NYU Tandon School of Engineering

CS-UY 1114 Fall 2022

Homework 07

Due: 12:00pm, Friday, November 11th, 2022

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: `hw7_q1.py` and `hw7_q2_3.py`, etc.
3. Each Python file you submit should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW7 - Q1 (depending on the file name)
Date due: 2022-11-11, 12:00pm
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.*

The use of `eval()` and `break` are no longer permitted in this class.

For this specific assignment, you may use everything we have learned up to, **and including**, lists 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, file i/o, exception handling, dictionaries, and/or object-oriented programming.

Problems

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 3 and 4 should both be written in the same file, `hw7_q2_3.py`. Your header's second line should thus look like this `Assignment / Part: HW7 - 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: *Dance Dance Revolution (Memory Remix)*

Note: You will need to have the file `dance_dance_revolution.py` included in your working directory for this problem.

There's a popular memory game out there that involves memorising a series of directions given to you at random by the computer, and you have to recall the order from memory. A classic example of this game can be seen in Pokémon™ Stadium 2's ***Clefairy Says* mini game**.

We're going to simulate this game by using a custom module that we have prepared for you, `dance_dance_revolution`. You may find and download it **here**. All you need to get started is to add the following two lines at the top of your file:

```
from dance_dance_revolution import DanceDanceRevolution

GAME = DanceDanceRevolution()
```

Note: If you're among the unfortunate few whose IDLE won't let them import local modules, please visit office hours so that the CAs can help you work around it.

The constant `GAME` contains the memory game that we will be interacting with. In order to do so, we need to write three functions:

1. `get_game_parameters()` (*sig.* *`None -> (int, float)`*): This function accepts no parameters, but will rather do the following...
    1. Ask the user to enter a positive integer value that will represent the **amount** of steps that the user will have to memorise. The program must continue asking the user to enter a number until they enter a positive, non-zero integer. You may assume that the user will always enter integers for this step.
    2. Ask the user to enter a positive integer value that will represent the **speed** at which the steps that the user will have to memorise will appear on screen. The program must continue asking the user to enter a number until they enter a positive, non-zero number. You may assume that the user will always enter a numerical value for this step.
    3. Return both the amount and speed of the steps as a tuple. Check out the sample behaviour below:

```
print(get_game_parameters())
```

Output (yours doesn't have to look exactly the same):

```
How many steps would you like to memorize? (positive non-zero integers only) -4
WARNING: Please enter a positive non-zero integer value.
How many steps would you like to memorize? (positive non-zero integers only) 0
WARNING: Please enter a positive non-zero integer value.
How many steps would you like to memorize? (positive non-zero integers only) 4
How fast would you like the game to run? (positive non-zero numerical values
only) -5
WARNING: Please enter a positive non-zero numerical value.
How fast would you like the game to run? (positive non-zero numerical values
only) 0.0
WARNING: Please enter a positive non-zero numerical value.
How fast would you like the game to run? (positive non-zero numerical values
only) 0.5
(4, 0.5)
```

2. `get_user_answers()` (*sig.* `None -> list[str]`): This function accepts no parameters, but will instead do the following:
   1. Prompt the user to enter either the string `'U'` for "up", `'D'` for "down", `'L'` for "left", `'R'` for "right". This function must continue doing this until the user enters the string `"DONE"` instead of anything else.
   2. Every time the user enters something (unless it is `"DONE"`), you must collect that input inside a **list**. If the user enters `"DONE"` but hasn't entered at least one piece of input, the program must continue asking for input.
   3. After the user enters `"DONE"` your program must return this list of inputs. Check out the sample behaviour below:

