NYU Tandon School of Engineering

CS-UY 1114 Spring 2023

Homework 08

Due: 11:59pm, April 13th, 2023

Submission instructions

1. You should submit your homework on **Gradescope**.
2. For this assignment you should turn in 4 separate `.py` files named according to the following pattern: `hw8_q1.py`, `hw8_q2_3.py`, etc.
3. Each Python file you submit should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW8 - Q1 (etc.)
Date due: 2023-04-13, 11:59pm
I pledge that I have completed this assignment without
collaborating with anyone else, in conformance with the
NYU School of Engineering Policies and Procedures on
Academic Misconduct.
"""
```

No late submissions will be accepted.

REMINDER: *Do not use any Python structures that we have not learned in class.*

For this specific assignment, you may use everything we have learned up to, **and including**, variables, types, mathematical and boolean expressions, user IO (i.e. `print()` and `input()`), number systems, and the `math` / `random` modules, selection statements (i.e. `if`, `elif`, `else`), and `for`- and `while`-loops, strings / string methods, custom functions, file IO, lists / list methods, and tuples. Please reach out to us if you're at all unsure about any instruction or whether a Python structure is or is not allowed.

Do **not** use, for example, dictionaries, lists comprehension, and/or object-oriented programming.

Failure to abide by any of these instructions will make your submission subject to point deductions.

Problems

1. **Nucleotidal Arithmetic**
2. **Of Code And Poetry**
3. **Now, Let's Make It Presentable**
4. **Gradescope™ Logic**
5. **Matryoshka Lists**

Problem 1: *Nucleotidal Arithmetic*

***NOTE**: The use of Python dictionaries is not allowed in this problem. At least not if you want any credit for it* 😃.

Remember them good ol' DNA sequences? In this case we want to create a function `update_frequencies()`, that will do just that: receive a list containing an existing list of frequencies, as well as a string representing a new DNA sequence, and update the previous numbers to reflect their added frequencies. Your function program must also print the nucleotides being added to each frequency.

That is, the following code:

```
def main():
    old_frequencies = [("A", 20), ("C", 23), ("T", 125), ("G", 4)]
    new_sequence = "ACCCGTTA"
    new_frequencies = update_frequencies(old_frequencies, new_sequence)

    print(new_frequencies)

main()
```

will result in the following output:

```
Number of nucleotides added: A -> 2 | C -> 3 | T -> 2 | G -> 1
[('A', 22), ('C', 26), ('T', 127), ('G', 5)]
```

You may assume that the list of old frequencies will be in that exact format and ordering (i.e. `A`s come first, etc.).

Problem 2: *Of Code And Poetry*

NOTE: Problems 2 and 3 should both be written in the same file, `hw8_q2_3.py`. Your header's second line should thus look like this `Assignment / Part: HW8 - Q2, Q3`.

Haiku are poems that follow a 5-7-5 syllabic structure. That is, the first line contains 5 syllables, the second contains 7 syllables, and the last contains 5 syllables:

> *Clouds murmur darkly*
>
> *it is a blinding habit—*
>
> *gazing at the moon*

— ***Basho***.

The first sentence can be broken down into five syllables (`clouds` + `mur` + `mur` + `dark` + `ly` = 5 syllables), the second sentence into 7 (`it` + `is` + `a` + `blind` + `ing` + `ha` + `bit` = 7 syllables), and the last into 5 (`ga` + `zing` + `at` + `the` + `moon` = 5 syllables).

Write a function, `is_haiku()`, that returns the bool `True` if an input string is a haiku, or the bool `False` if it is not. The function accepts one parameter, `input_string`, of the following format:

```
sample_input_string = "clouds ,mur,mur ,dark,ly /it ,is ,a ,blin,ding ,ha,bit
/ga,zing ,at ,the ,moon "
```

As you can see, syllables are separated by commas (`,`), and lines are separated by forward-slashes (`/`). Notice that the final syllables of each word contain an extra space (e.g. `"clouds "`). The function `is_haiku` will return `True` if and only if:

1. The haiku contains 3 lines.
2. The first line contains 5 syllables.
3. The second line contains 7 syllables.
4. The third line contains 5 syllables.

If your function is to return `False`, it should print a message explaining the reason to the user.

Consider the sample executions below and feel free to try your own:

```
print(is_haiku("clouds ,mur,mur ,dark,ly/it ,is ,a ,blin,ding ,ha,bit/ga,zing
,at ,the ,moon "))  # prints 'True'
print(is_haiku("clouds ,mur,mur ,dark,ly/it ,is ,a ,blin,ding ,ha,bit/ga,zing
"))  # prints 'False'
```

Output:

```
True

WARNING: The third line is not 5 syllables long.
False
```

HINT: While there are multiple ways to approach this problem, the string method `split()` may be particularly useful. See its official documentation **here**.

Problem 3: *Now, let's make it presentable*

Note: Output must **exactly** match the examples' for this problem.

Write a function, `haiku_string_parser()`, that takes in one parameter, `input_string`, checks if it is a haiku based on its structure. If it is, it will return a reformatted, easy-to-read string:

