NYU Tandon School of Engineering

CS-UY 1114 Spring 2023

Homework 05

Due: 11:59pm, March 22nd, 2023

Submission instructions

1. You should submit your homework on **Gradescope**.
2. For this assignment you should turn in 5 separate `.py` files named according to the following pattern: `hw5_q1.py`, `hw5_q2.py`, etc.
3. Each Python file you submit should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW5 - Q1 (etc.)
Date due: 2023-03-22, 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. 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, user-defined functions (*except* for `main()` if your instructor has covered it during lecture), string methods, file i/o, exception handling, dictionaries, lists, tuples, and/or object-oriented programming.

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

Problems

1. **Working With Limitations (`hw5_q1.py`)**
2. **Complementary Service (`hw5_q2.py`)**
3. **Lexicographical Trends (`hw5_q3.py`)**
4. **Read Between The Lines (`hw5_q4.py1`)**
5. **So It Goes When You Do Your Scales And Your Arpeggios (`hw5_q5.py`)**

Problem 1: *Working With Limitations*

Write a program that prompts the user for a string and prints the string in lowercase without using the `lower()`, `upper()`, `islower()` and `isupper()` string methods.

Your program must also tell the user how many letter were made lowercase, how many *letters* were left untouched, how many whitespace characters there were (i.e. `' '` and '`\n`'), and how many non-alphabetic characters the string contained. You must do this without using `isalnum()`, `isdigit()`, `isalpha()`, or any other **method** that checks for the type of character you are considering. In other words, all of these checks must be made *manually*.

For example, an execution of this program could look like this:

```
Say something: New Romantic Sailors!
new romantic sailors!
Number of changed letters: 3
Number of unchanged letters: 15
Number of whitespace characters: 2
Number of non-alphabetic characters: 3
```

Hint **ASCII**.

Problem 2: *Complementary Service*

Given two DNA sequences represented as string values, write a program that will:

1. Fuse the two sequences by adding a nucleotide from each in alternating order (i.e. `ACT` + `CA` = `ACCAT`). If any invalid nucleotides (i.e., not `A`, `C`, `T`, or `G`) are found in either sequence, you should not include them in the fused sequence. Instead, just keep track of how many of those invalid nucleotides you encountered.
2. Create a complement sequence from the new, fused sequence. (i.e `ACCAT` complements to `TGGTA`)
3. Print that complement sequence, as well all the amount of invalid nucleotides that we encountered.

Recall the DNA complements:

Nucleotide	Complement
A	T
C	G
T	A
G	C

Figure 1: **Nucleotide Complements**.

Here's an example to make it clearer:

```
Enter a DNA sequence: ACTGGGTAV
Enter a second DNA sequence: TTZAG
Fused sequence: TAGAACTCCCAT | Invalid characters: 2
```

Problem 3: *Lexicographic Trends*

Write a program that asks the user to input a string containing only lower case letters (you may assume that they will do so). Your program should determine if the input is ordered in **lexicographic decreasing order**. If it is *not* in decreasing order, then your program must also print the **location** at which the string stops being ordered in decreasing order.

For example, an execution would look like:

```
Please enter a string of lowercase letters: pkgba
pkgba is decreasing.
```

```
Please enter a string: abgcp
abgcp is not decreasing
It stopped being lexicographically decreasing at location 1
```

Problem 4: *Read Between The Lines*

Let's say you wanted to decode a message that used the following encryption: Starting from the last character of the sentence, you must read every X-th letter. If the character that you land on is a number, you must skip it. For instance:

```
Enter an encoded string: !thnsdosdhdft7g68yyrop
Enter a key: 2
Your message is 'python!'
```

Above, we start at the last character, `p`, skip 2 (as denoted by the decryption key), and ignore the one numeric character we landed on (`6`):

```
STEP 0:
!thnsdosdhdft7g68yyrop

STEP 1:
! [th] n [sd] o [sd] h [df] t [7g 6 8y] y [ro] p

STEP 2:
! n o h t y p

STEP 3
!nohtyp

STEP 4:
python!
```

You may assume that both user inputs will always be valid ones. You may **not** use the `reverse()` string method.

Problem 5: *So it goes when you do your scales and your arpeggios*

Note: The instructions for this problem might look really long, but the actual code that you need to write isn't long.

An **arpeggio** is a musical structure wherein one plays a series of notes, one after the other, usually at even intervals. For example, if you wanted to play an arpeggio based on the **C-major** chord, you would play the following notes, one after the other:

```
C -> E -> G -> C -> E -> G -> C -> E -> G
```

Figure 1: The C-major chord being arpeggiated in the "up" direction 3 times.

In music production, an **arpeggiator** is a type of synthesiser that collection a number of notes from the user and plays those notes over and over again, in any direction and as many times that the user may choose. A great example that makes awesome use of arpeggiators is the track ***Resurrections*** by **Lena Raine**, from the ***Celeste*** original soundtrack.

We will be simulating a user entering notes into an arpeggiator, which will then arpeggiate those notes either up or down as many times as the user wants. For this, we will use a module that we have created for you, included in the file `arpeggiator.py`, which you can download **here**. You do **not** need to know how this module works—you only need to use its contents, and it *must* be in the same folder as your `hw5_q5.py` file. Note that, on IDLE, importing `arpeggiator` may not work. If this happens to you, ask your CAs during office hours on how to fix it.

You can find the `arpeggiator` module documentation **here**.