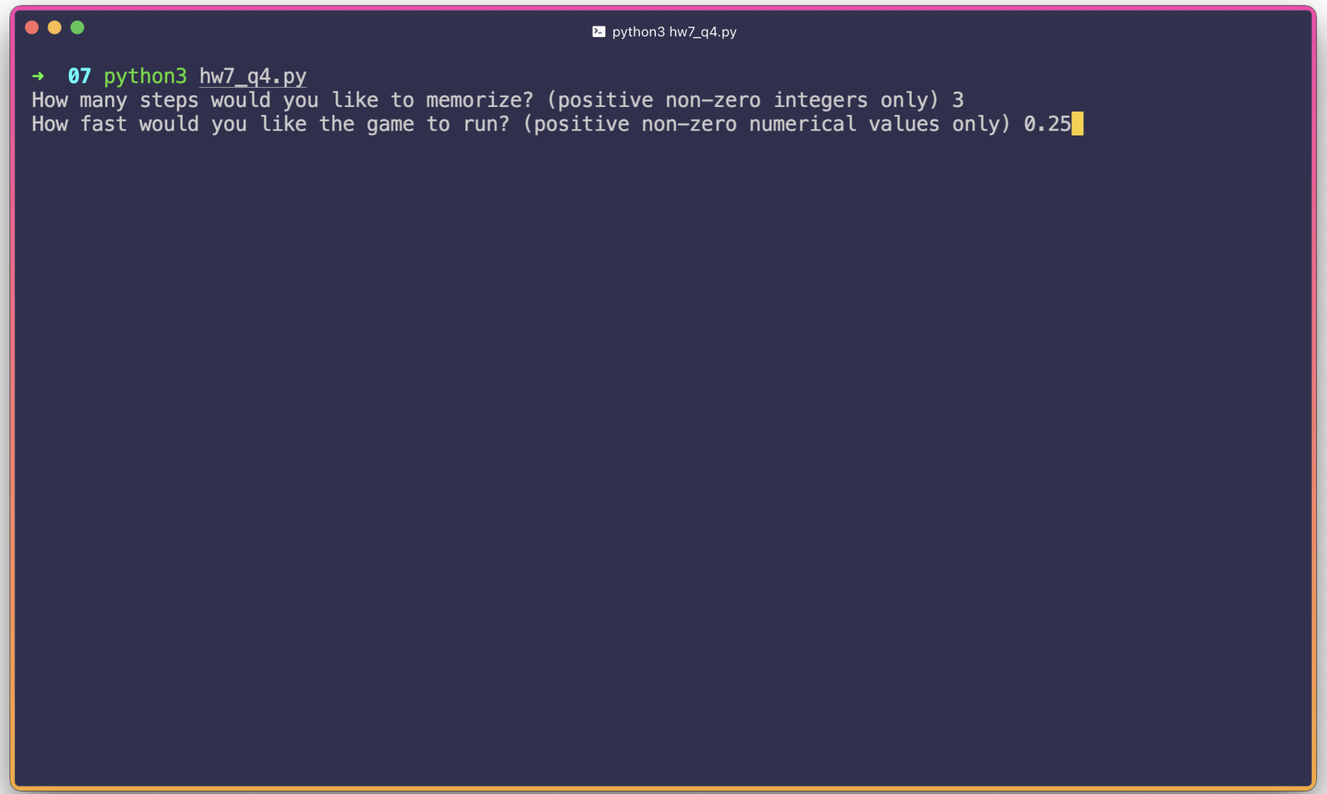
```
print(get_user_answers())
```

```
Enter a direction (U/D/L/R) or 'DONE' to finish: DONE
Please enter at least one answer before selecting 'DONE'.
Enter a direction (U/D/L/R) or 'DONE' to finish: U
Enter a direction (U/D/L/R) or 'DONE' to finish: d
Enter a direction (U/D/L/R) or 'DONE' to finish: Left
Enter a direction (U/D/L/R) or 'DONE' to finish: Done
Enter a direction (U/D/L/R) or 'DONE' to finish: DONE
['U', 'd', 'Left', 'Done']
```