```
def main():
    haiku_string = "clouds ,mur,mur ,dark,ly/it ,is ,a ,blin,ding
,ha,bit/ga,zing ,at ,the ,moon"
    formatted_haiku = haiku_string_parser(haiku_string)
    print(formatted_haiku)

main()
```

Output:

```
clouds murmur darkly
it is a blinding habit
gazing at the moon
```

If `input_string` is not a haiku based on its structure, then `haiku_string_parser()` will simply return an empty string (you *must* use the `is_haiku()` function from problem 2 in order to receive full credit for this problem).

HINTS:

- In order to create a line-break (newline) in a string, you can use the `\n` character.
- You might want to consider using the `join()` **string method.**

Problem 4: *Gradescope™ Logic*

We've prepared a (completely anonymous) CSV file containing the midterm 1 grades–you can download it **here**.

Write a function called `get_list_of_grades()` that will:

- Accept a single parameter, `grades_filepath` (*`str`*): The address of the CSV file containing the grades data.
- Return a **list of lists**, where each nested list corresponds to a question the exam, and itself contains the grade each student got on that question.

```
def main():
    grades = get_list_of_grades("Midterm1_scores.csv")
    print(grades)

main()
```

Output (note that the output here was heavily abbreviated using `...`, since the actual output is much larger):

```
[[0.0, 3.0, 3.0, ..., 3.0], [0.0, 3.0, 0.0, ..., 3.0], ..., [25.0, 25.0, 25.0,
..., 30.0]]
```

For this function:

1. As usual with questions that involve files, you may *not* assume that the file exists. If, for whatever reason, the grades file fails to open, your function should simply end.
2. You may assume that at least one grade exists per question, but not that each data point will be numerical (see last few lines of the midterm grades CSV file).
3. Your function must work for any grades file of this format, *regardless of how many questions the exam has*. You can assume that the question score columns always come last. (i.e. you will never have a column containing any other kind of data *after* the grade columns).

Now, write a second function, `generate_statistics_report()` that will:

- Accept two parameters:
  - **`grades_filepath`** (*`str`*): The address of the CSV file containing the grades data.
  - **`stats_filepath`** (*`str`*): The address of a file to be created containing the statistics of the exam. This parameter should default to `score_stats.csv` if the user does not specify a path.
- Use `get_list_of_grades()` to get a list of grades of each question from the `grades_filepath` file.
- Use this list of lists to calculate the **mean**, **standard deviation**, and **median** of ***each of the questions on the exam*** (`Q1_a` to `Q6`). To make this easier, import the **`statistics` Python module**, which contains functions to calculate the **mean**, **standard deviation**, and **median** of a list of numbers.
- Print these statistics onto a file (`stats_filepath`) in the following format:

```
Mean,Standard Deviation,Median
```

I.e. line 1 will contain the mean, standard deviation, and median of `Q1_a`, line 2 of `Q1_b`, and so on. So, the following:

```
def main():
    generate_statistics_report("Midterm1_scores.csv", "stats.csv")

main()
```

Will create the following file, called `stats.csv`, in your working directory (note that here, we've rounded the values to the second decimal point–this is optional):

```
Mean,Standard Deviation,Median
2.89,0.56,3.0
1.75,1.48,3.0
1.85,0.53,2.0
1.84,0.55,2.0
1.78,0.62,2.0
1.76,0.66,2.0
1.86,0.52,2.0
1.87,0.49,2.0
1.81,0.59,2.0
1.49,0.87,2.0
1.8,0.6,2.0
1.61,0.79,2.0
1.94,0.35,2.0
1.93,0.37,2.0
11.19,4.08,13.0
23.93,3.31,25.0
20.26,8.24,22.0
```

Or, in tabulated form:

Mean	Standard Deviation	Median
2.89	0.56	3.0
1.75	1.48	3.0
1.85	0.53	2.0
1.84	0.55	2.0
1.78	0.62	2.0
1.76	0.66	2.0
1.86	0.52	2.0
1.87	0.49	2.0
1.81	0.59	2.0
1.49	0.87	2.0
1.8	0.6	2.0
1.61	0.79	2.0
1.94	0.35	2.0
1.93	0.37	2.0
11.19	4.08	13.0
23.93	3.31	25.0
20.26	8.24	22.0

Please note that both functions must be defined in the same file.

Problem 5: *Matryoshka Lists*

The goal of our last problem is to turn a list full of numbers into a list of ascending lists, depending on its contents. Here's the algorithm:

1. The program will traverse the list starting at index 0.
2. The program will create a new temporary list for every section of the original list that includes numbers in **ascending** order.
3. When a number is encountered that is not in ascending order compared to the number preceding it, no additional items will be added to the temporary list, and the temporary list will be added to a "repository" list.
4. When the program reaches the end of the original list, the "repository" list of lists will be returned to the calling function.

Here's an example to make the behavior of our function, `get_matryoshka_list()`, clear:

```
def main():
    original_list = [1, 2, 3, 5, 20, 19, 3, 4, 7, 45, 100, 1, 1, 3]
    matryoshka_list = get_matryoshka_list(original_list)
```

```
    print(matryoshka_list)

main()
```

Output:

```
[[1, 2, 3, 5, 20], [19], [3, 4, 7, 45, 100], [1], [1, 3]]
```