ECS 102 Baruch

Spring 2019

HW 5

Due: Thursday, February 28

For this assignment you will be writing 2 programs in files, each with a main.

1. Start each program with a comment like this:

# Official Name: *Amanda Kitredge*

# Nickname: *Mandy*

# email: *ajkitred@syr.edu*

# Assignment: *Assignment 3, problem 1.*

# Date: *September 13, 2018*

# *a brief description of the problem.*

Replace the *italicised fields* with appropriate information for you and for this assignment.

* In the "Official Name" field write your name as it appears on University documents.
* "Nickname" is the name you would like to be called by the course staff. If you don't use a nickname you can leave this line out.

2. In addition, each program should have comments describing the main sections of the program. They should be specific to the problem you are solving. Under each of these comments should be a chunk of code implementing the comment.

3. Good readability:

* Use meaningful names for all variables.
* Use blank lines to delineate small portions of code that work together to accomplish a task.

4. Test you program. Run each program with my sample data. Then create your own data files and run your program.

**hw5Output.docx** should contain your sample input and output files for both **A** and **B** as in the examples for each problem.

**A. MergeFiles.py**

There will be two input files for this program. One will be a list of names, one name per line. The other file will be a list of exam scores. Each line has two scores for one person. The lines are in the same order as the names, so the first name in one file corresponds to the first 2 scores in the other file, and so on.

You will create a third file, an output file, called "GradeBook.txt". In the first row you will print the name of the first person, their 2 scores, and finally the average of their two scores. You will repeat this for each row. In the last row print the class average for each exam and the overall class average (the average of the student averages).

The program should ask for the name of the file with names and the name of the file with scores.

Here is are some sample input files, **StudentNames.txt** and **StudentScores.txt** and the corresponding output, **GradeBook.txt**, though the program should handle files with more or fewer students. (Use the links on the Assignments page) You may assume that the data is good and you have the same number of student names as you have pairs of scores. Get your program working with these input files. Your output file should have neat columns (use **format.** )

Make sample input files of your own, with a different number of students from mine (i.e. not eight), but where both of your files have the same number of students as each other. In the file **hw5Output.docx** put copies of your input and output files (labeled). So, for my data it would look like

StudentNames.txt

Mohammed bin Barak

Lina Lopez

Sarah Parker

Marcus Kim

David Weider

Ada Mae Freeman

Chongzong Li

Josh Cohen

StudentScores.txt

76 85

98 80

87 79

70 99

77 97

90 99

60 77

97 98

GradeBook.txt

Mohammed bin Barak 76 85 80.50

Lina Lopez 98 80 89.00

Sarah Parker 87 79 83.00

Marcus Kim 70 99 84.50

David Weider 77 97 87.00

Ada Mae Freeman 90 99 94.50

Chongzong Li 60 77 68.50

Josh Cohen 97 98 97.50

Averages 81.875 89.25 85.56

**B. Gibberish.py**

There is a silly way children try to encode what they say so others won't understand - unless they know the trick. Here is roughly how it works:

For each word, between the first letter and the rest of the word, insert **itheg**.

So monkey becomes mithegonkey and fish becomes fithegish.

Your job is to write encoder and decoder functions for Gibberish.

Encode: Ask for the name of an input file. Do not include the extension. The function should add the **.txt**. It should also create a name for the output file by adding **Encoded** before adding the **.txt**. So if the user types **lesson** the input file will be **lesson.txt** and the output file will be **lessonEncoded.txt**

Open these files for input and output, respectively. Read the input file (one line, no punctuation.) Build an output message by adding on encoded words to the message, one word at a time. Include a space before each new word. When you are done, you should print the message, first letter of each word upper case, to the output file. Close all your files.

Decode: Assuming you have a message already in Gibberish, reconstruct the original message. When asking for the input file, again add **.txt** to the entered name. For the output file add **Decoded.txt** So if the user types

**lessonEncode** the input file will be **lessonEncoded.txt** and the output file will be **lessonEncodedDecoded.txt**

Print the decoded message to the output file entirely in upper case.

For testing, you can test each of these functions independently. For the version to turn in,

main should call encode then decode. Try it by first encoding some message, then decoding its output.

Sample dialogue:

Enter name of file with message to encode: **food**

Enter name of file with message to decode: **foodEncoded**

Sample files:

food.txt

cows make milk trees give fruit

foodEncoded.txt

Cithegows Mithegake Mithegilk Tithegrees Githegive Fithegruit

foodEncodedDecoded.txt

COWS MAKE MILK TREES GIVE FRUIT

Run the program with this sample file and a file of your own design.

Include in the file **hw5Output.docx** the name and content of your file and the output files, as I have above.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Upload:**

**MergeFiles.py**

**Gibberish.py** (with 3 functions, including main())

**hw5Output.docx**