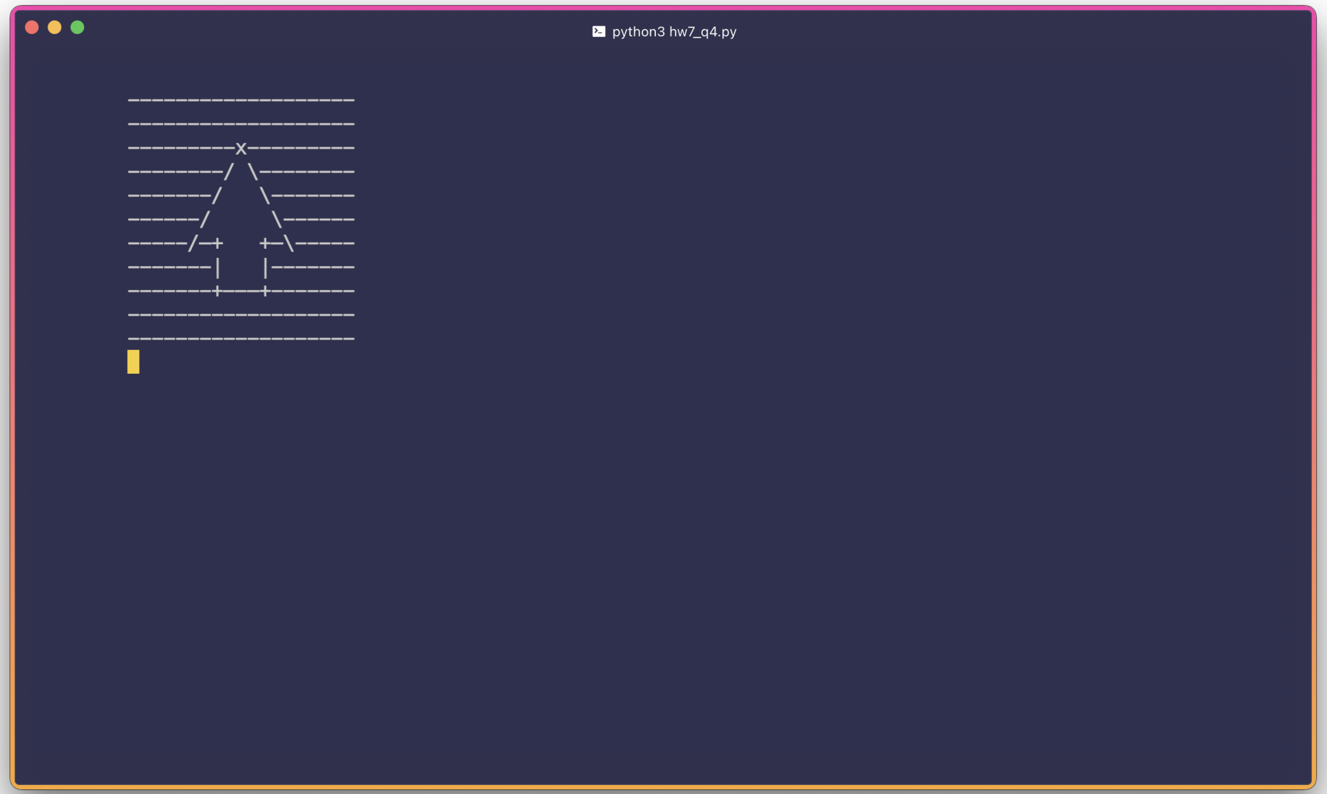
3. `run_game()` (*sig.* `None -> None`): This function accepts no parameters and returns no values. Instead, it puts together our two functions from above with the methods of our `GAME` object. The documentations for the `GAME` object can be found below, but here is an overview of the general steps:
   1. Ask the user for the game's parameters.

2. Enter those parameters into the `GAME` object.
3. Have `GAME` display the game to the user using the parameters that they entered.
4. Ask the user to enter the pattern than they memorised.
5. If the pattern matches the pattern that the game displayed, the game will display a "win" message. Otherwise, it will display an "lose" message.