Our goal is the following:

1. Ask the user to enter any notes into the arpeggiator. The arpeggiator will check whether these notes are actual valid musical notes, so you don't need to worry about checking yourself. When the user is done entering notes, they should enter the word `"DONE"` (caps-sensitive). The user *must* do this to proceed.
2. Next, the user will enter a direction in which they would like the arpeggiator to play. Their choices are limited to the characters `'U'` for "up" and `'D'` for "down". The user must enter either of these characters to proceed.
3. Next, the user must enter how many times they would like their arpeggiator to play. The user must enter a positive, non-zero number to proceed.
4. Finally, the program should play the arpeggiator as many times and in the direction that the user selected.

Once you're entire solution is complete, sample executions might look like this (note that I added some `print()` statements for clarity; you may format your output however you like):

```
Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: C
Note 'C' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: E
Note 'E' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: G
Note 'G' added!
```

```
Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: E
Note 'E' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: C
Note 'C' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: DONE

Arpeggiator (notes: C, E, G, E, C)

Enter an arpeggiator direction [U/D] u
Enter an arpeggiator direction [U/D] d
Enter an arpeggiator direction [U/D] U

How many times do you want your arpeggiator to play? 5

~C
~~~E
~~~~~G
~~~E
~C
~C
~~~E
~~~~~G
~~~E
~C
~C
~~~E
~~~~~G
~~~E
~C
~C
~~~E
~~~~~G
~~~E
~C
~C
~~~E
~~~~~G
~~~E
~C
```

```
Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: A
Note 'A' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: Fb
WARNING: 'Fb' is not a valid note.
VALID NOTES: Ab, A#, A, Bb, B, C#, C, Db, D#, D, Eb, E, F#, F, Gb, G#, G

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: D#
Note 'D#' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: Gb
```

```
Note 'Gb' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: L#
WARNING: 'L#' is not a valid note.
VALID NOTES: Ab, A#, A, Bb, B, C#, C, Db, D#, D, Eb, E, F#, F, Gb, G#, G

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: B-Sharp
WARNING: 'B-Sharp' is not a valid note.
VALID NOTES: Ab, A#, A, Bb, B, C#, C, Db, D#, D, Eb, E, F#, F, Gb, G#, G

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: D
Note 'D' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: E
Note 'E' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: F
Note 'F' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: E
Note 'E' added!

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: END
WARNING: 'END' is not a valid note.
VALID NOTES: Ab, A#, A, Bb, B, C#, C, Db, D#, D, Eb, E, F#, F, Gb, G#, G

Enter a musical note (i.e. A, Db, C#, etc.) or 'DONE' to end: DONE

Arpeggiator (notes: A, D#, Gb, D, E, F, E)

Enter an arpeggiator direction [U/D] D

How many times do you want your arpeggiator to play? -4
How many times do you want your arpeggiator to play? (Must be positive amount)
3

~E
~~~F
~~~~~E
~~~~~~~D
~~~~~Gb
~~~D#
~A
~A
~~~D#
~~~~~Gb
~~~~~~~D
~~~~~E
~~~F
~E
~E
~~~F
~~~~~E
~~~~~~~D
~~~~~Gb
```

```
~~~D#
~A
```

How on earth do we do this? Check the documentation for the `arpeggiator` module in the next page.

Setting Up Your File

At the top of your `hw5_q5.py` file, type the following four lines:

```
import arpeggiator

ARPEGGIATOR = arpeggiator.Arpeggiator()
UP = arpeggiator.Direction.UP
DOWN = arpeggiator.Direction.DOWN
```

You don't have to worry about how `ARPEGGIATOR`, `UP`, and `DOWN` work behind the hood—you simply have to use them in your solution.

The `add_note()` Method

You can use `ARPEGGIATOR` the way you use `math` or `random` (i.e. you can use its methods). For example, if you want to input a note into the arpeggiator, you would use its `add_note()` method:

```
ARPEGGIATOR.add_note("Eb")          # Adding an E-flat into the arpeggiator
ARPEGGIATOR.add_note("Bb")          # Adding a B-flat into the arpeggiator
ARPEGGIATOR.add_note('C')           # Adding a C into the arpeggiator
ARPEGGIATOR.add_note("G#")          # Adding a G-sharp into the arpeggiator
ARPEGGIATOR.add_note('G')           # Adding a G into the arpeggiator
ARPEGGIATOR.add_note(("D-minor"))   # Adding an invalid note to the arpeggiator
```

Output (this output is generated by `ARPEGGIATOR`—you do ***NOT*** need to print this yourself):

```
Note 'Eb' added!

Note 'Bb' added!

Note 'C' added!

Note 'G#' added!

Note 'G' added!

WARNING: 'D-minor' is not a valid note.
VALID NOTES: Ab, A#, A, Bb, B, C#, C, Db, D#, D, Eb, E, F#, F, Gb, G#, G
```

You can confirm which notes you have added to your arpeggiator by printing `ARPEGGIATOR`:

```
print(ARPEGGIATOR)
```

Output:

```
Arpeggiator (notes: Eb, Bb, C, G#, G)
```

The `play()` Method

You can also play your arpeggiator by using its `play()` method:

```
ARPEGGIATOR.play()
```

Output:

```
In order:
~G
~~~G#
~~~~~C
~~~Bb
~Eb
```

The default direction of the `play()` method is `UP`,but you can also ask it to play it backwards by passing `DOWN` into its parentheses:

```
print("In order:")
ARPEGGIATOR.play()

print("\nIn Order:")
ARPEGGIATOR.play(UP)

print("\nBackwards:")
ARPEGGIATOR.play(DOWN)
```

Output:

```
In order:
~G
~~~G#
~~~~~C
~~~Bb
~Eb

In Order:
~G
~~~G#
~~~~~C
~~~Bb
~Eb
```

```
Backwards:
~Eb
~~~Bb
~~~~~C
~~~G#
~G
```