dimension, or 729 gum balls in total. This is known as a `cubic lattice`.

Do the following:

1. First write two functions, `find_volume_sphere(radius)` and `find_volume_cube(side)` that calculate the volume of a sphere given a radius, and that calculate the volume of a cube given a side, respectively.
2. Now ask the user for the radius of the gum balls and the weekly sales.
3. Calculate the total number of gum balls that need to fit in the machine as 1.25 times the weekly sales and use this to calculate the side length of the machine in terms of an integer number of gum balls. **Hint:** You know the total number of gum balls and in a cubic lattice, you can fit the same number along each dimension, so if you can fit `N` gum balls along each dimension, then the machine holds N^3 gum balls. The math module function `ceil` will always round up and may be of use here. (We won't be cutting gum balls to fit them in the machine.)
4. Calculate a few more values: how many gum balls will actually fit given the dimension you chose (remember that there must be an integer number of gum balls along each dimension of the cube); the volume of the cube; the volume of the gum balls, and the wasted space if we put in both the number of gum balls we need to hold and how many we can hold.
5. Print these values out using the `.2f` format for all floating point values.

Two examples of the program run (how it will look when you run it using Spyder IDE) are provided in files hw2_part1_output_01.txt and hw2_part1_output_02.txt. (In order to access these files, you will need to download file `hw02_files.zip` from the `Course Materials` section of Submitty and unzip it into your directory for HW 2.)

We will test your code for the above values as well as a range of different values. Test your code well and when you are sure that it works, please submit it as a file named **hw2_part1.py** to Submitty for Part 1 of the homework.

Part 2: Find the Hidden Message (40 pts)

Write a program to determine if a simple substitution code is reversible for a given string. The program should ask the user for a sentence using `input`. The program should then encrypt the

string into a cipher, decode the cipher, and compare the result of the decode operation to the original sentence. If the decoded cipher matches the original, then the operation is reversible on the string. Otherwise, it is not reversible.

Along the way, the program should print out the cipher, the difference in length between the cipher and the original sentence, the decoded cipher, and a brief message saying whether the operation was reversible. Note that the difference in length should always be printed as a positive number.

Two examples of the program run (how it will look when you run it using Spyder IDE) are provided in files hw2_part2_output_01.txt and hw2_part2_output_02.txt (can be found inside the `hw02_files.zip` file).

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

`' a' => '%4%'`	replace any `a` after a space with `%4%`.
`'he' => '7!'`	replace all occurrences of string `he` with `7!`
`'e' => '9(*9('`	replace any remaining `e` with `9(*9(`
`'y' => '*%$'`	replace all occurrences of string `y` with `*%$`
`'u' => '@@@'`	replace all occurrences of string `u` with `@@@`
`'an' => '-?'`	replace all occurrences of string `an` with `-?`
`'th' => '!@+3'`	replace all occurrences of string `th` with `!@+3`
`'o' => '7654'`	replace all occurrences of string `o` with `7654`
`'9' => '2'`	replace all occurrences of string `9` with `2`
`'ck' => '%4'`	replace all occurrences of string `ck` with `%4`

For example the cipher for `methane` is `m2(*2(!@+3-?2(*2(`. Here is how we get this:

```
>>> 'methane'.replace('e','9(*9(')
'm9(*9(than9(*9('
>>> 'm9(*9(than9(*9('.replace('an','-?')
'm9(*9(th-?9(*9('
>>> 'm9(*9(th-?9(*9('.replace('th','!@+3')
'm9(*9(!@+3-?9(*9('
'm9(*9(!@+3-?9(*9('.replace('9', '2')
'm2(*2(!@+3-?2(*2('
```

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. We will test your code for the above values as well as a range of different values.

Test your code well and when you are sure that it works, please submit it as a file named **hw2_part2.py** to Submitty for Part 2 of the homework.

Part 3: How Do You Feel about Homework? (20 pts)

In this part of the homework, you will implement a very rough sentiment analysis tool. While the real tools use natural language processing, they all use word counts similar to the one we use here. Understanding the sentiment in messages is a crucial part of a lot of artificial intelligence tools.

Write a program that will ask the user for a string containing a sentence. The program will then compute the happiness and sadness level of the sentence using the two functions described below. If the happiness level is higher than sadness level, then the tone of the sentence is happy. If the sadness level is higher, then the tone of the sentence is sad. Otherwise, it is neutral. Find and print the tone of the sentence by first printing a sentiment line with a number of `+` equal to the number of happy words followed by the number of `-` equal to the number of sad words, followed by a simple statement of the analysis.

Two examples of the program run (how it will look when you run it using Spyder IDE 101) are provided in files hw2_part3_output_01.txt and hw2_part3_output_02.txt. (Can be found inside the `hw02_files.zip` file.)

To accomplish this you will write a function called `number_happy(sentence)` which returns the number of words in a given string called `sentence` that are happy. To do this, find the total count of the following 6 words: `laugh happiness love excellent good smile`. Here is an example run of this function:

```
>>> number_happy("I laughed and laughed at her excellent joke.")
3
```

This is because the count of happy words is 3 (laugh is repeated twice). Your code should work even if there are upper and lower case words and extra spaces in the beginning and end of the sentence.

```
>>> number_happy("   Happiness is the state of a student who started homework early.   ")
1
```

Next, write a second function called `number_sad(sentence)` that works the same way but instead counts the number of the following 6 sad words in English: `bad sad terrible horrible problem hate` (there are sadder words for sure, but no reason to depress ourselves).

```
>>> number_sad("Dr. Horrible's Sing-Along Blog is an excellent show.")
1
>>> number_sad("Alexander and the Terrible, Horrible, No Good, Very Bad Day")
3
```

Of courses, there are more than 6 words of each category. We will see how to feed them using a file and use lists to process them in future classes.

Test your code well and when you are sure that it works, please submit it as a file named **hw2_part3.py** to Submitty for Part 3 of the homework.