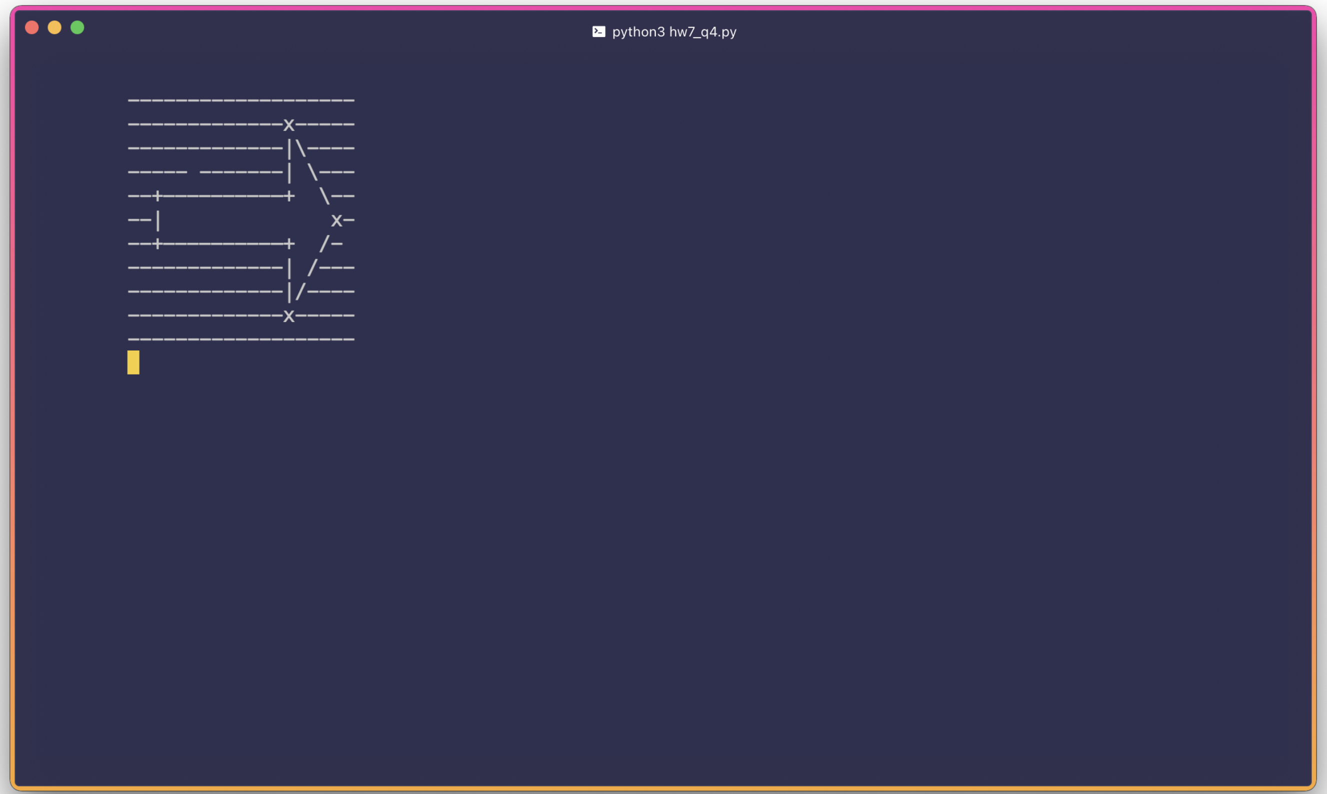
Running `run_game()`, if implemented correctly, should look something like this:



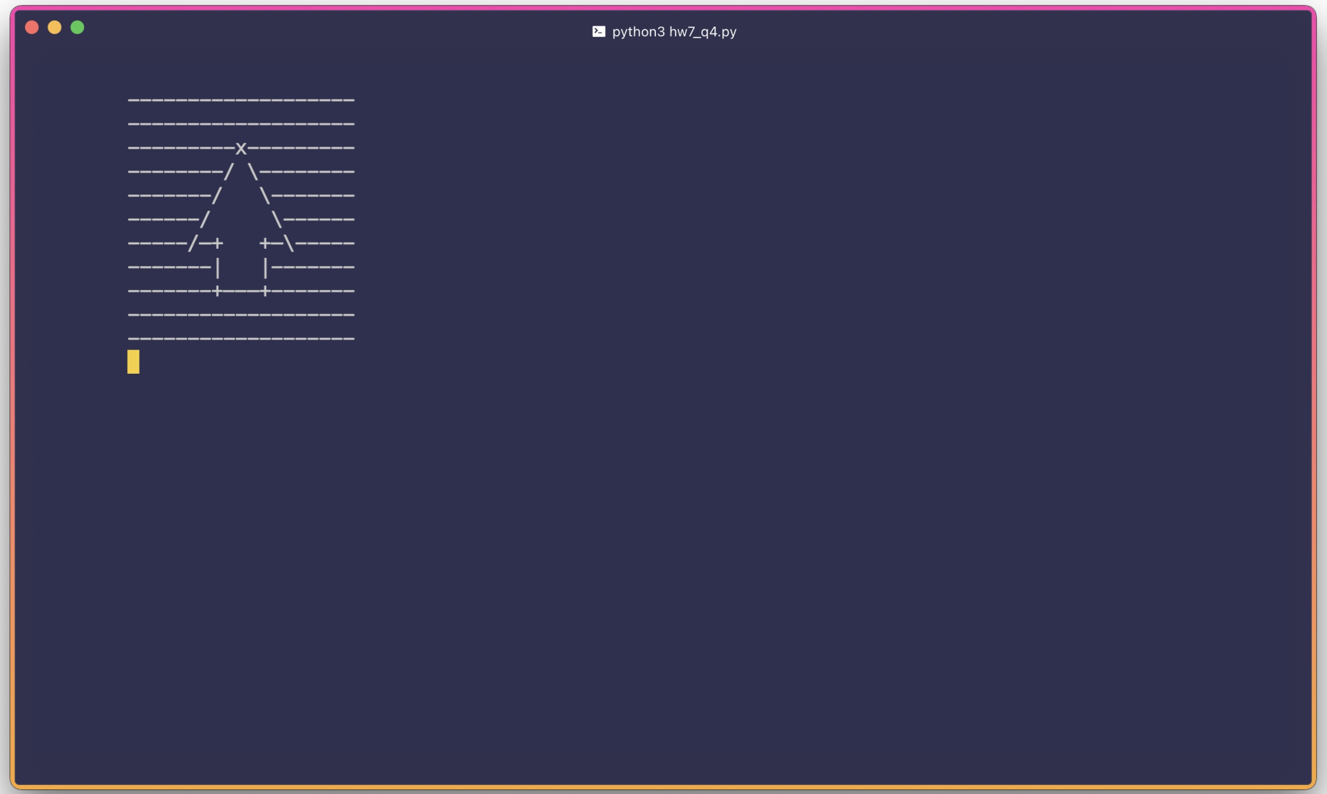
```
➜  07 python3 hw7_q4.py
How many steps would you like to memorize? (positive non-zero integers only) 3
How fast would you like the game to run? (positive non-zero numerical values only) 0.25
```



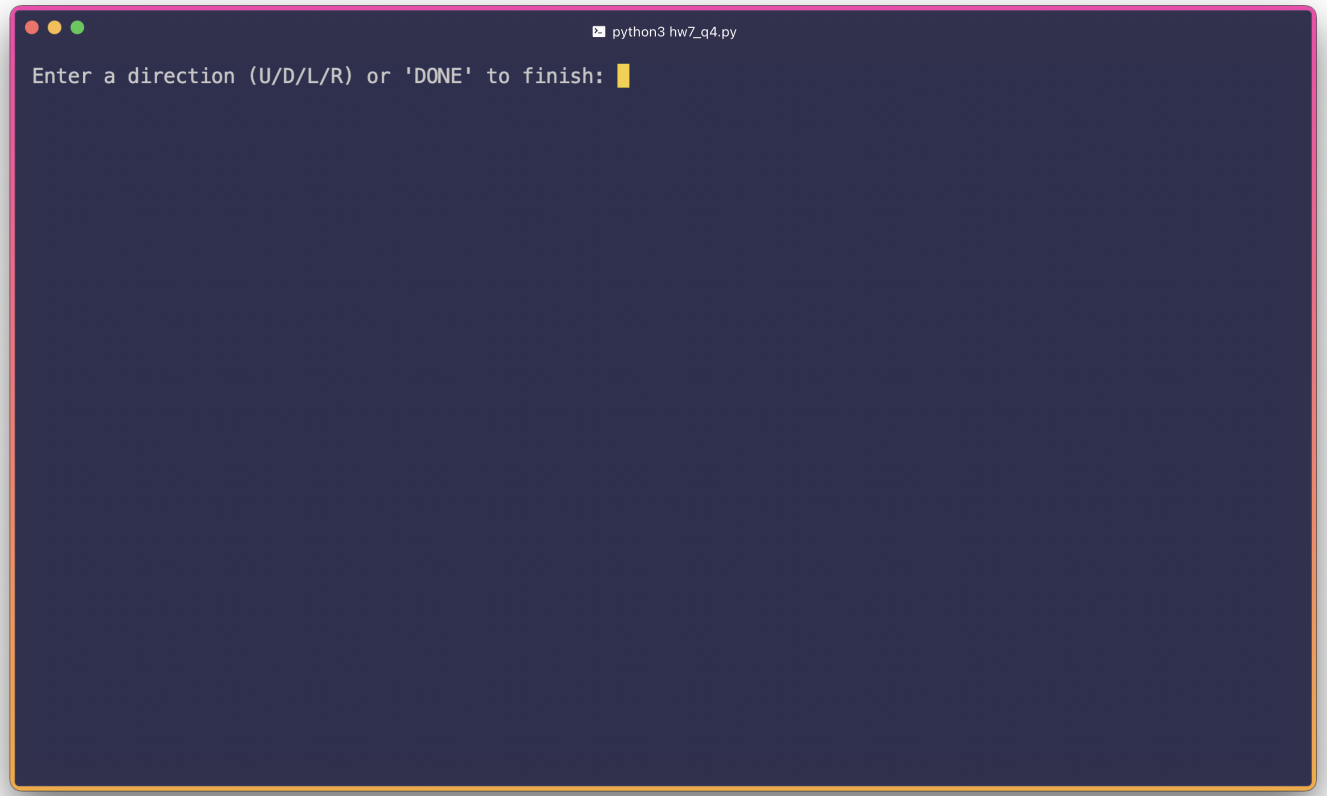
```
-------------------
-------------------
---------x---------
--------/ \--------
-------/   \-------
------/     \------
-----/-+   +-\-----
-------|   |-------
-------+---+-------
-------------------
-------------------
```



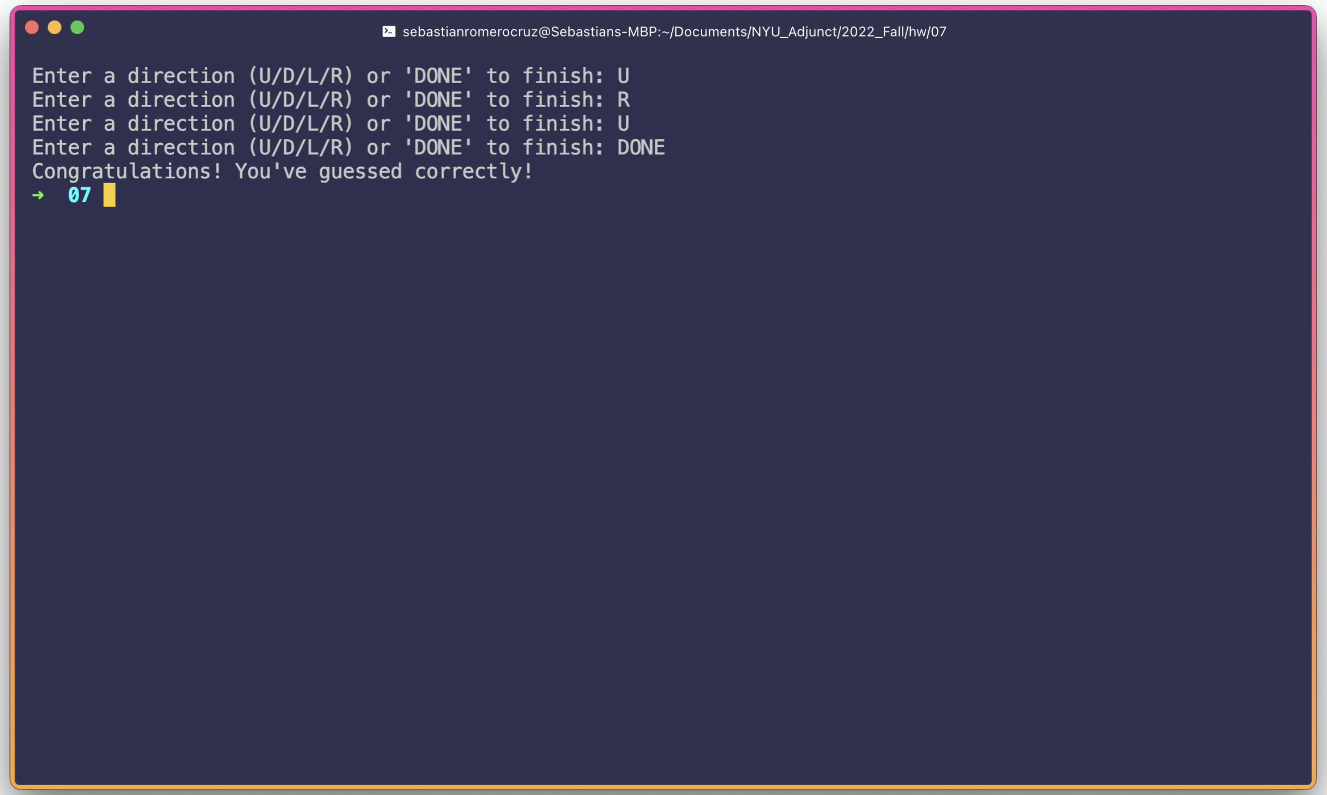
```
-------------------
-------------x-----
-------------|\----
----- -------| \---
--+----------+  \--
--|              x-
--+----------+  /-
-------------| /---
-------------|/----
-------------x-----
-------------------
```



```
-------------------
-------------------
---------x---------
--------/ \--------
-------/   \-------
------/     \------
-----/-+   +-\-----
-------|   |-------
-------+---+-------
-------------------
-------------------
```



```
Enter a direction (U/D/L/R) or 'DONE' to finish:
```


sebastianromerocruz@Sebastians-MBP:~/Documents/NYU_Adjunct/2022_Fall/hw/07

```
Enter a direction (U/D/L/R) or 'DONE' to finish: U
Enter a direction (U/D/L/R) or 'DONE' to finish: R
Enter a direction (U/D/L/R) or 'DONE' to finish: U
Enter a direction (U/D/L/R) or 'DONE' to finish: DONE
Congratulations! You've guessed correctly!
➜  07
```

Figures 1 - 6: A sample run of `run_game()` from a Terminal (MacOS).

`GAME` Documentation

The methods available to you in `GAME` are as follows:

- **`set_speed()`** (*sig.* *`float -> None`*): This method accepts a numerical value and sets the game's speed to it. The higher the value, the faster the game runs. This method will raise an error if this value is not a positive, non-zero numerical value.
- **`set_amount()`** (*sig.* *`int -> None`*): This method accepts an integer value and has the game, when it is run, display that many directional arrows. For example, if the user enters a `4`, the game will display four directional arrows that the user will have to memorise. This method will raise an error if this value is not a positive, non-zero integer.
- **`play_sequence()`** (*sig.* *`None -> None`*): Every time this method is run, the user's command line will be cleared and a random sequence of arrows will be displayed one after the other at some speed (determined by `set_speed()` and `set_amount()`). The sequence is different every time this method is called, and `GAME` saves the value of the latest-generated sequence.
- **`check_answers()`** (*sig.* *`list[str] -> bool`*): The method accepts a list of `str` objects, which it considers to be the user's answer to the latest round of memorisation. It will check whether the user's sequence matches the latest-generated sequence. If it does, this method will return `True`. If the list differs in any way from the correct sequence, it will return `False`. This method will raise an error if any of the objects inside the list parameter are not `str` objects.

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:

```python
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]]
```